MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Investigation of Methods Improving the Efficiency of Vision Transformers

Master's Thesis

Author
Shukhrat Kulboboev

Advisor
Bencsik Blanka

December 9, 2024

Contents

Abstract

Vision Transformers (ViTs) have gained significant attention in recent years as a powerful alternative to Convolutional Neural Networks (CNNs) for computer vision tasks. Unlike CNNs, which rely on local convolutions, ViTs leverage self-attention mechanisms to model long-range dependencies in images, enabling them to capture rich global context and achieve state-of-the-art performance across a variety of vision tasks. However, despite their impressive performance, ViTs suffer from high computational complexity, making them resource-intensive and unsuitable for deployment in resource-constrained environments such as mobile devices and embedded systems.

This thesis investigates methods to improve the efficiency of Vision Transformers, with a particular focus on token pruning and token pooling techniques. These methods aim to reduce the number of tokens processed by the transformer model, thus alleviating the computational burden. Building upon the Token Pruning and Pooling techniques, we propose an integrated approach that combines both token pruning, which removes less important or redundant tokens, and token pooling, which merges similar tokens. By heuristically adapting these techniques across different layers of the ViT architecture, we seek to optimize both the model's efficiency and accuracy.

In this work, we analyze the impact of varying pruning ratios on Vision Transformers and investigate how the architecture responds to these reductions. Our experiments are conducted on standard benchmark datasets, including CIFAR-100 and ImageNet, to evaluate the performance of the proposed method in terms of accuracy, computational efficiency (measured by FLOPs), and memory footprint. The results demonstrate that our approach achieves significant improvements in computational efficiency, reducing the number of floating-point operations (FLOPs) and memory usage , while maintaining competitive accuracy compared to the baseline ViT models. Moreover, we show that the proposed method does not suffer from the typical trade-offs between efficiency and performance, thus presenting a promising solution for resource-efficient ViT deployment.

This study contributes to the growing body of research aimed at making Vision Transformers more efficient for real-world applications. By combining token pruning and pooling in a single framework, this work highlights the potential of these techniques to scale Vision Transformers for deployment in resource-constrained environments without compromising performance. Our findings underscore the importance of model optimization strategies and set the stage for future research into efficient deep learning models for computer vision.

Chapter 1

Introduction

In recent years, advancements in deep learning have transformed the field of computer vision, enabling machines to achieve near-human performance on a variety of tasks, including image classification, object detection, semantic segmentation, and more. Convolutional Neural Networks (CNNs) have traditionally dominated the computer vision landscape due to their ability to efficiently capture local spatial features using convolutional operations. However, CNNs are inherently limited in their ability to model long-range dependencies due to their local receptive field, which can result in suboptimal performance for tasks requiring a global understanding of an image.

The advent of Vision Transformers (ViTs) has introduced a paradigm shift in computer vision by utilizing the self-attention mechanism, originally proposed for natural language processing, to model global relationships within an image. By dividing an image into patches and treating these patches as tokens, ViTs process the entire image contextually, capturing both local and global dependencies simultaneously. This capability has allowed ViTs to achieve state-of-the-art results on several benchmarks, surpassing traditional CNNs in many scenarios.

Despite their remarkable performance, the computational cost of ViTs remains a critical challenge. The quadratic scaling of the self-attention mechanism with respect to the number of tokens results in high memory and computational requirements, making ViTs impractical for resource-constrained environments such as mobile devices, embedded systems, or real-time applications. As a result, developing techniques to optimize the efficiency of Vision Transformers without compromising their performance has become a pressing need in the field of deep learning.

The primary objective of this thesis is to improve the efficiency of Vision Transformers by integrating token pruning and pooling techniques into a unified framework. Inspired by the Token Pruning and Pooling for Efficient Vision Transformers (PPT) , this work investigates the potential of combining these two techniques to address both inattentive and duplicative redundancy across different layers of the Vision Transformer architecture. By adaptively applying token pruning and pooling, the study aims to achieve the following objectives:

- **Efficiency Improvement.** Reduce the computational cost and memory footprint of Vision Transformers without compromising their predictive accuracy.
- **Comprehensive Evaluation.** Analyze the impact of varying pruning and pooling ratios on model performance, including accuracy, FLOPs, and inference speed.

- **Scalability for Real-World Applications.** Demonstrate the applicability of the proposed approach on standard benchmark datasets, such as CIFAR-100 and ImageNet, and evaluate its potential for deployment in resource-constrained environments.

This research contributes to the growing demand for efficient deep learning models by addressing a critical challenge in Vision Transformers. By integrating token pruning and pooling, this study proposes a scalable solution that can significantly reduce the computational complexity of ViTs while maintaining their competitive performance. The findings of this work have important implications for the deployment of Vision Transformers in practical applications, including autonomous vehicles, robotics, and edge computing, where resource efficiency is paramount.

The key contributions of this thesis are as follows:

1. **Optimizing a Unified Pruning and Pooling Method:** A reproducible method that combines token pruning and pooling to comprehensively reduce redundancy in Vision Transformers.
2. **Empirical Analysis and Benchmarking:** A detailed evaluation of reproducible method on standard datasets, including CIFAR-100 and ImageNet, to validate its effectiveness in terms of computational efficiency and accuracy.
3. **Insights into ViT Optimization:** A thorough analysis of the trade-offs between efficiency and performance, offering valuable insights into the resilience of Vision Transformers under varying degrees of token reduction with different methods.
4. **Practical Guidelines for Efficient ViT Deployment:** Recommendations for optimizing ViTs in resource-constrained environments, paving the way for future research in efficient model design.

The structure of this thesis is organized as follows:

1. **Chapter 2:** Provides a comprehensive review of related work, including an overview of Vision Transformers, existing pruning and pooling techniques, and other approaches to model optimization.
2. **Chapter 3:** Details the Vision Transformer architecture, token redundancy, and the proposed unified pruning methods for token pruning and pooling.
3. **Chapter 4:** Defines the problem statement and briefly introduces the proposed solution.
4. **Chapter 5:** Describes the experimental setup, including datasets, evaluation metrics, and implementation details.
5. **Chapter 6:** Presents the experimental results, including comparisons with baseline models, and provides a detailed analysis of the findings.
6. **Chapter 7:** Discusses the implications of the results, challenges encountered during the research, and potential directions for future work.

Chapter 2

Theoretical Background

2.1 Introduction to Vision Transformers (ViTs)

2.1.1 What are Vision Transformers?

The Vision Transformers, or ViTs for short, combine two influential fields in artificial intelligence: computer vision and natural language processing (NLP). The Transformer model, originally proposed in the paper titled "Attention Is All You Need" by Vaswani et al. in 2017 [24], serves as the foundation for ViTs. Transformers were designed as a neural network architecture that excels in handling sequential data, making them ideal for NLP tasks.Then,introduced by Dosovitskiy et al. in 2020 [6], Vision Transformers have significantly shifted the paradigm in computer vision by replacing convolutional operations—commonly used in Convolutional Neural Networks (CNNs)[20]—with the self-attention mechanism. Unlike Convolutional Neural Networks (CNNs), which rely on convolutional layers to extract local spatial features from images, Vision Transformers model global relationships between image regions, making them particularly effective for tasks requiring a comprehensive understanding of an image.

2.1.2 The Structure of Vision Transformers

Vision Transformer (ViT) is an innovative deep learning architecture designed to process visual data using the same transformer architecture that revolutionized natural language processing (NLP). Unlike convolutional neural networks (CNNs), which rely on convolutions to capture local spatial features, Vision Transformers adopt the self-attention mechanism to model global relationships across image patches. This architecture has demonstrated state-of-the-art performance in many computer vision tasks such as image classification, object detection, and segmentation.

Figure 2.1 provides an overview of the architecture. The standard Transformer processes a 1D sequence of token embeddings. For handling 2D images, we transform the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where H and W represent the height and width of the image, C is the number of channels, P is the patch size, and $N = \frac{H \times W}{P^2}$ is the total number of patches, which also serves as the sequence length for the Transformer. These patches are flattened and projected to a fixed dimension D through a trainable linear transformation(Eq. 2.1), generating the patch embeddings.

We add a learnable embedding at the beginning of the patch sequence. The final state of this token after passing through the Transformer encoder is used as the image representation y (Eq. 2.4). During both pre-training and fine-tuning, this token is followed by a classification head. At pre-training, the classification head is implemented as an MLP with one hidden layer, while during fine-tuning, it is a simple linear layer.

To preserve the spatial information of the patches, we add learnable 1D positional encodings to the patch embeddings. This resulting sequence of position-encoded embeddings is then fed into the Transformer encoder.

The Transformer encoder alternates between multi-head self-attention (MSA) layers and MLP blockss (Eq. 2.2, 2.3). Each block is preceded by layer normalization (LN) and followed by a residual connection[1]. The MLP consists of two layers with a GELU activation.

The process can be mathematically expressed as follows:

Generating patch embeddings with positional encodings:

$$\mathbf{z}_0 = [\mathbf{x}_{class}; \mathbf{x}_1^p E; \mathbf{x}_2^p E; \ldots; \mathbf{x}_N^p E] + E_{pos} \tag{2.1}$$

Applying multi-head self-attention:

$$\mathbf{z}_\ell = \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L \tag{2.2}$$

Applying MLP:

$$\mathbf{z}_\ell = \mathrm{MLP}(\mathrm{LN}(\mathbf{z}_\ell)) + \mathbf{z}_\ell, \quad \ell = 1 \ldots L \tag{2.3}$$

The final image representation:

$$\mathbf{y} = \mathrm{LN}(\mathbf{z}_L) \tag{2.4}$$

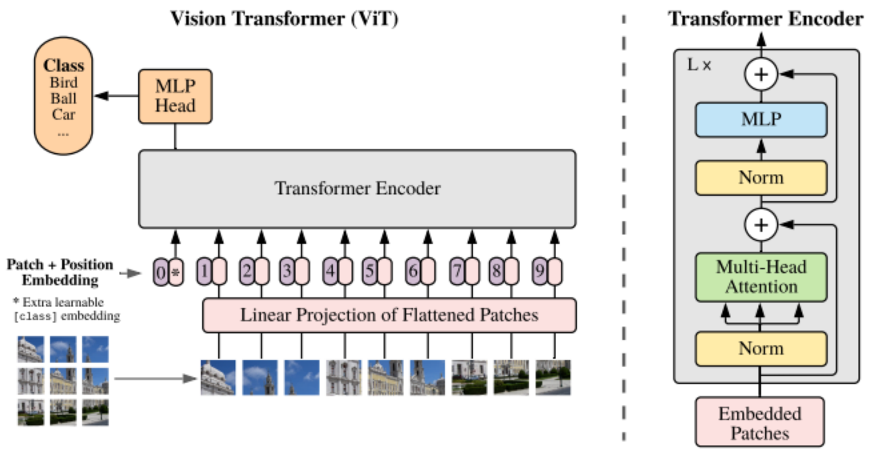

Figure 2.1: Model Overview: We divide an image into fixed-size patches, apply a linear embedding to each patch, incorporate position embeddings, and then pass the resulting sequence of vectors through a standard Transformer encoder. For classification, we follow the typical approach of appending an additional learnable "classification token" to the sequence.

We refer to our specific attention mechanism as "Scaled Dot-Product Attention" (Figure 2.2). The input consists of queries and keys(Eq. 2.5), each with dimension d_k, and values with dimension d_v. The dot products of the query with all keys are computed, scaled by $\sqrt{d_k}$, and then a softmax function is applied to obtain the weights for the values.

In practice, we process a set of queries simultaneously, packed together into a matrix Q, while the keys and values are grouped into matrices K and V, respectively. The matrix of outputs is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.5}$$

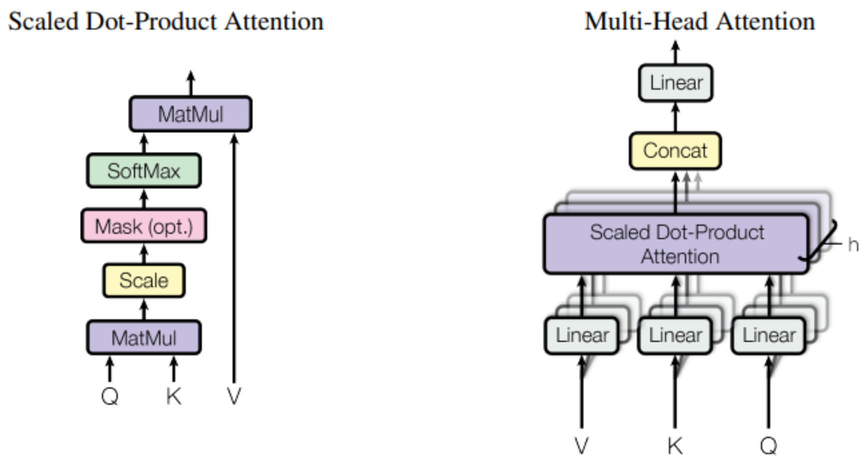

Figure 2.2: Scaled Dot-Product Attention. Multi-Head Attention

Instead of applying a single attention function with d_{model}-dimensional keys, values, and queries, we found it more beneficial to linearly project the queries, keys, and values h times using different learned linear projections to d_k, d_k, and d_v dimensions, respectively. We then perform the attention function in parallel on each projected version of the queries, keys, and values, yielding d_v-dimensional output values. These outputs are concatenated and projected once more, resulting in the final values, as shown in Figure 2.2. Multi-head attention(Eq. 2.6) enables the model to jointly attend to information from different representation subspaces at different positions. In contrast, a single attention head averages these contributions, limiting the model's ability to capture diverse information.

The multi-head attention mechanism can be defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O \tag{2.6}$$

where each attention head(Eq. 2.7) head_i is computed as:

$$\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i}) \tag{2.7}$$

The complete architecture of the transformer model is illustrated in Figure 2.3. Neural sequence transduction models often adopt an encoder-decoder architecture [4, 2, 23]. In this framework, the encoder transforms an input sequence of symbolic representations, $(x_1, \ldots, x_n)$, into a corresponding sequence of continuous representations, $(z_1, \ldots, z_n)$. The decoder then uses these continuous representations to produce an output sequence, $(y_1, \ldots, y_m)$, one symbol at a time. This process is auto-regressive [9], where the decoder incorporates previously generated symbols as additional inputs while predicting the next symbol.The Transformer model adheres to this general architecture, employing stacked self-attention mechanisms and point-wise fully connected layers for both its encoder and decoder components. The left and right sections of Figure 2.3 illustrate the encoder and decoder, respectively.

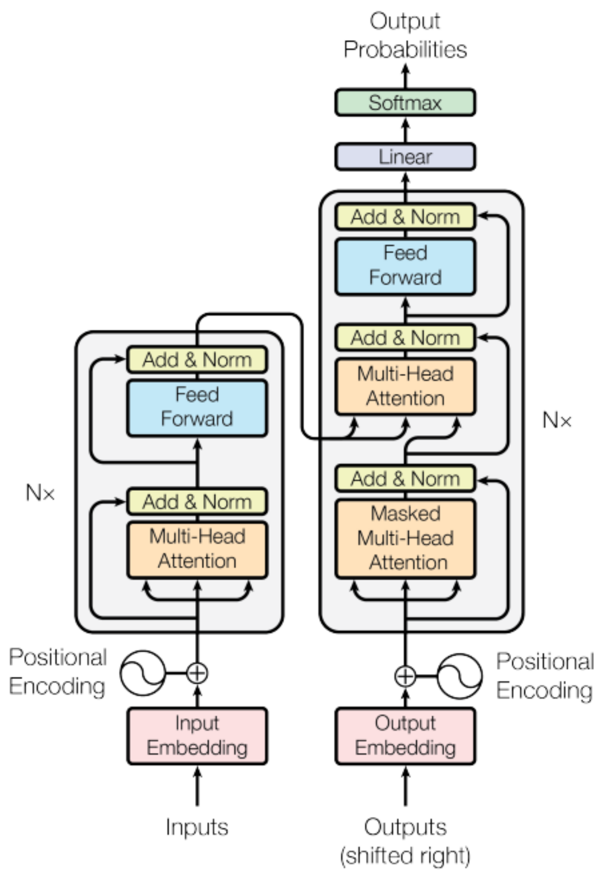

Figure 2.3: The Transformer - full model architecture

2.2 Understanding Efficiency in Vision Transformers

2.2.1 Definition of Efficiency

Efficiency in Vision Transformers (ViTs) refers to their ability to achieve high predictive performance while minimizing resource consumption in terms of computational power, memory, and inference time. The main aspects of efficiency in ViTs include:

- **Computational Cost:** Measured in floating-point operations (FLOPs), computational cost quantifies the operations required for processing an input. Reducing FLOPs is crucial for making ViTs practical for large-scale and real-time applications.

- **Memory Footprint:** Refers to the memory usage during training and inference. An efficient ViT reduces memory consumption, enabling deployment on resource-constrained devices.
- **Inference Speed:** Represents the time taken by the model to process an input and generate an output. Faster inference is a critical factor for real-time applications.
- **Model Size:** Defined by the number of parameters in the architecture, model size impacts storage requirements and transferability to edge devices.
- **Accuracy Retention:** Efficiency optimizations should maintain or minimally impact the model's predictive accuracy to ensure reliability.

Efficiency is particularly important for making Vision Transformers suitable for deployment in scenarios with resource limitations, such as mobile and edge devices, autonomous systems, and IoT applications.

2.2.2 Factors Affecting Efficiency

Several factors influence the efficiency of Vision Transformers. Understanding these factors is essential for designing models that balance performance and resource utilization. The key factors include:

1. **Token Redundancy:** Vision Transformers process images as sequences of tokens representing image patches. Redundant tokens, which carry limited or repetitive information, increase computational cost without contributing to improved accuracy. Efficient token pruning techniques address this issue by eliminating less important tokens.
2. **Self-Attention Complexity:** The quadratic scaling of the self-attention mechanism with the sequence length significantly impacts computational requirements. Efficient self-attention mechanisms aim to reduce this complexity without degrading performance.
3. **Model Depth and Width:** The number of layers and the size of each layer in a ViT influence its computational cost and memory requirements. Techniques such as layer pruning and width reduction can enhance efficiency.
4. **Parameter Redundancy:** Excessive parameters in the model can lead to inefficient resource usage. Compression techniques such as pruning, quantization, and knowledge distillation reduce redundancy while retaining performance.
5. **Positional Encoding Scheme:** Positional encodings are used to retain spatial information in ViTs. The choice of encoding (e.g., fixed vs. learnable) impacts both efficiency and accuracy.
6. **Hardware and Software Optimization:** Efficient implementation of ViTs also depends on optimized hardware and software. Techniques like mixed precision training, parallelization, and hardware accelerators (e.g., GPUs, TPUs) play a vital role.
7. **Dataset Characteristics:** The nature of the dataset, such as image resolution and diversity, influences the computational and memory demands of the ViT. Adjusting the model to match dataset requirements can improve efficiency.

By addressing these factors, researchers can design Vision Transformers that are not only accurate but also practical for deployment in real-world applications, especially in resource-constrained environments.

2.3 Pruning in Neural Networks

2.3.1 Overview of Pruning

Pruning refers to the process of reducing the number of parameters in a neural network by removing certain connections, neurons, or layers. The goal is to decrease the model's size and computational requirements, making it more efficient without compromising its performance[10, 5]. Pruning can be applied to different parts of the network, including weights, filters, neurons(Figure 2.4), or entire layers, depending on the type of pruning technique used. Pruning can be applied in both structured and unstructured forms, where unstructured pruning involves removing individual weights, and structured pruning targets larger components such as neurons or filters. A key advantage of pruning is that it can lead to a more compact and efficient model, which is easier to deploy in resource-constrained environments like mobile devices or edge computing systems.Pruning techniques can be categorized into three main types: **unstructured**, **structured**, and **semi-structured pruning**. Each approach targets different components of the neural network and offers different trade-offs in terms of computational efficiency and model performance.

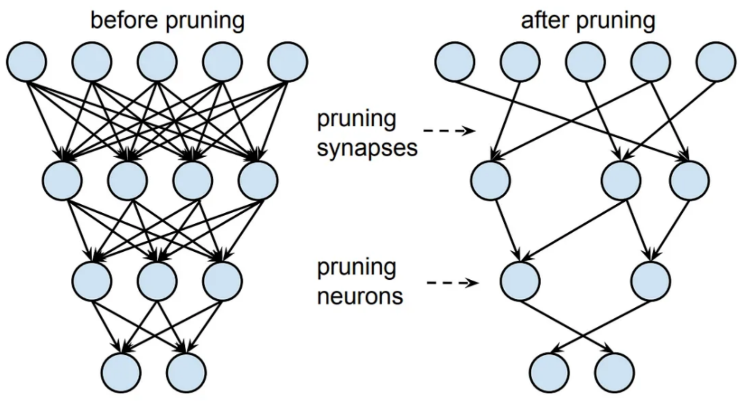

Figure 2.4: Example for pruning in order to understand the insight

2.3.2 Types of Pruning

Modern deep neural networks, especially large language models, have seen significant growth in size, demanding substantial computational and storage resources. This has led to challenges in deploying these models on devices with limited resources and ensuring fast inference times. Pruning techniques, which aim to reduce the size and complexity of neural networks, have emerged as a popular solution for addressing these challenges. By selectively removing unimportant weights or neurons, pruning can reduce both memory usage and computation time without significantly degrading model performance. Over the past few years, pruning has gained considerable attention, with over three thousand research papers published between 2020 and 2024. Below, we discuss the core concepts and techniques related to pruning.

2.3.2.1 Unstructured Pruning

Unstructured pruning, also known as non-structured pruning or weight-wise pruning, is considered the finest-grained case.

Definition 1 (Unstructured Pruning). Given a neural network with weights $W = \{w_0, w_1, ..., w_K\}$, a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ consisting of input-output pairs, and a target number of non-zero weights k (where k is less than K), unstructured pruning can be formulated as the following constrained optimization(Eq. 2.8) problem[11]:

$$\min_{W} \mathcal{L}(W; D) = \min_{W} \frac{1}{N} \sum_{i=1}^{N} \ell(W; (x_i, y_i)), \quad \text{subject to} \quad \|W\|_0 \leq k. \tag{2.8}$$

In practice, for small or medium-sized models, unstructured pruning typically does not directly set the weights to zero but instead applies a mask (or indicator) M [25, 14] that is set to zero. In this case, unstructured pruning can be considered as applying a binary mask to each weight. Thus, Eq. (2.8) is modified as follows:

$$\min_{W,M} \mathcal{L}(W \odot M; D) = \min_{W,M} \frac{1}{N} \sum_{i=1}^{N} \ell(W \odot M; (x_i, y_i)), \quad \text{subject to} \quad \|M\|_0 \leq k. \tag{2.9}$$

Usually, the network is retrained (via fine-tuning or training from scratch) with fixed masks M, and the weights corresponding to the mask are not involved in retraining. For large models, such as large language models (LLMs), assigning a mask to each weight is challenging due to the immense number of weights. Therefore, it is common to directly set the weights to zero[8, 22]. An example of weight-wise pruning is shown in Figure 2.5, where weights are either removed directly (as shown in Fig. 3(left)) or masked using their corresponding masks (as shown in Fig. 3(right)). Since this method can prune weights at arbitrary locations, its irregular pattern of removing non-zero weights enables actual acceleration, though it requires special software and hardware support. Consequently, unstructured pruning is classified as a specific speedup technique.

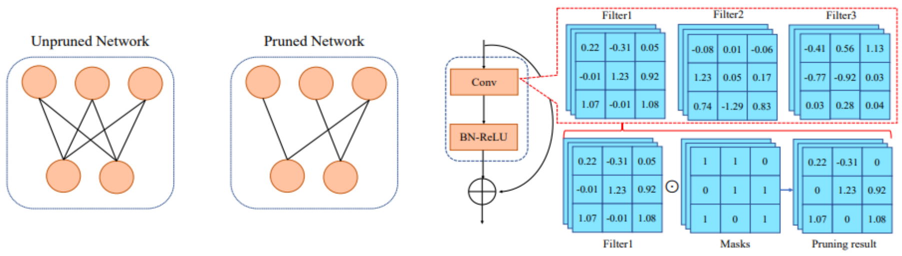

Figure 2.5: (left) From the view of neurons and connections . (right) From the view of weights and masks

Some of the common unstructured pruning methods include:

- **Magnitude-Based Pruning**: This method removes weights with the smallest absolute values, assuming that smaller weights contribute less to the model's output.

- **Gradient-Based Pruning**: This technique evaluates the gradient values for each weight, removing weights whose removal has minimal impact on the network's loss function.
- **Random Pruning**: This method removes a random subset of weights without regard to their magnitude or gradient, often used as a baseline.

Unstructured pruning typically results in sparse weight matrices, which can be stored efficiently. However, this sparsity does not necessarily lead to speedups during inference unless specialized hardware or algorithms are used to exploit the sparsity.

2.3.2.2 Structured Pruning

Given a specified pruning ratio and a neural network with $S = \{s_1, s_2, \dots, s_L\}$, where s_i can represent sets of channels, filters, neurons, or transformer attention heads in layer i, structured pruning seeks to determine $S' = \{s'_1, s'_2, \dots, s'_L\}$, where $s'_i \subseteq s_i$ for all $i \in \{1, \dots, L\}$, with the goal of minimizing performance degradation and maximizing speedup under the given pruning ratio.

Structured pruning removes entire components such as filters [30], channels [14], transformer attention heads [17], or even layers [18], as shown in Figures 2.6 right and left. This approach results in a more compact model with a regular structure. It does not require specialized hardware or software (e.g., sparse convolution libraries) and can directly accelerate networks and reduce the size of the neural networks [15].

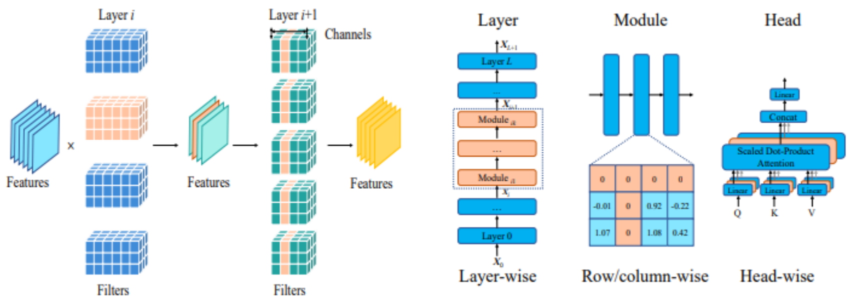

Figure 2.6: (left) Structured pruning for CNN . (right) Structured pruning for Transformers

Common types of structured pruning include:

- **Neuron Pruning**: This approach removes entire neurons from the network based on their importance, typically determined by their contribution to the network's output.
- **Filter Pruning**: Filters (or channels) in convolutional layers are removed based on their importance or contribution to the feature extraction process.
- **Layer Pruning**: Entire layers are removed from the network, typically in deeper layers where the contribution to performance may be less significant.

Structured pruning is particularly advantageous because it leads to reduced memory usage and faster computation during inference, as the network becomes more compact and efficient. However, it may result in more significant loss of accuracy compared to unstructured pruning, as entire components are discarded.

2.3.2.3 Semi-Structured Pruning

To improve the flexibility of structured pruning and reduce the accuracy drop when the pruning rate is high, some studies introduce **semi-structured pruning** (Figure 2.7), also referred to as **pattern-based pruning** [16]. This technique aims to achieve high accuracy and structural regularity simultaneously. For example, Meng et al. [19] treat a filter as several stripes and propose pruning individual stripes within each filter. SparseGPT [8] introduces a 2:4 or 4:8 sparsity pattern to halve the number of parameters in a large language model (LLM). In this configuration, at least two zeros must be present in every set of four consecutive values. The 2:4 pattern leverages NVIDIA Ampere GPU's sparse tensor cores to accelerate matrix multiplication [28].

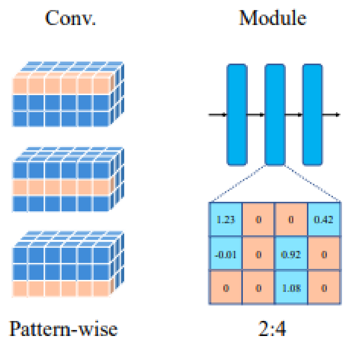

Figure 2.7: Example of unstructured pruning

In contrast, structured pruning is considered *coarse-grained* pruning [12], whereas semi-structured pruning is considered *fine-grained*.

Some techniques for semi-structured pruning include:

- **Block-Based Pruning**: This method removes blocks of weights, which may correspond to small groups of neurons or filter groups. The goal is to reduce the size of the network while maintaining efficient computation.
- **Group Pruning**: This method removes entire groups of weights or neurons, which may include filters or channels in CNNs, based on the importance of the group.
- **Hybrid Approaches**: These approaches combine unstructured and structured pruning techniques, pruning certain layers or neurons using structured methods and pruning individual weights in other layers using unstructured methods.

Semi-structured pruning offers a balance between flexibility and efficiency. While it may not lead to as much sparsity as unstructured pruning, it provides a more structured reduction in the network size, which is easier to implement in practice and can lead to better computational performance.

2.4 Pruning in Vision Transformers

The Vision Transformer (ViT) architecture consists of several key components, including Multi-Head Self-Attention (MHSA), Multi-Layer Perceptron (MLP), layer normalization, activation functions, and skip connections. The MHSA mechanism is the core element of transformers, facilitating interaction between different tokens. Given an input $X \in \mathbb{R}^{n \times d}$, it is transformed into query $Q \in \mathbb{R}^{n \times d}$, key $K \in \mathbb{R}^{n \times d}$, and value $V \in \mathbb{R}^{n \times d}$ through fully connected layers, where n is the number of patches and d is the embedding dimension. The self-attention mechanism models the relationships between the patches as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{2.10}$$

AA linear transformation is then applied to generate the output of MHSA:

$$Y = X + \text{FC}_{\text{out}}(\text{Attention}(\text{FC}_q(X), \text{FC}_k(X), \text{FC}_v(X))) \tag{2.11}$$

where layer normalization and activation functions are omitted for simplicity. For the two-layer MLP, it can be represented as:

$$Z = Y + \text{FC}_2(\text{FC}_1(Y)) \tag{2.12}$$

The widely-used fully connected layers in the transformer contribute to computational and storage burdens.

To reduce the size of the transformer architecture, the focus is placed on decreasing the number of floating-point operations (FLOPs) of MHSA and MLP. The proposed pruning strategy aims to reduce the dimensions of the linear projections by learning their corresponding importance scores. Let the features $X \in \mathbb{R}^{n \times d}$, where n is the number of features to be pruned and d is the dimension of each feature, represent the set of features. The objective is to retain the important features and remove the less informative ones from the linear projections. Let the optimal importance scores be $a^* \in \{0, 1\}^d$, where the components corresponding to important features are set to 1 and the ones corresponding to non-essential features are set to 0. The pruned features can be obtained as:

$$X^* = X \cdot \text{diag}(a^*) \tag{2.13}$$

However, since a^* consists of discrete values, it is difficult to optimize using backpropagation. Therefore, the scores are relaxed to real values $\hat{a} \in \mathbb{R}^d$, and the soft-pruned features are computed as:

$$\hat{X} = X \cdot \text{diag}(\hat{a}) \tag{2.14}$$

The relaxed importance scores $\hat{a}$ are learned end-to-end with the transformer network. To enforce sparsity in the importance scores, ℓ_1 regularization is applied:

$$\lambda \|\hat{a}\|_1 \tag{2.15}$$

where λ is a hyperparameter controlling the sparsity. After training with the sparsity penalty, the transformer will contain many importance scores close to zero. These scores are ranked, and a threshold τ is set based on the desired pruning rate. Using this threshold, the discrete importance scores a^* are obtained by setting values below τ to 0 and values above τ to 1:

$$a^* = \hat{a} \geq \tau \tag{2.16}$$

Once pruning is applied according to the importance scores a^*, the pruned transformer is fine-tuned to minimize accuracy degradation. The pruning procedure is summarized as:

$$X^* = \text{Prune}(X) \tag{2.17}$$

As depicted in Figure 2.8, pruning is applied to both the MHSA and MLP blocks. The pruning procedure for these blocks can be formulated as:

$$Q, K, V = \text{FC'}_q(\text{Prune}(X)), \text{FC'}_k(\text{Prune}(X)), \text{FC'}_v(\text{Prune}(X)) \tag{2.18}$$

$$Y = X + \text{FC'}_{\text{out}}(\text{Prune}(\text{Attention}(Q, K, V))) \tag{2.19}$$

$$Z = Y + \text{FC'}_2(\text{Prune}(\text{FC'}_1(\text{Prune}(Y)))) \tag{2.20}$$

where $\text{FC'}_q, \text{FC'}_k, \text{FC'}_v, \text{FC'}_{\text{out}}, \text{FC'}_1, \text{FC'}_2$ are the pruned linear projections corresponding to the pruned features and importance scores a^*. The proposed Vision Transformer Pruning (VTP) method offers a simple yet effective way to reduce the size of ViT models, serving as a solid baseline for future research and providing insights for the practical deployment of Vision Transformers.

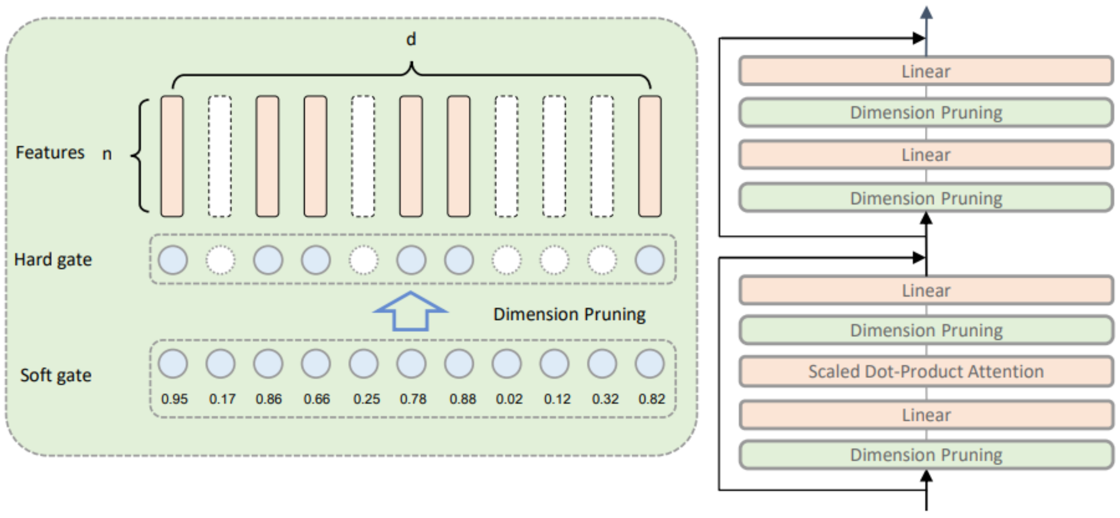

Figure 2.8: Vision Transformer Pruning

Chapter 3

Preliminaries

Vision Transformers (ViTs) have gained significant attention in the field of computer vision, achieving exceptional performance across a wide range of tasks. Despite their effectiveness, the high computational and memory demands of ViTs pose a significant challenge for their deployment in practical, resource-constrained scenarios. A key insight underlying these challenges is that not all tokens in the input sequence contribute equally to the final predictions. Consequently, reducing redundant tokens has emerged as a prominent strategy for accelerating Vision Transformers and making them more efficient.

Current approaches for token reduction typically fall into two categories: token pruning and token pooling. Token pruning focuses on removing inattentive tokens that contribute little to the model's output, while token pooling merges duplicative tokens to reduce redundancy. However, addressing only one of these redundancies in isolation may not fully optimize the computational efficiency and performance trade-offs.

To tackle this issue, we introduce a novel acceleration method called **Token Pruning and Pooling Transformers (PPT)**. PPT adaptively addresses both inattentive and duplicative redundancies at different layers of the Vision Transformer. By seamlessly integrating token pruning and pooling techniques without adding extra trainable parameters, the framework significantly reduces the computational complexity of ViTs while preserving their predictive accuracy. This unified approach provides a practical and efficient solution for enhancing the scalability and applicability of Vision Transformers in real-world applications.

3.1 Token Redundancy in ViTs

Token redundancy in Vision Transformers (ViTs) is a critical factor that impacts computational efficiency. Redundant tokens, which include those that carry negligible information or encode repetitive features, increase the computational cost and memory usage of the model without improving its performance. Addressing token redundancy is essential for deploying ViTs in resource-constrained environments such as mobile devices or edge systems. In this section, we discuss two main types of token redundancy: inattentive tokens and duplicative tokens.

3.1.1 Inattentive Tokens

Inattentive tokens are those that receive minimal attention weights during the self-attention computation in ViTs. These tokens contribute negligibly to the model's final predictions, often representing background or less informative parts of an image.

Impact on Computation. Inattentive tokens unnecessarily inflate the computational burden of ViTs, as they participate in every layer of the self-attention mechanism. Removing these tokens during inference can significantly enhance computational efficiency.

Mitigation Approaches. Several techniques exist to address inattentive tokens, such as attention-based scoring methods that assign a threshold to identify and prune these tokens. By dynamically removing inattentive tokens, models can maintain accuracy while reducing computational complexity. Our proposed framework builds upon these methods by integrating pruning techniques into different layers of ViTs.

3.1.2 Duplicative Tokens

Duplicative tokens are those that encode highly similar or repetitive information, often arising from patches with similar colors or textures within an image. These tokens lead to redundancy by encoding overlapping features.

Challenges. Identifying duplicative tokens is challenging because their representation may vary across layers. Addressing this redundancy requires mechanisms that preserve critical information while merging or pooling similar tokens.

Mitigation Approaches. Token pooling or merging strategies are commonly employed to reduce duplicative redundancy. These methods group similar tokens and represent them with a single pooled token, thereby reducing the sequence length and computational cost. Our framework introduces a novel token pooling method that adaptively tackles duplicative tokens at different layers of the ViT architecture, maintaining model performance while improving efficiency.

3.2 Token Reduction Methods

Begin with a brief introduction to token reduction methods, explaining their purpose and importance. Mention how these methods aim to optimize ViTs by reducing computational and memory demands while maintaining model performance.

3.2.1 Token Pruning

In general, the token pruning(Figure 3.1) paradigm consists two steps, token scoring and token selecting. Token scoring assigns a score to each token based on its importance for the task, and then token selecting determines which tokens to keep and which to discard.Several non-parametric scoring methods have been proposed for tokens in previous works, such as the attention scores A_{cls} between the classification token and the image tokens [13, 27]. Additionally, the Attention-based Token Scoring (ATS) method [7, 29] incorporates the value matrix V and introduces a more comprehensive token scoring metric:

$$A = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}}\right), \quad \text{Score}_i = \frac{A_{1,i+1} \times \|V_{i+1}\|}{\sum_{j=1}^{N} A_{1,j+1} \times \|V_{j+1}\|} \tag{3.1}$$

For token selection, the traditional Top-K selection policy is utilized to provide a more controllable compression ratio. This policy aims to retain the Top-K most important tokens while removing the inattentive ones, a technique that is commonly used in many studies [13, 21].

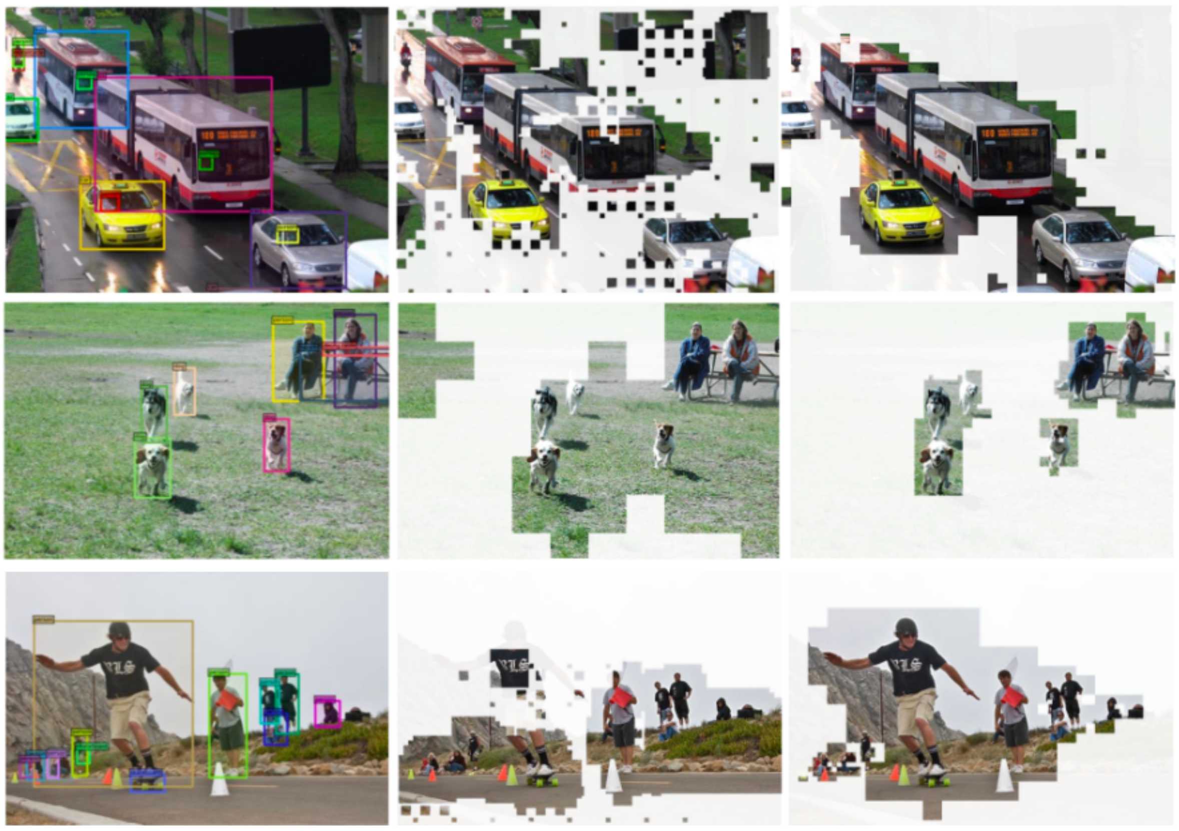

Figure 3.1: Example of token pruning

3.2.2 Token Pooling

The token pooling method (Figure 3.2, 3.3) aims to combine similar image tokens, effectively minimizing redundant information within the model. The Bipartite Soft Matching (BSM) algorithm [3] has recently demonstrated exceptional performance in the token merging process. The BSM approach divides the tokens into two nearly equal sets. It then establishes an edge between each token in one set and the most similar token in the other set based on cosine similarity. From these, the top-K most similar edges are chosen, and the tokens connected by these edges are merged by averaging their features.

Furthermore, a variable s is introduced to track the size of the tokens and minimize the loss of information. This vector s, which is a row vector, indicates the number of tokens and plays a key role in the token merging operation. Additionally, it acts as a weight reflecting the token's significance during the calculation of the attention matrix:

$$A = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d}} + \log s\right) \tag{3.2}$$

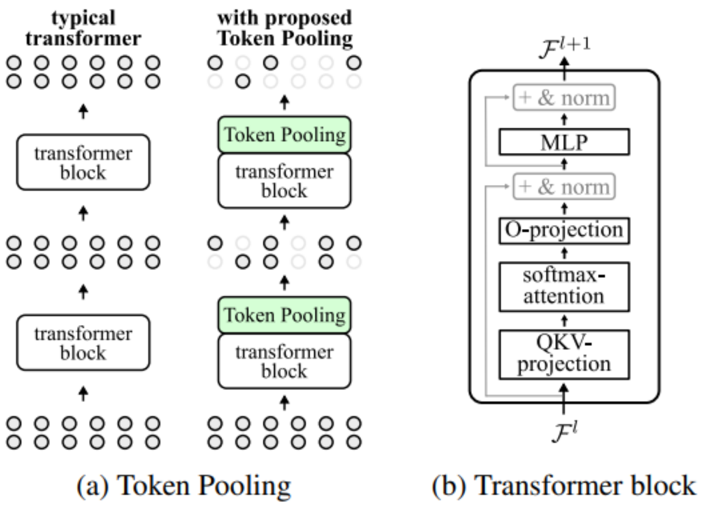

Figure 3.2: a) we insert token pooling after every transformer block . b) a transformer block.

3.3 Combining Pruning and Pooling

I begin by analyzing the importance scores of tokens at various layers. Our analysis uncovers an interesting observation: the variance in the significance scores assigned to image tokens increases as the model's layers deepen, as shown in Figure 3 (a more detailed analysis is presented in Section A of the appendix). This indicates that as the model depth increases, the importance of image tokens becomes more clearly defined. Therefore, token pruning techniques are more effective in deeper layers, where certain tokens exhibit significantly lower importance scores, making them more likely to be pruned. On the other hand, prematurely pruning tokens in shallow layers may result in irreversible information loss, which can adversely affect model performance.

In contrast, token pooling techniques are better suited for application in the shallow layers. Through our analysis, we observe that shallow layers contain many tokens with relatively high similarity, which makes it less problematic to merge dissimilar tokens using the Top-K strategy during token pooling. This approach is unlikely to have a significant negative impact on model performance. Additionally, by maintaining a variable s that reflects the size of each token, we ensure that any information loss due to the merging of highly similar tokens is minimal. However, as the number of similar tokens decreases in deeper layers, the application of token pooling techniques becomes less feasible.

3.3.1 Adaptive Token Pruning & Pooling Strategy

Based on these insights, we propose an adaptive token compression method that dynamically determines the optimal strategies for addressing the two types of redundancy at various layers.Based on the aforementioned findings, we propose an adaptive token compression strategy that dynamically determines the most suitable approach for removing redundant tokens. This method can adaptively choose between token pruning and token pooling based on the current significance scores of the tokens.

To address this, I introduce an adaptive strategy that considers both the layer and the sample. The approach is defined as follows:

$$S_{\text{opi}} = \text{var}(\text{Score}_i), \quad \text{Policy}_i = \begin{cases} \text{Token Pruning}, & \text{if } S_{\text{opi}} > \tau \\ \text{Token Pooling}, & \text{otherwise} \end{cases} \tag{3.3}$$

where Score_i is calculated using Formula (3.1), and τ is a hyperparameter that controls the decision threshold, determining which strategy is preferred. The effect of τ on model performance is discussed in detail [26]. Our method does not require any additional learnable parameters, which allows it to be seamlessly integrated with pre-trained Vision Transformers (ViTs) and demonstrates competitive performance, as shown by our experimental results.

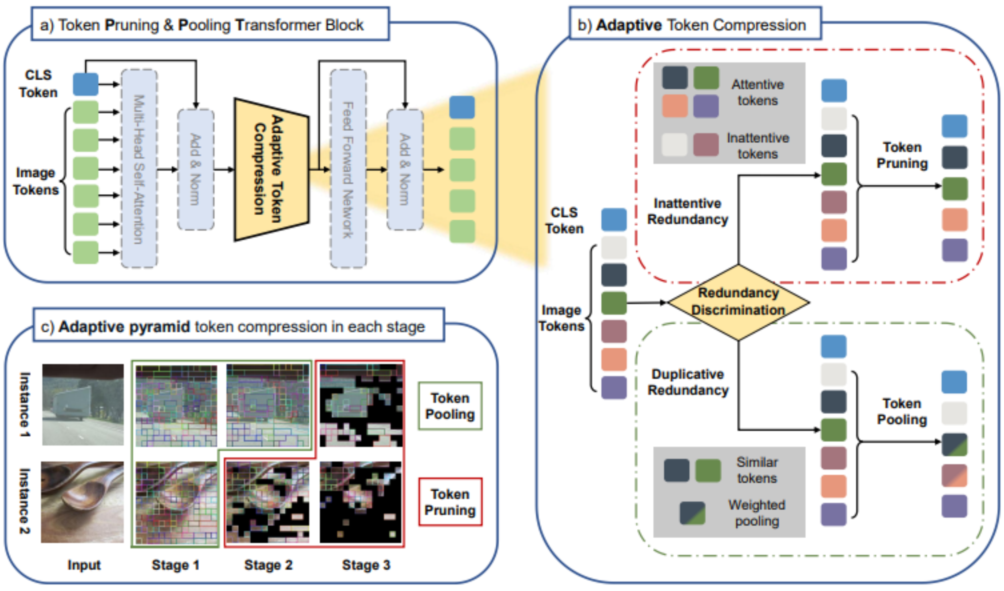

Figure 3.3: : Overview of the PPT approach.

(Figure 3.3 a) The Adaptive Token Compression module is simple and can be easily inserted inside the standard transformer block without additional trainable parameter. (Figure 3.3 b) Our module can adaptively select either token pruning or token pooling policy to tackle corresponding redundancy based on the current token distribution, which is intuitively reflected across various instances and layers in (Figure 3.3 c). Figure 3.3 c With PPT, similar patches within the same color bounding box are pooled into a single token, while the masked inattentive patches are pruned, resulting in promising trade-offs between the accuracy and FLOPs.

Chapter 4

Problem Statement

Vision Transformers (ViTs) have emerged as powerful tools in computer vision, achieving state-of-the-art results across numerous tasks. However, their widespread deployment is hindered by high computational and memory requirements, which are exacerbated in resource-constrained environments like mobile devices and edge systems.

The complexity of ViTs arises from their reliance on Multi-Head Self-Attention (MHSA), Multi-Layer Perceptrons (MLPs), and fully connected layers. These components introduce significant overhead in terms of floating-point operations (FLOPs), memory usage, and latency. While pruning has been widely explored for CNNs to address similar issues, the unique structure of ViTs poses distinct challenges. For instance, naive pruning of attention heads or embeddings often results in performance degradation, irregular sparsity, and limited compatibility with hardware accelerators.

Existing pruning techniques for ViTs are limited in their ability to balance computational efficiency and accuracy. This work addresses this gap by proposing a optimization of PPT method that prunes tokens in two distinctive way. The method aims to significantly reduce the computational demands of ViTs while preserving their performance and ensuring compatibility with modern hardware.

By focusing on these challenges, the PPT not only provides a robust baseline for future research but also offers practical insights for deploying ViTs in real-world scenarios.

Chapter 5

Proposed Methods

5.1 Metrics

Various metrics play a crucial role throughout the processes of training, fine-tuning, and pruning in machine learning and deep learning models. These metrics serve as benchmarks to evaluate the performance, efficiency, and optimization of the model. They help assess how well a model is learning during training, the extent to which fine-tuning improves its accuracy, and the effectiveness of pruning strategies in reducing computational complexity without significantly compromising performance. By analyzing these metrics, we can make informed decisions to improve model design and ensure that it meets the desired objectives for specific tasks.Here are some metrics:

Top-1 Accuracy

The Top-1 Accuracy measures the percentage of correctly classified samples and can be mathematically expressed as:

$$\text{Top-1 Accuracy (\%)} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}} \times 100$$

Where:

- **Number of Correct Predictions**: The count of predictions where the model's highest-confidence label matches the true label.
- **Total Number of Samples**: The total size of the evaluation dataset.

Parameters (Params)

The number of trainable parameters in the model, Params, is defined as:

$$\text{Params (M)} = \frac{\sum_{i=1}^{N} p_i}{10^6}$$

Where:

- p_i: The number of trainable parameters in the i^{th} layer.

- N: Total number of layers in the model.

FLOPs (Floating Point Operations Per Second)

The number of floating-point operations required for a single forward pass is:

$$\text{FLOPs (G)} = \frac{\sum_{i=1}^{N} \text{FLOPs}_i}{10^9}$$

Where:

- FLOPs_i: The floating-point operations required for the i^{th} layer.
- N: Total number of layers in the model.

Throughput

Throughput measures the number of images processed per second and can be written as:

$$\text{Throughput (images/s)} = \frac{\text{Total Number of Images Processed}}{\text{Total Inference Time (s)}}$$

Where:

- **Total Number of Images Processed**: Total size of the evaluation dataset.
- **Total Inference Time (s)**: Total time taken by the model to process the dataset.

Combined Representation for a Pruned Model

For a pruned model M_{pruned}, you might compare these metrics with a baseline model M_{baseline} as follows:

$$\text{Accuracy Improvement (\%)} = \text{Top-1}_{\text{pruned}} - \text{Top-1}_{\text{baseline}}$$

$$\text{Parameter Reduction (\%)} = \frac{\text{Params}_{\text{baseline}} - \text{Params}_{\text{pruned}}}{\text{Params}_{\text{baseline}}} \times 100$$

$$\text{FLOPs Reduction (\%)} = \frac{\text{FLOPs}_{\text{baseline}} - \text{FLOPs}_{\text{pruned}}}{\text{FLOPs}_{\text{baseline}}} \times 100$$

$$\text{Throughput Improvement (\%)} = \frac{\text{Throughput}_{\text{pruned}} - \text{Throughput}_{\text{baseline}}}{\text{Throughput}_{\text{baseline}}} \times 100$$

5.2 Development Environment

The source code for the proposed pruning system is implemented in a Python 3.9.10 environment, running on an Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz and an NVIDIA Titan Xp GPU. The development setup uses Ubuntu 22.04.3 LTS (GNU/Linux 6.5.0-28-generic x8.All the deep learning specific approaches are deployed using the Pytorch machine learning framework.

5.3 Model selection and Fine-tuning

5.3.1 Chosen Model: DEiT-small patch16_224

The DEiT (Data-efficient Image Transformer) small patch16_224 model was selected for this study. DEiT-small is a lightweight Vision Transformer specifically designed for efficient training and inference.This model is actually a more efficiently trained Vision Transformer (ViT).The Vision Transformer (ViT) is a transformer encoder model (BERT-like) pre-trained and fine-tuned on a large collection of images in a supervised fashion, namely ImageNet-1k, at a resolution of 224x224 pixels.Images are presented to the model as a sequence of fixed-size patches (resolution 16x16), which are linearly embedded. One also adds a token to the beginning of a sequence to use it for classification tasks. One also adds absolute position embeddings before feeding the sequence to the layers of the Transformer encoder.By pre-training the model, it learns an inner representation of images that can then be used to extract features useful for downstream tasks.

5.3.2 Rationale for Selection

The decision to use DEiT-small was motivated by the following factors:

- **Computational Efficiency**: DEiT-small is well-suited for GPU-constrained scenarios due to its relatively small architecture, making it manageable within limited hardware resources.
- **Performance Potential**: The model balances computational efficiency with high classification performance, providing a strong foundation for exploring optimization techniques such as token pruning.
- **Scalability**: Its design supports fine-tuning across different datasets and tasks, making it adaptable for CIFAR-100.

5.3.3 Dataset

CIFAR-100 is a benchmark dataset widely used in computer vision research. It contains:

- **Number of Classes**: 100 classes, each representing a unique category.
- **Image Size**: Images are of size 32x32 pixels.
- **Dataset Composition**: The dataset includes 50,000 training images and 10,000 test images, with a balanced distribution across all classes.

This dataset is ideal for testing fine-grained image classification models, given its diversity and large number of categories.

5.3.4 Fine-Tuning Process

To adapt DEiT-small to CIFAR-100:

- The model was fine-tuned for **40 epochs**.

- Pre-trained weights (from ImageNet) were used as the initialization point, providing a strong starting point for transfer learning.
- Adjustments were made to handle CIFAR-100's smaller image size:
  - Images were resized to match the model's expected input resolution of 224x224 pixels.
  - Classifier head was modified to predict 100 classes instead of the original 1,000.
- Data augmentation techniques like random cropping and horizontal flipping, were applied to improve generalization.
- The learning rate with 0.001, ADAM optimization,batch-size 64 were carefully tuned to optimize performance for CIFAR-100.

5.4 Significant Scores

Below, I have implemented various score functions to compare the results of the baseline with those of other score functions. The explanations and descriptions of each formula are provided.

5.4.1 1. Baseline Score Function

Formula:

$$\text{Significance Score}(b,t) = \frac{\text{attn}(b,h,0,t)\cdot \text{Unorm}(b,t)}{\sum_{t'} \text{attn}(b,h,0,t')\cdot \text{Unorm}(b,t')}$$

Explanation: The baseline score function computes the significance of each token based on the attention received from the CLS token (token 0). The attention score is normalized over all tokens in the sequence to compute the final significance score.

- $\text{attn}(b,h,0,t)$: This is the attention score between the CLS token (0) and token t at head h for batch b.
- $\text{Unorm}(b,t)$: This is the normalization term that normalizes the attention score.
- The summation normalizes the attention score by the sum of all attention-weighted values for each token.

Purpose: This function helps to identify tokens that are most significant to the CLS token. It is commonly used for tasks that focus on the importance of tokens in relation to the overall sequence representation.

5.4.2 Significance Score Computation Based on Attention Entropy

In this section, we present the mathematical formulation for computing the *significance score* of tokens in a transformer-based model. This score is derived using the attention weights, specifically through a combination of attention entropy and value norms.

Let $\text{attn\_probs}(b,h,t_1,t_2)$ represent the attention probability from token t_1 to token t_2 in head h for batch b. The significance score is computed using both the entropy of the attention distribution and the token importance (measured through value norms). The following steps outline the computation:

Attention Entropy Calculation

We first compute the entropy of the attention distribution for each token t_1, which captures how uniformly the attention is distributed across the other tokens t_2. The entropy for a given token t_1 and attention head h is given by:

$$\text{entropy}(b, h, t_1) = -\sum_{t_2=1}^{T} \text{attn\_probs}(b, h, t_1, t_2) \log \left(\text{attn\_probs}(b, h, t_1, t_2) + \epsilon \right)$$

Here, ϵ is a small constant added to avoid taking the logarithm of zero. The term $\text{attn\_probs}(b, h, t_1, t_2)$ represents the attention probability from token t_1 to token t_2 in the attention matrix for the given head h and batch b.

Averaging Entropy Across Attention Heads

Next, we average the entropy across all attention heads h for each token t_1. This gives us a measure of the overall uncertainty in the attention distribution for token t_1:

$$\text{average\_entropy}(b, t_1) = \frac{1}{H} \sum_{h=1}^{H} \text{entropy}(b, h, t_1)$$

Where H is the number of attention heads, and $\text{entropy}(b, h, t_1)$ is the entropy computed for each head.

Significance Score

The significance score for each token t is then computed by multiplying the average entropy by the value norm of the token $v_{\text{norm}}(b, t)$. The value norm $v_{\text{norm}}(b, t)$ is calculated as the L2-norm of the token's value vector. The significance score for each token t in batch b is given by:

$$\text{score}(b, t_1) = \left(\frac{1}{H} \sum_{h=1}^{H} \left(-\sum_{t_2=1}^{T} \text{attn\_probs}(b, h, t_1, t_2) \log \left(\text{attn\_probs}(b, h, t_1, t_2) + \epsilon \right) \right) \right) \times v_{\text{norm}}(b, t_1)$$

Normalization of Significance Scores

Finally, the significance scores are normalized so that they sum to 1 across the tokens (excluding the CLS token). This ensures that the scores are proportionally distributed and can be interpreted as relative importance scores. The normalized significance score for each token t_1 is given by:

$$\text{significance\_score}(b, t_1) = \frac{\text{significance\_score}(b, t_1)}{\sum_{t_1=1}^{T-1} \text{significance\_score}(b, t_1)}$$

Where the denominator sums over all tokens t_1 (excluding the CLS token) in batch b. This ensures that the significance scores are normalized across tokens in each batch.

Full Formula for Significance Score

Combining the above steps, the final formula for the significance score of each token t_1 in batch b is as follows:

$$\text{score}(b,t_1) = \frac{\left(\frac{1}{H}\sum_{h=1}^{H}\left(-\sum_{t_2=1}^{T}\text{attn\_probs}(b,h,t_1,t_2)\log\left(\text{attn\_probs}(b,h,t_1,t_2)+\epsilon\right)\right)\right)\cdot v_{\text{norm}}(b,t_1)}{\sum_{t_1=1}^{T-1}\left(\frac{1}{H}\sum_{h=1}^{H}\left(-\sum_{t_2=1}^{T}\text{attn\_probs}(b,h,t_1,t_2)\log\left(\text{attn\_probs}(b,h,t_1,t_2)+\epsilon\right)\right)\right)\cdot v_{\text{norm}}(b,t_1)}$$

This formula computes the significance score for each token by combining the attention entropy and value norms, and then normalizing the scores across all tokens.

5.4.3 Average Attention Score

Given an attention matrix attn $\in \mathbb{R}^{B\times H\times T\times T}$, where:

- B is the batch size,
- H is the number of attention heads,
- T is the sequence length,
- $\text{attn}(b,h,t_1,t_2)$ represents the attention weight from token t_1 to token t_2 in head h for batch b.

The average attention across all heads for each token pair is computed as:

$$\text{avg\_attention}(b,t_1,t_2) = \frac{1}{H}\sum_{h=1}^{H}\text{attn}(b,h,t_1,t_2) \quad \forall b,t_1,t_2$$

This results in an average attention matrix avg_attention $\in \mathbb{R}^{B\times T\times T}$.

Attention From CLS Token to Other Tokens

The function isolates the attention from the CLS token (t_0) to each of the other tokens $t \in [1, T-1]$:

$$\text{significance\_score}(b,t) = \text{avg\_attention}(b,t,0) \quad \forall b, t\in[1,T-1]$$

Excluding the CLS Token

The CLS token itself is excluded from the significance score:

$$\text{significance\_score} = \text{significance\_score}[:,1:] \quad \text{so that} \quad \text{significance\_score} \in \mathbb{R}^{B\times(T-1)}$$

Normalization of Significance Scores

The significance scores are normalized across the remaining tokens (excluding the CLS token):

$$\text{significance\_score}(b,t) = \frac{\text{significance\_score}(b,t)}{\sum_{t=1}^{T-1}\text{significance\_score}(b,t)} \quad \forall b$$

Final Formula

The final significance score for each token $t \in [1, T-1]$ in batch b is:

$$\text{significance\_score}(b,t) = \frac{\frac{1}{H}\sum_{h=1}^{H} \text{attn}(b,h,0,t)}{\sum_{t=1}^{T-1}\left(\frac{1}{H}\sum_{h=1}^{H} \text{attn}(b,h,0,t)\right)} \quad \forall b$$

Chapter 6

Results

6.1 Fine-Tuning Results and Observations

6.1.1 Accuracy Achieved

After fine-tuning, the DEiT-small model achieved an accuracy of **83.56%** (Figure 6.1) on the CIFAR-100 test set. This result demonstrates the effectiveness of Vision Transformers in handling fine-grained classification tasks even when applied to relatively small datasets.

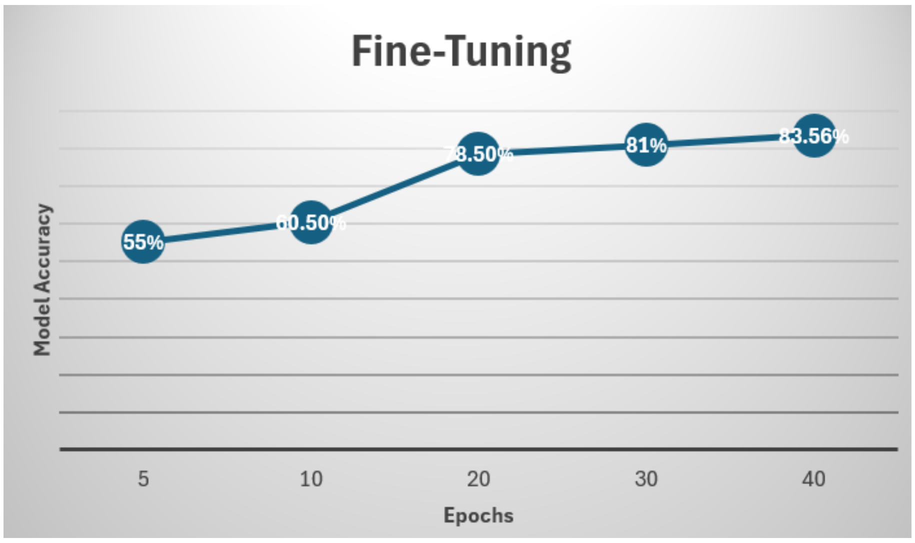

Figure 6.1: Model accuracy trends during the fine-tuning

6.1.2 Insights and Observations

- The achieved accuracy highlights the adaptability of DEiT-small for datasets with smaller images and diverse categories.
- This performance provides a baseline for subsequent experiments involving token pruning and other optimization techniques.
- Despite GPU constraints, the model demonstrated a robust ability to learn meaningful representations, paving the way for further improvements in computational efficiency.

6.2 Accuracy of model after implementing different score function

he following table presents the accuracy results for the different score functions evaluated in this study:

Score Function	**Accuracy**
Average Score Function	62.59%
Diversity Score Function	63.40%
Baseline Score Function	65.30%

From the results, we observe that the Baseline Score Function yields the highest accuracy at 65.30, followed by the Diversity Score Function with an accuracy of 63.40. The Average Score Function shows the lowest accuracy at 62.59. These results suggest that the Baseline Score Function provides the best performance in terms of accuracy, while the Diversity Score Function offers a slight improvement over the Average Score Function.

Chapter 7

Conclusion

While the Baseline Score Function performs the best overall, the slight improvement observed with the Diversity Score Function over the Average Score Function indicates that incorporating attention diversity can contribute to a marginal enhancement in accuracy. However, the pruning of the DEiT-small model resulted in a significant drop in performance, suggesting that further refinement of the pruning strategy or tuning of the model architecture may be required to retain accuracy after pruning. Additionally, exploring the interaction between pruning and attention-based score functions might yield further insights into optimizing model performance post-pruning.

Acknowledgements

Bibliography

[1] Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. Cloze-driven pretraining of self-attention networks. *arXiv preprint arXiv:1903.07785*, 2019.

[2] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4599–4603, 2023.

[4] Kyunghyun Cho. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5840–5848, 2017.

[6] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[7] Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive token sampling for efficient vision transformers. In *European Conference on Computer Vision*, pages 396–414. Springer, 2022.

[8] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[9] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[10] 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*, 2015.

[11] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[12] Zhengang Li, Geng Yuan, Wei Niu, Pu Zhao, Yanyu Li, Yuxuan Cai, Xuan Shen, Zheng Zhan, Zhenglun Kong, Qing Jin, et al. Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14255–14266, 2021.

[13] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. *arXiv preprint arXiv:2202.07800*, 2022.

[14] Liyang Liu, Shilong Zhang, Zhanghui Kuang, Aojun Zhou, Jing-Hao Xue, Xinjiang Wang, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pages 7021–7032. PMLR, 2021.

[15] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[16] Xiaolong Ma, Wei Niu, Tianyun Zhang, Sijia Liu, Sheng Lin, Hongjia Li, Wujie Wen, Xiang Chen, Jian Tang, Kaisheng Ma, et al. An image enhancing pattern-based sparsity for real-time inference on mobile devices. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*, pages 629–645. Springer, 2020.

[17] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36: 21702–21720, 2023.

[18] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

[19] Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning filter in filter. *Advances in Neural Information Processing Systems*, 33:17629–17640, 2020.

[20] K O'Shea. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[21] Roshan M Rao, Jason Liu, Robert Verkuil, Joshua Meier, John Canny, Pieter Abbeel, Tom Sercu, and Alexander Rives. Msa transformer. In *International Conference on Machine Learning*, pages 8844–8856. PMLR, 2021.

[22] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[23] I Sutskever. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[24] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[25] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[26] Xinjian Wu, Fanhu Zeng, Xiudong Wang, and Xinghao Chen. Ppt: Token pruning and pooling for efficient vision transformers. *arXiv preprint arXiv:2310.01812*, 2023.

[27] Yifan Xu, Zhijie Zhang, Mengdan Zhang, Kekai Sheng, Ke Li, Weiming Dong, Liqing Zhang, Changsheng Xu, and Xing Sun. Evo-vit: Slow-fast token evolution for dynamic vision transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2964–2972, 2022.

[28] Huanrui Yang, Hongxu Yin, Maying Shen, Pavlo Molchanov, Hai Li, and Jan Kautz. Global vision transformer pruning with hessian-aware saliency. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 18547–18557, 2023.

[29] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10809–10818, 2022.

[30] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.