AdaViT: Adaptive Vision Transformers for Efficient Image Recognition

Lingchen Meng[1,2,3*] Hengduo Li[4*] Bor-Chun Chen[5] Shiyi Lan[4]
Zuxuan Wu[1,2†] Yu-Gang Jiang[1,2] Ser-Nam Lim[5]
[1]Shanghai Key Lab of Intelligent Info. Processing, School of Computer Science, Fudan University
[2]Shanghai Collaborative Innovation Center on Intelligent Visual Computing
[3]Biren Technology [4]University of Maryland [5]Meta AI

Abstract

Built on top of self-attention mechanisms, vision transformers have demonstrated remarkable performance on a variety of tasks recently. While achieving excellent performance, they still require relatively intensive computational cost that scales up drastically as the numbers of patches, self-attention heads and transformer blocks increase. In this paper, we argue that due to the large variations among images, their need for modeling long-range dependencies between patches differ. To this end, we introduce AdaViT, an adaptive computation framework that learns to derive usage policies on which patches, self-attention heads and transformer blocks to use throughout the backbone on a per-input basis, aiming to improve inference efficiency of vision transformers with a minimal drop of accuracy for image recognition. Optimized jointly with a transformer backbone in an end-to-end manner, a light-weight decision network is attached to the backbone to produce decisions on-the-fly. Extensive experiments on ImageNet demonstrate that our method obtains more than 2× improvement on efficiency compared to state-of-the-art vision transformers with only 0.8% drop of accuracy, achieving good efficiency/accuracy trade-offs conditioned on different computational budgets. We further conduct quantitative and qualitative analysis on learned usage polices and provide more insights on the redundancy in vision transformers. Code is available at https://github.com/MengLcool/AdaViT.

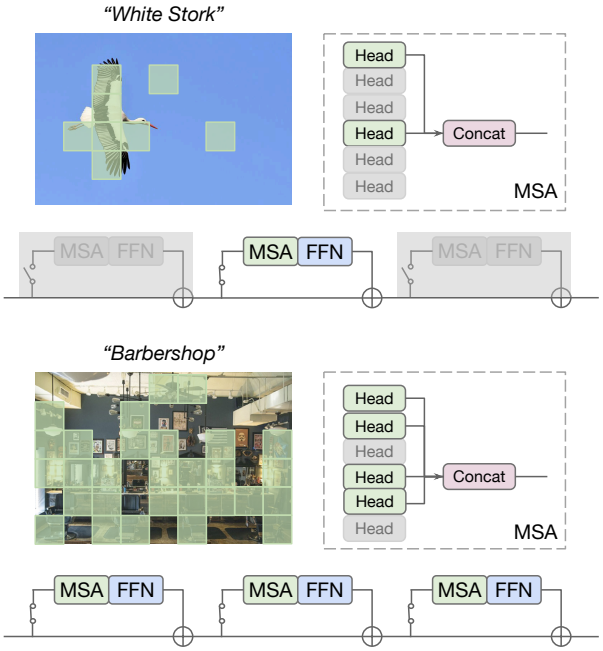

Figure 1. **A conceptual overview of our method.** Exploiting the redundancy in vision transformers, AdaViT learns to produce instance-specific usage policies on which patches, self-attention heads and transformer blocks to keep/activate throughout the network for efficient image recognition. Fewer computational resources are allocated for easy samples (top) while more are used for hard samples (bottom), reducing the overall computational cost with a minimal drop of classification accuracy. Green patches are activated in both figures.

1. Introduction

Transformers [40], the dominant architectures for a variety of natural language processing tasks, have been attracting an ever-increasing research interest in the computer vision community since the success of the Vision Transformer (ViT) [7]. Built on top of self-attention mechanisms, transformers are capable of capturing long-range dependencies among pixels/patches from input images effectively, which is arguably one of the main reasons that they outperform standard CNNs in vision tasks spanning from image classification [4, 12, 20, 22, 38, 49, 56] to object detection [3, 5, 43, 44], action recognition [9, 23, 58] and so forth.

Recent studies on vision transformers [4, 7, 38, 56] typically adopt the Transformer [40] architecture from NLP

*Equal contributions.
†Corresponding author.

with minimal surgery. Taking a sequence of sliced image patches analogous to tokens/words as inputs, the transformer backbone consists of stacked building blocks with two sublayers, *i.e*. a self-attention layer and a feed-forward network. To ensure that the model can attend to information from different representation subspaces jointly, multi-head attention is used in each block instead of a single attention function [40]. While these self-attention-based vision transformers have outperformed CNNs on a multitude of benchmarks like ImageNet [6], the competitive performance does not come for free—the computational cost of the stacked attention blocks with multiple heads is large, which further grows quadratically with the number of patches.

But are all patches needed to be attended to throughout the network for correctly classifying images? Do we need all the self-attention blocks with multiple heads to *look for* where to attend to and model the underlying dependencies for all different images? After all, large variations exist in images such as object shape, object size, occlusion and background complexity. Intuitively, more patches and self-attention blocks are required for complex images containing cluttered background or occluded objects, which require sufficient contextual information and understanding of the whole image so as to infer their ground-truth classes (*e.g*. the barber shop in Figure 1), while only a small number of informative patches and attention heads/blocks are enough to classify easy images correctly.

With this in mind, we seek to develop an adaptive computation framework that learns which patches to use and which self-attention heads/blocks to activate on a per-input basis. By doing so, the computational cost of vision transformers can be saved through discarding redundant input patches and backbone network layers for easy samples, and only using full model with all patches for hard and complex samples. This is an orthogonal and complementary direction to recent approaches on efficient vision transformers that focus on designing static network architectures [4, 11, 22, 56].

To this end, we introduce Adaptive Vision Transformer (AdaViT), an end-to-end framework that adaptively determines the usage of patches, heads and layers of vision transformers conditioned on input images for efficient image classification. Our framework learns to derive instance-specific inference strategies on: 1) which patches to keep; 2) which self-attention heads to activate; and 3) which transformer blocks to skip for each image, to improve the inference efficiency with a minimal drop of classification accuracy. In particular, we insert a light-weight multi-head subnetwork (*i.e*. a decision network) to each transformer block of the backbone network, which learns to predict binary decisions on the usage of patch embeddings, self-attention heads and blocks throughout the network. Since binary decisions are non-differentiable, we resort to Gumbel-Softmax [26] during training to make the whole framework end-to-end trainable. The decision network is jointly optimized with the transformer backbone with a usage loss that measures the computational cost of the produced usage policies and a normal cross-entropy loss, which incentivizes the network to produce policies that reduce the computational cost while maintaining classification accuracy. The overall *target* computational cost can be controlled by hyperparameter $\gamma \in (0, 1]$ corresponding to the percentage of computational cost of the full model with all patches as input during training, making the framework flexible to suit the need of different computational budgets.

We conduct extensive experiments on ImageNet [6] to validate the effectiveness of AdaViT and show that our method is able to improve the inference efficiency of vision transformers by more than $2\times$ with only 0.8% drop of classification accuracy, achieving good trade-offs between efficiency and accuracy when compared with other standard vision transformers and CNNs. In addition, we conduct quantitative and qualitative analyses on the learned usage policies, providing more intuitions and insights on the redundancy in vision transformers. We further show visualizations and demonstrate that AdaViT learns to use more computation for relatively hard samples with complex scenes, and less for easy object-centric samples.

2. Related Work

Vision Transformers. Inspired by its great success in NLP tasks, many recent studies have explored adapting the Transformer [40] architecture to multiple computer vision tasks [7–9, 14, 22, 27, 31, 32, 42, 46, 48, 54, 59]. Following ViT [7], a variety of vision transformer variants have been proposed to improve the recognition performance as well as training and inference efficiency. DeiT [38] incorporates distillation strategies to improve training efficiency of vision transformers, outperforming standard CNNs without pretraining on large-scale dataset like JFT [35]. Other approaches like T2T-ViT [56], Swin Transformer [22], PVT [44] and CrossViT [4] seek to improve the network architecture of vision transformers. Efforts have also been made to introduce the advantages of 2D CNNs to transformers through using convolutional layers [20, 53], hierarchical network structures [22, 23, 44], multi-scale feature aggregation [4, 9] and so on. While obtaining superior performance, the computational cost of vision transformers is still intensive and scales up quickly as the numbers of patches, self-attention heads and transformer blocks increase.

Efficient Networks. Extensive studies have been conducted to improve the efficiency of CNNs for vision tasks through designing effective light-weight network architectures [15, 16, 25, 34, 37, 57]. To match the inference efficiency of standard CNNs, recent work has also explored

developing efficient vision transformer architectures. T2T-ViT [56] proposes to use a deep-narrow structure and a token-to-token module, achieving better accuracy and less computational cost than ViT [7]. LeViT [11] and Swin Transformer [22] develop multi-stage network architectures with down-sampling and obtain better inference efficiency. These methods, however, use a fixed network architecture for all input samples regardless of the redundancy in patches and network architecture for easy samples. Our work is orthogonal to this direction and focuses on learning input-specific strategies that adaptively allocate computational resources for saved computation and a minimal drop in accuracy at the same time.

Adaptive Computation. Adaptive computation methods exploit the large variations within network inputs as well as the redundancy in network architectures to improve efficiency with instance-specific inference strategies. In particular, existing methods for CNNs have explored altering input samples [28, 29, 39, 50, 51, 55], skipping network layers [10, 18, 41, 45, 52] and channels [1, 21], early exiting with a multi-classifier structure [2, 17, 19], to name a few. A few attempts have also been made recently to accelerate vision transformers with adaptive inference policies exploiting the redundancy in patches, *i.e*. producing policies on what patch size [47] and which patches [30, 33] to use conditioned on input image. In contrast, we exploit the redundancy in the attention mechanism of vision transformer and propose to improve efficiency by adaptively choosing which self-attention heads, transformer blocks and patch embeddings to keep/drop conditioned on the input samples.

3. Approach

We propose AdaViT, an end-to-end adaptive computation framework to reduce the computational cost of vision transformers. Given an input image, AdaViT learns to adaptively derive policies on which *patches*, self-attention *heads* and transformer *blocks* to use or activate in the transformer backbone conditioned on the input image, encouraging using less computation while maintaining the classification accuracy. An overview of our method is shown in Figure 2.

3.1. Preliminaries

Vision transformers [7, 38, 56] for image classification take a sequence of sliced patches from image as input, and model their long-range dependencies with stacked multi-head self-attention layers and feed-forward networks*. Formally, for an input image $\mathcal{I}$, it is first split into a sequence of fixed-size 2D patches $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ where N is the number of patches (*e.g*. $N = 14 \times 14$). These raw patches are then mapped into D-dimensional patch embeddings $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_N]$ with a linear layer. A learnable embedding $\mathbf{z}_{cls}$ termed class token is appended to the sequence of patch embeddings, which serves as the representation of image. Positional embeddings $\mathbf{E}_{pos}$ are also optionally added to patch embeddings to augment them with positional information. To summarize, the input to the first transformer block is:

$$\mathbf{Z} = [\mathbf{z}_{cls}; \mathbf{z}_1; \mathbf{z}_2; ...; \mathbf{z}_N] + \mathbf{E}_{pos} \tag{1}$$

where $\mathbf{z} \in \mathbb{R}^D$ and $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D}$ respectively.

Similar to Transformers [40] in NLP, the backbone network of vision transformers consist of L blocks, each of which consists of a multi-head self-attention layer (MSA) and a feed-forward network (FFN). In particular, a single-head attention is computed as below:

$$\texttt{Attn}(Q, K, V) = \texttt{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2}$$

where Q, K, V are—in a broad sense—query, key and value matrices respectively, and d_k is a scaling factor. For vision transformers, Q, K, V are projected from the same input, *i.e*. patch embeddings. For more effective attention on different representation subspaces, multi-head self-attention concatenates the output from several single-head attentions and projects it with another parameter matrix:

$$\text{head}_{i,l} = \texttt{Attn}(\mathbf{Z}_l\mathbf{W}^Q_{i,l},\ \mathbf{Z}_l\mathbf{W}^K_{i,l},\ \mathbf{Z}_l\mathbf{W}^V_{i,l}) \tag{3}$$

$$\texttt{MSA}(\mathbf{Z}_l) = \texttt{Concat}(\text{head}_{1,l}, ..., \text{head}_{H,l})\mathbf{W}^O_l, \tag{4}$$

where $\mathbf{W}^Q_{i,l}, \mathbf{W}^K_{i,l}, \mathbf{W}^V_{i,l}, \mathbf{W}^O_l$ are the parameter matrices in the i-th attention head of the l-th transformer block, and $\mathbf{Z}_l$ denotes the input at the l-th block. The output from MSA is then fed into FFN, a two-layer MLP, and produce the output of the transformer block $\mathbf{Z}_{l+1}$. Residual connections are also applied on both MSA and FFN as follows:

$$\mathbf{Z}'_l = \texttt{MSA}(\mathbf{Z}_l) + \mathbf{Z}_l, \quad \mathbf{Z}_{l+1} = \texttt{FFN}(\mathbf{Z}'_l) + \mathbf{Z}'_l \tag{5}$$

The final prediction is produced by a linear layer taking the class token from last transformer block ($\mathbf{Z}^0_L$) as inputs.

3.2. Adaptive Vision Transformer

While large vision transformer models have achieved superior image classification performance, the computational cost grows quickly as we increase the numbers of patches, attention heads and transformer blocks to obtain higher accuracies. In addition, a computationally expensive one-size-fit-all network is often an overkill for many easy samples. To remedy this, AdaViT learns to adaptively choose 1) which patch embeddings to use; 2) which self-attention heads in MSA to activate; and 3) which transformer block to skip—on a per-input basis—to improve the inference efficiency of vision transformers. We achieve this by inserting

*In this section we consider the architecture of ViT [7], and extend it to other variants of vision transformers is straightforward.

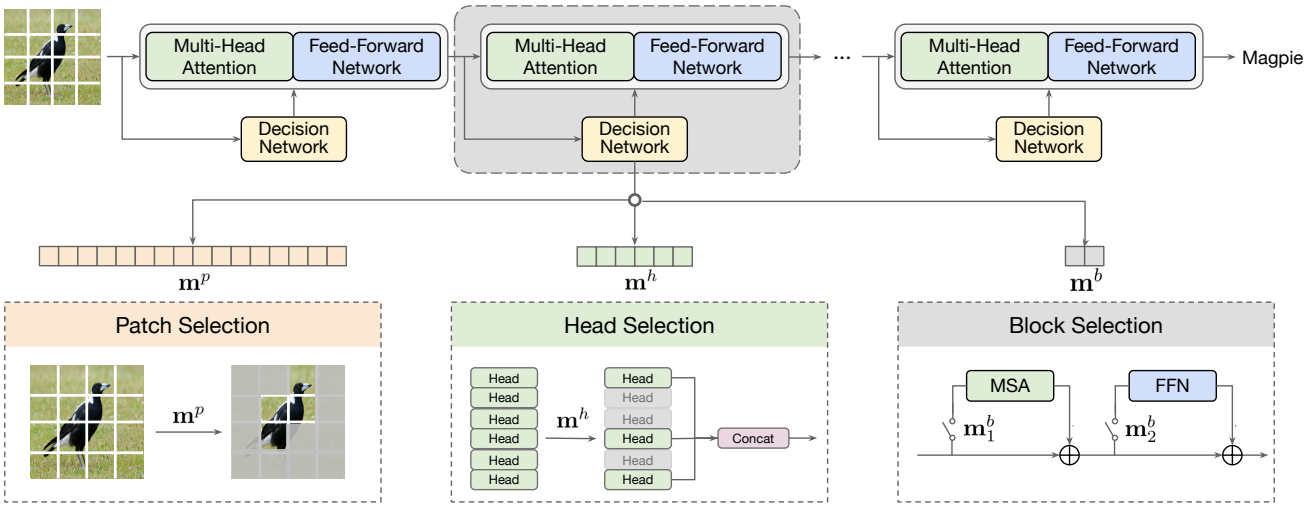

Figure 2. **An overview of our approach.** We insert a light-weight decision network before each block of the vision transformer backbone. Given an input image, the decision networks produce usage policies on which *patches*, self-attention *heads* and transformer *blocks* to keep/activate throughout the backbone. These instance-specific usage policies are incentivized to reduce the overall computational cost of vision transformers with minimal drop of accuracy. See texts for more details.

a light-weight decision network before each of the transformer blocks, and it is trained to produce the three sets of usage policies for this block.

Decision Network. The decision network at l-th block consists of three linear layers with parameters $\mathbf{W}_l = \{\mathbf{W}_l^p, \mathbf{W}_l^h, \mathbf{W}_l^b\}$ to produce computation usage policies for *patch selection*, attention *head selection* and transformer *block selection* respectively. Formally, given the input to l-th block $\mathbf{Z}_l$, the usage policy matrices for this block is computed as follows:

$$(\mathbf{m}_l^p, \mathbf{m}_l^h, \mathbf{m}_l^b) = (\mathbf{W}_l^p, \mathbf{W}_l^h, \mathbf{W}_l^b)\mathbf{Z}_l$$
$$s.t.\ \ \mathbf{m}_l^p \in \mathbb{R}^N,\ \mathbf{m}_l^h \in \mathbb{R}^H,\ \mathbf{m}_l^b \in \mathbb{R} \quad (6)$$

where N and H denote the numbers of patches and self-attention heads in a transformer block, and $l \in [1, L]$. Each entry of $\mathbf{m}_l^p$, $\mathbf{m}_l^h$ and $\mathbf{m}_l^b$ is further passed to a `sigmoid` function, indicating the probability of keeping the corresponding patch, attention head and transformer block respectively. The l-th decision network shares the output from previous $l-1$ transformer blocks, making the framework more efficient than using a standalone decision network.

As the decisions are binary, the action of keeping/discarding can be selected by simply applying a threshold on the entries during inference. However, deriving the optimal thresholds for different samples is challenging. To this end, we define random variables $\mathbf{M}_l^p, \mathbf{M}_l^h, \mathbf{M}_l^b$ to make decisions by sampling from $\mathbf{m}_l^p$, $\mathbf{m}_l^h$ and $\mathbf{m}_l^b$. For example, the j-th patch embedding in l-th block is kept when $\mathbf{M}_{l,j}^p = 1$, and dropped when $\mathbf{M}_{l,j}^p = 0$. We relax the sampling process with Gumbel-Softmax trick [26] to make it differentiable during training (see Sec. 3.3.)

Patch Selection. For the input to each transformer block, we aim at keeping only the most informative patch embeddings and discard the rest to speedup inference. More formally, for l-th block, the patches are removed from the input to this block if the corresponding entries in $\mathbf{M}_i^p$ equal to 0:

$$\mathbf{Z}_l = [\mathbf{z}_{l,cls}; \mathbf{M}_{l,1}^p \mathbf{z}_1; ...; \mathbf{M}_{l,N}^p \mathbf{z}_N] \quad (7)$$

The class token $\mathbf{z}_{l,cls}$ is always kept since it is used as representation of the whole image.

Head Selection. Multi-head self attention enables the model to attend to different subspaces of the representation jointly [40] and is adopted in most, if not all, vision transformer variants [4,7,22,38,56]. Such a multi-head design is crucial to model the underlying long-range dependencies in images especially those with complex scenes and cluttered background, but fewer attention heads could arguably suffice to look for where to attend to in easy images. With this in mind, we explore dropping attention heads adaptively conditioned on input image for faster inference. Similar to patch selection, the decision of activating or deactivating certain attention head is determined by the corresponding entry in $\mathbf{M}_l^h$. The "deactivation" of an attention head can be instantiated in different ways. In our framework, we explore two methods for head selection, namely partial deactivation and full deactivation. For *partial deactivation*, the softmax output in attention as in Eqn. 2 is replaced with predefined ones like an $(N+1, N+1)$ identity matrix $\mathbb{1}$, such that the cost of computing attention map is saved. The attention in i-th head of l-th block is then computed as:

$$\texttt{Attn}(Q, K, V)_{l,i} = \begin{cases} \texttt{softmax}(\frac{QK^T}{\sqrt{d_k}}) \cdot V & \text{if } \mathbf{M}_{l,i}^h = 1 \\ \mathbb{1} \cdot V & \text{if } \mathbf{M}_{l,i}^h = 0 \end{cases}$$

For *full deactivation*, the entire head is removed from the multi-head self attention layer, and the embedding size of the output from MSA is reduced correspondingly:

$$\texttt{MSA}(\mathbf{Z}_l)_{l,i} = \texttt{Concat}([\text{head}_{l,i:1\to H} \text{ if } \mathbf{M}^h_{l,i} = 1])\mathbf{W}^{O'}_l$$

In practice, full deactivation saves more computation compared with partial deactivation when same percentage of heads are deactivated , yet is likely to incur more classification errors as the embedding size is manipulated on-the-fly.

Block Selection. In addition to patch selection and head selection, a transformer block can also be favourably skipped entirely when it is redundant, by virtue of the residual connections throughout the network. To increase the flexibility of layer skipping, we increase the dimension of block usage policy matrix $\mathbf{m}^b_l$ from 1 to 2, enabling the two sublayers (MSA and FFN) in each transformer block to be controlled individually. Eqn. 5 then becomes:

$$\begin{aligned}\mathbf{Z}'_l &= \mathbf{M}^b_{l,0} \cdot \texttt{MSA}(\mathbf{Z}_l) + \mathbf{Z}_l \\ \mathbf{Z}_{l+1} &= \mathbf{M}^b_{l,1} \cdot \texttt{FFN}(\mathbf{Z}'_l) + \mathbf{Z}'_l\end{aligned} \tag{8}$$

In summary, given the input of each transformer block, the decision network produces the usage policies for this block, and then the input is forwarded through the block with the decisions applied. Finally, the classification prediction from the last layer and the decisions for all blocks $\mathbf{M} = \{\mathbf{M}^p_l, \mathbf{M}^h_l, \mathbf{M}^b_l, \text{for } l : 1 \to L\}$ are obtained.

3.3. Objective Function

Since our goal is to reduce the overall computational cost of vision transformers with a minimal drop in accuracy, the objective function of AdaViT is designed to incentivize correct classification and less computation at the same time. In particular, a usage loss and a cross-entropy loss are used to jointly optimize the framework. Given an input image I with a label $\mathbf{y}$, the final prediction is produced by the transformer $\mathbf{F}$ with parameters $\boldsymbol{\theta}$, and the cross-entropy loss is computed as follows:

$$L_{ce} = -\mathbf{y}\texttt{log}(\mathbf{F}(I;\boldsymbol{\theta})) \tag{9}$$

While the binary decisions on whether to keep/discard a patch/head/block can be readily obtained through applying a threshold during inference, determining the optimal thresholds is challenging. In addition, such an operation is not differentiable during training and thus makes the optimization of decision network challenging. A common solution is to resort to reinforcement learning and optimize the network with policy gradient methods [36], yet it can be slow to converge due to the large variance that scales with the dimension of discrete variables [26, 36]. To this end, we use the Gumbel-Softmax trick [26] to relax the sampling process and make it differentiable. Formally, the decision at i-th entry of $\mathbf{m}$ is derived in the following way:

$$\mathbf{M}_{i,k} = \frac{\texttt{exp}(\texttt{log}(\mathbf{m}_{i,k} + G_{i,k})/\tau)}{\sum_{j=1}^{K}\texttt{exp}(\texttt{log}(\mathbf{m}_{i,j} + G_{i,j})/\tau)} \quad \text{for } k = 1, 2, ..., K \tag{10}$$

where K is the total number of categories ($K = 2$ for binary decision in our case), and $G_i = -\texttt{log}(-\texttt{log}(U_i))$ is the Gumbel distribution in which U_i is sampled from $\texttt{Uniform}(0, 1)$, an i.i.d uniform distribution. Temperature τ is used to control the smoothness of $\mathbf{M}_i$.

To encourage reducing the overall computational cost, we devise the usage loss as follows:

$$\begin{aligned}L_{usage} = &(\frac{1}{D_p}\sum_{d=1}^{D_p}\mathbf{M}^p_d - \gamma_p)^2 + (\frac{1}{D_h}\sum_{d=1}^{D_h}\mathbf{M}^h_d - \gamma_h)^2 \\ &+ (\frac{1}{D_b}\sum_{d=1}^{D_b}\mathbf{M}^b_d - \gamma_b)^2 \\ \text{where } D_p = L \times N,\ &D_h = L \times H,\ D_b = L \times 2\end{aligned} \tag{11}$$

Here D_p, D_h, D_b denote the sizes of flattened probability vectors from the decision network for patch/head/block selection, *i.e.* the total numbers of patches, heads and blocks of the entire transformer respectively. The hyperparameters $\gamma_p, \gamma_h, \gamma_b \in (0, 1]$ indicate target computation budgets in terms of the percentage of patches/heads/blocks to keep.

$$\min_{\boldsymbol{\theta},\mathbf{W}} L = L_{ce} + L_{usage} \tag{12}$$

Finally, the two loss functions are combined and minimized in an end-to-end manner as in Eqn. 12.

4. Experiment

4.1. Experimental Setup

Dataset and evaluation metrics. We conduct experiments on ImageNet [6] with ∼1.2M images for training and 50K images for validation, and report the Top-1 classification accuracy. To evaluate model efficiency, we report the number of giga floating-point operations (GFLOPs) per image.

Implementation details. We use T2T-ViT [56] as the transformer backbone due to its superior performance on ImageNet with a moderate computational cost. The backbone consists of $L = 19$ blocks and $H = 7$ heads in each MSA layer, and the number of tokens $N = 196$. The decision network is attached to each transformer block starting from 2-nd block. For head selection, we use the *full deactivation* method if not mentioned otherwise. We initialize the transformer backbone of AdaViT with the pretrained weights released in the official implementation of [56]. We will release the code.

Method	Top-1 Acc (%)	FLOPs (G)	Image Size	# Patch	# Head	# Block
ResNet-50* [13, 56]	79.1	4.1	224×224	-	-	-
ResNet-101* [13, 56]	79.9	7.9	224×224	-	-	-
ViT-S/16 [7]	78.1	10.1	224×224	196	12	8
DeiT-S [38]	79.9	4.6	224×224	196	6	12
PVT-Small [44]	79.8	3.8	224×224	-	-	15
Swin-T [22]	81.3	4.5	224×224	-	-	12
T2T-ViT-19 [56]	81.9	8.5	224×224	196	7	19
CrossViT-15 [4]	81.5	5.8	224×224	196	6	15
LocalViT-S [20]	80.8	4.6	224×224	196	6	12
Baseline *Upperbound*	81.9	8.5	224×224	196	7	19
Baseline *Random*	33.0	4.0	224×224	∼ 118	∼ 5.6	∼ 16.2
Baseline *Random+*	71.5	3.9	224×224	∼ 121	∼ 5.6	∼ 16.2
AdaViT (Ours)	81.1	3.9	224×224	∼ 95	∼ 4.5	∼ 15.5

Table 1. **Main Results.** We compare AdaViT with various standard CNNs and vision transformers, as well as baselines including *Upperbound*, *Random* and *Random+*. * denotes training ResNets with our recipe following [56].

We use 8 GPUs with a batch size 512 for training. The model is trained with a learning rate 0.0005, a weight decay 0.065 and a cosine learning rate schedule for 150 epochs following [56]. AdamW [24] is used as the optimizer. For all the experiments, we set the input size to 224×224. Temperature τ in Gumbel-Softmax is set to 5.0. The choices of $\gamma_p, \gamma_h, \gamma_b$ vary flexibly for different desired trade-offs between classification accuracy and computational cost.

4.2. Main Results

We first evaluate the overall performance of AdaViT in terms of classification accuracy and efficiency, and report the results in Table 1. Besides standard CNN and transformer architectures such as ResNets [13], ViT [7], DeiT [38], T2T-ViT [56] and so on, we also compare our method with the following baseline methods:

- *Upperbound*: The original pretrained vision transformer model, with all patch embeddings kept as input and all self-attention heads and transformer blocks activated. This serves as an "upperbound" of our method regarding classification accuracy.
- *Random*: Given the usage policies produced by AdaVit, we generate random policies on patch selection, head selection and block selection that use similar computational cost and apply them to the pretrained models to validate the effectiveness of learned policies.
- *Random+*: The pretrained models are further finetuned with the random policies applied, in order to adapt to the varied input distribution and network architecture incurred by the random policies.

As shown in Table 1, AdaViT is able to obtain good efficiency improvement with only a small drop on classification accuracy. Specifically, AdaViT obtains 81.1% Top-1 accuracy requiring 3.9 GFLOPs per image during inference, achieving more than 2× efficiency than the original T2T-ViT model with only ∼ 0.8% drop of accuracy. Compared with standard ResNets [13] and vision transformers that use a similar backbone architecture of ours [4, 7, 38, 56], AdaViT obtains better classification performance with less computational cost, achieving a good efficiency/accuracy trade-off as further shown in Figure 3. It is also worth pointing out that compared with vision transformer variants [22, 44] which resort to advanced design choices like multi-scale feature pyramid and hierarchical downsampling, our method still obtains comparable or better accuracy under similar computational cost.

When using a similar computation budget, AdaViT outperforms *random* and *random+* baselines by clear margins. Specifically, Ada-ViT with T2T-ViT as the backbone network obtains 48.1% and 9.6% higher accuracy than *random* and *random+* respectively at a similar cost of 3.9 GFLOPs per image, demonstrating that the usage policies learned by AdaViT can effectively maintain classification accuracy and

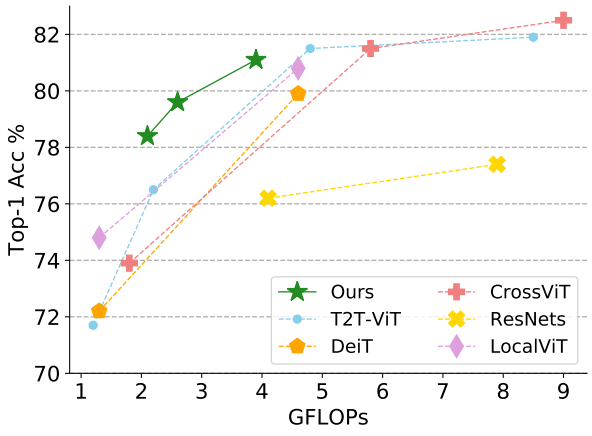

Figure 3. **Tradeoff between efficiency and accuracy.** AdaViT obtains good efficiency/accuracy tradeoffs compared with other static vision transformers.

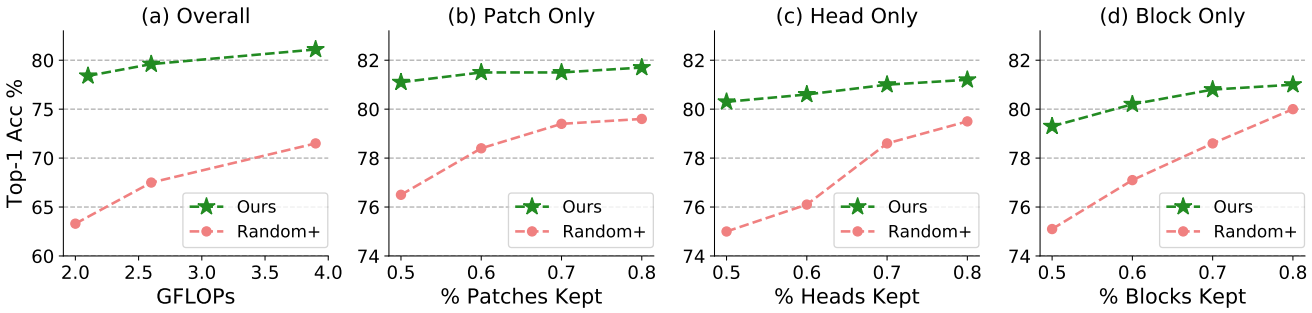

Figure 4. **Effectiveness of each component.** Efficiency/Accuracy tradeoffs of AdaViT with (a) all three selection methods; (b) patch selection; (c) head selection; (d) block selection and their *Random+* counterparts.

reduce computational cost at the same time.

AdaViT with different computational budgets. AdaViT is designed to accommodate the need of different computational budgets flexibly by varying the hyperparameters γ_p, γ_h and γ_b as discussed in Section 3.2. As demonstrated in Figure 4(a), AdaViT is able to cover a wide range of tradeoffs between efficiency and accuracy, and outperforms *Random+* baselines by a large margin.

4.3. Ablation Study

Effectiveness of learned usage policies. Here we validate that each of the three sets of learned usage policies is able to effectively maintain the classification accuracy while reducing the computational cost of vision transformers. For this purpose, we replace the learned usage policies with randomly generated policies that cost similar computational resources and report the results in Table 2. As shown in Table 2, changing any set of learned policies to a random one results in a drop of accuracy by a clear margin. Compared with random patch/head/block selection, AdaViT obtains $31.9\%/23.7\%/16.4\%$ higher accuracy under similar computational budget. This confirms the effectiveness of each learned usage policy.

Random Patch	Random Head	Random Block	Top-1 Accuracy
✓			49.2
	✓		57.4
		✓	64.7
Full AdaViT			81.1

Table 2. **Effectiveness of learned usage policies.** We replace each set of policies with randomly generated policies and compare with our method in its entirety.

Method	Top-1 Acc	% Head	GFLOPs
Upperbound	81.9	100%	8.5
Partial	81.7	50%	6.9
Full	80.3	50%	5.1
Full	80.8	60%	5.8
Full	81.1	70%	6.6

Table 3. **Partial *vs*. Full deactivation for head selection.**

Ablation of individual components. Having demonstrated the effectiveness of the jointly learned usage policies for patch, head and block selection, we now evaluate the performance when only one of the three selection methods is used. It is arguable that part of the performance gap in Table 2 results from the change of input/feature distribution when random policies are applied, and thus we compare each component with its further finetuned *Random+* counterparts. For faster training and evaluation, we train these models for 100 epochs. As shown in Figure 4(b-d), our method with only patch/head/block selection is also able to cover a wide range of accuracy/efficiency tradeoffs and outperforms *Random+* baselines by a clear margin, confirming the effectiveness of each component.

Partial *vs*. Full deactivation for head selection. As discussed in Sec. 3.2, we propose two methods to deactivate a head in the multi-head self-attention layer, namely partial deactivation and full deactivation. We now analyze their effectiveness on improving the efficiency of vision transformers. As demonstrated in Table 3, when deactivating the same percentage (*i.e*. 50%) of self-attention heads within the backbone, partial deactivation is able to obtain much higher accuracy than full deactivation (81.7% *vs*. 80.3%), but also incurs higher computational cost (6.9 *vs*. 5.1 GFLOPs). This is intuitive since partial deactivation only skips the computation of attention maps before `Softmax`, while full deactivation removes the entire head and its output to the FFN. As the number of heads increases, full deactivation obtains better accuracy gradually. In practice these different head selection methods provide more flexible options to suit different computational budgets.

4.4. Analysis

Computational saving throughout the network. AdaViT exploits the redundancy of computation to improve the efficiency of vision transformers. To better understand such re-

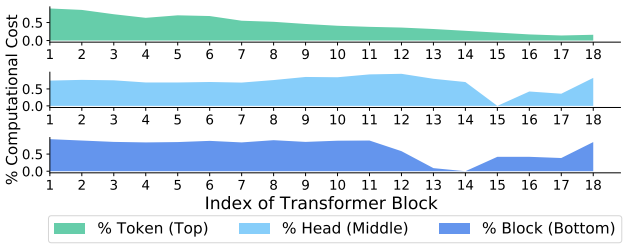

Figure 5. **Computational cost throughout the network.** The percentages of kept/activated patches (**top**), heads (**middle**) and blocks (**bottom**) throughout the backbone are reported.

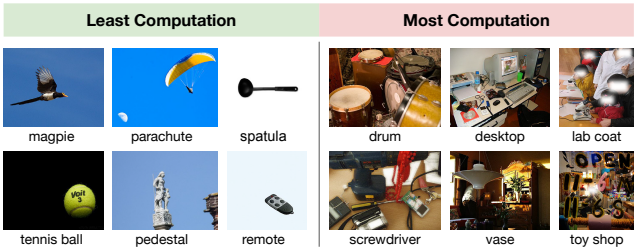

Figure 6. **Qualitative results.** Images allocated with the least and the most computational resources by AdaViT are shown.

dundancy, we collect the usage policies on patch/head/block selection predicted by our method on the validation set and show the distribution of computational cost (*i.e.* percentage of patches/heads/blocks kept) throughout the backbone network. As shown in Figure 5, AdaViT tends to allocate more computation in earlier stages of the network. In particular, for patch selection, the average number of kept patches in each transformer block gradually decrease until the final output layer. This is intuitive since the patches keep aggregating information from all other patches in the stacked self-attention layers, and a few informative patches near the output layer would suffice to represent the whole input image for correct classification. As visualized in Figure 7, the number of selected patches gradually decreases with a focus on the discriminative part of the images.

For head selection and block selection, the patterns are a bit different from token selection, where relatively more computation is kept in the last few blocks. We hypothesize that the last few layers in the backbone are more responsible for the final prediction and thus are kept more often.

Learned usage policies for different classes. We further analyze the distribution of learned usage policies for different classes. In Figure 8, we show the box plot of several classes that are allocated the most/least computational resources. As can be seen, our method learns to allocate more computation for difficult classes with complex scenes such as "shoe shop", "barber shop", "toyshop" but uses less computation for relatively easy and object-centric classes like "parachute" and "kite".

Qualitative Results. Images allocated with the least and the most computation by our method are shown in Figure 6. It can be seen that object-centric images with simple background (like the parachute and the tennis ball) tend to use less computation, while hard samples with clutter background (*e.g.* the drum and the toy shop) are allocated more.

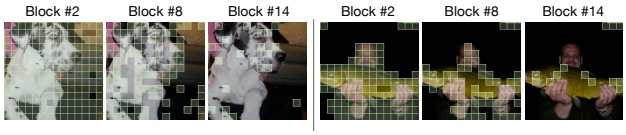

Figure 7. **Selected patches at different blocks.** Green color denotes that the patches are kept.

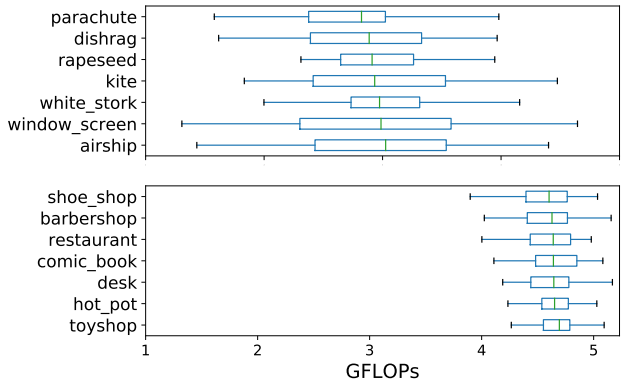

Figure 8. **Distribution of allocated computational resources** for classes using the least (**top**) and the most (**bottom**) computation.

Limitation. One potential limitation is that there is still a small drop of accuracy when comparing our method with the *Upperbound* baseline, which we believe would be further addressed in future work.

5. Conclusion

In this paper we presented AdaViT, an adaptive computation framework that learns which patches, self-attention heads and blocks to keep throughout the transformer backbone on a per-input basis for an improved efficiency for image recognition. To achieve this, a light-weight decision network is attached to each transformer block and optimized with the backbone jointly in an end-to-end manner. Extensive experiments demonstrated that our method obtains more than $2\times$ improvement on efficiency with only a small drop of accuracy compared with state-of-the-art vision transformers, and covers a wide range of efficiency/accuracy trade-offs. We further analyzed the learned usage policies quantitatively and qualitatively, providing more insights on the redundancy in vision transformers.

Acknowledgement This project was supported by National Key R&D Program of China (No. 2021ZD0112805). Y.-G. Jiang was sponsored in part by "Shuguang Program" supported by Shanghai Education Development Foundation and Shanghai Municipal Education Commission (No. 20SG01). Z. Wu was supported in part by NSFC (No. 62102092).

References

[1] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. In *ICLR*, 2020. 3

[2] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for fast test-time prediction. In *ICML*, 2017. 3

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 1

[4] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *ICCV*, 2021. 1, 2, 4, 6

[5] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. In *NeurIPS*, 2021. 1

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 5

[7] 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. In *ICLR*, 2021. 1, 2, 3, 4, 6

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

[9] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021. 1, 2

[10] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017. 3

[11] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *ICCV*, 2021. 2, 3

[12] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. In *NeurIPS*, 2021. 1

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 6

[14] Shuting He, Hao Luo, Pichao Wang, Fan Wang, Hao Li, and Wei Jiang. Transreid: Transformer-based object re-identification. In *ICCV*, 2021. 2

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 2

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[17] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018. 3

[18] Hengduo Li, Zuxuan Wu, Abhinav Shrivastava, and Larry S Davis. 2d or not 2d? adaptive 3d convolution selection for efficient video recognition. In *CVPR*, 2021. 3

[19] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *ICCV*, 2019. 3

[20] Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021. 1, 2, 6

[21] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NIPS*, 2017. 3

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 1, 2, 3, 4, 6

[23] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. *arXiv preprint arXiv:2106.13230*, 2021. 1, 2

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 6

[25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018. 2

[26] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017. 2, 4, 5

[27] Jiageng Mao, Yujing Xue, Minzhe Niu, Haoyue Bai, Jiashi Feng, Xiaodan Liang, Hang Xu, and Chunjing Xu. Voxel transformer for 3d object detection. In *ICCV*, 2021. 2

[28] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. Ar-net: Adaptive frame resolution for efficient action recognition. In *ECCV*, 2020. 3

[29] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. In *ICCV*, 2019. 3

[30] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-red^2: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*, 2021. 3

[31] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *CVPR*, 2021. 2

[32] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *ICCV*, 2021. 2

[33] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021. 3

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2

[35] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, 2017. 2

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998. 5

[37] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 2

[38] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 1, 2, 3, 4, 6

[39] Burak Uzkent and Stefano Ermon. Learning when and where to zoom with deep reinforcement learning. In *CVPR*, 2020. 3

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1, 2, 3, 4

[41] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018. 3

[42] Junke Wang, Zuxuan Wu, Jingjing Chen, Xintong Han, Abhinav Shrivastava, Yu-Gang Jiang, and Ser-Nam Lim. Objectformer for image manipulation detection and localization. In *CVPR*, 2022. 2

[43] Rui Wang, Dongdong Chen, Zuxuan Wu, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Yu-Gang Jiang, Luowei Zhou, and Lu Yuan. Bevt: Bert pretraining of video transformers. In *CVPR*, 2022. 1

[44] 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. In *ICCV*, 2021. 1, 2, 6

[45] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018. 3

[46] Xiaohan Wang, Linchao Zhu, Yu Wu, and Yi Yang. Symbiotic attention for egocentric action recognition with object-centric alignment. *IEEE TPAMI*, 2020. 2

[47] Yulin Wang, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang. Not all images are worth 16x16 words: Dynamic vision transformers with adaptive sequence length. In *NeurIPS*, 2021. 3

[48] Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. End-to-end video instance segmentation with transformers. In *CVPR*, 2021. 2

[49] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021. 1

[50] Zuxuan Wu, Hengduo Li, Caiming Xiong, Yu-Gang Jiang, and Larry Steven Davis. A dynamic frame selection framework for fast video recognition. *IEEE TPAMI*, 2022. 3

[51] Zuxuan Wu, Hengduo Li, Yingbin Zheng, Caiming Xiong, Yu-Gang Jiang, and Larry S. Davis. A coarse-to-fine framework for resource efficient video recognition. *IJCV*, 2021. 3

[52] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018. 3

[53] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. In *NeurIPS*, 2021. 2

[54] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *NeurIPS*, 2021. 2

[55] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *CVPR*, 2020. 3

[56] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *ICCV*, 2021. 1, 2, 3, 4, 5, 6

[57] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018. 2

[58] Yanyi Zhang, Xinyu Li, Chunhui Liu, Bing Shuai, Yi Zhu, Biagio Brattoli, Hao Chen, Ivan Marsic, and Joseph Tighe. Vidtr: Video transformer without convolutions. In *ICCV*, 2021. 1

[59] Linchao Zhu and Yi Yang. Label independent memory for semi-supervised few-shot video classification. *IEEE TPAMI*, 2020. 2