Speeding up Vision Transformers Through Reinforcement Learning

(Discussion Paper)

Francesco Cauteruccio[1,†], Michele Marchetti[2,†], Davide Traini[2,3,†], Domenico Ursino[2,†] and Luca Virgili[2,*,†]

[1]DIEM, University of Salerno
[2]DII, Polytechnic University of Marche
[3]CHIMOMO, University of Modena and Reggio Emilia

Abstract

In recent years, Transformers have led a revolution in Natural Language Processing, and Vision Transformers (ViTs) promise to do the same in Computer Vision. The main obstacle to the widespread use of ViTs is their computational cost. Indeed, given an image divided into a list of patches, ViTs compute, for each layer, the attention of each patch with respect to all others. In the literature, many solutions try to reduce the computational cost of attention layers using quantization, knowledge distillation, and input perturbation. In this paper, we aim to make a contribution in this setting. In particular, we propose AgentViT, a framework that uses Reinforcement Learning to train an agent whose task is to identify the least important patches during the training of a ViT. Once such patches are identified, AgentViT removes them, thus reducing the number of patches processed by the ViT. Our goal is to reduce the training time of the ViT while maintaining competitive performance.

Keywords

Vision Transformers, Training Time Reduction, Reinforcement Learning, Computer Vision, CIFAR10

1. Introduction

In recent years, thanks also to the massive development of deep learning systems, Artificial Intelligence is experiencing a golden age in many sectors, including Natural Language Processing (NLP) and Computer Vision (CV) [1, 2, 3]. Transformers are one of the key players in this development [3]. Initially designed in the context of NLP, they were adapted for Computer Vision tasks through the introduction of Vision Transformers (ViTs) [4]. The working principle of ViTs is similar to those of Transformers, but instead of dividing a sentence into words, they split an image into non-overlapping rectangular patches and look for semantic correlations between them. ViTs have proven to be very competitive, and in some contexts, their performance has been superior to that of Convolutional Neural Networks (CNNs) [4]. The main problem with

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
†These authors contributed equally.
fcauteruccio@unisa.it (F. Cauteruccio); m.marchetti@pm.univpm.it (M. Marchetti); davide.traini@unimore.it (D. Traini); d.ursino@univpm.it (D. Ursino); luca.virgili@univpm.it (L. Virgili)
0000-0001-8400-1083 (F. Cauteruccio); 0000-0003-3692-3600 (M. Marchetti); 0009-0007-3098-9349 (D. Traini); 0000-0003-1360-8499 (D. Ursino); 0000-0003-1509-783X (L. Virgili)

ViTs is their computational cost, since for each layer it is necessary to compute the attention of each token with respect to all others. To overcome this problem, several variants of ViTs have been proposed to reduce the cost of the attention layers [5, 6, 7, 8, 9].

Another area of Artificial Intelligence that has shown great promise in recent years is Reinforcement Learning (RL). In fact, it is being applied in a wide range of contexts, from robotics to intelligent transportation systems [10, 11, 12, 13, 14, 15].

In this paper, we propose AgentViT, a framework for ViT optimization. To achieve its goal, AgentViT uses RL to reduce the computational complexity of the attention layer and thus the training time of ViTs. In AgentViT, an RL agent selects a subset of the image patches so that the ViT has to process only them for its training, thus reducing the training time while maintaining competitive performances. The RL agent is a Deep Q-Learning Network [16] that returns a list of selected patches. The agent is composed of three dense layers. For each training batch, it observes the attention values produced by the first ViT layer and returns a subset of the original patches to use for the training of the ViT. After a certain number of training epochs, the agent receives a reward that takes into account training loss and training time. The user can decide how much weight to give to each of these two parameters, thus favoring a set of patches that guarantees a low training time or one that guarantees a low training loss.

Several approaches have been proposed in the literature to reduce the computational load of attention layers. They are based on different techniques such as quantization [17, 18, 19, 20, 21, 22, 23], pruning [24, 25, 26, 27], low-rank factorization [28, 29] and knowledge distillation [30, 31, 32, 33, 34, 35]. Other approaches perturbate the input of a ViT to optimize the resources it uses [36, 8, 37]. Others compute the importance of each token and remove less important tokens as inference proceeds [9, 6, 7, 38, 39, 40]. AgentViT shares with some of the above approaches the policy of setting a variable number of tokens based on the input images. This allows it to fit the images in the best possible way. However, it has a completely different fitting mechanism than the other approaches. In fact, the latter requires the user to specify the maximum number of tokens to be used. If, after resampling, the number of tokens is greater than the number specified by the user, the excess tokens are removed. AgentViT also allows the user to specify the maximum number of tokens desired, and it trains its RL agent to select a number of tokens as close as possible to the number specified by the user. However, if it obtains a particularly low training loss during training, its reward mechanism will prompt it to select a smaller number of tokens for that particular batch of images. Conversely, if it obtains a high training loss for a particular batch, its reward mechanism will prompt it to increase the number of tokens to be used, giving less weight to the number specified by the user.

This paper is organized as follows: in Section 2, we describe AgentViT. In Section 3, we illustrate our experimental campaign aimed at determining the values of its hyperparameters, comparing it with related approaches, and deriving interesting insights. Finally, in Section 4, we draw our conclusions and look at some possible future developments.

2. Description of AgentVIT

2.1. Schematic workflow of AgentViT

AgentViT uses a Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$ [41]. Here, $\mathcal{S}$ is a state space, $\mathcal{A}$ is a discrete set of actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition kernel, and $\gamma \in [0, 1)$ is a discount factor. In an MDP, a stationary policy $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from states to actions; it specifies the action an agent takes when it is in a given state. It is used to describe how an agent interacts with the environment.

AgentViT uses an Action Value Function $Q(s, a)$, introduced in Q-Learning [42], to estimate the expected cumulative reward an RL agent can obtain from a given state-action (s, a). Q-Learning uses a table with a row for each observable state and a column for each possible action. As the algorithm runs, the values in the table are updated using the formula expressed in Equation 2.1. This allows us to recursively obtain the cumulative reward $Q(s, a)$ associated with each action-state pair (s, a).

$$Q(s, a) \leftarrow Q(s, a) + \eta \cdot \left[\mathcal{R}_{s,a} + \gamma \cdot \max_{a' \in \mathcal{A}} (Q(s^*, a')) - Q(s, a) \right] \tag{2.1}$$

Here:

- η is the learning rate; it is a number in the real range $[0, 1]$ and specifies the rate at which the agent learns;
- γ is the discount factor; it represents the importance of the immediacy of the reward. If γ is closer to 0, actions with immediate rewards are favored; if γ is closer to 1, all rewards are given equal weight, regardless of their immediacy, which favors a long-term view;
- s^* is the next state, i.e., the state in which the agent arrives when it starts from s and executes the action a.

The Q-Learning algorithm struggles in the presence of a large number of states or when the states involved are continuous. In these cases, the table is replaced by a neural network called Deep Q-Network. It receives the vector representing the state s and computes the Q-values corresponding to each pair (s, a^*), where a^* represents any action the agent can take in s. The agent chooses the action with the highest Q-value.

Figure 1 illustrates the architecture of AgentViT. As the figure shows, there are two basic components in AgentViT, namely the ViT and the RL agent. The latter consists of a Deep Q-Network that implements the Deep Q-Learning algorithm. The agent computes the Q-values for each state-action pair and uses them to select the patches that the ViT will use for its training. Unlike a classical Deep Q-Network that returns only one patch, our Deep Q-Network can return multiple patches. The behavior of AgentViT can be described by the following steps: *(i)* the attention score matrix is extracted from the ViT; *(ii)* the Deep Q-Network uses this matrix to compute the Q-values and then select the patches; *(iii)* the ViT uses these patches for its training and, at the end of this last task, returns the reward to the Deep Q-Network; *(iv)* the Deep Q-Network uses this reward to tune its behavior for the next step.

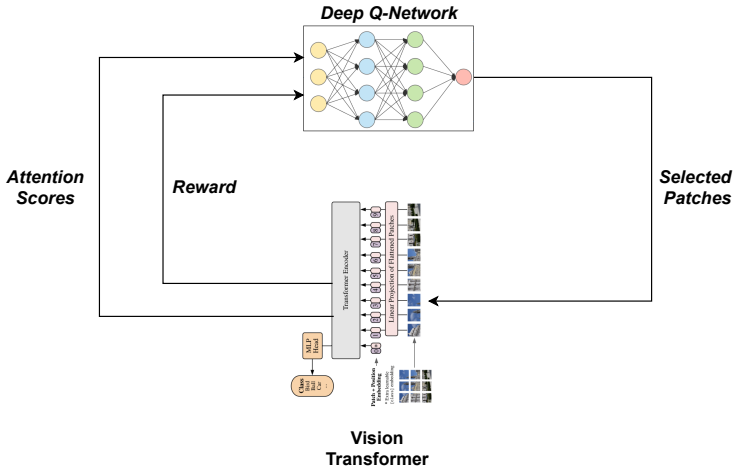

Figure 1: Schematic workflow of AgentViT

2.2. State

In the context of the MDP, a state $s \in \mathcal{S}$ observed by the agent is represented by a vector of real numbers and models the current conditions of the environment. In AgentViT, the state of an environment is represented by the attention score obtained from a batch of images processed by the first attention layer of the ViT.

More specifically, given an image composed of n patches, the output of the attention layer is represented by an $n \times d$ matrix, where d is the embedding size of the image. This matrix is given in input to a ViT module downstream of the attention layer, which transforms the matrix itself into a vector of n elements obtained by averaging the d values along the n dimensions. This vector represents the average values of attention (and thus importance) that the attention layer assigns to the different patches. It is also the state given as input to AgentViT's Deep Q-Network.

2.3. Action

Given a vector of n elements representing a state $s \in \mathcal{S}$, the agent returns a list of n elements. Each element is associated with a patch and is a real value that, according to the Deep Q-Learning algorithm, represents the cumulative reward associated with the corresponding patches as estimated by the agent [42]. The list is sorted in descending order so that the first elements represent the most important patches. AgentViT selects all patches in the list whose associated value is greater than the average value of the elements in the list. Therefore, the action $a \in \mathcal{A}$ associated with the state $s \in \mathcal{S}$ corresponds to the selection of the most promising patches.

The transition kernel function $\mathcal{P}$ (see Section 2.1) is the one provided by Deep Q-Learning. Based on it, after the agent chooses an action $a \in \mathcal{A}$ and receives a reward $\mathcal{R}_{s,a}$, the update of the Q-value associated with the pair (s, a) is done through the following formula [42]:

$$Q(s,a) \leftarrow Q(s,a) + \eta \cdot \left[\mathcal{L}_H(\mathcal{R}_{s,a} + \gamma \cdot \max_{a' \in \mathcal{A}}(Q(s^*, a')), Q(s,a))\right] \tag{2.2}$$

Similarly to Equation 2.1, this formula describes the update of $Q(s,a)$ by taking into account the previous value and the distance between the maximum cumulative reward associated with the next state s^* and the Q-value associated with the current state. In AgentViT, we adopted a Huber function $\mathcal{L}_H$ [43] to compute this distance (unlike [42] that used an algebraic difference). The reasoning behind this choice is that $\mathcal{L}_H$ is not sensitive to outliers and, in some cases, prevents the gradient explosion problem. The cumulative reward for the next state is predicted by the Target Network, which consists of an exact copy of the agent network except that its weights are not updated by backpropagation, but are periodically copied from the agent network by a soft-copy mechanism. As shown in [44], this way of proceeding allows us to stabilize the learning process.

AgentViT also has a mechanism to avoid falling into a local minimum. Indeed, the agent chooses a random action with a probability equal to ϵ instead of the action that maximizes the value of Q. The value of ϵ decays exponentially as training progresses to avoid instability. In this way, AgentViT is able to ensure good exploratory analysis in the early stages of ViT training and good stability of results as training progresses.

Finally, AgentViT uses a replay memory [45, 46] to improve the stability and generalizability of the agent. It can store observed data for later use during training in a way that breaks unwanted temporal correlations.

2.4. Reward

In AgentViT, the reward $\mathcal{R}_t$ obtained at iteration t by starting from a state s_t and executing an action a_t plays a key role since it serves to define the quality of training. As mentioned above, this quality must take into account the time required for training and the accuracy. Consequently, $\mathcal{R}_t$ must consider both the training loss and the training time. For this purpose, it is defined as a weighted mean of the training loss and the number of patches selected by the agent, which is proportional to the training time.

Based on this reasoning, $\mathcal{R}_t$ can be formulated as:

$$\mathcal{R}_t = \alpha \cdot \mathcal{R}_t^{loss} + (1-\alpha) \cdot \mathcal{R}_t^{patch} \tag{2.3}$$

Here:

- $\mathcal{R}_t^{loss}$ is the reward related to the training loss; it is equal to the ratio between the value $\mathcal{L}(0)$ of the loss function of the ViT at the starting iteration and the value $\mathcal{L}(t)$ of the same function at iteration t.
- $\mathcal{R}_t^{patch}$ is the reward related to the number of patches; it is defined as the ratio of the difference between the actual number of patches selected by the agent and the user's desired number of selected patches, to the user's desired number of selected patches.
- α is a value belonging to the real interval $[0, 1]$, that determines the weight to assign to $\mathcal{R}_t^{loss}$ with respect to $\mathcal{R}_t^{patch}$.

In this way, the agent is incentivized to select a number of patches close to the number desired by the user (or a very small number if the user does not specify a value). But, it is also incentivized to select a subset of patches that can minimize training loss. These two goals are represented by $\mathcal{R}_t^{loss}$ and $\mathcal{R}_t^{patch}$ in Equation 2.3. The weight α allows the user to specify how much importance to place on each of these goals. If α tends to 1, the agent has an incentive to choose a large number of patches. On the other hand, if α tends to 0, it is incentivized to minimize the number of patches selected, subject to the accuracy constraints to be achieved.

3. Experiments

3.1. Testbed

AgentViT can be applied to any ViT, since it is based on observing the attention scores it returns. Consequently, in our experiments, we could have applied AgentViT to any ViT proposed in the literature. To conduct the experiments in a reasonable time, we decided to employ SimpleViT [47] that splits images into 64 patches (SimpleViT64), since it can be trained faster than a classical ViT. We performed our experiments on the CIFAR10 dataset, which is a collection of 60,000 color images 32x32 divided into 10 different classes, designed for training and testing machine learning models in computer vision tasks. CIFAR10 is widely used for benchmarking classification algorithms in the deep learning field. For the training and testing phases of our experiments, we used Google Colab, which provides an Intel Xeon CPU with 2 vCPUs, 13 GB of RAM, and an NVIDIA Tesla K80 GPU with 12 GB of VRAM. We refer the reader to the link https://github.com/DavideTraini/RL-for-ViT for the code used to implement AgentViT.

As a first step in our experiments, we had to define the values of the hyperparameters of AgentViT. Due to space limitations, we cannot report in detail the tasks we performed to determine these values. At the end of these tasks, we obtained the values reported in Table 1.

As a next step, we decided to compare the performance of AgentViT with that of related approaches already proposed in the literature. Specifically, the approaches we considered for comparison are the original ViT, SimpleViT64, and ATSViT [6]; the latter, to the best of our knowledge, is the approach most similar to AgentViT. For each of these approaches, we computed their Cumulative Training Time (measured in seconds), Accuracy, Precision, Recall, and F1-Score. Table 2 shows the corresponding values.

This table shows that there are approaches able to guarantee low values of Cumulative Training Time, but at the expense of Accuracy, Precision, Recall, and F1-Score. Conversely, other approaches can obtain high values of these measures, but at the expense of Cumulative Training Time. AgentViT is able to achieve a suboptimal value for all five metrics. In other words, it does not achieve the maximum value for any metric but is able to ensure the best compromise among all metrics.

3.2. Discussion

As seen above, the goal of AgentViT is to use RL to select patches for optimal filtering. Comparison with other related approaches has shown us that it has a satisfactory performance. In fact, it is able to train a ViT in less time than the ViT itself, if it is trained without removing

Hyperparameter	*Value*
`patch_size`	4×4 pixels
`dim`	128
`depth`	6
`heads`	16
`mlp_dim`	512
`buffer_batch_size`	64
`buffer_size`	1,024
`gamma`	0.95
`eps_start`	1
`eps_end`	0.01
`eps_decay`	20,000
`eta`	0.01
`tau`	0.1
`update_every`	2
`frequency`	10
`n_patches_desired`	40
`alpha`	0.2

Table 1
Values of the hyperparameters of AgentViT

Approach	*Cumulative Training Time* (s)	*Accuracy*	*Precision*	*Recall*	*F1-Score*
OriginalViT	5,955	0.8377	0.8136	0.8571	0.8348
SimpleViT64	4,870	0.7844	0.7917	0.7857	0.7886
ATSViT	5,813	0.7429	0.7324	0.7544	0.7432
AgentViT	3,730	0.8011	0.8013	0.8010	0.7997

Table 2
Values of Cumulative Training Time, Accuracy, Precision, Recall and F1-Score obtained by the approaches considered in our experiments

patches. This saving in training time does not come at the expense of accuracy, which remains comparable to that of the SimpleViT trained without patch removal. Moreover, AgentViT allows the user to specify the desired trade-off between accuracy and training time. Finally, AgentViT is the approach capable of providing the best trade-off between Cumulative Training Time on the one hand, and Accuracy, Precision, Recall, and F1-Score on the other hand.

In addition, AgentViT has other interesting implications. One of them is the possibility of using larger Vision Transformers. In fact, AgentViT's ability to reduce training time makes it possible to adopt architectures that would not normally have been adopted due to their excessive computational load. A second implication concerns the use of AgentViT to build smaller synthetic datasets from the original ones, which can be used to train deep neural networks. A further implication concerns the possibility of extending the use of AgentViT to contexts other than Vision Transformers. In fact, the idea behind AgentViT is general and independent of the type of transformers to which it is applied; therefore, it could work with any transformers, such as those used in the context of NLP. The only condition is that the input of the RL agent within AgentViT can receive an attention matrix as input.

Finally, we highlight some limitations of AgentViT. The first concerns the fact that the decision to use Deep Q-Learning within AgentViT involves the need to set various hyperparameters, which makes the setup phase rather complex. A second limitation is related to the number of patches required for AgentViT to work properly. In fact, if the Vision Transformer underlying AgentViT works with only a few patches, the agent has difficulty selecting the most important ones.

4. Conclusion

In this paper, we proposed AgentViT, a framework that uses RL to reduce the training time of a ViT without significantly reducing its performance. The RL agent present in AgentViT uses a classical MDP-based mechanism to represent an environment for the image classification task. As for this process, we redefined the state, action, and reward needed to train our RL agent. We tested AgentViT using SimpleViT64 as the internal ViT and Deep Q-Network as the internal RL agent. Our experiments showed that AgentViT can achieve the best trade-off between Cumulative Training Time on the one hand and Accuracy, Precision, Recall, and F1-Score on the other hand. The experiments conducted allowed us to draw several implications regarding the strengths and limitations of our framework.

We can think of several possible future developments of our approach. For example, we could improve the reward function to consider validation loss instead of training loss. We could also define a new metric, similar to the Akaike information criterion [48], which takes into account both model performance and the number of tokens. Moreover, we could test other Reinforcement Learning algorithms, such as Multi-Agent RL and Contextual Multi-Armed Bandit, instead of the Deep Q-Network, and see if they can further improve the performance of ViTs. These algorithms could assist in selecting the best actions and the corresponding patches to speed up ViT training. Finally, we could evaluate the impact of our approach on different ViT architectures, possibly including multiple attention layers, which would make our framework more robust and versatile.

References

[1] B. Min, H. Ross, E. Sulem, A. Veyseh, T. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, ACM Computing Surveys 56 (2023) 1–40. ACM.

[2] J. Chai, H. Zeng, A. Li, E. Ngai, Deep learning in computer vision: A critical review of emerging techniques and application scenarios, Machine Learning with Applications 6 (2021) 100134. Elsevier.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is All you Need, Advances in Neural Information Processing Systems 30 (2017). Curran Associates, Inc.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, arXiv preprint arXiv:2010.11929 (2020).

[5] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, arXiv preprint arXiv:1904.10509 (2019).
[6] M. Fayyaz, S. A. Koohpayegani, F. R. Jafari, S. Sengupta, H. R. V. Joze, E. Sommerlade, H. Pirsiavash, J. Gall, Adaptive token sampling for efficient vision transformers, in: Proc. of the European Conference on Computer Vision (ECCV'22), Tel Aviv, Israel, 2022, pp. 396–414. Springer.
[7] H. Yin, A. Vahdat, J. Alvarez, A. Mallya, J. Kautz, P. Molchanov, A-vit: Adaptive tokens for efficient vision transformer, in: Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR'22), New Orleans, LA, USA, 2022, pp. 10809–10818.
[8] C. Renggli, A. S. Pinto, N. Houlsby, B. Mustafa, J. Puigcerver, C. Riquelme, Learning to merge tokens in vision transformers, arXiv preprint arXiv:2202.12015 (2022).
[9] Y. Liang, C. Ge, Z. Tong, Y. Song, J. Wang, P. Xie, Not all patches are what you need: Expediting vision transformers via token renotes, arXiv preprint arXiv:2202.07800 (2022).
[10] A. S. Polydoros, L. Nalpantidis, Survey of model-based reinforcement learning: Applications on robotics, Journal of Intelligent & Robotic Systems 86 (2017) 153–173. Springer.
[11] A. Coronato, M. Naeem, G. D. Pietro, G. Paragliola, Reinforcement learning for intelligent healthcare applications: A survey, Artificial Intelligence in Medicine 109 (2020) 101964. Elsevier.
[12] R. Nian, J. Liu, B. Huang, A review on reinforcement learning: Introduction and applications in industrial process control, Computers & Chemical Engineering 139 (2020) 106886. Elsevier.
[13] N. Luong, D. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, D. I. Kim, Applications of deep reinforcement learning in communications and networking: A survey, IEEE Communications Surveys & Tutorials 21 (2019) 3133–3174. IEEE.
[14] A. Haydari, Y. Yılmaz, Deep reinforcement learning for intelligent transportation systems: A survey, IEEE Transactions on Intelligent Transportation Systems 23 (2020) 11–32. IEEE.
[15] W. Fang, L. Pang, W. Yi, Survey on the application of deep reinforcement learning in image processing, Journal on Artificial Intelligence 2 (2020) 39–58.
[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).
[17] K. Wang, Z. Liu, Y. Lin, J. Lin, S. Han, Haq: Hardware-aware automated quantization with mixed precision, in: Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR'19), Long Beach, CA, USA, 2019, pp. 8612–8620.
[18] Y. Gong, Y. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, arXiv preprint arXiv:1412.6115 (2014).
[19] Z. Yuan, C. X. e Y. Chen e Q. Wu e G. Sun, Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization, in: Proc. of the European Conference on Computer Vision (ECCV'22), Tel Aviv, Israel, 2022, pp. 191–207. Springer.
[20] Y. Lin, T. Zhang, P. Sun, Z. Li, S. Zhou, Fq-vit: Post-training quantization for fully quantized vision transformer, arXiv preprint arXiv:2111.13824 (2021).
[21] Y. Ding, H. Qin, Q. Y. e Z. Chai e J. Liu e X. Wei e X. Liu, Towards Accurate Post-Training Quantization for Vision Transformer, in: Proc. of the International Conference on Multimedia (MM'22), Lisbon, Portugal, 2022, pp. 5380–5388.
[22] Z. Li, T. Yang, P. Wang, J. Cheng, Q-vit: Fully differentiable quantization for vision

transformer, arXiv preprint arXiv:2201.07703 (2022).
[23] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, W. Gao, Post-training quantization for vision transformer, Advances in Neural Information Processing Systems 34 (2021) 28092–28103.
[24] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proc. of the International Conference on Computer Vision (ICCV'17), Venice, Italy, 2017, pp. 1389–1397.
[25] Y. Rao, J. Lu, J. Lin, J. Zhou, Runtime network routing for efficient image classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 41 (2018) 2291–2304. IEEE.
[26] M. Zhu, Y. Tang, K. Han, Vision transformer pruning, arXiv preprint arXiv:2104.08500 (2021).
[27] F. Yu, K. Huang, M. W. e Y. Cheng e W. Chu e L. Cui, Width & depth pruning for vision transformers, in: Proc. of the International Conference on Artificial Intelligence (AAAI'22), volume 36, Virtual Only, 2022, pp. 3143–3151.
[28] X. Yu, T. Liu, X. Wang, D. Tao, On compressing deep models by low rank and sparse decomposition, in: Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR'17), Honolulu, HI, USA, 2017, pp. 7370–7379.
[29] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, arXiv preprint arXiv:1405.3866 (2014).
[30] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015).
[31] B. Liu, Y. Rao, J. Lu, J. Zhou, C. Hsieh, Metadistiller: Network self-boosting via meta-learned top-down distillation, in: Proc. of the European Conference on Computer Vision (ECCV'20), Glasgow, Scotland, UK, 2020, pp. 694–709. Springer.
[32] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, M. Zhou, Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, Advances in Neural Information Processing Systems 33 (2020) 5776–5788.
[33] R. Liu, K. Yang, A. Roitberg, J. Zhang, K. Peng, H. Liu, R. Stiefelhagen, TransKD: Transformer knowledge distillation for efficient semantic segmentation, arXiv preprint arXiv:2202.13393 (2022).
[34] X. Chen, Q. Cao, Y. Zhong, J. Zhang, S. G. e D. Tao, Dearkd: data-efficient early knowledge distillation for vision transformers, in: Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR'22), New Orleans, LA, USA, 2022, pp. 12052–12062.
[35] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, H. Jégo, Training data-efficient image transformers & distillation through attention, in: Proc. of the International Conference on Machine Learning (ICML'21), Virtual Only, 2021, pp. 10347–10357. PMLR.
[36] X. He, C. Li, P. Zhang, J. Yang, X. E. Wang, Parameter-efficient model adaptation for vision transformers, in: Proc. of the International Conference on Artificial Intelligence (AAAI'23), volume 37, Washington, DC, USA, 2023, pp. 817–825.
[37] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, C. Hsieh, Dynamicvit: Efficient vision transformers with dynamic token sparsification, Advances in Neural Information Processing Systems 34 (2021) 13937–13949.
[38] Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. X. e D. Tao, Patch slimming for efficient vision transformers, in: Proc. of the International Conference on Computer Vision and Pattern

Recognition (CVPR'22), New Orleans, LA, USA, 2022, pp. 12165–12174.
[39] Y. Rao, Z. Liu, W. Zhao, J. Zhou, J. Lu, Dynamic spatial sparsification for efficient vision transformers and convolutional neural networks, IEEE Transactions on Pattern Analysis and Machine Intelligence (2023). IEEE.
[40] L. Meng, H. Li, B. Chen, S. Lan, Z. Wu, Y. J. e S.N. Lim, Adavit: Adaptive vision transformers for efficient image recognition, in: Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR'22), New Orleans, LA, USA, 2022, pp. 12309–12318.
[41] C. L. Lan, S. Tu, A. Oberman, R. Agarwal, M. Bellemare, On the generalization of representations in reinforcement learning, arXiv preprint arXiv:2203.00543 (2022).
[42] K. Arulkumaran, M. Deisenroth, M. Brundage, A. Bharath, Deep Reinforcement Learning: A brief survey, IEEE Signal Processing Magazine 34 (2017) 26–38. IEEE.
[43] P. J. Huber, Robust estimation of a location parameter, in: Breakthroughs in statistics: Methodology and distribution, Springer, 1992, pp. 492–518.
[44] J. Fan, Z. Wang, Y. Xie, Z. Yang, A theoretical analysis of deep Q-learning, in: Proc. of the International Conference on Learning for Dynamics and Control (L4DC'20), Berkeley, CA, USA, 2020, pp. 486–489. PMLR.
[45] R. Liu, J. Zou, The effects of memory replay in reinforcement learning, in: Proc. of the Annual Allerton Conference on Communication, Control, and Computing (Allerton'18), Monticello, IL, USA, 2018, pp. 478–485. IEEE.
[46] L. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Machine Learning 8 (1992) 293–321. Springer.
[47] L. Beyer, X. Zhai, A. Kolesnikov, Better plain ViT baselines for ImageNet-1k, arXiv preprint arXiv:2205.01580 (2022).
[48] H. Akaike, A new look at the statistical model identification, IEEE Transactions on Automatic Control 19 (1974) 716–723. IEEE.