FlexiViT: One Model for All Patch Sizes

Lucas Beyer\textsuperscript{1} Pavel Izmailov\textsuperscript{1,3} Alexander Kolesnikov\textsuperscript{1} Mathilde Caron\textsuperscript{2} Simon Kornblith\textsuperscript{1} Xiaohua Zhai\textsuperscript{1} Matthias Minderer\textsuperscript{1} Michael Tschannen\textsuperscript{1} Ibrahim Alabdulmohsin\textsuperscript{1} Filip Pavetic\textsuperscript{1}

Google Research

Figure 1. FlexiViT is a standard ViT model that sees randomized patch sizes, hence sequence lengths, during training. The patch embedding weights are resized adaptively for each patch size and the model weights are shared as-is across all patch sizes.

Abstract

Vision Transformers convert images to sequences by slicing them into patches. The size of these patches controls a speed/accuracy tradeoff, with smaller patches leading to higher accuracy at greater computational cost, but changing the patch size typically requires retraining the model. In this paper, we demonstrate that simply randomizing the patch size at training time leads to a single set of weights that performs well across a wide range of patch sizes, making it possible to tailor the model to different compute budgets at deployment time. We extensively evaluate the resulting model, which we call FlexiViT, on a wide range of tasks, including classification, image-text retrieval, open-world detection, panoptic segmentation, and semantic segmentation, concluding that it usually matches, and sometimes outperforms, standard ViT models trained at a single patch size in an otherwise identical setup. Hence, FlexiViT training is a simple drop-in improvement for ViT that makes it easy to add compute-adaptive capabilities to most models relying on a ViT backbone architecture. Code and pre-trained models are available at github.com/google-research/big_vision.
1. Introduction

Vision Transformers (ViTs) cut images into non-overlapping patches and perform all computations on tokens created from these patches. This “patchification” procedure represents a significant shift away from the previously dominant convolutional neural network (CNN) approach [32], where an image is processed with small local and typically overlapping filters. Patchification has unlocked new capabilities, such as (random) dropping of image patch tokens [10, 20, 44, 53, 61], adding specialized tokens for new tasks [54, 56] or mixing image tokens with tokens from other modalities [1, 38, 64].

Despite the importance of patchification for ViT models, the role of the patch size has received little attention. While the original ViT paper [15] works with three patch sizes (32×32, 16×16, and 14×14 pixels), many follow-up works fix the patch size at 16×16 pixels [54, 55, 65]. In this work, we show that the patch size provides a simple and effective lever to change the compute and predictive performance of a model, without changing model parametrization. For example, a ViT-B/8 model achieves 85.6% top-1 accuracy on ImageNet1K with 156 GFLOPs and 85 M parameters, while a ViT-B/32 model achieves only 79.1% accuracy with 8.6 GFLOPs and 87 M parameters. Despite the major difference in performance and compute, these models have essentially the same parametrization. However, standard ViT models perform well only at the patch size that they have been trained at. Tuning the patch size therefore requires complete re-training of the model.

To overcome this limitation, we propose FlexiViT, a flexible ViT which matches or outperforms standard fixed-patch ViTs across a wide range of patch sizes with no added cost. To train FlexiViT, we randomize the patch size during training, and resize the positional and patch embedding parameters adaptively for each patch size, as shown in Figure 1. These simple modifications are already sufficient for strong performance, but we also propose an optimized resizing operation and a training procedure based on knowledge distillation which achieves even better results.

We demonstrate the efficiency of FlexiViT models in many downstream tasks, such as image classification, transfer learning, panoptic and semantic segmentation, image-text retrieval and open-world recognition, and provide a general recipe for flexifying existing ViT-based training setups. Furthermore, we show that flexibility of the backbone, i.e. strong performance across patch sizes, is often preserved even after fine-tuning with a fixed patch size. We leverage this observation to perform resource-efficient transfer learning: we finetune the model cheaply with a large patch size, but then deploy it with a small patch size for strong downstream performance. We further show that flexible patch size can be used to accelerate pre-training.

To explain the effectiveness of FlexiViT, we analyze the model’s representations. We find that the representations are often similar across different patch sizes, especially in the deeper layers. Finally, we show that FlexiViT outperforms alternative architectural ways of controlling the performance-compute trade-off in ViT models.

2. Related work

Several recent works explore improving ViT’s efficiency by exploiting patchification. Some suggest removing tokens, either in randomized [20] or structured [10] fashion throughout training. Others aim to quantify a token’s importance and remove the least important ones, during [44, 61] or after [53] training. [57] trained a cascade of Transformers using increasing number of tokens to allow early exiting during inference. Conversely, we always keep all tokens and do not discard any information. It may be possible to combine such approaches with FlexiViT in future work.

More similar to our approach, the Neural Architecture Search (NAS) field is converging towards training one “supernet” from which individual, differently-shaped “subnets” can be extracted [8, 18, 63]. Since these works aim for changes in most or all model dimensions, they usually involve multiple specialized architectural additions. SuperViT [34] is most related to FlexiViT as it patchifies an image at multiple scales, passes all these patches to ViT, while dropping random tokens [20] to reduce the sequence length. In contrast to the aforementioned works, our sharpened focus on ViT’s patch size only, allows benefiting from existing pretrained models, future ViT improvements, and is an easy drop-in to any existing training procedure.

Matryoshka representation learning [31] proposes training models whose output vector contains meaningful sub-vectors. This can be seen as the complement of FlexiViT.

3. Making ViT flexible

In this section we show that standard ViT models are not flexible, and introduce the FlexiViT model and training procedure in the supervised image classification setting. We perform all experiments in this section on the public ImageNet-21k dataset [46]. We use the base (ViT-B) scale model and unregularized light2 setting from [50], and train the models for 90 epochs following [36].

3.1. Background and notation

FlexiViT is based on the Vision Transformer (ViT) architecture [15]. Here, we briefly describe the ViT architecture and introduce the necessary notation.

Consider an image \( x \in \mathbb{R}^{h \times w \times c} \), where \((h, w, c)\) are the width, height and number of channels respectively. ViT first tokenizes the input image into a sequence of \( s \) patches \( x_i \in \mathbb{R}^{p \times p \times c} \), where \( i \in \{1, \ldots, s\} \). We refer to this procedure as patchification and illustrate it in Figure 1.
The sequence length \( s = \lfloor h/p \rfloor \cdot \lfloor w/p \rfloor \) is the number of patches (or tokens) after patchification and controls the amount of compute used by the ViT: self-attention scales as \( O(s^2) = O(h^4) = O(w^4) \), i.e. quartically in terms of image height (or width).

Next, we compute patch embeddings \( e_i = (e_i^k)_{k=1}^d \in \mathbb{R}^d \) for each patch \( x_i; e_i^k = \langle x_i, \omega_k \rangle = \text{vec}(x_i)^T \text{vec}(\omega_k) \), where \( \omega_k \in \mathbb{R}^{p \times p \times c} \) are the patch embedding weights, \( \langle \cdot, \cdot \rangle \) denotes the dot product, and \( \text{vec} \) is the operation flattening a multi-dimensional array to a vector. Finally, we add learned position embeddings \( \pi_i \in \mathbb{R}^d \) to the patch embeddings \( t_i = e_i + \pi_i \). We then pass the sequence of \( s \) tokens \( t_i \) as input to the Transformer encoder, as illustrated in Figure 1.

In summary, for a given image size \( h \times w \), the patch size \( p \) determines the length \( s \) of the input sequence to the Transformer model: smaller patch sizes correspond to longer input sequences and slower, more expressive models. Following [15], we denote ViT models as ViT-\( S/p \), where \( S \in \{ S, M, B, L, \ldots \} \) is the model scale (small, medium, base, large, \ldots) and \( p \) is the patch size. Note that there are only two parts of the model where the parameter vectors depend on the patch size: the patch embedding weights \( \omega_k \) and the position embedding \( \pi \). In the following sections, we will develop a flexible ViT model which works simultaneously for any patch size.

### 3.2. Standard ViTs are not flexible

We first show that evaluating a standard pre-trained ViT model at different patch sizes yields poor performance. In order to change the patch size, we simply resize the patch embedding weights \( \omega \) and the position embeddings \( \pi \) with bilinear interpolation. For the position embeddings, this resize approach was already proposed in the original ViT paper [15] to fine-tune at higher resolution.

The result is shown in Figure 3, where we see that the performance of standard ViT models (dashed and dotted lines) rapidly degrades as the inference-time patch size departs from the one used during training.

### 3.3. Training flexible ViTs

In Figure 3 we also show the performance of our FlexiViT-B model (solid line), which matches both ViT-B/16 and ViT-B/30 when evaluated at their training patch sizes, and significantly outperforms them for all other patch sizes. This model was trained in the same setting as the ViT-B/16 and ViT-B/30 models, except that at each step of training, the patch size was chosen uniformly at random from a set of pre-defined patch sizes.\(^2\) In order to do so, two small changes to the model and training code are necessary.

First, the model needs to define an underlying parameter shape for \( \omega \) and \( \pi \). The learnable parameters are of that shape, and resized on-the-fly as part of the model’s forward pass. We show in Appendix B that the exact shape of these underlying learnable parameters does not matter much, and we use an underlying size of \( 32 \times 32 \) for patches and \( 7 \times 7 \) for position embeddings in all experiments.

Second, to have a large variety of patch sizes that perfectly tile the image, we use an image resolution of 240\( p \) px, which allows for patch sizes \( p \in \{ 240, 120, 60, 48, 40, 30, 24, 20, 16, 15, 12, 10, 8, 6, 5, 4, 2, 1 \} \), of which we use all between 48 and 8, inclusive.\(^3\) At each iteration we sample \( p \) from the uniform distribution \( P \) over these patch sizes.

These are all the changes necessary to flexify an existing ViT training procedure. Algorithm 1 summarizes them.

Note that changing the patch size is related to, but not identical to, changing the image size. The patch size is purely a change to the model while changing the image size may drastically reduce the available information. This distinction is further explored in Section 3.4.

We explore two alternative ways to flexify ViTs in Sec-

---

\(^2\)We sample patch sizes uniformly in most experiments. Some early runs used a distribution which slightly favors intermediate patch sizes. Later experiments showed that the distribution makes little difference (Appendix C). We therefore did not re-run the early experiments.

\(^3\)Perfect tiling may not be strictly necessary, and it may be fine to use arbitrary patch sizes and ignore a small border of the image. For simplicity, we focus on the perfect tiling setting.
Figure 4. Various ways of “resizing” ViTs. We load a ViT-B/8 from [50] trained on 224\(^2\) px, resize patch-embeddings and input images by the same factor, and compute validation accuracy. PI-resize is the only method that stays accurate when upscaling.

### 3.4. How to resize patch embeddings

Consider a patch \(x \in \mathbb{R}^{p \times p}\) of the input image, and the patch embedding weights \(\omega \in \mathbb{R}^{p \times p}\) and let’s assume a simple scenario when we are dealing with non-negative values. If we resize both the patch and the embedding weights with bilinear interpolation, the magnitude of the resulting tokens will differ greatly; for example \(\langle x, \omega \rangle \approx \frac{1}{4} \langle \text{resize}^{2p}(x), \text{resize}^{2p}(\omega) \rangle\). We hypothesize that this dramatic change in token norm is part of the reason of ViT’s inflexibility, and an inductive bias that hinders learning of a single FlexiViT. Ideally, as long as there is no loss of information during resizing, the patch embeddings \(e_i = \langle x, \omega \rangle\) after resizing both the input \(x\) and the embedding \(\omega\) should remain the same.

One way to achieve this equality is to normalize the tokens right after their embedding, either explicitly or by using a LayerNorm [2] module. However, this approach requires changing the model architecture and is not compatible with existing pre-trained ViTs. Further, it does not exactly preserve the patch embeddings. As we will show, there is a more principled way of achieving this goal, which is compatible with existing pre-trained models and does not require any architectural change.

First, we note that the linear resize operation introduced in Section 3.2 can be represented by a linear transformation:

\[
\text{resize}^{p^*}_p(o) = B^{p^*}_p \text{vec}(o),
\]

where \(o \in \mathbb{R}^{p \times p}\) is any input, and \(B^{p^*}_p \in \mathbb{R}^{p^* \times p^2}\). We resize channels of multi-channel inputs \(o\) independently.

Intuitively, we would like to find a new set of patch-embedding weights \(\hat{\omega}\) such that the tokens of the resized patch match the tokens of the original patch. Formally, we want to solve the optimization problem:

\[
\hat{\omega} \in \arg \min_{\omega} \mathbb{E}_{x \sim \mathcal{X}} \left[ \langle x, \omega \rangle - \langle Bx, \hat{\omega} \rangle^2 \right],
\]

where \(B = B^{p^*}_p\) and \(\mathcal{X}\) is some distribution over the patches. In case when we are increasing the patch size, i.e. \(p^* \geq p\), we can use \(\hat{\omega} = \text{Proj} \omega\) where \(P = B(B^T B)^{-1} = (B^T)^+\) is the pseudoinverse of \(B^T\):

\[
\langle Bx, \hat{\omega} \rangle = x^T B^T B(B^T B)^{-1} w = x^T w = \langle x, w \rangle.
\]

This way we match the patch embeddings exactly for all \(x\).

In the case of downsampling, i.e. when \(p^*_s < p\), the solution to the optimization problem in Eq. (2) will in general depend on the patch distribution \(\mathcal{X}\). In Appendix A.2, we show that for \(\mathcal{X} = \mathcal{N}(0, I)\), we recover the pseudoinverse \(\hat{\omega} = P \omega = (B^T)^+ \omega\) as the optimal solution\(^4\). To sum up, we define PI-resize (pseudoinverse resize) as:

\[
\text{PI-resize}^{p^*}_p(w) = \left( (B^{p^*}_p)^T \right)^+ \text{vec}(\omega) = P^{p^*_s} \text{vec}(\omega),
\]

where \(P^{p^*_s} \in \mathbb{R}^{p^*_s \times p^2}\) is the matrix corresponding to the PI-resize transformation. The PI-resize operation resizes the patch embedding weights, serving as an inverse of the bilinear resize operation.

To experimentally validate the effectiveness of PI-resize and compare it to several alternative heuristics, including standard linear resize, we load a pre-trained ViT-B/8 model from [50] and evaluate it after resizing both the image and the model, thus preserving its sequence length \(s = (224/8)^2 = 784\). The results, shown in Figure 4, demonstrate that PI-resize maintains nearly constant performance when upsampled, and degrades gracefully when downsampled. None of the heuristics works as well as thoughtful PI-resize across the board.

For completeness, in Appendix A.1 we experimentally compare the remaining ways of dealing with variable patch sizes when one does not care about maintaining model compatibility. These methods include fixed normalization, LayerNorm, and learning separate parameters \(\omega\) for each patch size. Adding a LayerNorm works best, but otherwise, PI-resize and bilinear resize are among the best techniques.

### 3.5. Connection to knowledge distillation

Knowledge distillation [23] is a popular technique, where a typically smaller \textit{student} model is trained to mimic the predictions of a typically larger \textit{teacher} model. This can significantly improve the performance of the student model compared to standard label-supervised training [5, 12, 60].

\(^4\)We can also target the patch distribution in the data in place of \(\mathcal{X}\), producing a resize operation which depends on the data. In our preliminary experiments, we did not observe significant benefits from this approach.
Figure 5. **The effect of initialization** when distilling to FlexiViT.

It was recently shown that knowledge distillation corresponds to a much more challenging optimization problem than standard supervised training [5, 49], and that initializing the student close to the teacher simplifies alleviates this [49]. Unfortunately, this solution is impractical since the teacher usually has a different (larger) architecture than the student [5]. However, with FlexiViT, we can initialize a student FlexiViT with the weights of a powerful ViT teacher and significantly improve distillation performance.

Unless otherwise stated, the model we use for the remaining experiments in this paper is a FlexiViT-B initialized and distilled from the powerful ViT-B/8 model of [50]. At initialization, we PI-resize the teacher’s patch embedding weights to $32 \times 32$, and bilinearly resample its position embeddings to $7 \times 7$. We then train the student model following the FunMatch [5] approach, minimizing the KL-divergence between the predictions of the teacher and the student FlexiViT with a randomized patch size:

$$
\mathbb{E}_{x \in \mathcal{D}} \mathbb{E}_{p \sim \mathcal{P}} \ KL ( f_{\text{FlexiViT}}(x, p) \| f_{\text{ViT-B/8}}(x) ),
$$

where $f_{\text{FlexiViT}}(x, p)$ is the distribution over classes for the FlexiViT model on an input $x$ with patch size $p$, $f_{\text{ViT-B/8}}(x)$ is the predictive distribution of the teacher on the exact same input, $\mathcal{D}$ is the training data distribution with random flips, crops, and mixup, and $\mathcal{P}$ is the distribution over patch sizes used for training the FlexiViT model.

Figure 5 compares the effect of distilling using teacher initialization to random initialization and to supervised training from labels. The comparison was performed for 90 epochs and shows considerable benefits of this unique initialization capability of FlexiViT. Since distillation needs patience [5, 54], we additionally run for 300 and 1000 epochs, shown as pale green curves in the figure. FlexiViT matches the teacher’s performance at small patch sizes, and teacher initialization provide a large improvement in accuracy at the largest patch sizes. In the following sections, we use the FlexiViT that was trained for 300 epochs and train two fixed ViT-B/30 and ViT-B/16 models in the same setting (including the initialization) as baselines.

### 3.6. FlexiViT’s internal representation

Does FlexiViT process inputs with different patch sizes in similar ways? We investigate this by analyzing the model’s internal representations. We apply minibatch centered kernel alignment (CKA) [14, 28, 39], a widely-used approach for comparing representations within and across neural networks. For visualization purposes, we apply an arccosine transform to transform CKA/cosine similarity to proper metrics [58] and then perform t-SNE.

Results are shown in Figure 6. Feature map representations are similar across grid sizes from the first layer until the MLP sublayer of block 6. At the MLP sublayer of block 6, layer representations diverge, before converging again at the final block. By contrast, CLS token representations remain aligned across grid sizes. Thus, although internal representations of a substantial portion of FlexiViT differ by grid size, output representations are generally aligned.

### 4. Using pre-trained FlexiViTs

We have shown that ViTs can be trained flexibly without significant loss of *upstream* performance. Next, we verify that pre-trained FlexiViTs are still comparable to individual fixed patch-size ViTs when transferred to other tasks. We check this by transferring the single pre-trained FlexiViT with its patch size fixed to either $16^2$ or $30^2$ during transfer. We compare FlexiViT to ViT-B/16 and a ViT-B/30 models that were pre-trained using the same distillation setup as FlexiViT (Section 3.5), but with a fixed patch size. We perform this transfer on the following set of diverse tasks.

For each task, we provide more details along with many more results, all with the same take-away, in Appendix E.
**Classification** We fine-tune on small- (Pet [41], Flowers [40]) and medium-scale (CIFAR10, CIFAR100 [30], Food101 [7], SUN397 [59]) classification datasets following the setup of [15] at 240^2 px resolution.

**Locked-image Tuning (LiT)** We follow [66] to train a text model contrastively [24, 43] for the frozen FlexiViT, which we evaluate in terms of 0-shot classification and retrieval.

**Open-vocabulary detection** We test the transferability of FlexiViT to object detection using OWL-ViT [37], an open-vocabulary object detector based on image-text models such as LiT or CLIP [43]. We evaluate its zero-shot open-vocabulary detection performance on LVIS [19].

**Panoptic segmentation** The Universal Vision Model (UViM) is a general-purpose modeling approach for vision [27]. We train UViM on the COCO panoptic segmentation dataset [25, 35] and use FlexiViT as initialization for the image encoder in UViM.

**Semantic segmentation** We transfer to semantic segmentation following Segmenter’s linear decoder setup [51]. We report mean IoU for single scale evaluation and evaluate on Cityscapes [13] and ADE-20k [67].

**4.1. Results**

The results of these transfer experiments are summarized in Figure 7. Across the diverse set of tasks, a single FlexiViT model roughly matches the two fixed ViT models, barely lagging behind at large patch size and leading to a small or significant improvement at smaller patch size.

These results confirm that there is no significant downside in using a pre-trained FlexiViT, as opposed to pre-training multiple ViTs for different patch sizes.

**4.2. Resource-efficient transfer via flexibility**

FlexiViT enables a new way of making transfer learning more resource efficient, saving accelerator memory and compute. This is possible because, surprisingly, *flexibility is largely retained even after transfer at a fixed patch size.* We can therefore perform transfer training cheaply with large input patches (small input grid), but later deploy the resulting model using small patch sizes (large input grid). We preform experiments by transferring a FlexiViT-B model (pre-trained on ImageNet-21k with distillation) to the ImageNet-1k dataset, and use a similarly pretrained fixed ViT-B/30 model as the baseline. The pretrained FlexiViT works well at larger grid sizes even after fixed-size transfer. For example, we can perform relatively cheap finetuning at 8 × 8 grid size. When evaluated at 8 × 8 grid size, the model achieves 81.8% accuracy, but when evaluated at the 24 × 24 grid size, it achieves 85.3% top-1 accuracy gaining 3.5% accuracy at no additional training cost (Figure 8). More details on the finetuning setup can be found in the Appendix D.
5. Flexifying existing training setups

So far, we have focused on flexifying models during pre-training. We now show that existing pre-trained models can also be flexified during transfer to downstream tasks. Below, we flexify a diverse set of existing training setups.

5.1. Transfer learning

We use the same set of 6 transfer datasets from Section 4, with the same settings. We again show the results for SUN397 in Figure 9 and all other datasets in Appendix E. The difference is that we now also randomize the patch size during transfer, and evaluate the single resulting model at different patch sizes (x-axis, three groups of bars).

Flexible transfer of FlexiViT works best, but flexifying a fixed model during transfer also works surprisingly well, considering the very short training and low learning rate used for transfer. The baseline of a fixed-size model transferred at a fixed patch size and evaluated at that same size is indicated by a small horizontal line.

5.2. Multimodal image-text training

Next, we discuss two ways to flexify multimodal image-text training: FlexiLiT and FlexiCLIP. In FlexiLiT, we train a text tower to produce text embeddings that align well with visual embeddings from various patch sizes (B/flexi). LiT baselines with direct use of either ViT models are provided. Figure 10 shows zero-shot image to text retrieval results on the Flickr30k [42] dataset. FlexiLiT-B/flexi performs the best on average, while LiT with FlexiViT-B/30 and FlexiViT-B/16 both get very close results. Flexification additionally provides the possibility of fast transfer as discussed in Section 4.2. The LiT-ViT baselines shown in Figure 10 match FlexiLiT on the sequence length it has been trained for, but performance drops quickly when using a different sequence length during inference. We observe similar conclusions with a from-scratch image-text training setup, i.e. FlexiCLIP (see Appendix G for more results).

5.3. Open-vocabulary detection

Beyond image-level tasks, we find that flexification also works for object detection training. We modify the training of OWL-ViT to introduce flexible patch sizes as described in Algorithm 1. Similar to classification, flexible OWL-ViT detection models perform close to or better than fixed-size models at any patch size during inference (Figure 11). In addition, we find that for detection, the optimal patch size is not necessarily the smallest. When evaluated on a set of 35 detection datasets [33], inference-time tuning of the patch size leads to improved results over evaluation at the smallest patch size (Appendix E). This makes flexification especially valuable for detection.

5.4. Training times and flexification

Besides having a flexible model, one can use FlexiViT’s machinery to pre-train fixed ViTs faster. In this case, we specify a curriculum: a sequence $(p_k)_{k=1}^K$ of probability distributions over the patch sizes along with a mapping $c : \mathbb{N} \to [K]$ that identifies which distribution $p_{c(t)}$ to use at training step $t$. For example, if the desired patch size is $16 \times 16$, the last probability distribution in the sequence $p_K$ would place its entire mass on said patch size. A multitude of curricula can be designed, see Appendix H. In Figure 12 we show that in general training with a patch size curriculum leads to better performance per compute budget than standard training as we vary the training length.
ViT-B/16
FlexiViT-B/30
ViT-B/16
FlexiViT-B/16
/30
/16 (Seed)
FlexiViT-B/48
FlexiViT-B/30
FlexiViT-B/16
FlexiViT-B/8
Relevance map for the ”fork” class
Similarity of center token representation

Figure 13. Analysis of FlexiViT attention and token representations across scales. Top: Attention relevance (as in [9]) can significantly change at different patch sizes. For example, FlexiViT-B/48 and FlexiViT-B/8 consider different areas of the input most relevant for class ’fork’. See Appendix L for more examples. Bottom: Cosine similarity between a seed token representation at the center of the feature map of FlexiViT-B at patch size 16 and representations of tokens at other patch sizes. Representations are taken from block 6 and averaged across our I21K validation set. See Appendix I for similar plots for other blocks and patch sizes.

6. Analyzing FlexiViTs

Attention relevance patterns across scales We find that decreasing the patch size results in attention relevance [9] to concentrate into a larger number of smaller areas throughout the image. In Figure 13 (top) we observe that attention can significantly change at different scales.

Relation of token representations across scales As we decrease FlexiViT’s patch size, each token “splits” into multiple tokens. A natural question is how token representations at larger patch sizes relate to token representations at smaller patch size. To answer this question, we measure cosine similarity between the representation of a “seed” token at the center of a feature map at one patch size and representations of other tokens at the same and different patch sizes. As shown in Figure 13 (bottom), we are indeed able to find correspondences between tokens across scales.

Ensembling We explored whether it is possible to improve prediction accuracy by ensembling the predictions of the same FlexiViT at multiple scales. We find that, in terms of total compute spent, it is nearly always better to run a single FlexiViT at that compute budget than to ensemble multiple smaller ones. Full results are provided in Appendix J.

Shape or texture bias ViT’s bias towards using shape or texture features [17] has been shown to largely depend on its patch size [6]. In Appendix K, we show that FlexiViT evaluated at each patch size has a similar texture bias to a ViT trained and evaluated at that same patch size.

Model and dataset size Throughout the paper, we focus on FlexiViT models of the base size (-B) trained on 12M images. In order to validate that neither of these two settings are required, we train FlexiViT-S,B,L models on ImageNet-1k (1.2M images) using the ImageNet-1k DeiT III model [55] as teacher. We can see in Fig 2 that a single FlexiViT-L model matches or outperforms all three DeiT III models and EfficientNetV2. However, there is still a point at which it becomes more effective to change model width than patch size. Numerical results and evaluation on ImageNet-ReaL/v2/A/R [3,21,22,45] are in Appendix F.

7. Discussion of alternatives

Changing the input patch size is not the only way to trade off sequence length and compute in ViTs. We explore two alternatives in our core setup: distillation on ImageNet-21k.

Varying patch embedding stride One alternative is to fix the patch size and change its sampling stride, i.e. extract overlapping patches to increase sequence length. Intuitively, the advantage of this approach is that the intrinsic patch size is fixed and we avoid any special care when computing patch embeddings. Results in Figure ?? suggest varying the stride works almost as well, only slightly lagging behind our baseline.

Varying model depth Another alternative is adding flexibility in terms of depth, i.e. number of layers. Depth pruning has been explored in the context of NLP [16,47] and more recently also for ViTs [62]. Depth pruning differs fundamentally from FlexiViT: it scales linearly in depth, uses a subset of parameters, and allows progressively refining a prediction. We randomize the depth by attaching the shared head to various intermediate layers. We also tried separate heads, which worked worse. In these experiments, FlexiViT provided a significantly better compute-accuracy trade-off than depth pruning.

8. Conclusion

FlexiViT is a simple and efficient way of trading off compute and predictive performance with a single model, enabled by the unique patch embedding strategy of ViTs. FlexiViT can be used to significantly reduce pre-training costs by only training a single model for all scales at once, and performs well at a variety of downstream tasks. There are many exciting directions for future work, and we hope that our results inspire the community to explore additional creative applications of patchification.
References


[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy,


