SliceViT

Andriy Sergiyenko
Stanford University
andriys@stanford.edu

Abstract

Vision transformers have rapidly advanced the state of the art in computer vision, yet their deployment is often limited by significant computational and memory demands. To address these challenges, this paper introduces SliceViT, a novel post-training pruning technique adapted from the successful SliceGPT method used in language models. SliceViT effectively reduces the embedding dimension and overall model size by slicing rows and columns from each weight matrix. Through comprehensive experimentation, we demonstrate that SliceViT can eliminate up to 30% of the parameters in Google ViT Base and Large models while preserving 91% and 95% of the original top-5 accuracy, respectively, in image classification tasks. Notably, our pruned models enhance deployment on resource-constrained devices such as smartphones and tablets, improving inference speed by 6.6% and 18% on an iPad Pro with an M2 processor, without the need for additional code optimizations. The implementation of SliceViT is made publicly available at: https://github.com/sndre/pytorch-image-models.

1. Introduction

In recent years, the computational expense associated with training and deploying vision models has increased significantly, prompting the research community to explore efficient model compression techniques. These techniques have proved useful in deploying advanced AI models on resource-constrained devices such as smartphones and tablets. Among the prevalent model compression methods—distillation, tensor decomposition, pruning, and quantization—this work focuses on pruning. Specifically, it adapts the "SliceGPT" method [3], originally developed for large language models (LLMs), to vision transformers. The SliceGPT approach is particularly compelling because it eliminates entire rows and columns of a model's weight matrix without necessitating recovery fine-tuning, which traditionally poses a challenge in model compression.

Furthermore, SliceGPT enhances computational efficiency by reducing the embedding dimensions of feature representations, thereby lowering the computational overhead required during inference. The method's ability to maintain model performance while significantly cutting down computational demands makes it a promising candidate for real-time AI applications on mobile devices.

This paper contributes to the field by:

1. Adapting the SliceGPT method, initially designed for text-based models, to the domain of vision transformers.
2. Modifying the original method to enable selective pruning of transformer blocks and to apply incremental PCA computations.
3. Conducting extensive experiments to evaluate the accuracy-performance trade-offs introduced by this method. These experiments are specifically tailored to measure the method's performance on Apple's M2 processors, setting a benchmark for its feasibility in real-time applications.

We call our method SliceViT. The following sections will expand on the methodology, experimental setup, results, and implications of applying the SliceViT to vision transformers, emphasizing its potential to impact mobile-based AI applications.

1.1. Problem Statement

Reduce the inference time and computational load of Vision Transformer models without significantly compromising accuracy for their use in real-time applications on resource-constrained devices such as smartphones and tablets.

Inputs and Outputs

- **Input:** The method takes as input a pre-trained vision transformer (ViT) model designed for image recognition tasks, specifically trained or fine-tuned on the ImageNet 1k dataset.
- **Output:** The output is a pruned vision transformer model that maintains acceptable levels of accuracy while reducing the computational requirements and memory footprint.

Baseline Method

The baseline method involves measuring both top-1 and top-5 accuracy, as well as runtime latency, of a non-pruned vision transformer model. This model serves as a benchmark for evaluating the effectiveness of the SliceViT pruning method in reducing model size and computational complexity while attempting to preserve the accuracy levels necessary for practical applications.

2. Background and Related Work

There are four main categories of model compression techniques: distillation, tensor decomposition, pruning, and quantization ([11], [7], [21], and [8]). In this work, we focus on pruning, specifically applying the SliceGPT method [3] to Vision Transformers [6].

2.1. Vision Transformer Networks

Vision Transformers [6] are a class of neural networks that adapt the transformer architecture [18], used for natural language processing, to handle computer vision tasks. The ViT architecture treats image patches as the equivalent of tokens in NLP, processing these patches through a series of transformer encoder layers. Each encoder is composed of multi-head self-attention mechanisms followed by a feed-forward neural network (FFN), which are normalized by a LayerNorm operation.

Embeddings: Let D be the dimension of each image patch embedding, and N be the total number of patches derived from splitting the image. The Vision Transformer takes a flattened list of patch embeddings as input, where each patch embedding is generated by linearly projecting a fixed-size portion of the input image. Additionally, a positional embedding is added to the patch embeddings to retain positional information.

LayerNorm: After the embedding stage, the signal matrix undergoes a LayerNorm [4] operation, which normalizes the data by subtracting the mean and dividing by the standard deviation of each layer's inputs. The LayerNorm operation is given by:

$$\text{LayerNorm}(X) = \text{RMSNorm}(XM)\text{diag}\,(\alpha)\,\sqrt{D} + 1_N\beta^T$$

where RMSNorm [20] applies $x \leftarrow x/\|x\|$ to each row of X. The parameters α and β are learned separately for each LayerNorm instance in the network. The constant matrix $M = I - \frac{1}{D}11^T$ is a $D \times D$ matrix which is a matrix-form expression for subtracting the mean of X from each row of X.

Attention Blocks: The core of the Vision Transformer is its attention mechanism, where the input matrix X is projected into queries Q, keys K, and values V, each obtained by multiplying X with corresponding learned weight matrices. The self-attention mechanism computes the relevance of different patches to one another and aggregates this information to form the output matrix.

FFN Blocks: Each attention block is followed by a feed-forward network, consisting of two linear transformations with a GeLU [10] activation in between.

Forward Pass: In the ViT, the image passes sequentially through the transformer blocks, where each block updates the representation of the image patches. This sequence of operations transforms the initial patch-based representation into a more abstract representation that captures both individual and relational features of the patches, which is then used for classification or other vision tasks.

2.2. SliceGPT

The SliceGPT method represents a promising advancement in the compression of transformer-based models, particularly focusing on maintaining computational invariance while reducing model complexity through a targeted pruning approach [14].

Computational Invariance in Transformers

An *invariant* function is one for which a transformation to the input does not result in a change to the output. Let Q denote an orthogonal matrix, for which $Q^TQ = QQ^T = I$. Note that multiplying a vector x by Q does not change the norm of the vector, since:

$$\|Qx\| = \sqrt{x^TQ^TQx} = \sqrt{x^Tx} = \|x\|.$$

This invariance property can be used to prove the following expression:

$$\text{RMSNorm}(XQ)Q^T = \text{RMSNorm}(X).$$

Since each Attention or FFN transformer block has linear operations on both the input and output, we can integrate the orthogonal transformations Q into the linear layers of these blocks. This foundational insight enables the application of pruning based on computational invariance, as described in the next section.

Pruning Method

The key steps of SliceGPT method are outlined below:

1. **Conversion from LayerNorm to RMSNorm**: This transformation is achieved by absorbing the scaling matrix diag(α) into the linear layer that follows the LayerNorm, and the mean-subtraction matrix M into the linear layer that precedes it.
2. **Computation of Orthogonal Matrices**: The orthogonal matrices Q are computed using Principal Component Analysis (PCA) [9] on the covariance matrix X^TX of the signals. These signals are typically the inputs and outputs of the Attention and FFN blocks within the transformer and are projected onto their principal components.

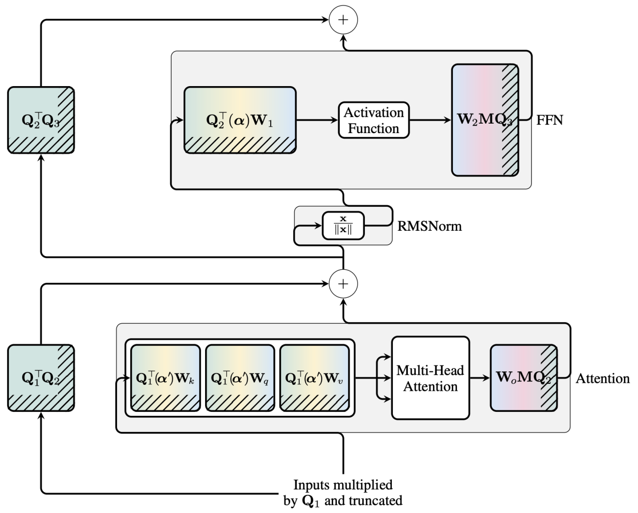
$\mathbf{Q}_2^\top\mathbf{Q}_3$
$\mathbf{Q}_2^\top(\boldsymbol{\alpha})\mathbf{W}_1$
Activation Function
$\mathbf{W}_2\mathbf{M}\mathbf{Q}_3$
FFN
$\frac{\mathbf{x}}{\|\mathbf{x}\|}$
RMSNorm
$\mathbf{Q}_1^\top\mathbf{Q}_2$
$\mathbf{Q}_1^\top(\boldsymbol{\alpha}')\mathbf{W}_k$
$\mathbf{Q}_1^\top(\boldsymbol{\alpha}')\mathbf{W}_q$
$\mathbf{Q}_1^\top(\boldsymbol{\alpha}')\mathbf{W}_v$
Multi-Head Attention
$\mathbf{W}_o\mathbf{M}\mathbf{Q}_2$
Attention
Inputs multiplied by $\mathbf{Q}_1$ and truncated

Figure 1. A pruned transformer block using SliceGPT method based on computational invariance (adapted from [3]).

3. **Applying Orthogonal Transformations**: The input linear layers of the Attention and FFN blocks are pre-multiplied by Q^T, and the output linear layers are post-multiplied by Q. In the residual connections a new linear layer $Q_l^T Q_{l+1}$ is introduced.
4. **Pruning via Row and Column Deletion**: The final step is the actual pruning, where rows of the input and columns of the output linear layer weights are deleted. Additionally, both rows and columns of the matrix $Q_l^T Q_{l+1}$ in the residual connections are also removed.

Fig. 1 illustrates this entire process, demonstrating how the transformations and deletions lead to a compressed yet effective model architecture.

2.3. Hardware-specific Optimizations

In addition to model pruning, another avenue for improving inference speed lies in hardware-specific optimizations. Such optimizations take advantage of the unique architecture of target hardware to maximize computational efficiency. Apple Inc. has explored this domain extensively in [2] and [1], providing insights into how models can leverage Apple Neural Engine features effectively. Through our initial investigations into these optimizations, we have found that while the scope for novel contributions might be limited due to the comprehensive work already done by Apple, these techniques can nonetheless be combined with our SliceViT methodology. The integration of model pruning via SliceViT with hardware-specific optimizations is out of scope for this work however.

3. Method

3.1. SliceViT: Transformation and Optimization of Patch Embedding Layers

In the standard Vision Transformer architecture, the initial embedding of image patches is typically performed using a convolutional layer. This layer, denoted as a 2D convolution (Conv2D), maps input image patches into a latent space conducive to subsequent attention mechanism. Unlike linear layers, convolutional layers cannot directly undergo orthogonal transformations due to their structured kernel operations. Therefore, a transformation approach is necessary to integrate these layers into the SliceViT framework, which involves orthogonal transformations and pruning.

Conversion of Conv2D to Linear Layer

The conversion process starts by reshaping the weights of the pre-trained Conv2D layer to form a linear layer (Linear). Given a convolutional layer with parameters $\mathbf{W}_{\text{conv}}$ and biases $\mathbf{b}_{\text{conv}}$, the transformation to a linear layer is mathematically represented as:

1. **Weight Reshaping**:

$$\mathbf{W}_{\text{linear}} = \text{reshape}(\mathbf{W}_{\text{conv}}, (C_{\text{out}}, C_{\text{in}} \times K_h \times K_w))$$

Where C_{in}, C_{out} are the number of input and output channels, and K_h, K_w are the heights and widths of the convolutional kernel.

2. **Bias Transfer**:

$$\mathbf{b}_{\text{linear}} = \mathbf{b}_{\text{conv}}$$

This process ensures that the newly formed linear layer retains the functional characteristics of the original convolutional layer but in a form amenable to further transformations.

Orthogonal Transformation and Pruning

Upon converting the convolutional layer to a linear format, the next steps involve applying orthogonal transformations and slicing to reduce the dimensionality while preserving essential features:

1. **Rotation**:

$$\mathbf{W}_{\text{rotated}} = \mathbf{Q}^T \cdot \mathbf{W}_{\text{linear}}$$

$$\mathbf{b}_{\text{rotated}} = \mathbf{Q}^T \cdot \mathbf{b}_{\text{linear}} \text{ (if biases exist)}$$

Here, $\mathbf{Q}$ is an orthogonal matrix derived from PCA method, ensuring that the transformation maintains the representational capacity of the embeddings.

2. **Slicing**:

$$\mathbf{W}_{\text{sliced}} = \mathbf{W}_{\text{rotated}}[: d', :]$$

$$\mathbf{b}_{\text{sliced}} = \mathbf{b}_{\text{rotated}}[: d'] \text{ (if biases exist)}$$

Where d' is the new reduced dimensionality of the output features, effectively pruning the less significant components.

These steps ensure that the patch embeddings are prepared for subsequent integration into the Vision Transformer where remaining blocks are rotated and sliced using SliceGPT method described in Sec. 2.2.

3.2. Incremental PCA Computation

The original SliceGPT method used between 128 and 2048 samples from the training set of the calibration dataset to calculate orthogonal matrices using PCA. However, there is strong evidence validated in Sec. 4.4 suggesting that using the entire training dataset may result in a more robust PCA calculation. PCA aims to find directions (principal components) that maximize the variance in the dataset. With more data, the calculated directions are more likely to represent the true underlying patterns in the entire dataset.

However, the entire ImageNet 1k training dataset does not fit into available RAM, which necessitates converting the original SliceGPT PCA calculation to an incremental PCA to manage memory usage effectively.

The PCA calculation algorithm used in SliceGPT is detailed in Tab. 1.

Step	Operation
1	Load samples from dataset to inputs array
2	Use inputs array to calculate orthogonal matrix Q
3	Rotate & prune embeddings
4	For each transformer block:
	4.1. Rotate & prune attention input layer
	4.2. Use inputs to compute attention outputs
	4.3. Use attention outputs to calculate orthogonal matrix Q
	4.4. Rotate & prune attention output layer
	4.5. Rotate & prune MLP input layer
	4.6. Use attention outputs to compute MLP outputs
	4.7. Rotate & prune MLP output layer
5	Rotate & prune LM head layer

Table 1. SliceGPT PCA calculation algorithm

To address the memory constraints, the PCA calculation in SliceViT was adapted to an incremental approach, as described in Tab. 2.

Step	Operation
1	Calculate orthogonal matrix Q using incremental PCA with model inputs
2	Rotate & prune embeddings
3	Rotate & prune patch embeddings
4	For each transformer block:
	4.1. Rotate & prune attention input layer
	4.3. Calculate orthogonal matrix Q using incremental PCA with attention outputs
	4.4. Rotate & prune attention output layer
	4.5. Rotate & prune MLP input layer
	4.6. Calculate orthogonal matrix Q using incremental PCA with MLP outputs
	4.7. Rotate & prune MLP output layer
5	Rotate & prune LM head layer

Table 2. SliceViT Incremental PCA calculation algorithm

3.3. Implementation Details

The implementation of this project is based on the PyTorch Image Models provided by the repository available at https://github.com/huggingface/pytorch-image-models. We integrated the SliceGPT code from Microsoft's Transformer Compression GitHub repository at https://github.com/microsoft/TransformerCompression/tree/main/src/slicegpt into it to enable efficient pruning and validation operations.

The Vision Transformer model structure is crucial for understanding how to apply the pruning algorithm. The model structure is as follows:

```
VisionTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(...)
  )
  (blocks): Sequential(
    (0-11): 12 x Block(
      (norm1): LayerNorm(...)
      (attn): Attention(
        (qkv): Linear(...)
        (proj): Linear(...)
      )
      (norm2): LayerNorm(...)
      (mlp): Mlp(
        (fc1): Linear(...)
        (act): GELU(...)
        (fc2): Linear(...)
      )
    )
  )
  (norm): LayerNorm(...)
  (head): Linear(...)
)
```

The pruning algorithm is applied as follows:

1. **Conversion of Patch Embeddings Convolutional Layer to Linear Layer:** The `patch_embed.proj`

layer, originally a Conv2d layer, was converted into a linear layer. This was necessary to allow for the application of orthogonal transformations. The conversion process is desribed in Sec. 3.1.

2. **Fusing LayerNorm Operations:** The centering and scaling operations performed by LayerNorm in each block were fused into adjacent linear layers to enable the network pruning:
   - The centering operation (mean subtraction) was fused into the projection linear layers (`attn.proj` and `mlp.fc2`) of the attention and MLP blocks.
   - The scaling operation (multiplication by learned weights) was integrated into the input layers (`attn.qkv` and `mlp.fc1`), as well as the final classifier head (`head`).
3. **Orthogonal Transformations and Pruning:**
   - **Projection Layers:** The projection linear layers `patch_embed.proj`, `attn.proj`, and `mlp.fc2` were rotated by pre-multiplying by Q^T and pruned.
   - **Input Layers:** Embedding layers `cls_token` and `pos_embed` (not shown in the model structure) and input linear layers `attn.qkv`, `mlp.fc1`, and `head`, were rotated by post-multiplying by Q and pruned.
4. **Model and Layer Adapters:** SliceGPT provides plug-in model for pruning transformers. We implemented ViT model and layer adapters in `CompressedBlock`, `VitModelAdapter`, and `VitLayerAdapter` within `vit_adapter.py`. We modified `layernorm_fusion.py` to center and scale ViT specific embedding layers and `rotate.py` to rotate and prune patch embedding layer. We also implemented the `optimized_rotate_and_slice_sequential` function in `rotate.py`, which uses the Incremental PCA calculation algorithm described in Sec. 3.2.

To validate the correctness of our fusion, rotation, and pruning implementation, we performed pruning with 0% sparsity on the Google ViT base model, which yielded accuracies matching the baseline established in Sec. 4.3.

We used the CoreML Tools [12] Python framework to convert PyTorch [16] models to CoreML format and measured runtime performance using the performance evaluation tool built into Xcode.

4. Experimental Validation

4.1. Computing Environment

Our experiments were conducted using the Google Colab platform. The specific configurations utilized were **NVIDIA Tesla T4 GPU** with 15GB of GPU RAM for general less resource-intensive tasks and **NVIDIA A100 GPU** with 40GB of GPU RAM for more demanding computations and larger dataset processing needs.

Despite the sufficient hardware support, a notable limitation of this computing environment was the performance impact at the beginning of each session—a noticeable lag was observed due to the time required to download necessary dataset images from an attached Google Drive. This was particularly evident when large volumes of data were used.

4.2. Dataset

For the experiments conducted in this study, the ImageNet 1k dataset [5] was utilized both for calculating PCA and for evaluating model performance. Specifically, the PCA was computed using the ImageNet 1k training dataset, which consists of over 1.2 million images spanning 1000 different classes. This extensive dataset provides a comprehensive and diverse range of visual features necessary for effective PCA computation.

Model evaluations were performed using the ImageNet 1k validation dataset, which includes 50,000 images. This separate dataset ensures that the pruning method's performance is assessed on unseen data, providing a reliable measure of its generalization capabilities.

4.3. Baseline

For establishing a baseline performance, two types of pre-trained Vision Transformer models provided by Google were used. These models are available via PyTorch Image Models as:

- Google ViT base: `vit_base_patch16_224.orig_in21k_ft_in1k`
- Google ViT large: `vit_large_patch32_384.orig_in21k_ft_in1k`

Accuracy Benchmarking

The baseline performance of these models was measured in terms of top-1 and top-5 accuracy on the ImageNet 1k validation dataset. The results are summarized in the table below:

Model	Acc@1 (%)	Acc@5 (%)
Google ViT base, dense	81.718 (18.282)	96.134 (3.866)
Google ViT large, dense	81.546 (18.454)	96.082 (3.918)

Table 3. Baseline accuracy results on ImageNet 1k validation dataset

Performance Evaluation

Further, both models were converted to CoreML format to assess their performance on an iPad equipped with an M2 processor. The evaluation focused on inference time, with results detailed in the following table:

Model	Inference Time (ms)
Google ViT base, dense	128.06
Google ViT large, dense	752.32

Table 4. Baseline performance results on iPad with M2 processor

These benchmarks establish a solid baseline for further experiments applying SliceViT pruning method to these models.

4.4. Impact of Data Volume on PCA Robustness and Accuracy of Pruned Vision Transformers

To validate the assumption that using a larger dataset might yield more robust PCA calculations as alluded to in Sec. 3.2, we used the ImageNet 1k validation dataset for both PCA calculation and accuracy evaluation. This approach was taken to gauge the potential upper bound of accuracy when PCA is "overfit" to a specific dataset. By calculating PCA with different subsets of this validation dataset and subsequently evaluating accuracy on the same dataset, we aimed to understand how variations in the PCA input size affect model performance. The experiment indicated that the more batches used in PCA calculation, the higher the accuracy achieved, as summarized in Tab. 5.

Batch Configuration	Acc@1 (%)	Acc@5 (%)
1 batch, Dense	81.718 (18.282)	96.134 (3.866)
1 batch, 25%	71.310 (28.690)	90.860 (9.140)
10 batches, 25%	71.736 (28.264)	90.568 (9.432)
50 batches, 25%	73.758 (26.242)	91.766 (8.234)
75 batches, 25%	73.790 (26.210)	91.512 (8.488)
All 196 batches, 25%	77.078 (22.922)	94.396 (5.604)

Table 5. Accuracy results across different batch configurations

4.5. Using ImageNet 1k Training Dataset for PCA Calculation

Despite the promising evidence presented in Sec. 4.4 suggesting that using more data for PCA computation should result in higher accuracy of the pruned model, our experiments with the entire ImageNet 1k training dataset, which consists of 934 batches, did not yield better results compared to using significantly fewer batches. Specifically, using the entire dataset yielded the same level of accuracy as using as few as 29 batches. The accuracy results are detailed in the table below:

Please refer to Appendix A.1 where we explored potential reasons for this in greater detail.

Batch Configuration	Acc@1 (%)	Acc@5 (%)
29 batches	73.258 (26.742)	91.506 (8.494)
58 batches	73.272 (26.728)	91.534 (8.466)
116 batches	73.268 (26.732)	91.612 (8.388)
All 934 batches	73.266 (26.734)	91.524 (8.476)

Table 6. Accuracy results across different training batch configurations when used with Random Sampler and 25% Sparsity Level

4.6. Effects of Different Sparsity Levels on Pruned Model Accuracy

The impact of varying sparsity levels on the accuracy of pruned Vision Transformer models was systematically evaluated. The results for both the Google ViT base and large models with different sparsity levels are shown in Tab. 7.

Model and Sparsity	Acc@1 (%)	Acc@5 (%)
Google ViT Base, Dense	81.718 (18.282)	96.134 (3.866)
Google ViT Base, 20%	75.960 (24.040)	93.178 (6.822)
Google ViT Base, 25%	73.258 (26.742)	91.506 (8.494)
Google ViT Base, 30%	67.872 (32.128)	87.684 (12.316)
Google ViT Large, Dense	81.546 (18.454)	96.082 (3.918)
Google ViT Large, 20%	78.228 (21.772)	94.546 (5.454)
Google ViT Large, 25%	76.110 (23.890)	93.592 (6.408)
Google ViT Large, 30%	72.882 (27.118)	91.438 (8.562)

Table 7. Accuracy results for Google ViT models at various sparsity levels

To better visualize these findings, we plotted Fig. 2 that helps to illustrate the trade-off between model compactness and accuracy retention.

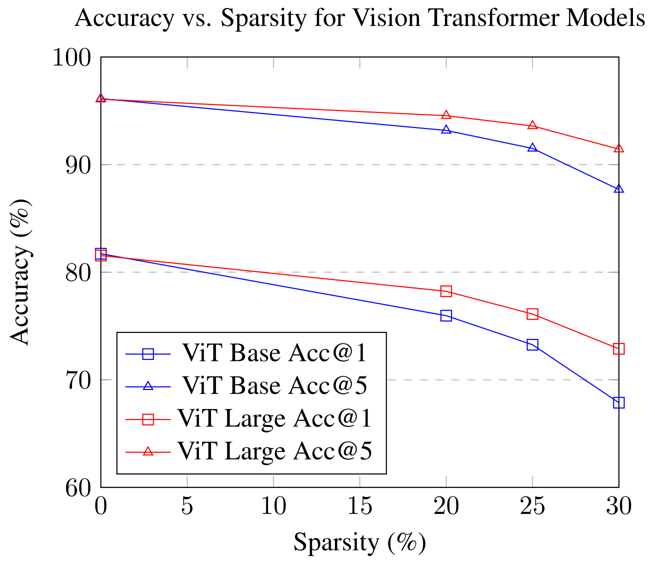

Figure 2. The effect of sparsity on Acc@1 and Acc@5 for Google ViT Base and Large models.

The Google ViT Large model consistently demonstrated an ability to retain higher accuracy compared to the ViT Base model under equivalent sparsity conditions. This can be observed in the lesser decline of accuracy metrics as sparsity increases. Moreover, the rate at which accuracy decreases as sparsity levels increase is smaller for the ViT Large model compared to the ViT Base model. This suggests that the ViT Large model's architecture is better at coping with the loss of parameters.

These findings underscore the importance of model architecture in the context of pruning. The inherent advantage of having more weights allowed the ViT Large model to handle reductions in model complexity more gracefully.

4.7. Effects of Different Sparsity Levels on Pruned Model Runtime Performance

An analysis was conducted to evaluate the impact of varying sparsity levels on the runtime performance of pruned Google ViT Base and Large models. Measurements focused on inference time reductions as a result of increased sparsity.

Tab. 8 shows inference times and parameter counts for models at different sparsity levels:

Model and Sparsity	Parameters	Inference (ms)
Google ViT Base, 0%	86,529,256	128.06
Google ViT Base, 20%	78,136,872	122.63
Google ViT Base, 25%	73,768,552	120.11
Google ViT Base, 30%	68,374,392	119.62
Google ViT Large 0%	306,532,328	752.32
Google ViT Large, 20%	277,494,088	651.96
Google ViT Large, 25%	259,755,496	604.79
Google ViT Large, 30%	239,334,200	616.83

Table 8. Runtime performance and parameter counts for Google ViT models at various sparsity levels

To better visualize the relationship between sparsity and runtime performance, we plotted Fig. 3.

The analysis shows diminishing returns in runtime performance gains when sparsity exceeds 25%. A possible explanation is the effect of additional shortcut matrix multiplications in the network, which become more significant as the model size decreases. These operations might offset the computational savings from reducing the number of parameters as seen in Fig. 3, especially at higher sparsity levels.

4.8. Effects of Different Sparsity Levels on Pruned Model Efficiency Score

The efficiency of pruned models is quantified using an efficiency score that integrates reductions in accuracy and inference speed. This score helps to understand the trade-offs between maintaining model accuracy and enhancing

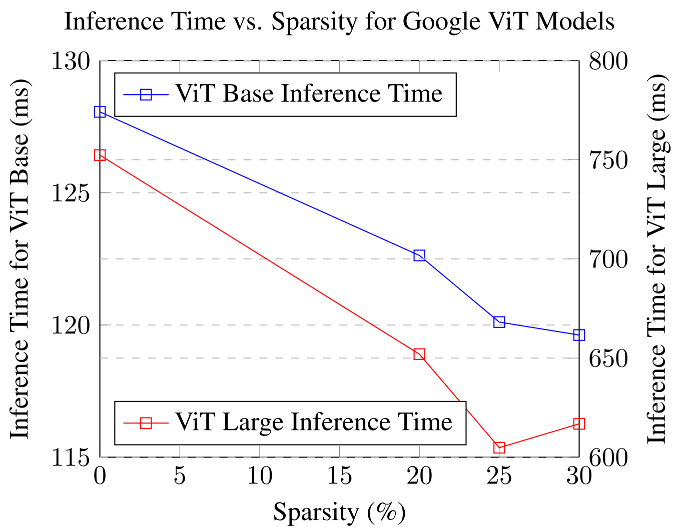

Figure 3. The effect of sparsity on inference time for Google ViT Base and Large models.

operational efficiency through faster inference times. The efficiency score is computed as follows:

$$\text{Efficiency Score} = \alpha \times (1 + \beta)$$

where α indicates how much accuracy pruned model retains compared to the base model and β — reduction in inference time. A higher score indicates a better balance of accuracy retention and inference speed improvement.

Fig. 4 illustrates the variation of efficiency scores with sparsity levels, highlighting the differences between the Google ViT Base and Large models.

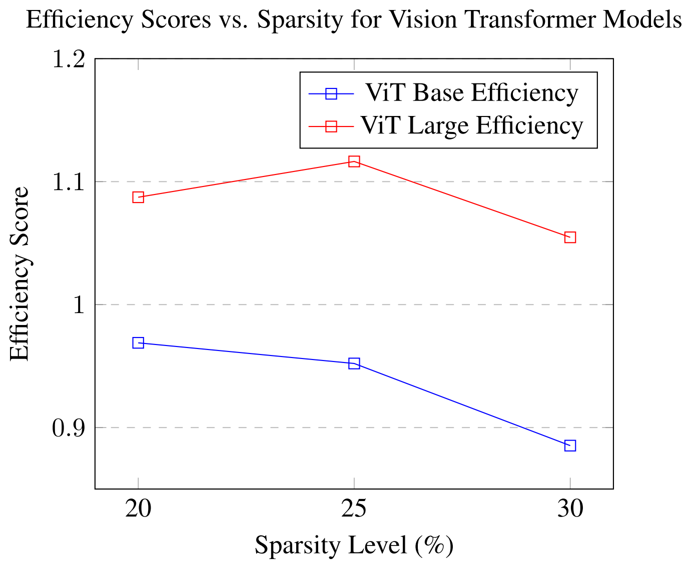

Figure 4. The effect of sparsity on efficiency scores for Google ViT Base and Large models.

The Google ViT Large model consistently demonstrates superior efficiency scores across various sparsity levels when compared to the Google ViT Base model. This pattern indicates that larger models may possess a greater capacity to retain operational efficiency—even as significant portions of their parameters are pruned.

4.9. Effects of Selective Transformer Blocks Pruning on Model Accuracy

This section examines the impact of selective pruning on the accuracy of Google ViT Base model, based on the hypothesis that some transformer blocks may be more robust to pruning than others. We tested this by progressively pruning both the initial (head) and final (tail) blocks of the transformer to observe how it affects the model accuracy. Head pruning hypothesizes that early transformer blocks are more robust to pruning, and tail pruning assumes that the last transformer blocks, responsible for understanding higher-level concepts, may be more susceptible to accuracy loss. The results are provided in Tab. 9 and Fig. 5.

Pruning Configuration	Top-1 Acc (%)	Top-5 Acc (%)
Baseline (Dense Model)	81.718	96.134
Head Pruning		
Prune 0-6 blocks	80.418	95.502
Prune 0-7 blocks	79.398	95.000
Prune 0-8 blocks	77.654	94.224
Prune 0-9 blocks	75.768	93.260
Prune 0-10 blocks	73.776	91.962
Tail Pruning		
Prune 11 block	81.178	95.818
Prune 10-11 blocks	79.280	94.778
Prune 9-11 blocks	77.224	93.746
Prune 8-11 blocks	75.830	93.056
Prune 7-11 blocks	74.958	92.650
Prune 6-11 blocks	74.320	92.194

Table 9. Accuracy results for selective pruning of Google ViT Base model

This experiment yielded a few interesting observations:

- Pruning the first nine blocks (blocks 0-8) yields similar accuracy (around 77% top-1) as pruning the last three blocks (blocks 9-11) while allowing for the removal of approximately 5 times more parameters.
- A more pronounced drop in accuracy is observed when pruning beyond the 8th block, indicating that the 9th block may be where the model begins to learn high-level features and thus becomes less robust to pruning.
- An interesting observation is that pruning the first 8 blocks affects the top-5 accuracy by only 1% as opposed to 5% loss when pruning all 12 blocks, suggesting that such pruning comes with minimal loss to top-5 accuracy, effectively being "almost free."

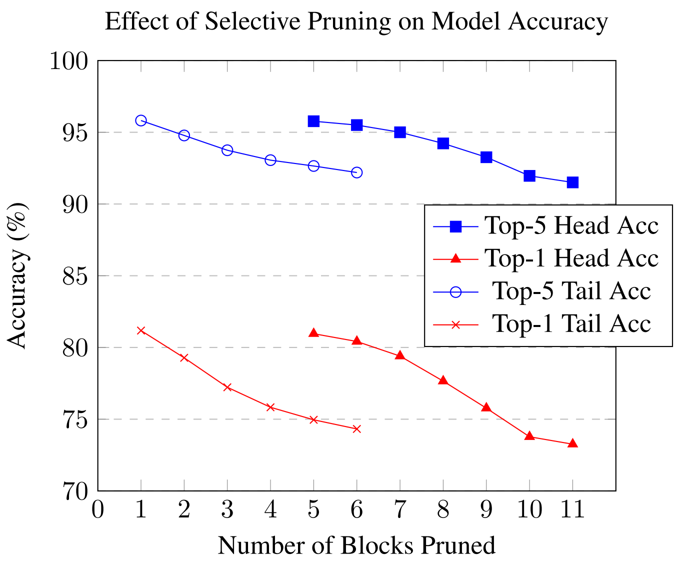

Figure 5. Plot showing the effects of head and tail pruning on Google ViT Base model accuracy. As a starting point, note how pruning the last 1 block yields the same top-1 and top-5 accuracies as pruning the first 5 blocks.

5. Conclusions and Future Work

This study has demonstrated that the SliceViT method of pruning yields notably higher efficiency for larger vision models, leveraging their extensive parameter sets to maintain performance while achieving significant reductions in model complexity. Our experiments with selective pruning further underscore the nuanced potential of this approach; particularly, pruning early blocks can substantially reduce parameter counts with minimal impact on accuracy, showcasing the method's capability to fine-tune performance trade-offs. For smaller models, where accuracy is not the paramount concern, SliceViT provides an effective means to improve inference speed, albeit at the expense of greater accuracy loss. This trade-off highlights the importance of adjusting optimization strategies to specific application needs.

Future investigations might explore pruning vision-language models such as CLIP [17], ALIGN [13], and Florence [19] using a combined approach of SliceViT and SliceGPT. We believe it holds substantial promise. CLIP models are designed to map both images and text into a shared embedding space. This unique characteristic might lend itself well to a pruning strategy aimed at maintaining the alignment between these embeddings. Since both modalities contribute to the same final task (contrastive learning), the impact of pruning might be similar across both pathways, allowing for consistent reductions in parameters without disproportionately affecting one side of the model.

Acknowledgements

We thank Saleh Ashkboos and Maximilian L. Croci for validating the applicability of the SliceGPT method to vision transformers and for their encouragement to explore this path. We also thank Nikil Ravi for suggesting the experiment with selective pruning.

References

[1] Apple. Deploying attention-based vision transformers to apple neural engine. *Machine Learning Research*. 3

[2] Apple. Deploying transformers on the apple neural engine. *Machine Learning Research*. 3

[3] S. Ashkboos, M. L. Croci, M. G. do Nascimento, T. Hoefler, and J. Hensman. Slicegpt: Compress large language models by deleting rows and columns, 2024. 1, 2, 3

[4] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. 2

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 5

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 2

[7] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference, 2021. 2

[8] M. Gupta and P. Agrawal. Compression of deep learning models for text: A survey, 2021. 2

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009. 2

[10] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus), 2023. 2

[11] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021. 2

[12] A. Inc. Coreml tools. `https://github.com/apple/coremltools`, 2020. Version 4.0. 5

[13] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. V. Le, Y. Sung, Z. Li, and T. Duerig. Scaling up visual and vision-language representation learning with noisy text supervision, 2021. 8

[14] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. 2

[15] S. L. Lohr. *Sampling: Design and Analysis*. Brooks/Cole, Cengage Learning, 2 edition, 2010. 10

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 5

[17] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. 8

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023. 2

[19] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, C. Liu, M. Liu, Z. Liu, Y. Lu, Y. Shi, L. Wang, J. Wang, B. Xiao, Z. Xiao, J. Yang, M. Zeng, L. Zhou, and P. Zhang. Florence: A new foundation model for computer vision, 2021. 8

[20] B. Zhang and R. Sennrich. Root mean square layer normalization, 2019. 2

[21] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. A survey on model compression for large language models, 2023. 2

A Appendix

A.1. Impact of Training Data Volume on PCA Robustness and Accuracy of Pruned Vision Transformers

As mentioned in Sec. 4.5, using the entire training dataset, which consists of 934 batches, for PCA computation yielded the same level of accuracy as using as few as 29 batches.

Moreover, the computation of Q on the entire ImageNet 1k training dataset took approximately 60 hours on Nvidia A100 compared to about an hour on Nvidia T4 when using 29 batches. This significant increase in computation cost did not correlate with a proportionate improvement in model accuracy.

The invariant accuracy observed across varying batch sizes during PCA calculations may be attributed to the intrinsic properties and limitations of the PCA method when applied to complex, high-dimensional data like images. The initial gains in accuracy with small increases in batch size suggest that a critical amount of variance is captured quickly, which is essential for pruning efficiency. However, as more data is added, the marginal increase in the captured variance diminishes, possibly because the additional data are highly correlated with what has already been captured or because the added variance does not significantly influence the principal components relevant for model performance.

This phenomenon can also be influenced by overfitting during PCA computation. When PCA is calculated on the entire training dataset, it might be overfitting to nuances in the training data that do not generalize well to unseen data. This overfitting could explain why the application of PCA on the entire training set does not lead to better accuracy compared to using a smaller, yet statistically significant, subset of the training data.

To investigate whether overfitting during PCA computation impacts the efficiency of model pruning, an additional experiment was conducted using a stratified sampling approach [15]. This method aimed to ensure that each batch used for PCA calculation more evenly represents the entire dataset, potentially avoiding the overfitting to specific characteristics that might be prevalent in randomly selected batches.

The stratified sampling technique was employed to select batches for PCA computation, ensuring a diverse and representative selection of the dataset's variability. The experiment was conducted with three different batch sizes, and the results are summarized in the table below:

The results from the stratified sampling approach reveal only marginal improvements in accuracy with increased batch sizes, similar to the findings when using traditional random sampling. This consistency across different sampling methods and batch sizes suggests a fundamental limitation in the application of PCA for pruning Vision Transformers. The marginal gains indicate that while PCA is effective at capturing major variations, its ability to leverage incremental variance from larger or more diverse datasets is limited.

Batch Configuration	Acc@1 (%)	Acc@5 (%)
29 batches	73.146 (26.854)	91.512 (8.488)
58 batches	73.256 (26.744)	91.566 (8.434)
116 batches	73.352 (26.648)	91.534 (8.466)

Table 10. Accuracy results across different training batch configurations when used with Stratified Sampler and 25% Sparsity Level

This observation points to a potential ceiling effect where the principal components derived from a subset of the data are already capturing most of the critical information necessary for the model. Further increase in data volume does not contribute significantly to discovering new or more effective principal components that could enhance the model's performance post-pruning.