LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference

Ben Graham Alaaeldin El-Nouby Hugo Touvron Pierre Stock Armand Joulin
Hervé Jégou Matthijs Douze

Abstract

We design a family of image classification architectures that optimize the trade-off between accuracy and efficiency in a high-speed regime. Our work exploits recent findings in attention-based architectures, which are competitive on highly parallel processing hardware. We revisit principles from the extensive literature on convolutional neural networks to apply them to transformers, in particular activation maps with decreasing resolutions. We also introduce the attention bias, a new way to integrate positional information in vision transformers.

As a result, we propose LeViT: a hybrid neural network for fast inference image classification. We consider different measures of efficiency on different hardware platforms, so as to best reflect a wide range of application scenarios. Our extensive experiments empirically validate our technical choices and show they are suitable to most architectures. Overall, LeViT significantly outperforms existing convnets and vision transformers with respect to the speed/accuracy tradeoff. For example, at 80% ImageNet top-1 accuracy, LeViT is 5 times faster than EfficientNet on CPU. We release the code at https://github.com/facebookresearch/LeViT.

1. Introduction

Transformer neural networks were initially introduced for Natural Language Processing applications [1]. They now dominate in most applications of this field. They manipulate variable-size sequences of token embeddings that are fed to a residual architecture. The model comprises two sorts for residual blocks: Multi-Layer Perceptron (MLP) and an original type of layer: the self-attention, which allows all pairs of tokens in the input to be combined via a bilinear function. This is in contrast to 1D convolutional approaches that are limited to a fixed-size neighborhood.

Recently, the vision transformer (ViT) architecture [2] obtained state-of-the-art results for image classification in the speed-accuracy tradeoff with pre-training on large scale dataset. The Data-efficient Image Transformer [3] obtains competitive performance when training the ViT models only on ImageNet [4]. It also introduces smaller models

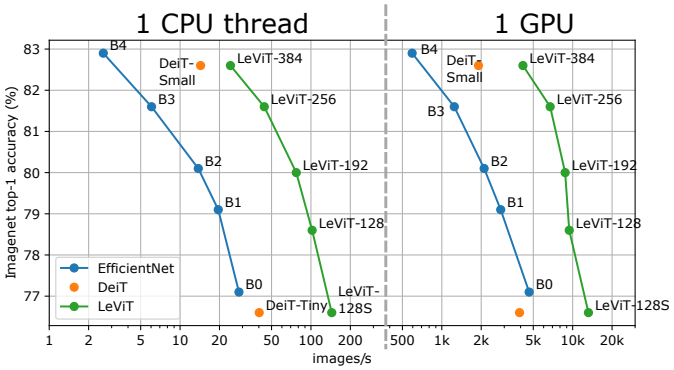

Figure 1. Speed-accuracy operating points for convolutional and visual transformers. *Left* plots: on 1 CPU core, *Right:* on 1 GPU. LeViT is a stack of transformer blocks, with pooling steps to reduce the resolution of the activation maps as in classical convolutional architectures.

adapted for high-throughput inference.

In this paper, we explore the design space to offer even better trade-offs than ViT/DeiT models in the regime of small and medium-sized architectures. We are especially interested in optimizing the performance–accuracy trade-off, such as the throughput (images/second) performance depicted in Figure 1 for Imagenet-1k-val [5].

While many works [6, 7, 8, 9, 10] aim at reducing the memory footprint of classifiers and feature extractors, inference speed is equally important, with high throughput corresponding to better energy efficiency. In this work, our goal is to develop a Vision Transformer-based family of models with better inference speed on both highly-parallel architectures like GPU, regular Intel CPUs, and ARM hardware commonly found in mobile devices. Our solution re-introduces convolutional components in place of transformer components that learn convolutional-like features. In particular, we replace the uniform structure of a Transformer by a pyramid with pooling, similar to the LeNet [11] architecture. Hence we call it LeViT.

There are compelling reasons why transformers are faster than convolutional architectures for a given computational complexity. Most hardware accelerators (GPUs, TPUs) are optimized to perform large matrix multiplica-

tions. In transformers, attention and MLP blocks rely mainly on these operations. Convolutions, in contrast, require complex data access patterns, so their operation is often IO-bound. These considerations are important for our exploration of the speed/accuracy tradeoff.

The contributions of this paper are techniques that allow ViT models to be shrunk down, both in terms of the width and spatial resolution:

- A multi-stage transformer architecture using attention as a downsampling mechanism;
- A computationally efficient patch descriptor that shrinks the number of features in the first layers;
- A learnt, per-head translation-invariant attention bias that replaces ViT's positional embeddding;
- A redesigned Attention-MLP block that improves the network capacity for a given compute time.

2. Related work

The convolutional networks descended from LeNet [11] have evolved substantially over time [12, 13, 14, 15, 16, 17]. The most recent families of architectures focus on finding a good trade-off between efficiency and performance [18, 17, 19]. For instance, the EfficientNet [17] family was discovered by carefully designing individual components followed by hyper-parameters search under a FLOPs constraint.

Transformers. The transformer architecture was first introduced by Vaswani *et al*. [1] for machine translation. Transformer encoders primarily rely on the self-attention operation in conjunction with feed-forward layers, providing a strong and explicit method for learning long range dependencies. Transformers have been subsequently adopted for NLP tasks providing state-of-the-art performance on various benchmarks [20, 21]. There have been many attempts at adapting the transformer architecture to images [22, 23], first by applying them on pixels. Due to the quadratic computational complexity and number of parameters involved in attention mechanisms, most authors [23, 24] initially considered images of small sizes like in CIFAR or Imagenet64 [25]. Mixed text and image embeddings already use transformers with detection bounding boxes as input [26], *i.e*. the bulk of the image processing is done in the convolutional domain.

The vision transformer (ViT) [2]. Interestingly, this transformer architecture is very close to the initial NLP version, devoid of explicit convolutions (just fixed-size image patch linearized into a vector), yet it competes with the state of the art for image classification. ViT achieves strong performance when pre-trained on a large labelled dataset such as the JFT300M (non-public, although training on Imagenet-21k also produces competitive results).

The need for this pre-training, in addition to strong data augmentation, can be attributed to the fact that transformers have less built-in structure than convolutions, in particular they do not have an inductive bias to focus on nearby image elements. The authors hypothesized that a large and varied dataset is needed to regularize the training.

In DeiT [3], the need for the large pre-training dataset is replaced with a student-teacher setup and stronger data augmentation and regularization, such as stochastic depth [27] or repeated augmentation [28, 29]. The teacher is a convolutional neural network that "helps" its student network to acquire an inductive bias for convolutions. The vision transformer has been thereafter successfully adapted for a wider range of computer vision tasks including object detection [30], semantic segmentation [31] and image retrieval [32].

Positional encoding. Transformers take a set as input, and hence are invariant to the order of the input. However, in language as well as in images, the inputs come from a structure where the order is important. The original Transformer [1] incorporates absolute non-parametric positional encoding with the input. Other works have replaced them with parametric encoding [33] or adopt Fourier-based kernelized versions [22]. Absolute position encoding enforce a fixed size for the set of inputs, but some works use relative position encoding [34] that encode the relative position between tokens. In our work, we replace these explicit positional encoding by positional biases that implicitly encode the spatial information.

Attention along other mechanisms. Several works have included attention mechanisms in neural network architectures designed for vision [35, 36, 37, 38]. The mechanism is used channel-wise to capture cross-feature information that complements convolutional layers [39, 40, 41], select paths in different branch of a network [42], or combine both [43]. For instance, the squeeze-and-excite network of Hu *et al*. [44] has an attention-like module to model the channel-wise relationships between the features of a layer. Li *et al*. [37] use the attention mechanism between branches of the network to adapt the receptive field of neurons.

Recently, the emergence of transformers led to hybrid architectures that benefit from other modules. Bello [45] proposes an approximated content attention with a positional attention component. Child *et al*. [23] observe that many early layers in the network learn locally connected patterns, which resemble convolutions. This suggests that hybrid architectures inspired both by transformers and convnets are a compelling design choice. A few recent works explore this avenue for different tasks [46, 47]. In image classification, a recent work that comes out in parallel with ours is the Pyramid Vision Transformer (PVT) [48], whose design is heavily inspired by ResNet. It is principally intended to address object and instance segmentation tasks.

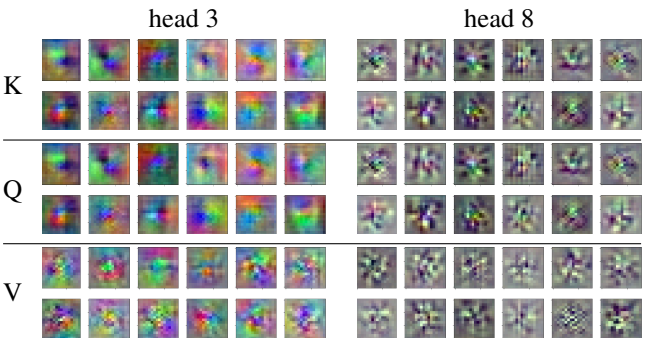

Figure 2. Patch-based convolutional masks in the pre-trained DeiT-base model [3]. The figure shows 12 of the 64 filters per head. Note that the K and Q filters are very similar, this is because the weights are entangled in the $W_{\mathrm{Q}}W_{\mathrm{K}}^{\top}$ multiplication.

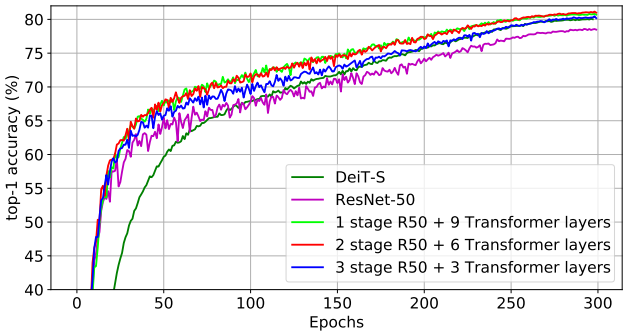

Figure 3. Models with convolutional layers show a faster convergence in the early stages compared to their DeiT counterpart.

Also concurrently with our work, Yuan et al. [49] propose the Tokens-to-Tokens ViT (T2T-ViT) model. Similar to PVT, its design relies on re-tokenization of the output after each layer by aggregating the neighboring tokens such number of tokens are progressively reduced. Additionally, Yuan et al. [49] investigate the integration of architecture design choices from CNNs [44, 50, 51] that can improve the performance and efficiency of vision transformers. As we will see, these recent methods are not as much focused as our work on the trade-off between accuracy and inference time. They are not competitive with respect to that compromise.

3. Motivation

In this section we discuss the seemingly convolutional behavior of the transformer patch projection layer. We then carry out "grafting experiments" of a transformer (DeiT-S) on a standard convolutional architecture (ResNet-50). The conclusions drawn by this analysis will motivate our subsequent design choices in Section 4.

3.1. Convolutions in the ViT architecture

ViT's patch extractor is a 16x16 convolution with stride 16. Moreover, the output of the patch extractor is multiplied by learnt weights to form the first self-attention layer's q, k and v embeddings, so we may consider these to also be convolutional functions of the input. This is also the case for variants like DeiT [3] and PVT [48]. In Figure 2 we visualize the first layer of DeiT's attention weights, broken down by attention head. This is a more direct representation than the principal components depicted by Dosovitskiy *et al*. [2]. One can observe the typical patterns inherent to convolutional architectures: attention heads specialize in specific patterns (low-frequency colors / high frequency graylelvels), and the patterns are similar to Gabor filters.

In convolutions where the convolutional masks overlap significantly, the spatial smoothness of the masks comes from the overlap: nearby pixels receive approximately the same gradient. For ViT convolutions there is no overlap. The smoothness mask is likely caused by the data augmentation: when an image is presented twice, slightly translated, the same gradient goes through each filter, so it learns this spatial smoothness.

Therefore, in spite of the absence of "inductive bias" in transformer architectures, the training *does* produce filters that are similar to traditional convolutional layers.

3.2. Preliminary experiment: grafting

The authors of the ViT image classifier [2] experimented with stacking the transformer layers above a traditional ResNet-50. In that case, the ResNet acts as a feature extractor for the transformer layers and the gradients can be propagated back through the two networks. However, in their experiments, the number of transformer layers was fixed (e.g. 12 layers for ViT-Base).

In this subsection, we investigate the potential of mixing transformers with convolutional network *under a similar computational budget*: We explore trade-offs obtained when varying the number of convolutional stages and transformer layers. Our objective is to evaluate variations of convolutional and transformer hybrids while controlling for the runtime.

Grafting. The grafting combines a ResNet-50 and a DeiT-Small. The two networks have similar runtimes.

We crop the upper stages of the ResNet-50 and likewise reduce the number of DeiT layers (while keeping the same number of transformer and MLP blocks). Since a cropped ResNet produces larger activation maps than the 14×14 activations consumed by DeiT, we introduce a pooling layer between them. In preliminary experiments we found average pooling to perform best. The positional embedding and classification token are introduced at the interface between the convolutional and transformer layer stack. For the ResNet-50 stages, we use ReLU activation units [52] and batch normalization [53].

#ResNet stages	#DeiT-S layers	nb. of Params	FLOPs (M) conv	FLOPs (M) transformer	Speed im/s	IMNET top-1
0	12	22.0M	57	4519	966	79.9
1	9	17.1M	820	3389	995	80.6
2	6	13.1M	1876	2260	1048	80.9
3	3	15.1M	3385	1130	1054	80.1
4	0	25.5M	4119	0	1254	78.4

Table 1. DeiT architecture grafted on top of a truncated ResNet-50 convolutional architecture.

Results. Table 1 summarizes the results. The grafted architecture produces better results than both DeiT and ResNet-50 alone. The smallest number of parameters and best accuracy are with two stages of ResNet-50, because this excludes the convnet's large third stage. Note that in this experiment the setup is similar to DeiT: we train for 300 epochs, we measure the top-1 validation accuracy on ImageNet, and we measure the speed as the number of images that one GPU can process per second.

One interesting observation that we show Figure 3 is that the convergence of grafted models during training seems to be similar to a convnet during the early epochs and then switch to a convergence rate similar to DeiT-S. A hypothesis is that the convolutional layers have the ability to learn representations of the low-level information in the earlier layers more efficiently due to their strong inductive biases, noticeably their translation invariance. They rely rapidly on meaningful patch embeddings, which can explain the faster convergence during the first epochs.

Discussion. It appears that in a runtime controlled regime it is beneficial to insert convolutional stages below a transformer. Most of the processing is still done in the transformer stack for the most accurate variants of the grafted architecture. Thus, the priority in the next sections will be to reduce the computational cost of the transformers. For this, instead of just grafting, the transformer architecture needs to be merged more closely with the convolutional stages.

4. Model

In this section we describe the design process of the LeViT architecture and what tradeoffs were taken. The architecture is summarized in Figure 4.

4.1. Design principles of LeViT

LeViT builds upon the ViT [2] architecture and DeiT [3] training method. We incorporate components that were proven useful for convolutional architectures. The first step is to get a compatible representation. Discounting the role of the classification embedding, ViT is a stack of layers that processes activation maps. Indeed, the intermediate "token" embeddings can be seen as the traditional $C \times H \times W$ activation maps in FCN architectures ($BCHW$ format). Therefore, operations that apply to activation maps (pooling, convolutions) can be applied to the intermediate representation of DeiT.

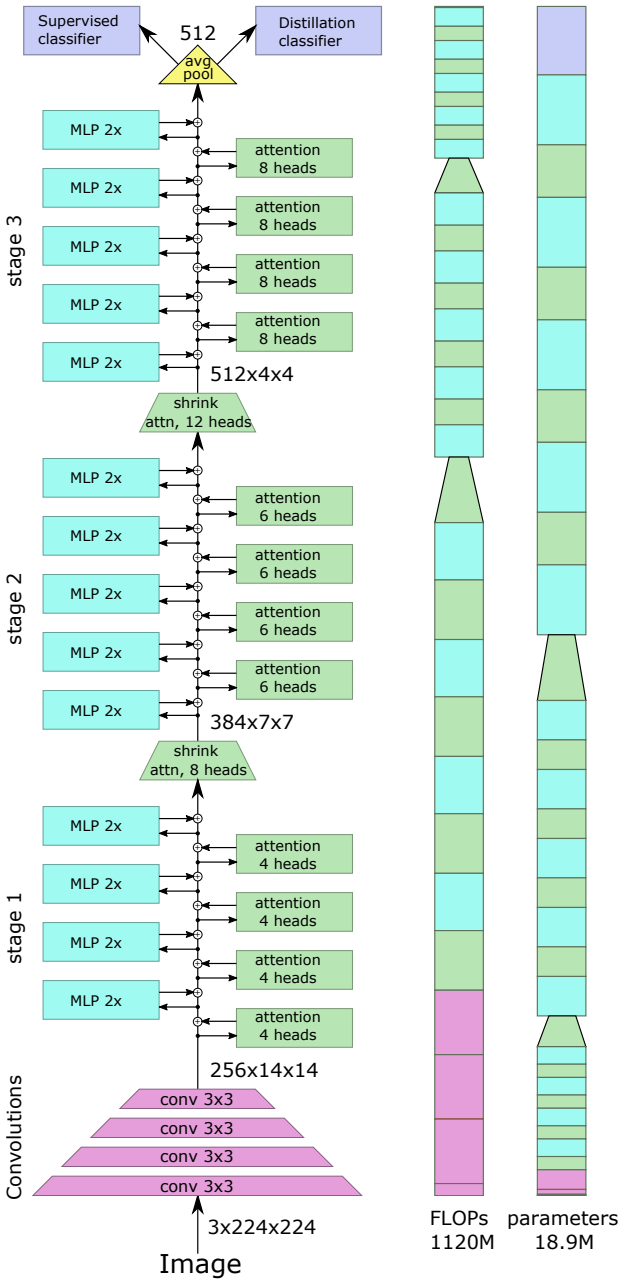

Figure 4. Block diagram of the LeViT-256 architecture. The two bars on the right indicate the relative resource consumption of each layer, measured in FLOPs, and the number of parameters.

In this work we optimize the architecture for compute, not necessarily to minimize the number of parameters. One of the design decisions that makes the ResNet [14] family more efficient than the VGG network [13] is to apply strong resolution reductions with a relatively small computation budget in its first two stages. By the time the activation map reaches the big third stage of ResNet, its resolution

has already shrunk enough that the convolutions are applied to small activation maps, which reduces the computational cost.

4.2. LeViT components

Patch embedding. The preliminary analysis in Section 3 showed that the accuracy can be improved when a small convnet is applied on input to the transformer stack. In LeViT we chose to apply 4 layers of 3×3 convolutions (stride 2) to the input to perform the resolution reduction. The number of channels goes $C = 3, 32, 64, 128, 256$. This reduces the activation map input to the lower layers of the transformer without losing salient information. The patch extractor for LeViT-256 transforms the image shape $(3, 224, 224)$ into $(256, 14, 14)$ with 184 MFLOPs. For comparison, the first 10 layers of a ResNet-18 perform the same dimensionality reduction with 1042 MFLOPs.

No classification token. To use the $BCHW$ tensor format, we remove the classification token. Similar to convolutional networks, we replace it by average pooling on the last activation map, which produces an embedding used in the classifier. For distillation during training, we train separate heads for the classification and distillation tasks. At test time, we average the output from the two heads. In practice, LeViT can be implemented using either BNC or $BCHW$ tensor format, whichever is more efficient.

Normalization layers and activations. The FC layers in the ViT architecture are equivalent to 1×1 convolutions. The ViT uses layer normalization before each attention and MLP unit. For LeViT, each convolution is followed by a batch normalization. Following [54], each batch normalization weight parameter that joins up with a residual connection is initialized to zero. The batch normalization can be merged with the preceding convolution for inference, which is a runtime advantage over layer normalization (for example, on EfficientNet B0, this fusion speeds up inference on GPU by a factor 2). Whereas DeiT uses the GELU function, all of LeViT's non-linear activations are Hardswish [19].

Multi-resolution pyramid. Convolutional architectures are built as pyramids, where the resolution of the activation maps decreases as their number of channels increases during processing. In Section 3 we used the ResNet-50 stages to pre-process the transformer stack.

LeViT integrates the ResNet stages within the transformer architecture. Inside the stages, the architecture is similar to a visual transformer: a residual structure with alternated MLP and activation blocks. In the following we review the modifications of the attention blocks (Figure 5) compared to the classical setup [1].

Downsampling. Between the LeViT stages, a *shrinking attention block* reduces the size of the activation map: a subsampling is applied before the Q transformation, which

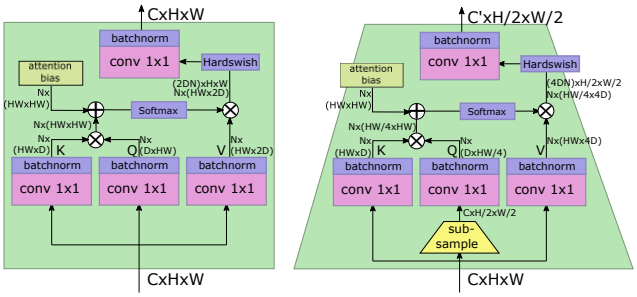

Figure 5. The LeViT attention blocks, using similar notations to [39]. Left: regular version, Right: with 1/2 reduction of the activation map. The input activation map is of size $C \times H \times W$. N is the number of heads, the multiplication operations are performed independently per head.

then propagates to the output of the soft activation. This maps an input tensor of size (C, H, W) to an output tensor of size $(C', H/2, W/2)$ with $C' > C$. Due to the change in scale, this attention block is used without a residual connection. To prevent loss of information, we take the number of attention heads to be C/D.

Attention bias instead of a positional embedding. The positional embedding in transformer architectures is a location-dependent trainable parameter vector that is added to the token embeddings prior to inputting them to the transformer blocks. If it was not there, the transformer output would be independent to permutations of the input tokens. Ablations of the positional embedding result in a sharp drop of the classification accuracy [55].

However positional embeddings are included only on input to the sequence of attention blocks. Therefore, since the positional encoding is important for higher layers as well, it is likely that it remains in the intermediate representations and needlessly uses representation capacity.

Therefore, our goal is to provide positional information within each attention block, and to explicitly inject relative position information in the attention mechanism: we simply add an *attention bias* to the attention maps. The scalar attention value between two pixels $(x, y) \in [H] \times [W]$ and $(x', y') \in [H] \times [W]$ for one head $h \in [N]$ is calculated as

$$A^h_{(x,y),(x',y')} = Q_{(x,y),:} \bullet K_{(x',y'),:} + B^h_{|x-x'|,|y-y'|}. \quad (1)$$

The first term is the classical attention. The second is the translation-invariant attention bias. Each head has $H \times W$ parameters corresponding to different pixel offsets. Symmetrizing the differences $x - x'$ and $y - y'$ encourages the model to train with flip invariance.

Smaller keys. The bias term reduces the pressure on the keys to encode location information, so we reduce the size of the key matrices K and Q relative to the values matrix V. Restricting the size of the keys reduces the time needed

to calculate the matrix product $QK^\top$. If the keys have size $D \in \{16, 32\}$, then we construct V to have $2D$ channels.

For downsampling layers, where there is no residual connection, we set the dimension of V to $4D$ to prevent loss of information.

Attention activation. We apply a Hardswish activation to the product A^hV before the regular linear projection is used to combine the output of the different heads. This is akin to a ResNet bottleneck residual block, in the sense that V is the output of a 1×1 convolution, A^hV corresponds to a spatial convolution, and the projection is another 1×1 convolution.

Reducing the MLP blocks. The MLP residual block in ViT is a linear layer that increases the embedding dimension by a factor 4, applies a non-linearity and reduces it back with another non-linearity to the original embedding's dimension. For vision architectures, the MLP is usually more expensive in terms of runtime and parameters than the attention block. For LeViT, the "MLP" is a 1×1 convolution, followed by the usual batch normalization. To reduce the computational cost of that phase, we reduce the expansion factor of the convolution from 4 to 2. Our design objective is that attention and MLP blocks should consume approximately the same number of FLOPs.

4.3. The LeViT family of models

The LeViT models can spawn a range of speed-accuracy tradeoffs by varying the size of the computation stages. We identify them by the number of channels input to the first transformer, *e.g.* LeViT-256 has 256 channels on input of the transformer stage. Table 2 shows how the stages are designed for the models that we evaluate in this paper.

5. Experiments

5.1. Experimental context

Datasets and evaluation. We model our experiments on the DeiT work, that is closest to our approach. It builds upon PyTorch [56] and the Timm library [57]. We train on the ImageNet-2012 dataset and evaluate on its validation set. We do not explore using more training data in this work.

Resource consumption. The generally accepted measure for inference speed is in units of multiply-add operations (aka FLOPs) because floating-point matrix multiplications and convolutions can be expressed as those.

However, some operations, most notably non-linear activations, do not perform multiply-add operations. They are generally ignored in the FLOP counts (or counted as a single FLOP) because it is assumed that their cost is negligible w.r.t. the cost of higher-order matrix multiplications and convolutions. However, for a small number of channels, the runtime of complicated activations like GELU is comparable to that of convolutions. Moreover, operations with the same number of FLOPs can be more or less efficient depending on the hardware and API used.

Therefore, we additionally report raw timings on reference hardware, like recent papers [2, 58]. The efficiency of transformers relies almost exclusively on matrix multiplications with a large reduction dimension.

Hardware. In this work, we run all experiments in PyTorch, thus we are dependent on the available optimizations in that API. In an attempt to obtain more objective timings, we time the inference on three different hardware platforms, each corresponding to one use case:

- One 16GB NVIDIA Volta GPU (peak performance is 12 TFLOP/s). This is a typical training accelerator.
- An Intel Xeon 6138 CPU at 2.0GHz. This is a typical server in a datacenter, that performs feature extraction on streams of incoming images. PyTorch is well optimized for this configuration, using MKL and AVX2 instructions (16 vector registers of 256 bits each).
- An ARM Graviton2 CPU (Amazon C6g instance). It is a good model for the type of processors that mobile phones and other edge devices are running. The Graviton2 has 32 cores supporting the NEON vector instruction set with 32 128-bit vector registers (NEON).

On the GPU we run timings on large image batches because that corresponds to typical use cases; following DeiT we use the maximum power-of-two batchsize that fits in memory. On the CPU platforms, we measure inference time in a single thread, simulating a setting where several threads process separate streams of input images.

It is difficult to dissociate the impact of the hardware and software, so we experiment with several ways to optimize the network with standard PyTorch tools (the just-in-time compiler, different optimization profiles).

5.2. Training LeViT

We use 32 GPUs that perform the 1000 training epochs in 3 to 5 days. This is more than the usual schedule for convolutional networks, but visual transformers require a longer training than convnets. For example, DeiT training for 1000 epochs improves by another 2 points of top-1 precision over 300 epochs. To regularize the training, we use distillation driven training, similar to DeiT. This means that LeViT is trained with two classification heads with a cross entropy loss. The first head receives supervision from the ground-truth classes, the second one from a RegNetY-16GF [18] model trained on ImageNet. In fact, the LeViT training time is dominated by the teacher's inference time.

5.3. Speed-accuracy tradeoffs

Table 3 shows the speed-precision tradeoffs that we obtain with LeViT, and a few salient numbers are plotted

Model	LeViT-128S ($D=16, p=0$)	LeViT-128 ($D=16, p=0$)	LeViT-192 ($D=32, p=0$)	LeViT-256 ($D=32, p=0$)	LeViT-384 ($D=32, p=0.1$)
Stage 1: 14×14	$2\times\begin{bmatrix}C=128\\N=4\end{bmatrix}$	$4\times\begin{bmatrix}C=128\\N=4\end{bmatrix}$	$4\times\begin{bmatrix}C=192\\N=3\end{bmatrix}$	$4\times\begin{bmatrix}C=256\\N=4\end{bmatrix}$	$4\times\begin{bmatrix}C=384\\N=6\end{bmatrix}$
Subsample	$[\ N=8\ ]$	$[\ N=8\ ]$	$[\ N=6\ ]$	$[\ N=8\ ]$	$[N=12]$
Stage 2: 7×7	$3\times\begin{bmatrix}C=256\\N=6\end{bmatrix}$	$4\times\begin{bmatrix}C=256\\N=8\end{bmatrix}$	$4\times\begin{bmatrix}C=288\\N=5\end{bmatrix}$	$4\times\begin{bmatrix}C=384\\N=6\end{bmatrix}$	$4\times\begin{bmatrix}C=512\\N=9\end{bmatrix}$
Subsample	$[N=16]$	$[N=16]$	$[N=9]$	$[N=12]$	$[N=18]$
Stage 3: 4×4	$4\times\begin{bmatrix}C=384\\N=8\end{bmatrix}$	$4\times\begin{bmatrix}C=384\\N=12\end{bmatrix}$	$4\times\begin{bmatrix}C=384\\N=6\end{bmatrix}$	$4\times\begin{bmatrix}C=512\\N=8\end{bmatrix}$	$4\times\begin{bmatrix}C=768\\N=12\end{bmatrix}$

Table 2. LeViT models. Each stage consists of a number of pairs of Attention and MLP blocks. N: number of heads, C: number of channels, D: output dimension of the Q and K operators. Separating the stages are shrinking attention blocks whose values of C, C' are taken from the rows above and below respectively. Drop path with probability p is applied to each residual connection. The value of N in the stride-2 blocks is C/D to make up for the lack of a residual connection. Each attention block is followed by an MLP with expansion factor two.

	# params	FLOPs	inference speed				ImageNet	
			top-1	GPU	Intel	ARM	-Real	-V2.
Architecture	(M)	(M)	%	im/s	im/s	im/s	%	%
LeViT-128S **(ours)**	7.8	305	76.6	12880	131.1	39.1	83.1	64.3
EfficientNet B0	5.3	390	77.1	4754	30.1	3.5	83.5	64.3
LeViT-128 **(ours)**	9.2	406	78.6	9266	94.0	30.8	84.7	66.6
LeViT-192 **(ours)**	10.9	658	80.0	8601	65.0	24.2	85.7	68.0
EfficientNet B1	7.8	700	79.1	2882	20.0	2.3	84.9	66.9
EfficientNet B2	9.2	1000	80.1	2149	13.1	1.3	85.9	68.8
LeViT-256 **(ours)**	18.9	1120	81.6	6582	42.5	16.4	86.8	70.0
DeiT-Tiny	5.9	1220	76.6	3973	39.1	16.8	83.9	65.4
EfficientNet B3	12	1800	81.6	1272	5.9	0.8	86.8	70.6
LeViT-384 **(ours)**	39.1	2353	82.6	4165	23.1	9.4	87.6	71.3
EfficientNet B4	19	4200	82.9	606	2.5	0.5	88.0	72.3
DeiT-Small	22.5	4522	82.6	1931	13.7	7.6	87.8	71.7

Table 3. Characteristics of LeViT w.r.t. two strong families of competitors: DeiT [3] and EfficientNet [17]. The top-1 numbers are accuracies on ImageNet or ImageNet-Real and ImageNet-V2 (two last columns). The others are images per second on the different platforms. LeViT models optimize the trade-off between efficiency and accuracy (and not #params). The rows are sorted by FLOP counts.

in Figure 1. We compare these with two competitive architectures from the state of the art: EfficientNet [17] as a strong convolutional baseline, and likewise DeiT [3] a strong transformer-only architecture. Both baselines are trained under to maximize their accuracy. For example, we compare with DeiT trained during 1000 epochs.

In the range of operating points we consider, the LeViT architecture largely outperforms both the transformer and convolutional variants. LeViT-384 is on-par with DeiT-Small in accuracy but uses half the number of FLOPs. The gap widens for faster operating points: LeViT-128S is on-par with DeiT-Tiny and uses 4× fewer FLOPs.

The runtime measurements follow closely these trends. For example LeViT-192 and LeViT-256 have about the same accuracies as EfficientNet B2 and B3 but are 5× and 7× faster on CPU, respectively. On the ARM platform, the float32 operations are not as well optimized compared to Intel. However, the speed-accuracy trade-off remains in LeViT's favor.

5.4. Comparison with the state of the art

Table 4 reports results with other transformer based architectures for comparison with LeViT (Table 3). Since our approach specializes in the high-throughput regime, we do not include very large and slow models [61, 62].

We compare in the FLOPs-accuracy tradeoff, since the other works are very recent and do not necessarily provide reference models on which we can time the inference. All Token-to-token ViT [49] variants take around 5× more FLOPs than LeViT-384 and more parameters for comparable accuracies than LeViT. Bottleneck transformers [46] and "Visual Transformers" [47] (not to be confused with ViT) are both generic architectures that can also be used for detection and object segmentation. Both are about 5× slower than LeViT-192 at a comparable accuracy. The same holds for the pyramid vision transformer [48] (not reported in the table) but its design objectives are different. The ad-

Architecture	#params	FLOPs	INET top-1
T2T-ViTt-14 [49]	21.5M	5200M	80.7
T2T-ViTt-19	39.0M	8400M	81.4
T2T-ViTt-24	64.1M	13200M	82.2
BoT-S1-50 [46]	20.8M	4270M	79.1
VT-R34 [47]	19.2M	3236M	79.9
VT-R50	21.4M	3412M	80.6
VT-R101	41.5M	7129M	82.3
PiT-Ti [59]	4.9M	700M	74.6
PiT-XS	10.6M	1400M	79.1
PiT-S	23.5M	2900M	81.9
CvT-13-NAS [60]	18M	4100M	82.2

Table 4. Comparison with the recent state of the art in the high-throughput regime. All inference are performed on images of size 224×224, and training is done on ImageNet only.

#id↓	Ablation of LeViT-128S	#params	FLOPs	INET top-1
	Base model	7.4M	305M	71.9
A1	– without pyramid shape	1.2M	308M	56.5
A2	– without PatchConv	7.4M	275M	65.3
A3	– without BatchNorm	7.4M	305M	66.6
A4	– without distillation	7.4M	305M	69.7
A5	– without attention bias	7.4M	305M	70.4
A6	– without wider blocks	6.2M	312M	70.9
A7	– without attention activ.	7.4M	305M	71.1

Table 5. Ablation of various components w.r.t. the baseline LeViT-128S. Each row is the baseline minus some LeViT component (1st column: experiment id). The training is run for 100 epochs only.

vantage of LeViT compared to these architectures is that it benefited from the DeiT-like distillation, which makes it much more accurate when training on ImageNet alone. Two architecture that comes close to LeViT are the pooling-based vision transformer (PiT) [59] and CvT [60], ViT variants with a pyramid structure. PiT, the most promising one, incorporates many of the optimization ingredients for DeiT but is still 1.2× to 2.4× slower than LeViT.

Alternaltive evaluations. In Table 3 we evaluate LeViT on alternative test sets, Imagenet Real [63] and Imagenet V2 matched frequency [64]. The two datasets use the same set of classes and training set as ImageNet. Imagenet-Real has re-assessed labels with potentially several classes per image. Imagenet-V2 (in our case match frequency) employs a different test set. It is interesting to measure the performance on both to verify that hyper-parameters adjustments have not led to overfitting to the validation set of ImageNet. Thus, we measure the classification performance on the alternative test sets for models that have equivalent accuracies on ImageNet validation. LeViT-256 and EfficientNet B3: the LeViT variant achieves the same score on -Real, but is slightly worse (-0.6) on -V2. LeViT-384 and DeiT-Small: LeViT is slightly worse on -Real (-0.2) and -V2 (-0.4). Although in these evaluations LeViT is relatively slightly less accurate, the speed-accuracy trade-offs still hold, compared to EfficientNet and DeiT.

5.5. Ablations

To evaluate what contributes to the performance of LeViT, we experiment with the default setting and replace one parameter at a time. We train the LeViT-128S model, and a number of variants, to evaluate the design changes relative to ViT/DeiT. The experiments are run with only 100 training epochs to magnify the differences and reduce training time. The conclusions remain for larger models and longer training schedules. We replace one component at a time. When the network needs to be reworked, we make sure the FLOP count remains roughly the same (see Appendix A.2 for details). Table 5 shows that all changes degrade the accuracy:

A1– The *without pyramid shape* ablation makes a straight stack of attention and MLPs (like DeiT). However, in order to keep the FLOP count similar to the baseline, the network width is reduced, resulting in a network with a small number of parameters, resulting in a very low final accuracy. This evidences that the reduction of the resolution in LeViT is the main tool to keep computational complexity under control.

A2– *without PatchConv:* we remove the four pre-processing convolutions with a single size-16 convolution. This has little effect on the number of parameters, but the number of flops is 10% less. The , and has a strong negative impact on the accuracy. This can be explained because in a low-capacity regime, the convolutions are an effective way to compress the $3 \cdot 16^2 = 768$ dimensional patch input.

A3– In *without BatchNorm,* we replace BatchNorm with preactivated LayerNorm, as used in the ViT/DeiT architecture. This slows down the model slightly, as batch statistics need to be calculated at test time. Removing the BatchNorm also removes the zero-initialization of the residual connections, which disrupts training.

A4– Removing the use of hard distillation from a RegNetY-16GF teacher model reduces performance, as seen with DeiT.

A5– The *without attention bias* ablation replaces the attention bias component with a classical positional embedding added on input to the transformer stack (like DeiT). Allowing each attention head to learn a separate bias seems to be useful.

A6– We use DeiT style blocks, i.e. Q,K and V all have dimension $D = C/N$, and the MLP blocks have expansion factor 4.

A7– LeViT has an extra Hardswish non-linearity added to the attention, in addition to the softmax non-linearity. Removing it, the *without attention activation* ablation degrades performance, suggesting that extra non-linearity is helpful for learning classification class boundaries.

6. Conclusion

This paper introduced LeViT, a transformer architecture inspired by convolutional approaches. The accuracy of LeViT stems mainly from the training techniques in DeiT. Its speed comes from a series of carefully controlled design choices. Compared to other efficient neural nets used for feature extraction in datacenters or on mobile phones, LeViT is 1.5 to 5 times faster at comparable precision. Thus to the best of our knowledge, it sets a new state of the art in the trade-off between accuracy and precision in the high-speed domain. The corresponding PyTorch code and models is available at https://github.com/facebookresearch/LeViT.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017. 1, 2, 5

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020. 1, 2, 3, 4, 6

[3] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou, "Training data-efficient image transformers & distillation through attention," *arXiv preprint arXiv:2012.12877*, 2021. 1, 2, 3, 4, 7

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Conference on Computer Vision and Pattern Recognition*, 2009. 1

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "Imagenet large scale visual recognition challenge," *International journal of Computer Vision*, 2015. 1

[6] Song Han, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2016. 1

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *arXiv preprint arXiv:1511.00363*, 2016. 1

[8] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2018. 1

[9] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han, "Haq: Hardware-aware automated quantization with mixed precision," *arXiv preprint arXiv:1811.08886*, 2019. 1

[10] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou, "And the bit goes down: Revisiting the quantization of neural networks," *arXiv preprint arXiv:1907.05686*, 2020. 1

[11] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989. 1, 2

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012. 2

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015. 2, 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2016. 2, 4

[15] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016. 2

[16] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He, "Aggregated residual transformations for deep neural networks," *Conference on Computer Vision and Pattern Recognition*, 2017. 2

[17] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019. 2, 7

[18] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár, "Designing network design spaces," *Conference on Computer Vision and Pattern Recognition*, 2020. 2, 6

[19] A. Howard, Mark Sandler, G. Chu, Liang-Chieh Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Quoc V. Le, and H. Adam, "Searching for MobileNetV3," in *International Conference on Computer Vision*, 2019. 2, 5

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. 2

[21] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, "Improving language understanding with unsupervised learning," OpenAI, Tech. Rep., 2018. 2

[22] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran, "Image transformer," in *International Conference on Machine Learning*. PMLR, 2018, pp.

4055–4064. 2

[23] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019. 2

[24] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi, "On the relationship between self-attention and convolutional layers," *arXiv preprint arXiv:1911.03584*, 2020. 2

[25] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017. 2

[26] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei *et al.*, "Oscar: Object-semantics aligned pre-training for vision-language tasks," in *European Conference on Computer Vision*, 2020. 2

[27] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*, 2016. 2

[28] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze, "Multigrain: a unified image embedding for classes and instances," *arXiv preprint arXiv:1902.05509*, 2019. 2

[29] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry, "Augment your batch: Improving generalization through instance repetition," in *Conference on Computer Vision and Pattern Recognition*, 2020. 2

[30] Josh Beal, Eric Kim, Eric Tzeng, Dong Huk Park, Andrew Zhai, and Dmitry Kislyuk, "Toward transformer-based object detection," *arXiv preprint arXiv:2012.09958*, 2020. 2

[31] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr *et al.*, "Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers," *arXiv preprint arXiv:2012.15840*, 2020. 2

[32] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou, "Training vision transformers for image retrieval," *arXiv preprint arXiv:2102.05644*, 2021. 2

[33] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin, "Convolutional sequence to sequence learning," *arXiv preprint arXiv:1705.03122*, 2017. 2

[34] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018. 2

[35] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang, "Residual attention network for image classification," in *Conference on Computer Vision and Pattern Recognition*, 2017. 2

[36] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon, "Cbam: Convolutional block attention module," in *European Conference on Computer Vision*, 2018, pp. 3–19. 2

[37] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang, "Selective kernel networks," in *Conference on Computer Vision and Pattern Recognition*, 2019, pp. 510–519. 2

[38] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le, "Attention augmented convolutional networks," in *Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3286–3295. 2

[39] X. Wang, Ross B. Girshick, A. Gupta, and Kaiming He, "Non-local neural networks," *Conference on Computer Vision and Pattern Recognition*, 2018. 2, 5

[40] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu, "Dynamic convolution: Attention over convolution kernels," in *Conference on Computer Vision and Pattern Recognition*, 2020. 2

[41] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun, "Exploring self-attention for image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2020. 2

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," in *Conference on Computer Vision and Pattern Recognition*, 2015. 2

[43] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola, "Resnest: Split-attention networks," *arXiv preprint arXiv:2004.08955*, 2020. 2

[44] Jie Hu, Li Shen, and Gang Sun, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, 2017. 2, 3

[45] Irwan Bello, "Lambdanetworks: Modeling long-range interactions without attention," *arXiv preprint arXiv:2102.08602*, 2021. 2

[46] A. Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon

Shlens, P. Abbeel, and Ashish Vaswani, "Bottleneck transformers for visual recognition," *arXiv preprint arXiv:2101.11605*, 2021. 2, 7

[47] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Peter Vajda, "Visual transformers: Token-based image representation and processing for computer vision," *arXiv preprint arXiv:2006.03677*, 2020. 2, 7

[48] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," *arXiv preprint arXiv:2102.12122*, 2021. 2, 3, 7

[49] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Francis EH Tay, Jiashi Feng, and Shuicheng Yan, "Tokens-to-token vit: Training vision transformers from scratch on imagenet," *arXiv preprint arXiv:2101.11986*, 2021. 3, 7

[50] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016. 3

[51] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," in *Conference on Computer Vision and Pattern Recognition*, 2017. 3

[52] Vinod Nair and Geoffrey E Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010. 3

[53] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015. 3

[54] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017. 5

[55] Xiangxiang Chu, Bo Zhang, Zhi Tian, Xiaolin Wei, and Huaxia Xia, "Do we really need explicit position encodings for vision transformers?" *arXiv preprint arXiv:2102.10882*, 2021. 5

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019. 6

[57] Ross Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019. 6

[58] Zhuoran Shen, Irwan Bello, Raviteja Vemulapalli, Xuhui Jia, and Ching-Hui Chen, "Global self-attention networks for image recognition," *arXiv preprint arXiv:2010.03019*, 2020. 6

[59] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh, "Rethinking spatial dimensions of vision transformers," 2021. 7, 8

[60] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang, "Cvt: Introducing convolutions to vision transformers," 2021. 7, 8

[61] A. Brock, Soham De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," *arXiv preprint arXiv:2102.06171*, 2021. 7

[62] Mingxing Tan and Quoc V. Le, "Efficientnetv2: Smaller models and faster training," 2021. 7

[63] Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aaron van den Oord, "Are we done with imagenet?" *arXiv preprint arXiv:2006.07159*, 2020. 8

[64] B. Recht, Rebecca Roelofs, L. Schmidt, and V. Shankar, "Do imagenet classifiers generalize to imagenet?" *arXiv preprint arXiv:1902.10811*, 2019. 8