
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
CHENNAI – 600036

# Exploration of Optimization Strategies for Vision Transformers

*A Dual Degree Project Report*

*Submitted by*

**ARJUN MENON VADAKKEVEEDU**

*For the award of the degree*

*Of*

**MASTER OF TECHNOLOGY**

June 2023

# THESIS CERTIFICATE

This is to undertake that the Dual Degree Project Report titled **EXPLORATION OF OPTIMIZATION STRATEGIES FOR VISION TRANSFORMERS**, submitted by me to the Indian Institute of Technology Madras, for the award of **Master of Technology**, is a bona fide record of the research work done by me under the supervision of **Dr. Nitin Chandrachoodan, Dr. Ankit Rai and Dr. Pradeep Ramachandran**. The contents of this Dual Degree Project Report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Chennai 600036** **Arjun Menon Vadakkeveedu**

**Date: June 2023**

**Dr. Nitin Chandrachoodan**
Research advisor
Professor
Department of Electrical Engineering
IIT Madras

**Dr. Ankit Rai**
Research advisor
AI Specialist (Research & Development)
KLA AI-Advanced Computing Lab

**Dr. Pradeep Ramachandran**
Research advisor
Director & Head of Research
KLA AI-Advanced Computing Lab

# ACKNOWLEDGEMENTS

My association with Prof. Nitin Chandrachoodan began in my third year, when I took his elective course on Computer Organization. Since then, I have got opportunites to work with him on two separate projects (including this dual degree project), take his course on DSP Architectures and work as a Teaching Assistant under him in my final year. These opportunities allowed me to play with FPGAs, GPUs and explore the various facets of AI acceleration. It was tremendous fun to work with him on these projects and I look forward to more opportunities of working with him in the future. I personally thank Prof. Nitin for his active guidance and mentorship, and for helping me gain expertise and insights into these problem domains.

This work was jointly done with KLA's Advanced Computing Labs in its AI Computing Group at IIT Madras Research Park. I was actively mentored by Dr. Pradeep Ramachandran and Dr. Ankit Rai from KLA over the course of this project. The discussions I had with Prof. Nitin, Pradeep and Ankit during our regular meeting sessions were quite helpful in laying out a strategy for this project, identifying alternate approaches to the problems I faced and developing clarity in my ideas whenever I found the path ahead to be cluttered. I thank Pradeep and Ankit for their valuable mentorship, and for taking out time from their schedules to guide me.

# ABSTRACT

**KEYWORDS** Vision Transformers; Graphics Processing Units; Knowledge Distillation; Sparse Neural Networks

Vision Transformers (ViT) are artificial neural networks based on the Transformer architecture of Natural Language Processing (NLP) fame. The success of Transformer models such as BERT and GPT in NLP tasks has seeped into Image Processing, with Vision Transformer models attaining state-of-the-art accuracy on classification, object detection, and segmentation tasks. However, compared to Convolutional Neural Network (CNN) models that are typically used in Image Processing tasks, Vision Transformers have a higher model complexity- owing to their higher number of trainable parameters and floating-point computations (FLOPs) per forward pass. In this project, we analyze the impact of various model optimization strategies on the ViT architecture, for their efficient deployment on modern parallel processors such as GPUs.

# CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION AND BACKGROUND

The transformer architecture [1] has demonstrated its capability on a variety of Deep Learning tasks, pitching itself as a better alternative to convolution based architectures such as Recurrent Neural Networks (RNNs) and Long Short-term Memory Networks (LSTMs). Transformers are perhaps most well-known for their impact on Natural Language Processing (NLP) tasks, through Large Language Models (LLMs) such as GPT-3 [2], GPT-4 [3], BERT [4] and PaLM [5].

A. Dosovitskiy, et. al. [6] proposed using transformers for computer vision tasks. Vision Transformer (ViT) networks employ the encoder sub-network in a transformer model, similar to the BERT architecture. Patches of pixels in the input image are treated as tokens by the transformer, analogous to how words are perceived by language transformer models. ViTs beat Convolutional Neural Networks (CNNs) on image classification, object detection and image segmentation tasks. As per papers-with-code [7], vision transformer-based models dominate the ImageNet classification leaderboard. The success of vision transformers has led to other works exploring transformer models for multi-modal data inputs, as the architecture allows projecting each mode (audio, text, image or video) into tokens and then evaluating the correlations between tokens across modes.

At the core of the transformer architecture is the self-attention operation. This operation computes a correlation score between all pairs of input tokens. This allows the operation to evaluate how each token is related to every other token in the input. In the context of image processing, this allows self-attention to evaluate the significance of features present in a token patch with respect to all other patches the input image is broken into- enabling attention to extract features present locally and globally in a single shot. In contrast, operations such as convolution and transposed convolution, the workhorses of CNNs, only compute "correlation scores" with pixels in their neighborhood, and are thus limited in their capability to extract only those features that appear locally.

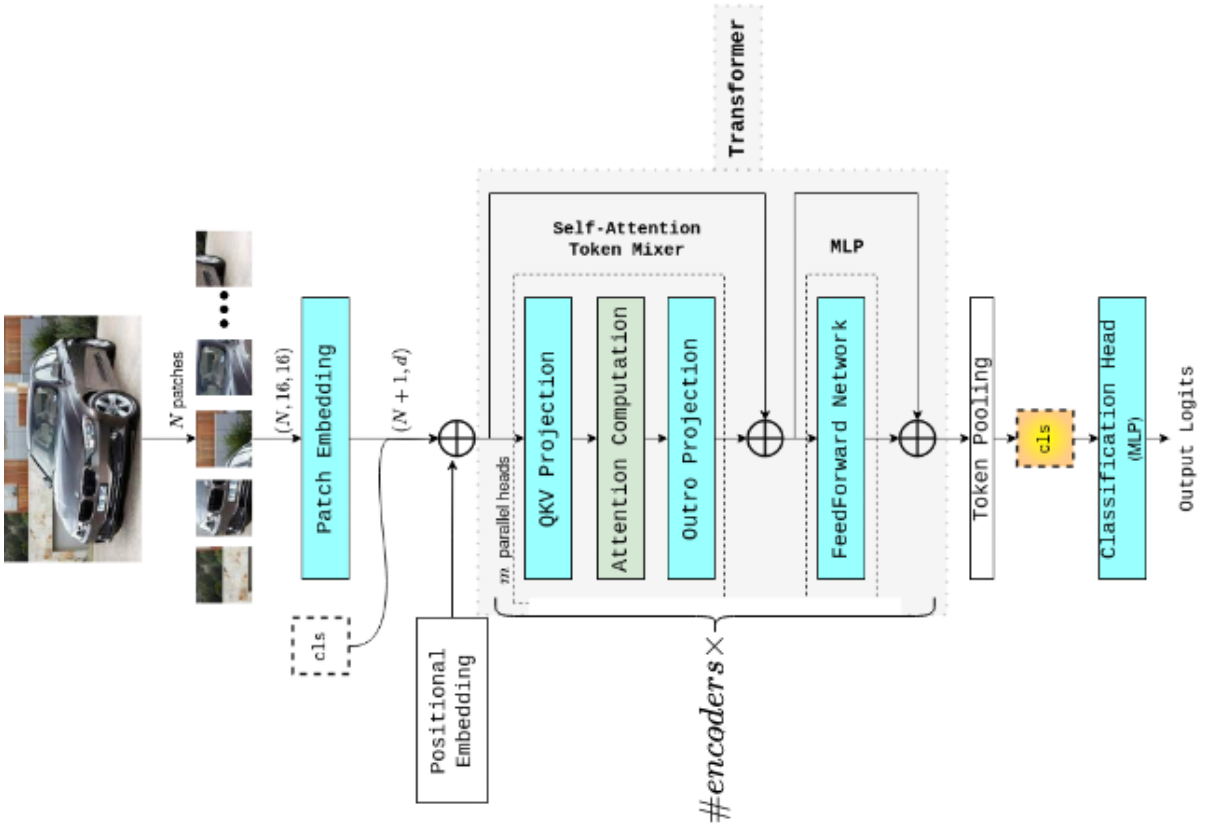



Figure 1.1: **The Architecture of a Vision Transformer**

### 1.1 VISION TRANSFORMER ARCHITECTURE

The architecture of a Vision Transformer is summarized in figure 1.1. The input image is broken into $N$ patches- each patch is of a fixed size, say $16 \times 16$. Each patch is projected into a latent vector space by the `Patch Embedding` linear projection layer. These patches then go through repeating blocks of Multi-Headed Self-Attention (MHSA) and Multi-Layer Perceptrons (MLPs). The attention and MLP blocks comprise the encoder portion of the network. For classification tasks, the patches at the end of the encoder are sent to a classification head through a token pooling layer.

Unlike convolution, self-attention is position-invariant. In order to provide information regarding the original positions of patches in the image, a positional embedding vector is added to the patches before sending them to the encoder layers. The positional embedding vector can be tuned by hand, using sine and cosine functions for example, or can be learned values that are set through gradient descent during the training step.

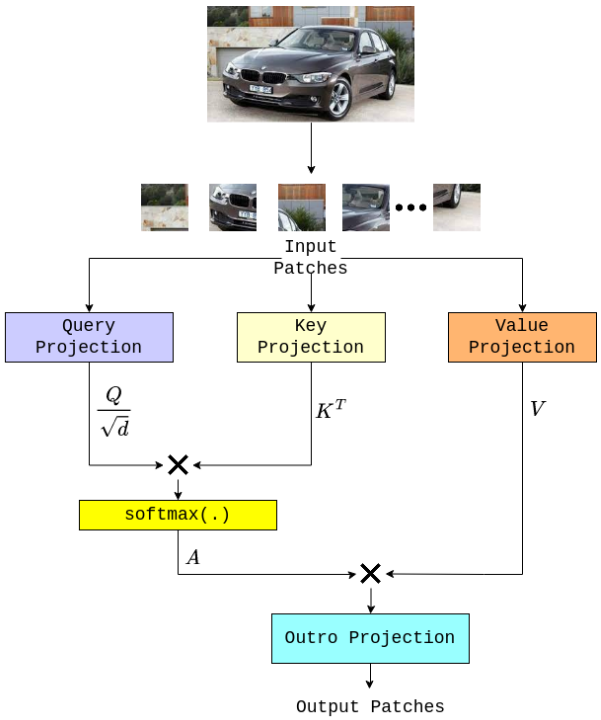



Figure 1.2: **Self-Attention Operation**

The input to the classification head is a single token that is supposed to represent all the tokens at the end of the encoder. A dummy token (`cls`) is added to the set of patches at the beginning of the encoder and is made to interact with all the tokens from the image over the course of the layers of the encoder. Thus, information regarding the features present in each patch of the image would eventually get accumulated into the `cls` token. The token pooling function at the end of the encoder picks the `cls` token and sends it across to the classification head.

### 1.1.1 Self-Attention Mechanism

The encoder block uses Multi-Headed Self-Attention (MHSA) layers in order to extract features in a content-driven manner. An MHSA layer uses multiple parallel instances of self-attention that individually extract features from the image by generating attention maps.

The operation performed by each attention head can be divided into three stages. The input patches to the attention layer are passed through a linear projection layer (`QKV-Projection`) to generate the query ($Q$), key ($K$) and value ($V$) matrices (figure 1.2). Next, the attention map ($A$) is generated by taking the dot-product between the query and key matrices, and computing the

softmax of the result.

$$A = softmax(\frac{QK^T}{\sqrt{d}})$$

*where $d$ is the channel dimension of the query and key matrices*

The attention map estimates correlations between patches in the query matrix with patches in the key matrix. The softmax operation is used to convert the correlation scores to a pseudo-probability distribution. That is, all the elements in the $A$ matrix lie in the range $[0, 1]$ and the sum of elements along any row is $1$. This attention map is used to compute a weighted sum of the tokens in the value matrix, and the result is passed through a projection layer (`Outro-Projection`) to generate the output of the attention layer. The `Outro-Projection` layer accumulates the attention-value products ($A \times V$) of all the attention heads present in the layer.

The attention map behaves like a weight matrix that gets multiplied with the input. The matrix preceding the outro-projection layer is $A \times V = A \times W^V \times X$ where $W^V$ is the value projection matrix, and $X$ is the input matrix $\implies$ $A \times W^V$ behaves effectively as the weights that the input is multiplied by. However, the values in the $A$ matrix are not learned, they are instead evaluated at runtime using the query and key matrices. This content-driven nature allows for greater versatility in the operation's ability to extract features present in the image. The success of vision transformers can be attributed to this versatility of the attention mechanism, and its ability to extract long-range dependencies in a single shot.

The MLP (feed-forward network) layers appearing in the encoder perform projections on the patches individually, using learned weights to amplify features identified in the attention maps. This leads to the learned MLP layers working in tandem with the content-driven attention layers on feature extraction. A two-layer linear network is used for the MLP layers in the ViT architecture.

### 1.1.2 ViT Model Configurations

The network architecture parameters of different ViT configurations are listed in table 1.1. These configurations differ in the number of attention and MLP layers in the encoder block (depth), the dimensions of the attention heads and the tokens, and the hidden dimension of the MLP network. These configurations have widely varying FLOPs and trainable parameter counts, and accordingly

| Parameter | ViT-Tiny | ViT-Small | ViT-Base | ViT-Large |
|---|---|---|---|---|
| Patch Size | 16 | 16 | 16 | 16 |
| Depth | 12 | 12 | 12 | 24 |
| # Attention Heads | 3 | 6 | 12 | 16 |
| Attention Head Channel Dimension | 64 | 64 | 64 | 64 |
| Token Outer Dimension | 192 | 384 | 768 | 1024 |
| MLP Hidden Dimension | 768 | 1536 | 3072 | 4096 |
| **Summary** | | | | |
| # trainable parameters | 5.790M | 22.197M | 86.859M | 304.716M |
| FLOPs per forward pass | 9.36B | 30.98B | 110.97B | 382.13B |

Table 1.1: **Network architecture parameters of different ViT configurations**

vary on their accuracy metrics. We use the Tiny, Small and Base variants as the baseline models for comparison in our experiments.

### 1.1.3 Challenges in using Vision Transformers

Compared to Convolutional Neural Networks, deploying Transformers for image processing tasks comes with the following challenges:

1. Vision Transformers lack the inductive bias that is inherent to CNNs. Since self-attention and MLPs are generic operations, they can learn to approximate a wide range of functions. However, they do not come with the prior knowledge of "where to look" when it comes to approximating functions that can appear in the image processing context. In contrast, operations like Convolution and Pooling are restricted to focus on neighboring patches in the image- this serves as an effective inherent bias while performing image processing.
2. It is seen that pre-training the transformer network on large datasets can make up for the lack of inductive bias in the model [6]. It is hence common for Vision Transformer models to be trained on larger variants of the ImageNet dataset [8], including one with over 21,000 classes. The authors of [6] also use some proprietary datasets from Google such as JFT-300M with 300 million images [9] and JFT-3B with 3 billion images [10], leading

to models with higher accuracy. This heavy reliance of transformer models on massive datasets poses a serious challenge when it comes to using them. It is imperative to use pre-trained models and adapt them for the task at hand when making modifications to the transformer architecture, or when building transformers for downstream image-processing tasks - training models from scratch isn't a viable option. The weights of models pre-trained on ImageNet-21k are made publicly available by Google Research; we use these weights for the baseline models in our experiments.

3. Compared to CNNs, Vision Transformers have more trainable parameters. In addition to a larger training cost, this also leads to higher memory requirements, which impacts the compute requirements during inference. Plus, transformers have a larger FLOP count per forward pass.

In effect, testing out modifications to the transformer architecture, and building models for downstream tasks have longer turnaround times and higher compute requirements than when dealing with CNNs and similar architectures.

### 1.2 PERFORMANCE OF VISION TRANSFORMER MODELS

Table 1.2 lists the accuracies of some vision transformer models along with their parameter counts. This list includes model architectures proposed in the original paper [6] and a couple of ViT variants- SWiN transformer [11] and Multi-Axis Vision transformer [12]. We see that the dataset used for pre-training the model significantly impacts the accuracy that can be attained by the model. Training the ViT-B/16 variant on ImageNet-1k alone leads to an accuracy of 78% while pre-training it on the larger ImageNet-21k dataset improves the accuracy to 85%. Pre-training ViT-B/16 on JFT-300M doesn't seem useful as the accuracy dips to 84.15%; however, pre-training ViT-L/16 on the dataset improves the accuracy by 2 percentage points compared to the corresponding ImageNet-21k model.

SWiN and MaxViT are variants of the vision transformer architecture that use simpler versions of the self-attention operation. This leads to models that are less versatile than the original

transformer but are sufficiently complex enough to perform well on image processing tasks. These models have fewer FLOP counts than ViTs of similar size while maintaining similar or better accuracy metrics. The trick that these architectures employ is to reduce the complexity of the self-attention layer and compensate for it by increasing the depth of the network (the increased depth leads to them having more trainable parameters than ViTs).

| Model | Pre-trained Dataset | Parameter Count | Top-1 on ImageNet-1k |
|---|---|---|---|
| ViT-Ti/16 | ImageNet-21k | 5.790M | 74.61 |
| ViT-S/16 | | 22.197M | 82.44 |
| ViT-B/16 | | 86.859M | 84.92 |
| ViT-L/16 | | 304.716M | 85.08 |
| ViT-B/16 | ImageNet-1k | 86.859M | 77.91 |
| ViT-B/16 | JFT-300M | 86.859M | 84.15 |
| ViT-L/16 | | 304.716M | 87.12 |
| SWiN-Ti | ImageNet-21k | 28.288M | 80.90 |
| SWiN-S | | 49.606M | 83.27 |
| SWiN-B | | 87.768M | 85.16 |
| MaxViT-Ti | ImageNet-1k | 30.964M | 84.84 |
| MaxViT-S | | 68.999M | 86.09 |
| MaxViT-B | | 119.615M | 86.39 |
| MaxViT-B | ImageNet-21k | 119.615M | 87.81 |

Table 1.2: **Accuracies of some Vision Transformer Model Variants**

### 1.3 PROBLEM DEFINITION

Through this project, we try to find suitable Algorithm-level optimizations of the ViT architecture for their efficient deployment on GPU-like hardware. We evaluate the efficacy of optimization

strategies based on their impact on the model accuracy and computational cost. The optimization strategies would be subject to the following constraints:

1. **Low turnaround time**: In order to fully explore the design space available to each optimization strategy, the time and compute required to build models with the optimized architecture must be sufficiently low. This means the optimization strategies must avoid long training schedules to the extent possible, and maximize the reuse of pre-trained models (that is, changes to the architecture, if any, should not require training the entire model from scratch). Further, the experiments would be performed on smaller datasets to expedite the process.
2. **Hardware-friendly algorithms**: The target hardware for our experiments are NVidia GPUs from the Turing and Ampere architectures (the models that were readily available on Google-Cloud during the course of the project). The optimization techniques considered here would have to fit well onto these GPUs- this constrains the precision formats, matrix sizes, and operation scheduling schemes that may be used.

In chapter 2, we analyze the bottlenecks in the ViT architecture and zero in on possible optimization schemes. Chapters 3 and 4 discuss the two key optimization strategies that were tested out in this work- building light-weight "MetaFormer" models and applying sparsity to ViTs, respectively.

# CHAPTER 2

# OPTIMIZATION OPPORTUNITIES IN VISION TRANSFORMERS

## 2.1 BOTTLENECKS IN VIT ARCHITECTURE

The operations performed within an encoder in a vision transformer are shown in figure 2.1. Among these, the layers marked in blue are Linear Projection operations. These layers perform dense matrix multiplications on the input activations using learned weight matrices. The number of float operations performed by these layers varies linearly with the input size, that is, the number of patches the image is broken into and the dimension of each patch vector. The linear projection layers contribute towards a significant chunk of the operations performed in the ViT encoder, and can thus be potential bottlenecks in the model.

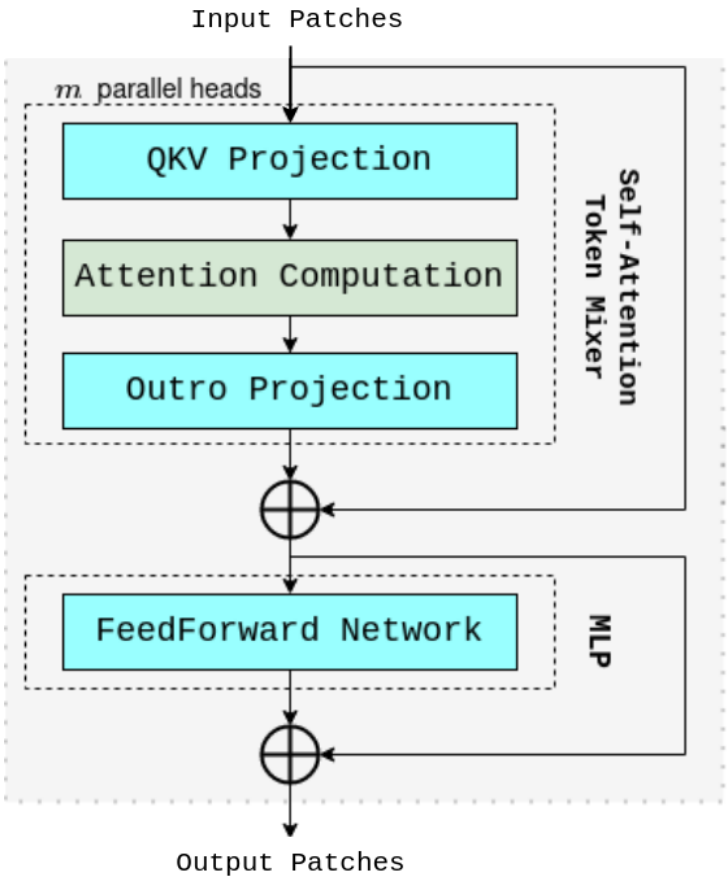
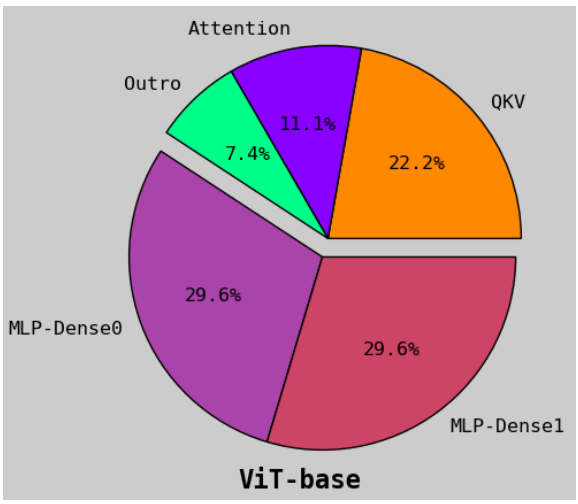
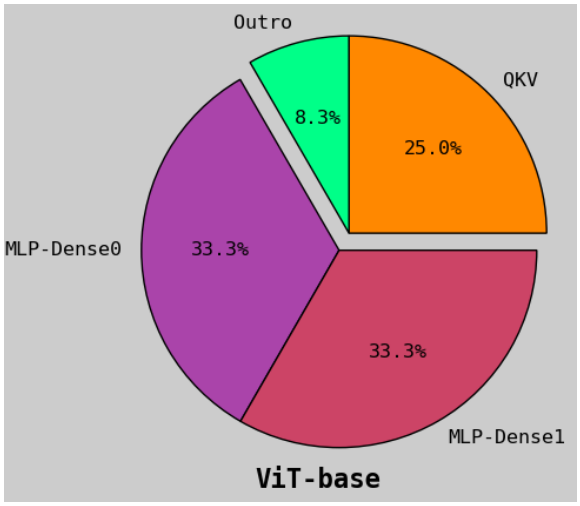



Figure 2.1: **Layers in a ViT Encoder**




(a) **Layer-wise FLOP break-up of ViT-B encoder**




(b) **Layer-wise parameter break-up of ViT-B encoder**

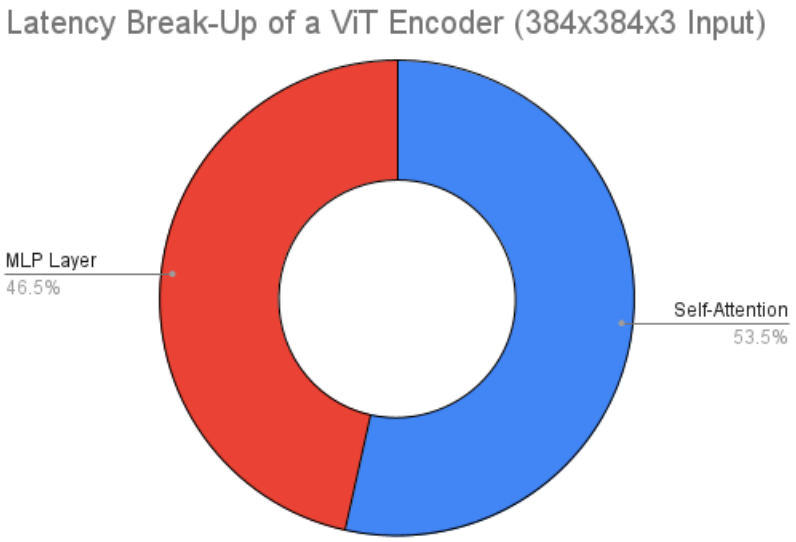
Figure 2.2: **Layer-wise cost analysis of ViT-B encoder for input image of size** $384 \times 384 \times 3$

The attention computation block involves computing the dot product between the query ($Q$) and key ($K$) matrices, evaluating the softmax of the product to generate the attention map ($A$), and computing the attention head output ($A \times V$). The dot product step has a quadratic cost with respect to the number of patches as every patch in $Q$ is multiplied by every patch in $K$. The softmax step involves fewer FLOPs per instance compared to the other Matrix Multiplication operations but requires the usage of the GPU's Special Function Units (SFUs) in order to evaluate the `exp()` function. This causes the computations to move away from the high throughput Tensor Cores to the relatively lower throughput SFUs, and can hence slow the entire computation down.

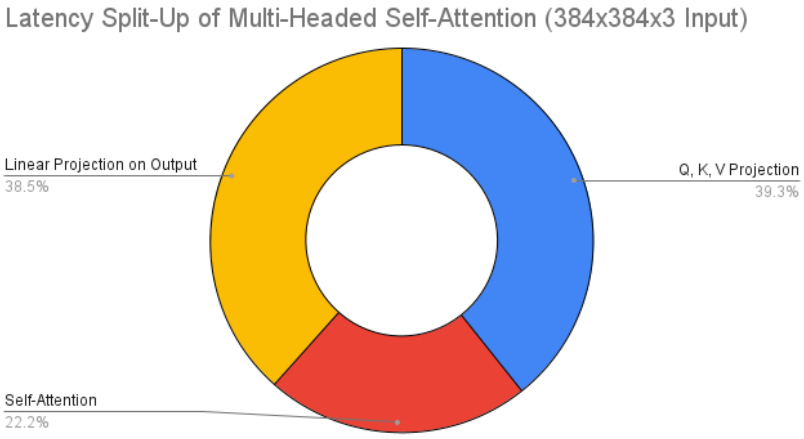
## 2.2 LAYER-WISE COST ANALYSIS

We evaluate the number of float operations, trainable parameters, and the execution time for each layer in the transformer's encoder as a first step towards zeroing in on the optimization strategies. We take the ViT-B/16 variant and use input images of size $384 \times 384 \times 3$ for this exercise.

In figure 2.2a the theoretical split of FLOPs from each layer in the encoder is plotted. The theoretical calculation only considered the operations arising from matrix-matrix multiplications- the activation functions contribute significantly fewer FLOPs. First, we see that the MLP layers make up close to 60% of the floating point computations performed in the encoder. Within the




(a) **Latency split of ViT-B encoder**




(b) **Latency split of operations within Attention Block**

Figure 2.3: **Latency Split of operations in a ViT-B encoder**

attention block, the $Q \times K^T$ and $A \times V$ products (labeled "Attention") contribute only around 11% of the overall FLOPs, the remaining 29% comes from the linear projection operations ($QKV$ and Outro Projections). This shows that for the given input size, the attention dot product isn't a significant bottleneck. At larger input sizes, the share of the "Attention" operation in the FLOPs split-up would increase due to the quadratic scaling of the computational cost of the operation. A similar split is seen for the number of trainable parameters (figure 2.2b). Note that since the attention operation is entirely content-driven, it does not feature in this plot.

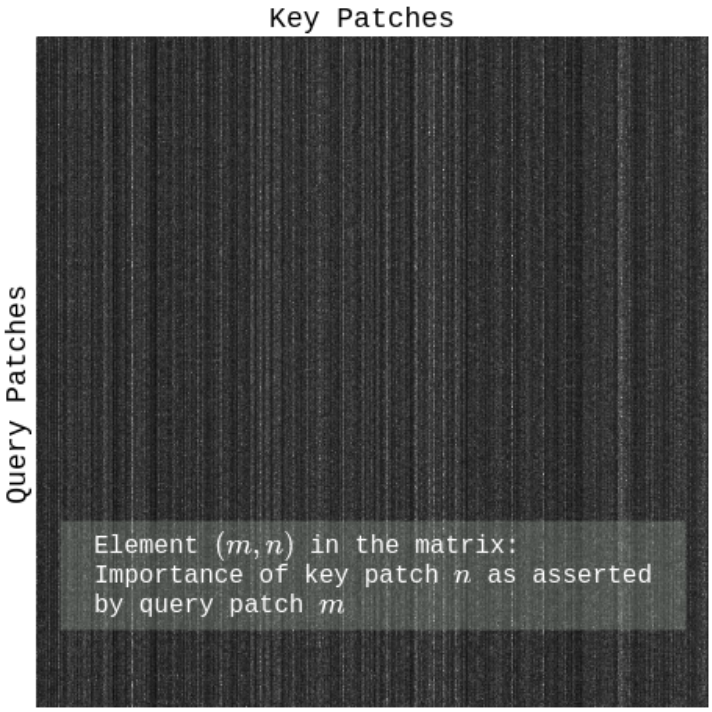
From figure 2.3a, we see that the MLP and Multi-headed attention blocks take similar amounts of time to complete execution. Within the self-attention block (figure 2.3b), around 80% of the execution time is spent on the linear projection layers. The attention computation ($Q \times K^T$, softmax evaluation and $A \times V$) take up only 20% of the execution time. This clears up uncertainties from the previous section regarding the bottlenecks in the network:

1. While the cost of the attention computation scales quadratically with the number of patches, its cost in absolute terms is lower than the cost of the linear projection operations for the input sizes of interest.
2. The effect of going away from the Tensor Cores to the SFUs to evaluate the softmax seems to be negligible.

Thus, the primary bottlenecks in the network are the linear layers from the attention and MLP blocks. This is unlike the transformers used for language modeling where the attention computation is almost always the bottleneck. This is because language transformer models typically have arbitrary input sequence lengths, often long enough for the attention computation's quadratic cost to become apparent. For vision transformers, the input sizes are typically fixed, and as we see here, the quadratic cost of the attention computation does not kick in at moderate image sizes.

### 2.3 OPTIMIZATION OPPORTUNITIES

The attention map from an encoder layer in ViT-B/16 that is generated when an image from the ImageNet dataset is used as an input is visualized in figure 2.4. Larger values in the attention




Attention Map Generated by Head #9, Encoder #6
of ViT-B for an image from the ImageNet val dataset

Figure 2.4: **Visualization of an Attention Map Matrix $A$ from a ViT-B/16 model**

map appear as brighter pixels in this image. Element $(m, n)$ in the attention map matrix quantifies the correlation between the $m^{th}$ query patch and the $n^{th}$ key patch. A characteristic feature of the attention map in figure 2.4 is the presence of vertical stripe-like features. These features indicate that most query patches give high correlation scores to the same set of patches in the key matrix. That is, a subset of patches from the input activation is considered to be important by the self-attention mechanism and these patches are given a higher weightage in the attention map. Interestingly, this pattern is seen in attention maps across attention heads and encoder layers for different images from the dataset. This indicates the presence of some order in the features learned by the transformer, which can perhaps be caught by a mechanism simpler than self-attention.

As noted in the previous section, the focus of the optimization task must be on the linear layers in the attention block and MLP layers. In this project, we focus on the following two optimization strategies:

1. Replacing the attention block with a simpler mechanism that can extract features from

image patches. This would replace the linear projection layers ($QKV$ and Outro projection) and the Attention Computation operations from the encoder (figure 2.1).

2. Inducing sparsity into the linear layers from the attention and MLP blocks. Sparsity enables reducing the number of trainable parameters and forward pass FLOPs by ignoring elements in the weight matrices that are 0 or have a magnitude close to 0. This usually involves a phase of pruning where weight elements are nudged towards values with low magnitude and then rounded off to 0. Post pruning, the weights undergo few iterations of re-training, with the pruned elements frozen, in order to compensate for any loss in the model's learning. NVidia GPUs from the Ampere generation have support for structured sparsity in hardware. We believe the linear layers in the network are suitable targets for structured sparsity.

### 2.4 RELATED WORKS

We look at previous works on transformer model optimization that share a similar approach as ours. The original work on Vision Transformers was followed by multiple attempts to replace self-attention with similar but less expensive mechanisms [11; 12; 13; 14]. I. Tolstikhin, et. al., [15] propose replacing the attention block with an MLP token mixer, resulting in an all-MLP network architecture. The authors of [16; 17] explore replacing attention blocks with trivially simple token mixing operations, such as Average Pooling, Separable Convolution and Identity. This leads to a generic transformer architecture, the "Metaformer", where an arbitrary function is used to mix image tokens in place of self-attention. The above-mentioned works demonstrate commendable classification accuracies with their respective attention replacement operations on the ImageNet dataset. We explore this direction further, the experiments and their results are discussed in chapter 3.

Inducing sparsity in the Multi-Headed Attention block has been proposed by multiple works [18; 19; 20; 21] primarily in the context of language transformer models. Z. Li, et. al. [20] propose to prune away attention scores below a certain threshold. The threshold is determined through a gradient-descent search during model training, by modifying the regularization term in the loss function. Y. Rao, et. al. [21] propose dynamic token pruning in vision transformers. Motivated

by trends similar to the attention map in figure 2.4, the authors claim that only a small subset of tokens in the input activation are relevant for inference. The authors train a prediction module that samples the relevant tokens from the input and discards the rest. Applying this sampling scheme hierarchically, the work demonstrates pruning of image tokens by up to 66%, resulting in a FLOPs reduction of 31-37%.

In this project, we target the sparse Tensor Cores on NVidia Ampere GPUs that support 2:4 structured sparsity. We induce sparsity into the linear layers from the Multi-Headed Attention and MLP blocks. These experiments are orthogonal to the dynamic token pruning idea proposed in [21]; hence, we believe the reduction in computational costs of the two methods would be additive.

Using reduced floating-point precision formats for models has been another direction that is pursued in transformer model optimization. In particular, the FP8 formats launched by NVidia on their Hopper generation of GPUs [22] (the successor to the Ampere generation) have garnered interest. This is a future direction of interest that could not be pursued in this project due to the unavailability of the Hopper generation GPUs.

# CHAPTER 3

# METAFORMER EXPERIMENTS ON VIT

This set of experiments deals with reducing the computational complexity of the Multi-Headed Self-Attention layers in a Vision Transformer, by replacing them with simpler operations.

The self-attention operation in a layer of the transformer encoder learns function mappings between latent feature spaces of the image. The versatility of the operation enables it to generalize a wide variety of common DNN operations that accumulate information across patches of an image, such as `Conv2D` and average pooling [23]. While this versatility of self-attention makes it a suitable operation in generic architectures like Transformers, it may be sufficient to use a simpler operation in its place when focusing on specific tasks. Thus, a viable model optimization strategy would be to replace self-attention layers in the transformer with appropriate token-mixing functions, that can attain an accuracy comparable to the original transformer at a lower computational cost.

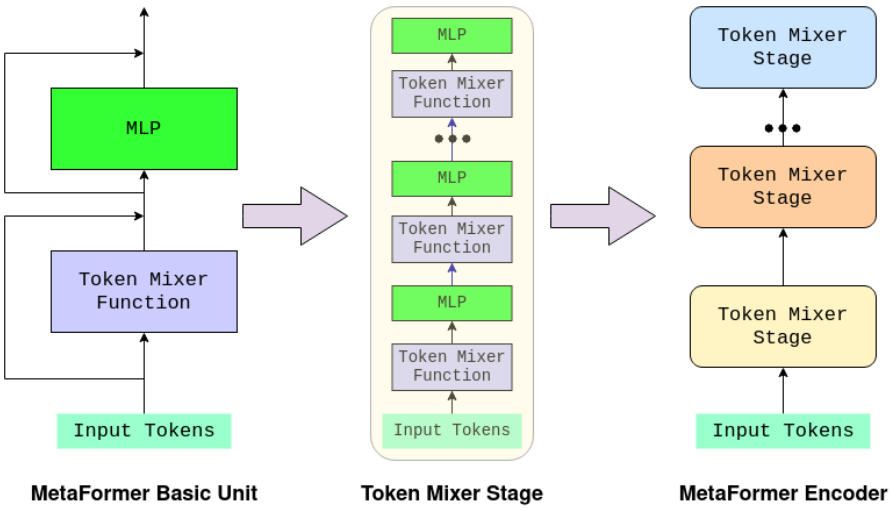



Figure 3.1: **Anatomy of a MetaFormer Encoder**

Yu, et. al. [16; 17] demonstrate commendable accuracy of "Metaformer" models where self-attention layers of a ViT are replaced by trivially simpler operations such as Average Pooling and Identity, on the ImageNet-1k dataset. The authors claim that the macro-architecture of the

transformer (skip-connections, token-mixing module, MLP layers and choice of activation functions) is more critical to the accuracy than self-attention.

The basic unit of a Metaformer encoder architecture comprises of a token mixer function followed by an MLP layer. Multiple identical basic units stacked together forms a monolithic token mixer stage, and multiple token mixer stages stacked together forms an arbitrary Metaformer encoder (figure 3.1). For instance, a ViT encoder can be thought of as a Metaformer with a single monolithic token mixer stage of depth 12, with multi-headed self-attention being the token mixer function. In an arbitrary MetaFormer, each monolithic stage can employ different token mixer functions and use different values for hyperparameters such as the patchsize, token embedding length, MLP hidden dimension, etc.

Here, we leverage the idea of MetaFormers and devise a framework to generate simpler Transformer models from a pre-trained attention-based Vision Transformer. The details of our approach are discussed in the following sections.

### 3.1 PLUG-AND-REPLACE METAFORMERS

The authors of [16; 17] train metaformer models from scratch and rely on pre-training on large datasets such as Imagenet-21k for attaining high classification accuracy. In contrast, we replace a few layers of a pre-trained Vision Transformer with custom token mixer stages, train these new layers on the dataset of interest (much smaller than datasets like ImageNet-21k), and reuse the remaining layers from the pre-trained network. This allows us to analyze the significance of the different layers of a ViT and the ability of different token-mixing functions to replace self-attention, at a lower cost of training.

Figure 3.2 highlights the approach followed for replacing layers in a ViT. The transformer is broken into $L$ stages, with each stage $i$ comprising of $d_i$ (stage depth) self-attention + MLP layers. The parameters $L$ and $\{d_i\}$ are left as hyperparameters to be tuned during model training. The ViT variants tested (tiny, small and base) have a total depth of 12 layers.

A metaformer configuration is generated by replacing a stage from the pre-trained ViT with a custom stage from a bag of token-mixer contenders. The parameters of the custom token mixer

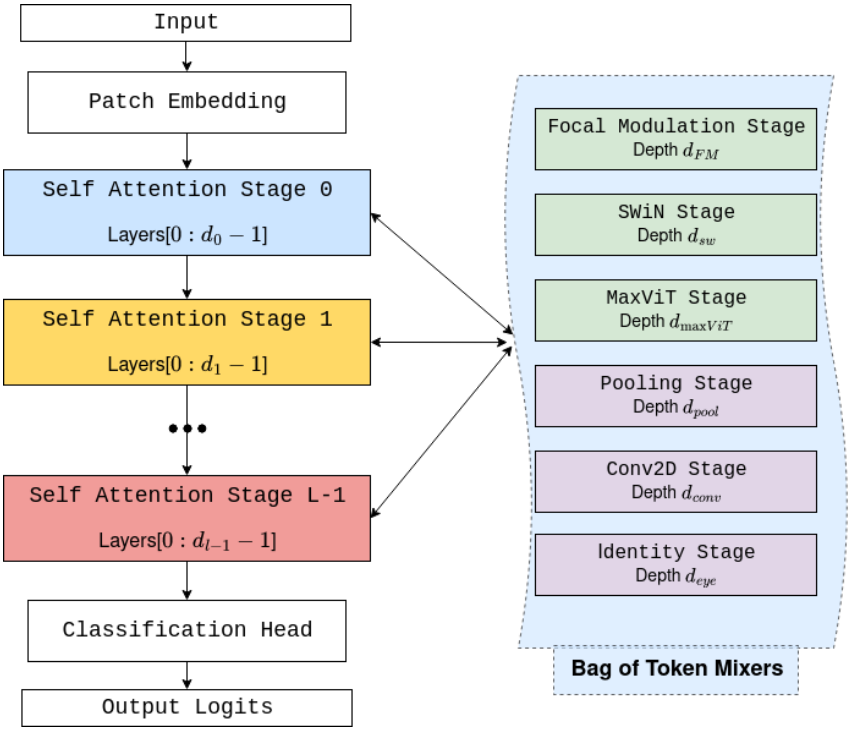



Figure 3.2: **Replacing stages in a Vision Transformer with operations from a bag of token mixers**

stage are set as trainable, while the weights of all remaining stages are frozen to that of the pre-trained network. The cost of training a metaformer is now limited to the parameters of the custom token mixer stage, which is considerably lower than training the entire model. Further, the low number of trainable parameters in the metaformer configuration and the inherent inductive biases of the simpler token mixer functions in the replaced stage allow training to be performed on the dataset of interest without pre-training on a massive dataset. Hence, the cost of generating metaformer variants from a pre-trained vision transformer model is considerably lower than training the metaformer from scratch.

We also explore using knowledge distillation to speed up model training, by using the pre-trained ViT as a teacher model and the metaformer as a student. Through distillation, we aim to train the student model to better mimic the features of the pre-trained teacher model. Details on the distillation methodology and their results are discussed in sections 3.4 and 3.5 respectively.

This work differs from other works that propose replacing self-attention, such as [17; 14; 11; 12] in the following ways:

| Number of Classes | 10 |
|---|---|
| **Image Size** | (320, 320, 3)<br>resized to (384, 384, 3) |
| **Dataset Size** | 346 MB |
| **# Training Images** | 9025 (69.7%) |
| **# Validation Images** | 3929 (30.3%) |

Table 3.1: **Summary of the ImageWoof dataset**

- We develop a framework for generating hybrid metaformers composed of multiple token mixer operations, including self-attention.
- We leverage information learned by self-attention through pre-training on large datasets in two ways.
  - First, we train only those layers where attention is replaced, using pre-trained weights for all other layers.
  - Second, we employ knowledge distillation to hand-hold token mixer functions to mimic the pre-trained self-attention layer.
- The training experiments are conducted on the smaller ImageWoof dataset, and have smaller training time and lower training hardware requirements than the other works.

### 3.2 IMAGEWOOF DATASET AND BASELINE RESULTS

We conduct the model optimization experiments on the ImageWoof dataset [24] (table 3.1). This dataset is a subset of ImageNet-1k containing 10 classes of dog breeds. Since this dataset is considerably smaller than the ImageNet-1k dataset, the efficacy of various model optimization strategies can be evaluated with a shorter turnaround time. Next, since the dataset is derived from ImageNet, it is reasonable to assume that the accuracy trends observed with this dataset would extend to ImageNet as well. Finally, the classes of images in the ImageWoof dataset are correlated (dog breeds), making the classification objective task somewhat difficult. This is in contrast to other ImageNet subsets such as ImageNette [24], which are largely composed of uncorrelated classes.

### 3.2.1 Classification Accuracy on ImageWoof

We compare Vision Transformer variants against CNNs on the ImageWoof validation dataset. ImageNet pre-trained variants of ViT (*tiny*, *small*, *base*), EfficientNet (*B0*, *B3*, *B5*, *B7*) [25] and VGG (*-16*, *-19*) [26] architectures are used, with modifications made to the final layers of the models to adapt to ImageWoof. The transformer models are modified by replacing the original MLP classification head with a 1-layer deep MLP block (with 10 output classes). Similarly, the output of the last convolutional layer in the CNN models is flattened and passed to a dense layer to generate the output logits.

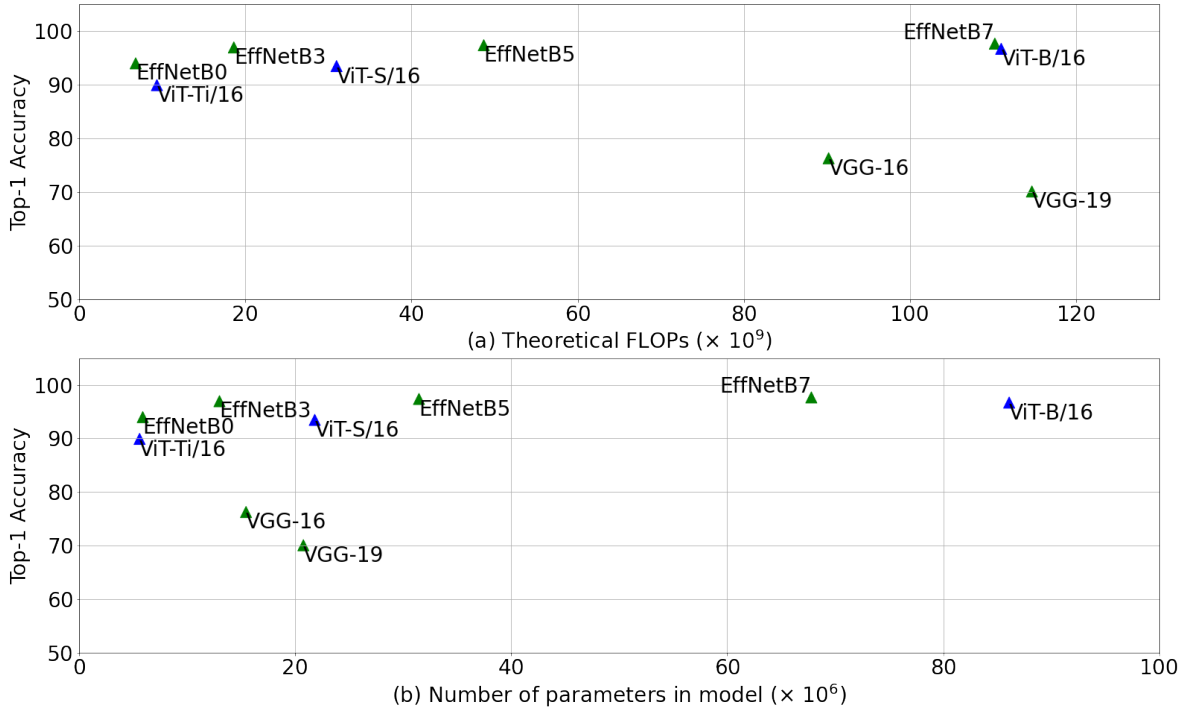



Figure 3.3: **Comparison of ViT and CNN models: (a) FLOPs vs accuracy, (b) Number of parameters vs accuracy**

The FLOPs vs accuracy and parameters vs accuracy trends for the models tested are plotted in figure 3.3. Models that feature at the top-left in the two plots attain a good trade-off between accuracy and model complexity. Comparing the families of architectures tested here, we see that ViTs do better at this trade-off when compared to VGG models, but are beaten by the EfficientNet models. However, it is worth noting that the ViT uses a generic architecture with

scope for complexity reduction unlike EfficientNet, whose architecture has been tuned over a Neural Architecture Search to attain an optimal accuracy-complexity trade-off [25].

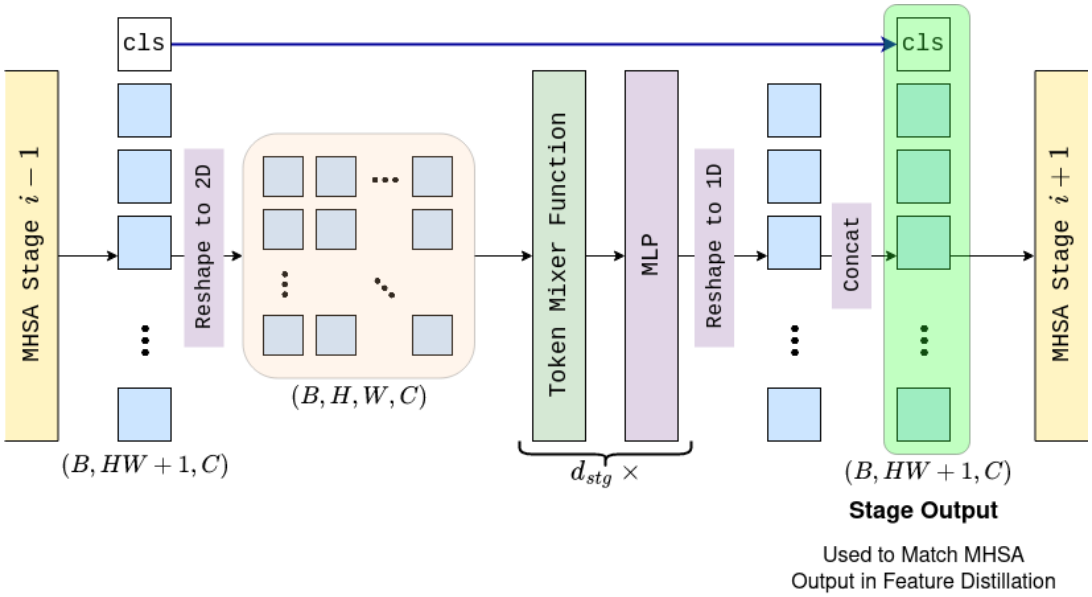
The goal of the following experiments is to evaluate the amount of room that is available for the ViT architecture on the accuracy-complexity landscape. We hypothesize that substantial reductions in FLOPs and parameters can be made without hurting the accuracy. This hypothesis is supported by the observations made in section 2.3.

### 3.3 TOKEN MIXER STAGE ARCHITECTURE

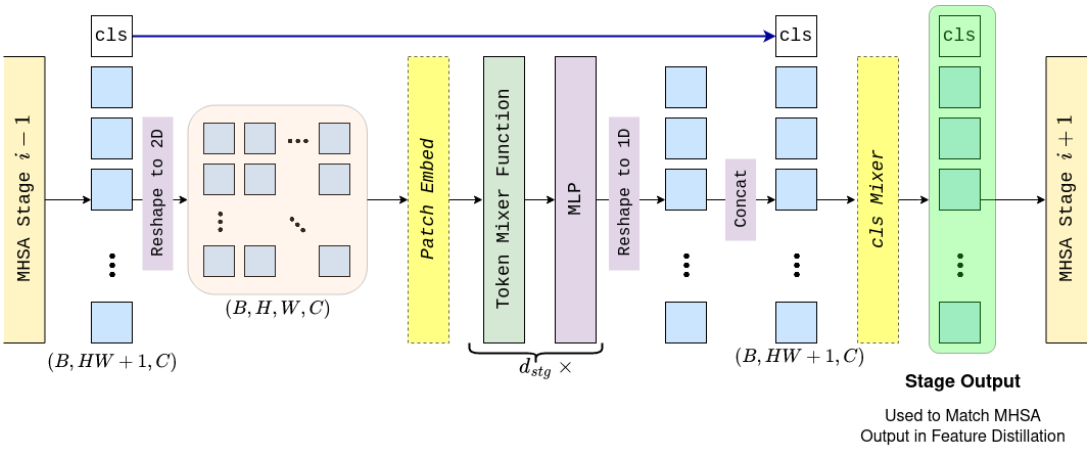
The functions in the bag of token mixer contenders (figure 3.2) come from two classes. Functions in the first class are variants of the attention mechanism and are operations that were proposed to enhance or maintain the accuracy of self-attention with a reduced computational cost. This class includes the following functions: focal modulation [14], shifted and windowed attention (SWiN transformers [11]), and block and grid attention (MaxViT [12]). Functions in the second class are relatively simpler operations such as separable convolution, average pooling, and identity operation [17]. These provide a baseline for the accuracy that can be attained using the MetaFormer architecture.

The encoder architecture used in the original works that propose these functions is slightly different from that of the Vision Transformer. The associated transformer models are not monolithic, instead, they are composed of multiple stages with each stage operating at a different patch size resolution. Further, the operations are performed only on patches from the image, additional tokens like the `cls` token are not used. Finally, since these functions involve a spatially local operation, the input to the operations is a 2D grid of patches. This is unlike the Vision Transformer where the image patches are flattened into a 1D sequence and input to the encoder. The architecture of the custom token mixer stages used in this study mirrors that of the parent models from these works. The skeletal architecture of a generic token mixer stage is shown in figure 3.4a. The 1D sequence of tokens output by the previous MHSA stage (stage $i-1$) includes the `cls` token and $HW$ image tokens. The image tokens are reshaped to a 2D grid and passed on to the token mixer function and MLP layers. The output 2D grid from the last MLP layer in the stage is reshaped back to a

1D sequence and then concatenated with the `cls` token. Note that the layers in the stage do not operate on the `cls` token, hence this architecture relies on the following MHSA stage (stage $i+1$) to accumulate features in the image tokens in the stage into the `cls` token.




(a) **Layers in a Token Mixer Stage**




(b) **Modified Token Mixer Stage with Patch-Embed and `cls` Mixer layers**

Figure 3.4: **Architecture of a generic Token Mixer Stage used in the MetaFormer experiments**

Figure 3.4b depicts a modified architecture for the stage with the following additional layers (marked in yellow):

- **Patch Embed**: The 2D grid of patches is passed through a unit-strided $4 \times 4$ convolutional layer before the token mixer function operates on it. This allows for some local accumulation

of information across the tokens, potentially aiding the token mixer function.

- **`cls` Mixer**: The `cls` mixer layer uses linear layers to update the `cls` token with a weighted sum of the old `cls` token and the output image tokens of the token mixer stage. For the experiments, we used two `Dense` layers for the mixer layer- a depth-wise accumulator layer that first computes a weighted sum of every token along each channel, and a channel-wise accumulator that updates the `cls` token with a weighted sum of the elements from the depth-wise accumulator output. Note that the `cls` Mixer updates only the `cls` token, the image patches preserve the value from the token mixer stage output.

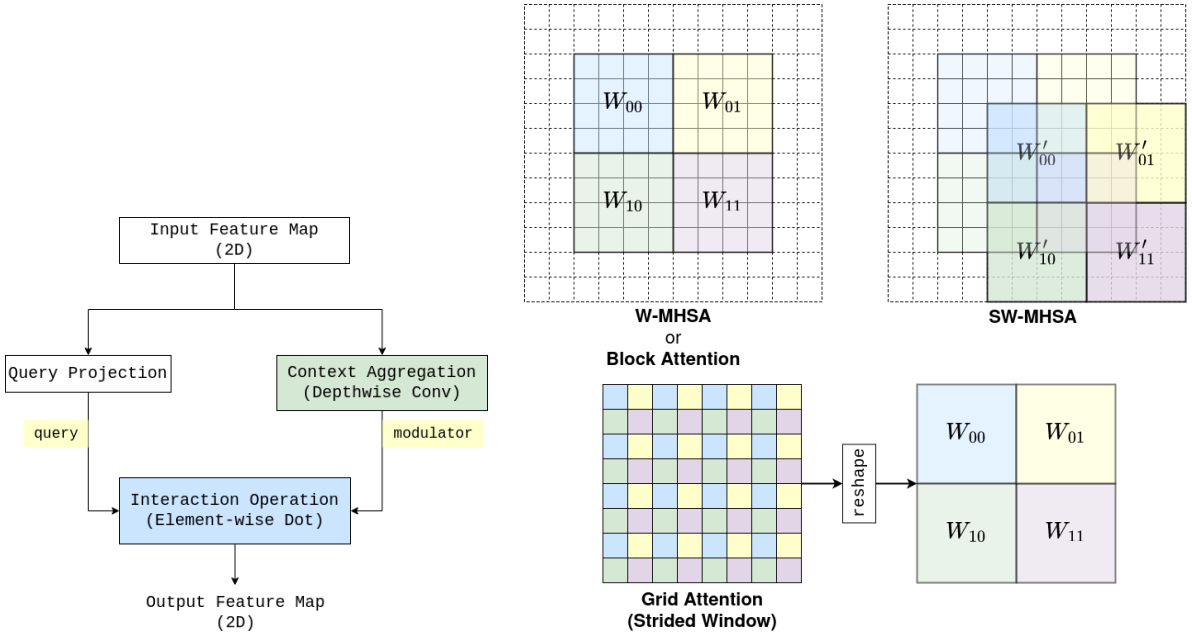
### 3.3.1 Token Mixer Functions

The functions used in the bag of contenders are briefly described here:

#### Focal Modulation

A generic feature encoding operation on a feature map $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ can be thought of as performing an interaction operation $\mathcal{T}$ on every token $\mathbf{x}_i \in \mathbb{R}^C$ in the feature map over its neighbouring tokens, and aggregating them using an operation $\mathcal{M}$. For instance, in self-attention, the interaction operation $\mathcal{T}$ is performed by the attention computation between the query and key tokens, and the aggregation operation $\mathcal{M}$ is performed by the multiplication of the attention map with the value tokens.

Focal Modulation [14] differs from self-attention in that it first performs context aggregation $(\mathcal{M})$ on the input tokens through a depth-wise convolution, and then computes the interaction $(\mathcal{T})$ with a query matrix using an element-wise dot-product. Performing an early aggregation helps reduce the computational cost of the interaction operation when compared to self-attention. The computational cost of the interaction operation is linear in the number of tokens here, unlike self-attention where the cost is quadratic in the number of tokens. This distinction, however, does not affect the accuracy of the model- Focal Modulation networks in fact manage to perform better than vanilla ViTs. Figure 3.5a highlights the operations in a focal modulation layer. Some other ideas used in the architecture, such as hierarchical contextualization and gated aggregation, can be found in the original paper [14].




(a) **Focal Modulation Architecture**

(b) **Different variants of attention: Windowed & Shifted Windowed (SWiN), and Grid Attention (MaxViT)**

Figure 3.5: **Token Mixer Functions that approximate self-attention with a lower computational cost**

### Shifted Window Attention

One way to limit the computational complexity of self-attention from scaling quadratically with the number of tokens is to restrict the attention computation to windows of patches in the image. For a fixed window size, the number of windows in an image scale linearly with the image size, while the cost of attention within a window is a constant factor. Thus, the overall cost of such an operation is linear in the number of tokens.

A drawback of the windowed attention operation is that it can no longer accumulate global information in a single shot, which is one of the unique selling points of self-attention. The SWiN architecture [11] addresses this issue by shifting the windows every alternate layer within a stage. This allows for some inter-window information accumulation across layers. The authors of [11] demonstrate an improved accuracy on the ImageNet dataset over the original ViT models with the SWiN architecture (table 1.2). Figure 3.5b illustrates the windowing and shifting schemes used by the windowed and shifted-window MHSA operations.

In our experiments, we use window MHSA with a window of size $4 \times 4$ patches and shift the windows by $2 \times 2$ patches every alternate layer in the stage. In addition, we add a relative position

bias vector to each patch in the window, as suggested by the original paper.

#### Block and Grid Attention

The block and grid attention mixer is inspired by the architecture used in Multi-Axis Vision Transformers [12]. This token mixer stage follows an approach similar to the SWiN stage, alternating between two variants of attention that accumulate local and global information respectively. Local information is accumulated using Windowed MHSA (the block attention operation), while grid attention is used to gather global information. In the latter operation, the attention scores are calculated amongst patches placed regular strides apart in the image. An example of grid attention for a window size of $4 \times 4$ using a stride of $2$ patches is shown in figure 3.5b. The cost of grid attention too scales only linearly with the number of tokens, as the cost of attention within a window is fixed for a given window size.

#### Separable Convolution

This stage is composed of an input projection layer, a separable convolution layer and an output projection layer. We use a kernel size of $7 \times 7$ for the separable convolution, and upsample the feature map dimensions by a factor of 2 in the input projection layer. A separable convolution is used due to its lower parameter and FLOP count, compared to a standard convolution operation.

#### Average Pooling

In this stage, a 2D average pooling layer with a filter size of $3 \times 3$ and a stride of $1$ is used. This allows for some local inter-token information accumulation in the token mixing stage, but its representation power is severely limited compared to other mixer operations used in the experiment.

#### Identity Operation

Forming the baseline of the experiment is the identity operation, where the token mixer layer is replaced by an identity layer. The only operations performed on the tokens are layer normalization and the projections by the MLP layers. Since the MLP projections are performed on each token individually, this stage does not perform any inter-token information accumulation.

## 3.4 KNOWLEDGE DISTILLATION

Knowledge distillation [27] is a model optimization technique that is used to train and build *student* models with reduced complexity (parameter count and MAC operations), from larger pre-trained *teacher* models. The key idea employed here is to use the output features of the pre-trained teacher network as soft labels to train the student network, so that the student learns to mimic the teacher network. The student model is trained with the objective to perform well on two metrics: teacher-student fidelity and student generalization. The former metric quantifies how well the student model mimics the teacher's features and is evaluated using a distillation loss function, while the latter measures the classification accuracy of the student model on the final validation/test dataset.

### 3.4.1 Distillation Hypothesis for MetaFormers

An MHSA layer in a transformer encoder learns a function mapping $f_{MHSA} : \mathbb{R}^{HW+1 \times C} \mapsto \mathbb{R}^{HW+1 \times C}$ that encodes features from an input latent feature space of patches to another output feature space. We hypothesise that a token mixer operation from the bag of contenders can be trained to mimic the function learned by MHSA through knowledge distillation, and hence be used to replace MHSA layers in the ViT network. We base the hypothesis on the following assumptions:

1. For most layers in a transformer, there exists some arbitrary operation $\tau^*$ that is considerably simpler than MHSA (fewer parameters and MAC operations), and learns a mapping $f_{\tau^*}$ between the input and output latent feature spaces of the layer that is at least good as $f_{MHSA}$ on the dataset of interest. The goal of this experiment hence becomes to find the operations $\tau$ from the bag of token mixers that fit the criteria of $\tau^*$.
2. A function $f_\tau$ is a good approximation of $f_{MHSA}$ if it achieves a low distillation loss on the dataset of interest. That is, if the outputs of a student layer are similar to the outputs of a teacher layer across a training dataset, the functions learned by the two layers are similar.
3. Reducing the distillation loss between a student and teacher results in better classification accuracy of the student model on the dataset of interest. Note that this assumption is trivially true if the student attains a distillation loss of 0 (student accuracy = teacher accuracy), but

it's not obvious if the assumption holds for non-zero distillation losses.

Once the suitable set of operations from the bag of token mixers are found, different configurations of metaformer models can be constructed based on where they stand on the model accuracy versus complexity trade-off. Leveraging knowledge distillation to build metaformer models has the following potential benefits:

- Although the distillation is performed on a smaller dataset (ImageWoof), the student layer learns to mimic a teacher layer that was pre-trained on a larger dataset. Hence it is possible that the student layer learns features that would have not been learned had the training solely relied on the smaller dataset.
- If the size of the distillation dataset required to effectively train the student through knowledge distillation is smaller than when training from scratch, this technique can give rise to significant savings on the training time of metaformer models.

### 3.4.2 Design Choices

There are four broad design choices in incorporating knowledge distillation into the metaformer training framework. These are: (i) the granularity at which the layers are replaced and distilled, (ii) the features that are picked to match through distillation, (iii) the loss function used to quantify teacher-student fidelity and student generalization, and (iv) the dataset and data augmentation strategies employed during student model training. We try out multiple options for each of these design choices, as discussed below.

#### Layer Replacement Granularity

The layer replacement may be carried out at the level of a basic unit layer in the encoder replacing each attention layer with an alternative token mixer operation (figure 3.6). This approach can lead to more diverse metaformer configurations, with each layer (basic unit) of the encoder employing a different token mixer operation. The objective of distillation would now be to get each basic unit layer in the metaformer to mimic the corresponding layer in the ViT.

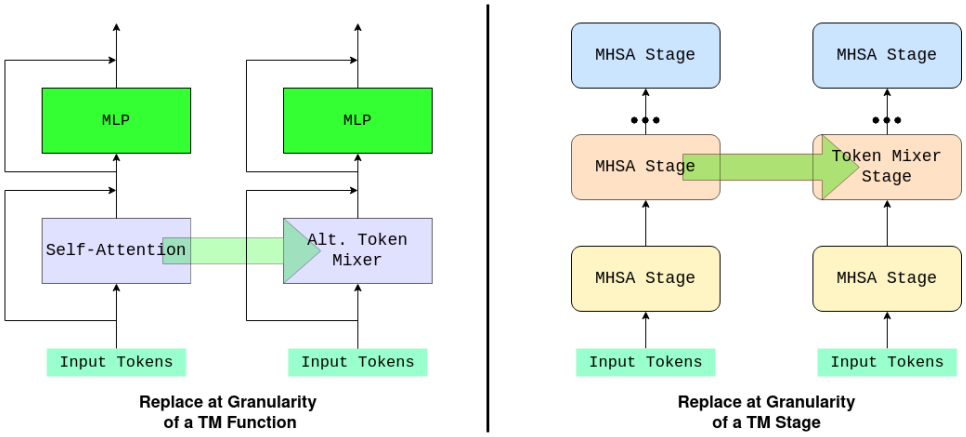



Figure 3.6: **Choices for granularity at which feature distillation may be performed**

The second choice is to replace layers at the granularity of a stage, where an arbitrary number of layers make up a stage. All the layers within the stage (attention and MLP) are replaced with untrained token mixer and MLP layers. This approach allows for more flexibility in the distillation process as the objective here is to get the output of the token mixer stage to match the output of the stage being replaced- the individual basic units within the token mixer stage are not forced to mimic the corresponding unit in the ViT. The depths of the outgoing MHSA stage and the incoming token mixer stage are hyperparameters to be tuned during training.

The first approach succeeds only if each token mixer replacement attains a high fidelity score with the corresponding attention layer. If each layer were to deviate slightly from the attention's outputs, there could be a compounding effect resulting in a drastically different prediction at the network's output.

Further, the cost of the accompanying operations in the token mixer stage (patch reshape, `concat`, patch embed and `cls` mixer) is amortized when replacing layers at the level of a stage. Hence, we proceed with the second choice in our experiments.

### Feature Matching

Training the student network to mimic the teacher can be enforced in multiple ways. The first, most obvious, way is to drive the output logits of the two networks to be similar. In this case, the distillation framework tries to minimize an error metric between the output logits of the teacher

and student networks, using the softmax of the logits of the teacher as soft labels [27]. In our experiments, we found it difficult to tune the interior layers of the network (layers 0-8 in the 12-layer ViT), when matching the output logits. We reckon this is because the loss landscape w.r.t. the trainable parameters is not smooth, making the output logits less tractable from the layers being trained. An update to the weights in the direction of the gradient may take it far away that the updated point now corresponds to a loss value drastically different from the previous value. In fact, on multiple occasions during the experiments, the distillation loss was driven to `inf` and `NaN` values. This issue may be circumvented by using a smaller learning rate or by setting all the layers that follow the token mixer stage to also be trainable, but both these steps would increase the training time substantially.

An alternative way to get the student model to mimic the teacher is by matching the intermediate features of the network [28; 29]. The output features of a token mixer stage are trained to resemble the features of a corresponding stage in the ViT. Since the layers that follow the token mixer stage in the student are common to both networks, matching intermediate features would invariably cause the output logits of the networks to be similar. We use this approach in our experiments. However, the output feature maps of token mixer stages are much larger than the output logits, having a shape of $(HW + 1, C)$ (upwards of 100,000 elements per image) as opposed to the $(\texttt{numClasses})$ elements in the output logits. Having to match large feature maps could cause the distillation optimizer to progress slowly. The feature matching task may be simplified by performing a reduction operation on the distillation feature map, bringing down its size. We conduct some feature-matching experiments with the distillation feature reduced down to a shape of $(HW + 1, 1)$ by taking the mean of the features along the channel dimension. This dimension reduction scheme, however, comes with a loss of information on the features being matched, making the feature distillation task less direct.

**Distillation Loss Functions**

A distillation loss function maps the teacher and student feature maps to a scalar value, that can be intuitively interpreted to quantify a measure of distance between the two feature maps. It is desired to have a loss function that closely obeys assumption #3 of the hypothesis- attaining a

lower distillation loss should translate to better generalization, i.e. higher classification accuracy on the ground truth labels. Additionally the landscape of the loss w.r.t. the trainable parameters must be 'well-behaved', avoiding extremely large or extremely low gradients- this ensures that the distillation loss progresses consistently over multiple epochs.

If the student and teacher feature maps have different dimensions due to the custom token mixer stage using a different patch size or patch dimension size, a linear projection is applied on the student feature map before evaluating the loss. We try the following loss functions in our experiments:

- **Mean Squared Error:** The distance of the student distillation feature map $\mathbf{x}_{student} \in \mathbb{R}^{(HW+1)\times C}$ from the teacher feature map $\mathbf{x}_{teacher} \in \mathbb{R}^{(HW+1)\times C}$ is calculated as:

$$\mathcal{L}^2_{dist,MSE} = \frac{1}{(HW+1)\cdot C} \sum_{i=0}^{HW+1} \sum_{j=0}^{C} (\mathbf{x}_{teacher}[i,j] - \mathbf{x}_{student}[i,j])^2$$

- **Cross Entropy:** This loss function measures how close the probability distributions represented by the feature maps are. The feature maps are first flattened to a vector and converted to a pseudo probability distribution by computing the `softmax` of the vector. The cross-entropy of the student distribution $\mathbf{p}_{student}$ w.r.t. the teacher distribution $\mathbf{p}_{teacher}$ is calculated as:

$$\mathcal{L}_{dist,CE} = -\sum_{i=0}^{(HW+1)\cdot C} \mathbf{p}_{teacher} \log(\mathbf{p}_{student})$$

- **Maximum Mean Discrepancy:** The mean square error between the feature vectors computes the Euclidean distance between the points represented by the two vectors in the vector space. Attaining an MSE value of $\epsilon$ ensures that the student feature vector is inside a hypersphere of radius $\epsilon$ centered at the teacher feature vector. However, not all points within the sphere may correspond to points with good classification accuracy on the ground truth labels, making it difficult to enforce assumption #3 of the hypothesis with the MSE loss. This is more pronounced if the classification accuracy inherently depends on some patterns amongst elements in the feature vectors that are not captured by the MSE loss.

  One possible way to address this is by projecting the feature vectors up to a higher

dimensional space and computing the distance in the projected space. The distance metric in the higher dimensional space can possibly capture more patterns in the feature vectors, possibly improving the synergy between the distillation loss and the classification objective on the ground truth.

Maximum Mean Discrepancy (MMD) loss [29; 30] is a generalization of the MSE loss that employs the kernel trick to evaluate the inner product of two vectors in a higher dimensional space without explicitly projecting the vectors to the higher dimensional space. For a given kernel function $k : \mathbb{R}^{(HW+1)\times C} \mapsto \mathbb{R}$, there exists a higher dimensional vector space $\mathcal{V} = \mathbb{R}^V$ where the kernel output $k(\mathbf{f}_{teacher}, \mathbf{f}_{student})$, for $\mathbf{f}_{teacher}, \mathbf{f}_{student} \in \mathbb{R}^{(HW+1)\times C}$ is equal to the inner product of the projected vectors $\mathbf{f}'_{teacher}, \mathbf{f}'_{student}$ in $\mathcal{V}$ [31]. The MMD loss for a given kernel function $k(\mathbf{f}_{teacher}, \mathbf{f}_{student})$ is defined as:

$$\begin{aligned}\mathcal{L}^2_{dist,MMD} = &\frac{1}{((HW+1)\cdot C)^2}\sum_{i=0}^{(HW+1)\cdot C}\sum_{i'=0}^{(HW+1)\cdot C} k\left(\frac{\mathbf{f}^i_{teacher}}{||\mathbf{f}^i_{teacher}||_2}, \frac{\mathbf{f}^{i'}_{teacher}}{||\mathbf{f}^{i'}_{teacher}||_2}\right)\\ &+\frac{1}{((HW+1)\cdot C)^2}\sum_{j=0}^{(HW+1)\cdot C}\sum_{j'=0}^{(HW+1)\cdot C} k\left(\frac{\mathbf{f}^j_{student}}{||\mathbf{f}^j_{student}||_2}, \frac{\mathbf{f}^{j'}_{student}}{||\mathbf{f}^{j'}_{student}||_2}\right)\\ &-\frac{2}{((HW+1)\cdot C)^2}\sum_{i=0}^{(HW+1)\cdot C}\sum_{j=0}^{(HW+1)\cdot C} k\left(\frac{\mathbf{f}^i_{teacher}}{||\mathbf{f}^i_{teacher}||_2}, \frac{\mathbf{f}^j_{student}}{||\mathbf{f}^j_{student}||_2}\right)\end{aligned}$$

The following kernel functions were tested in our experiments:

1. Linear Kernel: $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$; using this kernel in the above expression is equivalent to the MSE loss.
2. Polynomial Kernel: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + c)^d$; the inner product is computed in a feature space over polynomials of the original features. By inspecting the binomial expansion of this kernel function, it is evident that the loss captures correlations in the cross terms of the feature vectors ($\mathbf{f}^i_t \cdot \mathbf{f}^j_s$ for $i \neq j$)- this is expected to aid in a better distance metric.
3. Gaussian Kernel: $k(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{2\sigma^2}(||\mathbf{x} - \mathbf{y}||_2^2))$; the inner product computed by this kernel function can be inspected by evaluating the Taylor series expansion of the expression- the 'higher-dimensional' space that the inner product is computed in is in

fact infinite dimensional.

**Classification Loss:** In addition to the distillation loss between the teacher and student feature maps, we add a classification loss between the output logits of the student metaformer model and the ground truth labels. The classification loss term can help drive the metaformer model towards points in the loss landscape with a higher generalization on the data, if the distillation loss fails at satisfying the assumptions of our hypothesis. We used the categorical cross entropy function to evaluate the loss between the metaformer output logits (post a `softmax` operation) and the ImageWoof data labels.

**Distillation Dataset**

The distillation experiments are primarily conducted on the ImageWoof dataset, training the metaformer model to mimic the ViT outputs for inputs from the dataset. Another approach would be to perform knowledge distillation on datasets different from the one of interest, and then fine tune the metaformer model on the dataset of interest, after the token mixer stages learn to mimic self-attention sufficiently well.

- **Random Dataset:** The outputs of the teacher and student models are matched on a generated dataset of random images. Attaining low training and validation distillation losses on a random dataset is indicative of the token mixer stage indeed learning to mimic the self-attention operation, enabling the token mixer operation to replace attention in multiple datasets. An added benefit of using a generated dataset is that we are no longer limited by the size of the classification dataset, generating as many images as required for achieving a low distillation loss (possibly online, resulting in savings on storage). However, a low distillation loss on random images need not extend to the dataset of interest, and can lead to heavier dependence on model fine-tuning post distillation. We conduct a few distillation experiments with randomly generated datasets.
- **Ensemble of Multiple Datasets:** Another approach is to use a mixture of real-world datasets for distillation. This provides more samples and diversity in the data used for

training, and hence may fare better than performing distillation on the dataset of interest alone [32]. We did not try this approach in our experiments due to the costs involved (higher training time and storage requirements), but this is a future direction that can be pursued.

Additionally, we also evaluate the impact of data augmentation on training the metaformer model. In the experiments, we use MixUp data augmentation [33], where synthetic images are generated from the dataset by computing a weighted sum of two images sampled at random from the dataset. In addition to improved generalization from mixing patterns from different images into a single image, augmentation also allows for increasing the samples used for training.

### 3.4.3 Training Methodology

Figure 3.7 summarizes the training methodology used in the experiments. The pre-trained vision transformer is cloned, and a self-attention stage in the clone replaced with a custom token mixer stage, to create the metaformer model. The metaformer model is then trained on the ImageWoof dataset using a linear combination (determined by hyperparameter $\alpha$) of the distillation loss $\mathcal{L}_{dist}$ between the stage outputs, and the classification loss $\mathcal{L}_{clf}$ w.r.t. the ground truth labels. Since the custom token mixer stage may not always be a perfect drop-in replacement of the MHSA stage ($\mathcal{L}_{dist} > 0$), we also set the MHSA layer that immediately follows the token mixer stage as trainable. This allows the MHSA layer to recover any loss of feature information arising from the custom token mixer stage.

## 3.5 PERFORMANCE OF METAFORMER MODELS

### 3.5.1 Comparison of Token Mixers and Distillation Loss Functions

We study how the token mixer operation used to replace self-attention affects the accuracy of MetaFormer models. The attention layers at the transformer's beginning, middle, and end are replaced in separate experiments; this helps us identify the layers of the transformer that are critical for accuracy, and those that are most suited for replacement. The loss function used for training is a combination of the classification and distillation losses- we use an $\alpha$ value of 0.35. For this experiment, we use the Mean Squared Error between the means of the student and

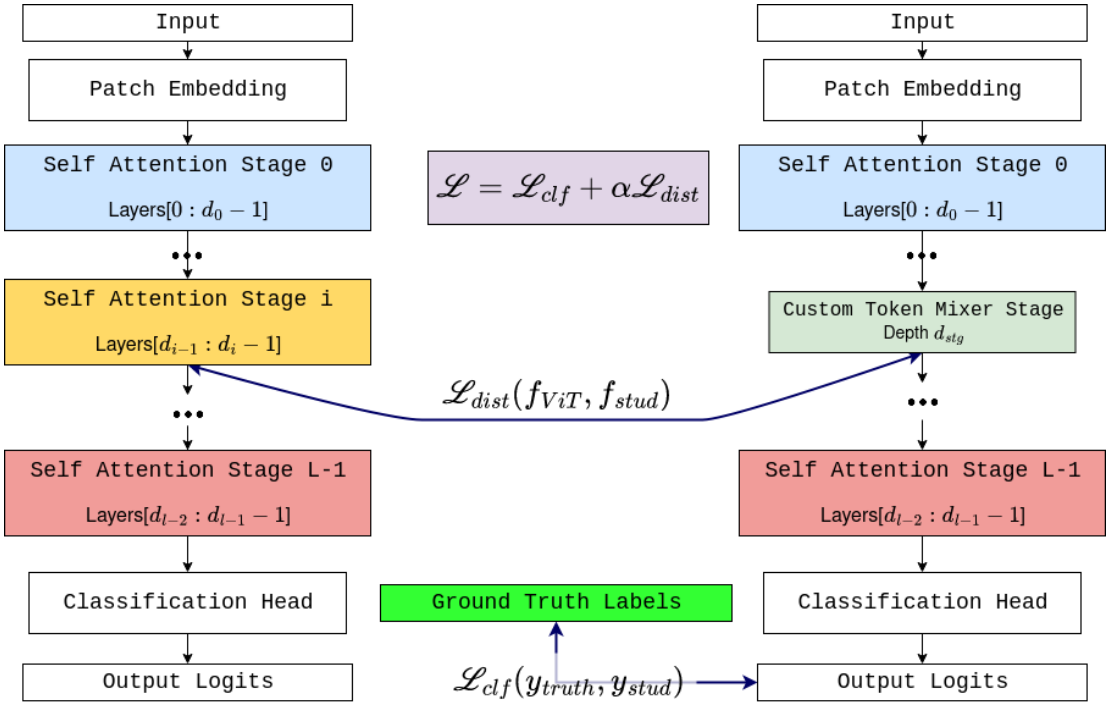



Figure 3.7: **MetaFormer Knowledge Distillation Training Methodology**

teacher feature maps, computed along the channel dimension, as the distillation loss.

The accuracies of MetaFormer models generated by replacing attention blocks at the beginning of the network are plotted in figure 3.8. The x-axis corresponds to different model configurations, the labels describe the attention layers of the transformer that were replaced, and the token mixer operation employed. For example, "Focal_2_6" implies that attention layers 2-6 and the accompanying MLP layers in the ViT were replaced by a Focal Modulation stage. The accuracy of the parent ViT model is marked by the horizontal line in the figure.

It is quite apparent that replacing the initial attention layers in the network adversely impacts the model's accuracy, with all the configurations tested showing a drop in accuracy of around 15%.

Figure 3.9 plots the accuracies of the MetaFormer models when attention layers towards the end of the network are replaced. Note that the last self-attention layer in the network is left untouched, as it may help recover features that may have been missed by the token mixer layers prior to it. We observe that the MetaFormer models fare better here, with most configurations attaining the same accuracy, if not a slightly higher accuracy, than the baseline parent model.

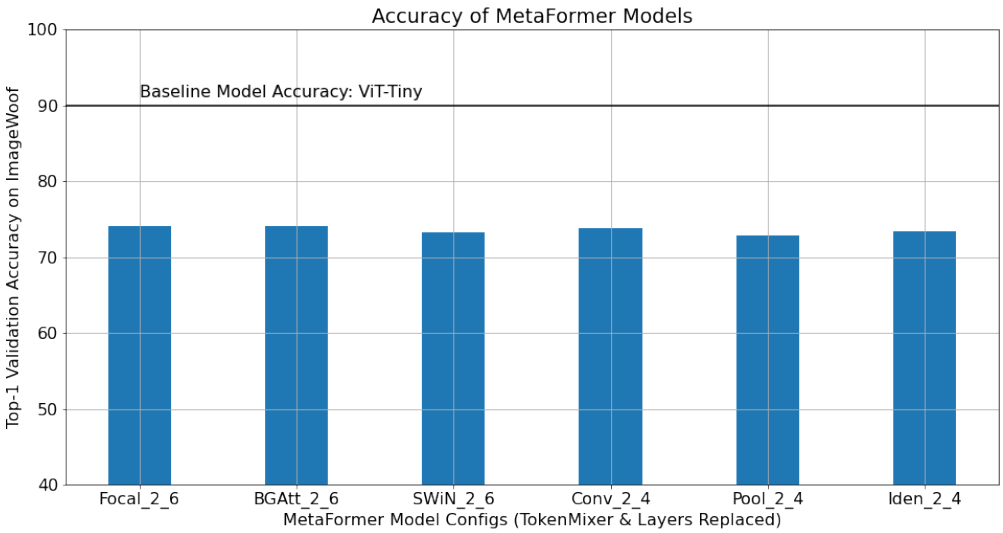



Figure 3.8: **Accuracy of MetaFormer Models when Attention Layers at the beginning are replaced**

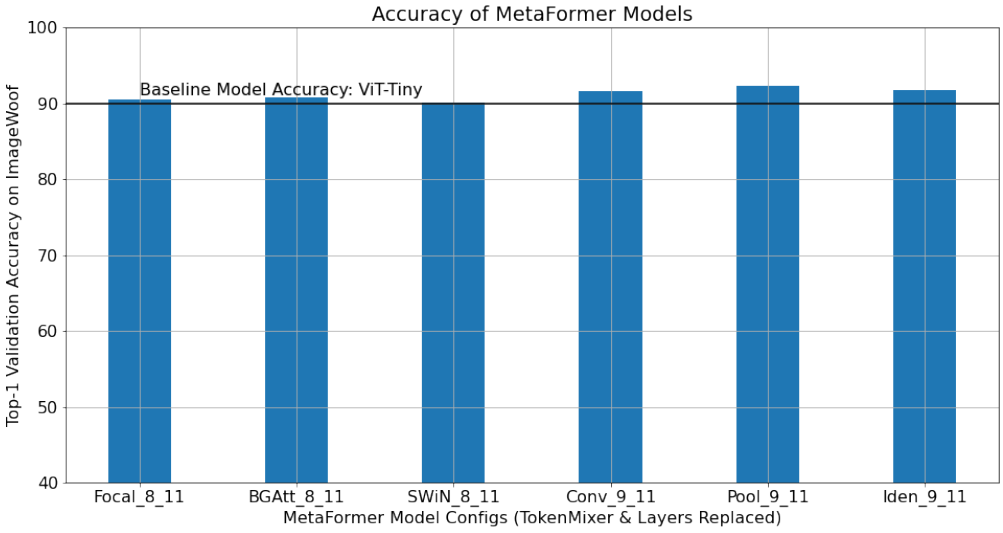



Figure 3.9: **Accuracy of MetaFormer Models when Attention Layers at the end are replaced**

The trends from the two plots suggest that the initial attention layers of the network focus on accumulating both local and global feature information from the image, while the later layers focus more on feature information present locally. This explains why replacing attention with token mixers that predominantly focus on extracting local information results in poor accuracy when the initial layers are replaced, but works well when replacing the later layers.

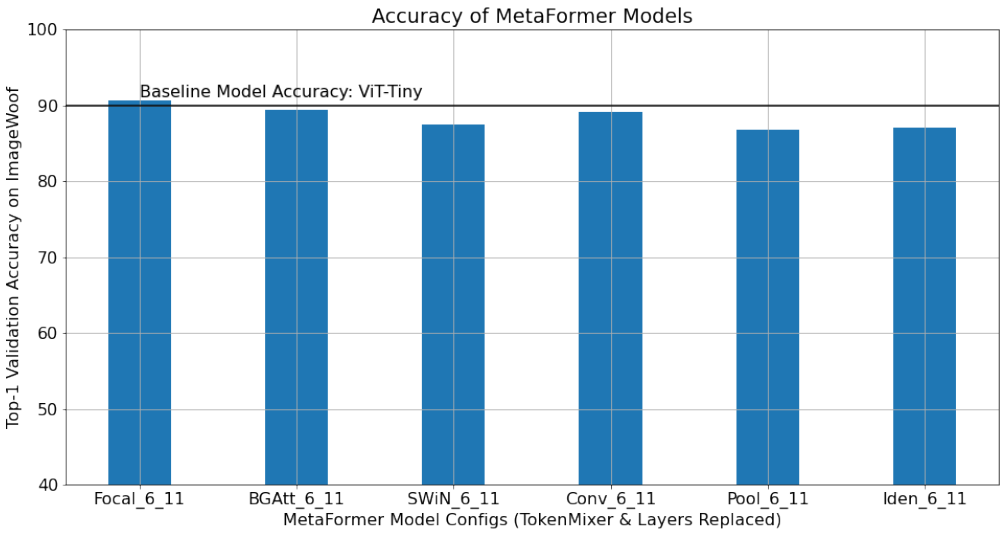



Figure 3.10: **Accuracy of MetaFormer Models when Attention Layers 6-11 are replaced**

Finally, figure 3.10 plots the accuracies of MetaFormer models when layers 6-11 in the network are replaced. These models too attain accuracies in the vicinity of the parent model, and promise higher savings in compute cost over those listed in figure 3.9.

It is also worth noting that in all three cases, there isn't a substantial diversity amongst different token mixer operations in terms of the accuracy attained. In fact, the "MetaFormer baseline" accuracy provided by the Identity Former configuration is quite high, hitting around 73% when it is used to replace layers at the beginning, and upwards of 85% when it replaces layers towards the end of the network.

Next, we compare different distillation loss functions. Here, we consider replacing layers 6-11 of the parent ViT with Focal Modulation, SWiN, or untrained Self-Attention layers. We vary the loss function used and the features to be matched while training the corresponding MetaFormer models. Comparing the results of the experiment for Focal Modulation and SWiN with that of the untrained Self-Attention MetaFormer allows us to identify differences in the behaviour of the token mixer contenders and self-attention, if any, during model training.

The distillation loss functions used here are Cross Entropy, Mean Squared Error (MSE), and Maximum Mean Discrepancy (MMD) losses, computed on the flattened feature vector. We also conduct experiments with MSE loss computed on the reduced mean of the feature vector (labelled

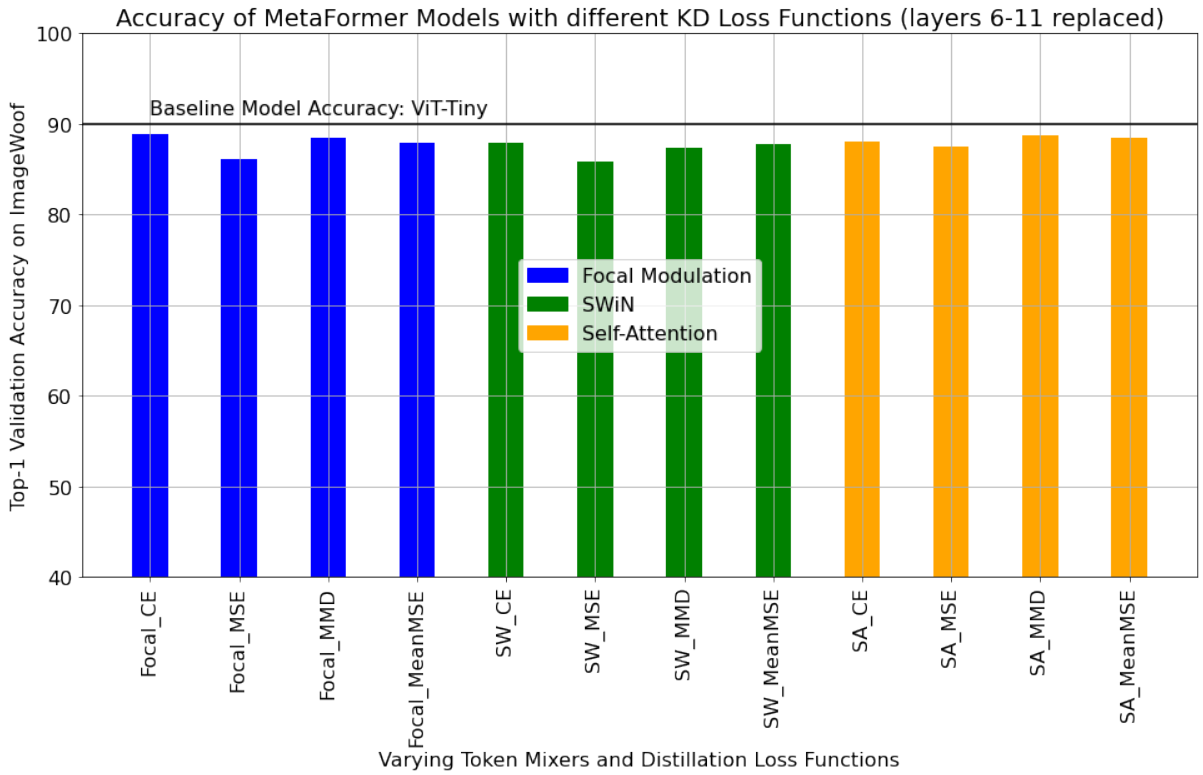



Figure 3.11: **Accuracies of MetaFormer Models when trained using different Knowledge Distillation Loss Functions**

as 'MeanMSE' in the fig. 3.11). The MMD loss uses a polynomial kernel of degree 4. The results of the experiment are depicted in figure 3.11.

The results in the above plot highlight three interesting trends:

1. Using Cross Entropy and MMD loss functions for distillation leads to higher accuracies of the MetaFormer model when compared with the MSE loss function. This could be due to the fact that the two loss functions use more sophisticated mappings from the teacher-student feature vector space to scalar loss values, while MSE computes the Euclidean distance between the two vectors.
2. Evaluating the MSE loss on the reduced mean feature vector ("MeanMSE") leads to a higher accuracy than evaluating it on the entire flattened vector. This is seen despite the obvious loss in feature information that arises from the reduction operation. However, performing the reduction on the feature vectors simplifies the MSE minimization task, which means that it is more likely for the loss to get driven toward the minimum when the loss is

calculated on the reduced feature vectors. Further, if the elements along the channel axis are correlated, the loss in feature information upon performing the reduction operation may not be drastic, allowing the model to hit higher accuracy values.

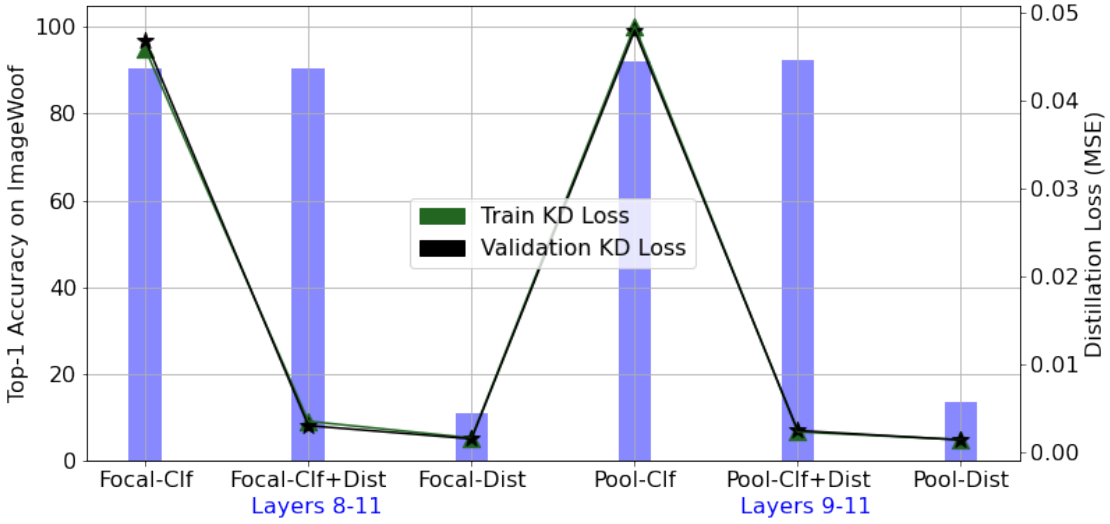
3. A similar trend is observed when the attention layers are replaced with untrained self-attention layers. This shows that the trends observed are related to the loss functions and the distillation methodology used, and are not heavily reliant on the token mixer operation of choice. Plus, as is evident from the plot, the Self-Attention MetaFormer models (marked in orange in fig. 3.11) are unable to attain the accuracy of the parent ViT, despite having an identical architecture. Thus, the distillation-based training methodology used here dwarfs in comparison to pre-training the layers on large datasets.

### 3.5.2 How Effective is Knowledge Distillation?

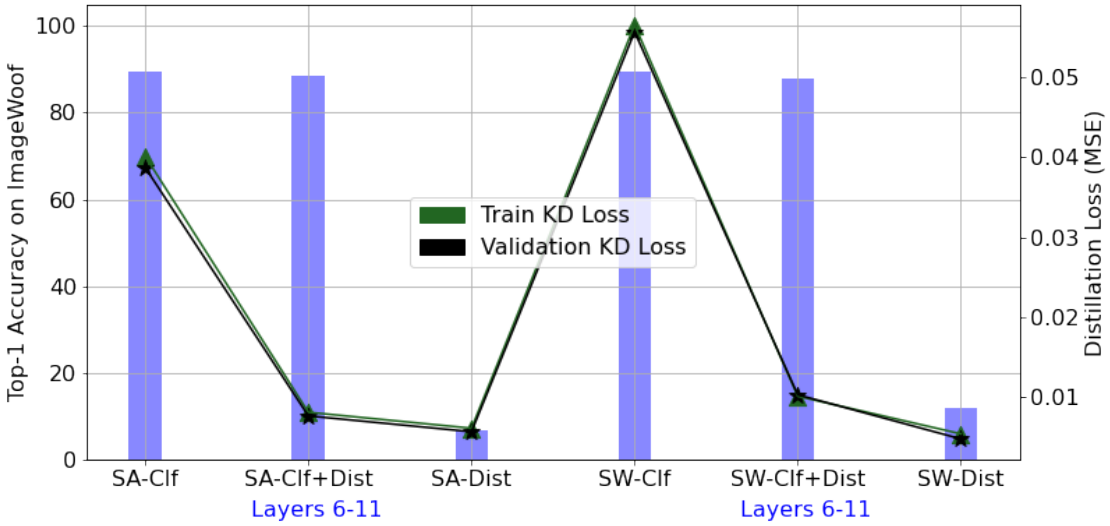
The MetaFormer models are trained with a dual objective of reducing the classification loss on the ground truth labels and the distillation loss on the features from the parent ViT model. It is imperative to evaluate the role of each of these loss objectives in arriving at the final MetaFormer model. To do so, we devise an experiment wherein we train models separately with only one of the two loss functions. The accuracies of the models trained with the three loss objective schemes (classification only, distillation only, classification + distillation losses) are compared to gain insights into the roles of the loss functions.

Figures 3.12a and 3.12b depict the results of this experiment for different token mixer operations. In these figures, we plot the top-1 accuracy attained by the model along with the training and validation distillation loss values. While training with a classification loss alone, the corresponding distillation loss is monitored for the sake of comparison. The trends from this experiment are summarized below:

1. Training with a classification loss allows the MetaFormer model to attain the same accuracy as when trained on classification + distillation losses, but the validation distillation loss is substantially higher. This implies that the MetaFormer model can achieve a decent accuracy without mimicking the parent ViT model.




(a) **Replace with Focal Modulation and Pooling Operations (layers replaced marked in blue)**




(b) **Replace with untrained Self-Attention and SWiN Operations (layers replaced marked in blue)**

Figure 3.12: **Comparison of Model Accuracy and Distillation Loss when trained with and without the Knowledge Distillation Loss Objective**

2. Training on the distillation loss alone leads to MetaFormer models with poor accuracy ($\approx 10\%$, which is equivalent to uniformly random guessing). However, the models have a low distillation loss with respect to the parent ViT's features. This indicates that the model learns to reduce the distance between its features and the parent ViT without effectively learning to mimic the parent ViT.

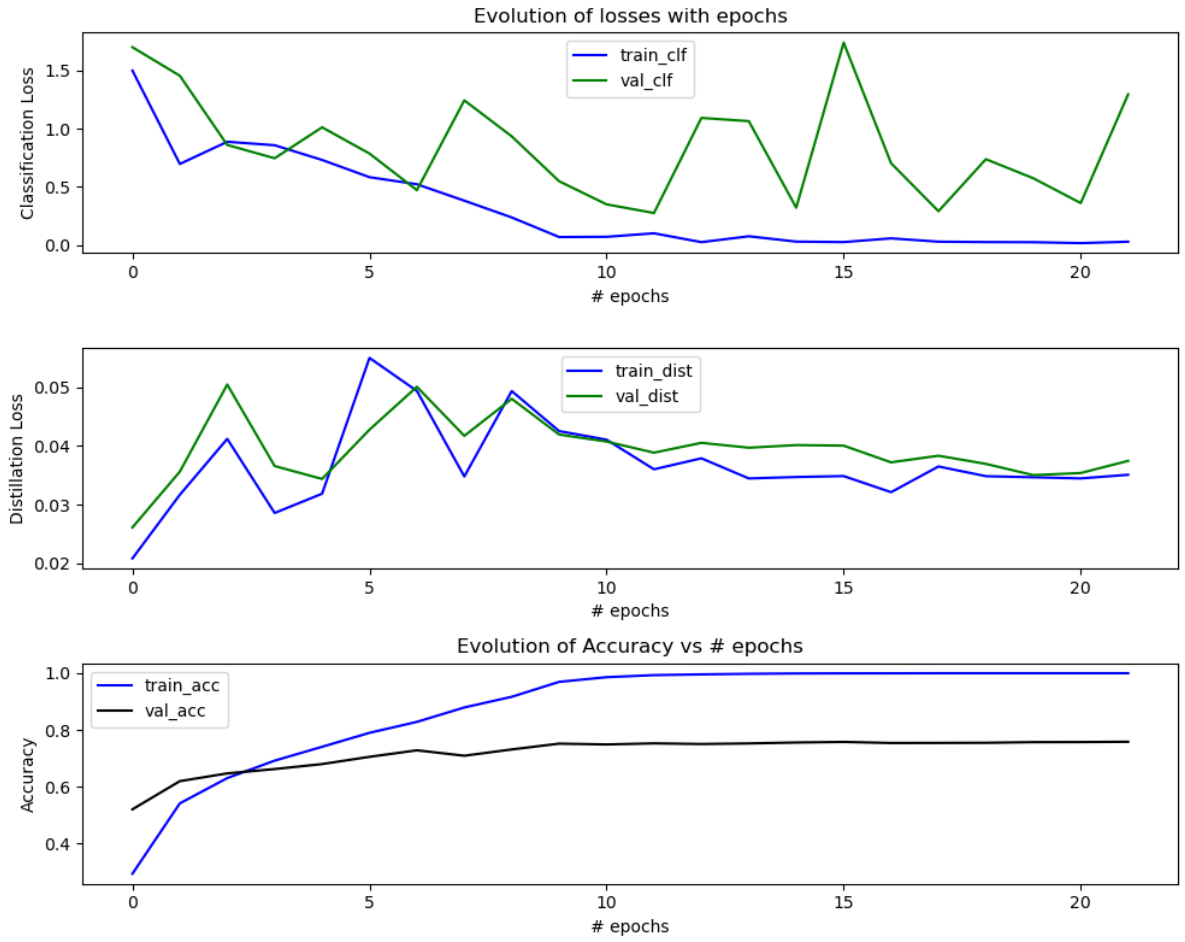



Figure 3.13: **Evolution of Classification, Distillation Losses and Accuracy with the number of epochs when trained with a Classification Objective alone**

The evolution of the classification and distillation loss values, and the model accuracy with the number of epochs during a training run are plotted in figures 3.13, 3.14 and 3.15. Figure 3.13 depicts the progression of the losses and accuracies for a Focal Modulation MetaFormer trained solely on the classification loss (replacing layers 2-6; final accuracy $\approx 73\%$). We see that the

corresponding model suffers overfitting (validation accuracy and classification loss stagnates while training accuracy and classification loss improve consistently), without making any improvements on the distillation loss front.

Next, in figure 3.14, we see that when trained with both loss functions, the distillation loss takes a smaller value from the first epoch and improves consistently. Further, there is no evidence of overfitting on the distillation loss as the values are quite similar when evaluated on the training and validation sets.

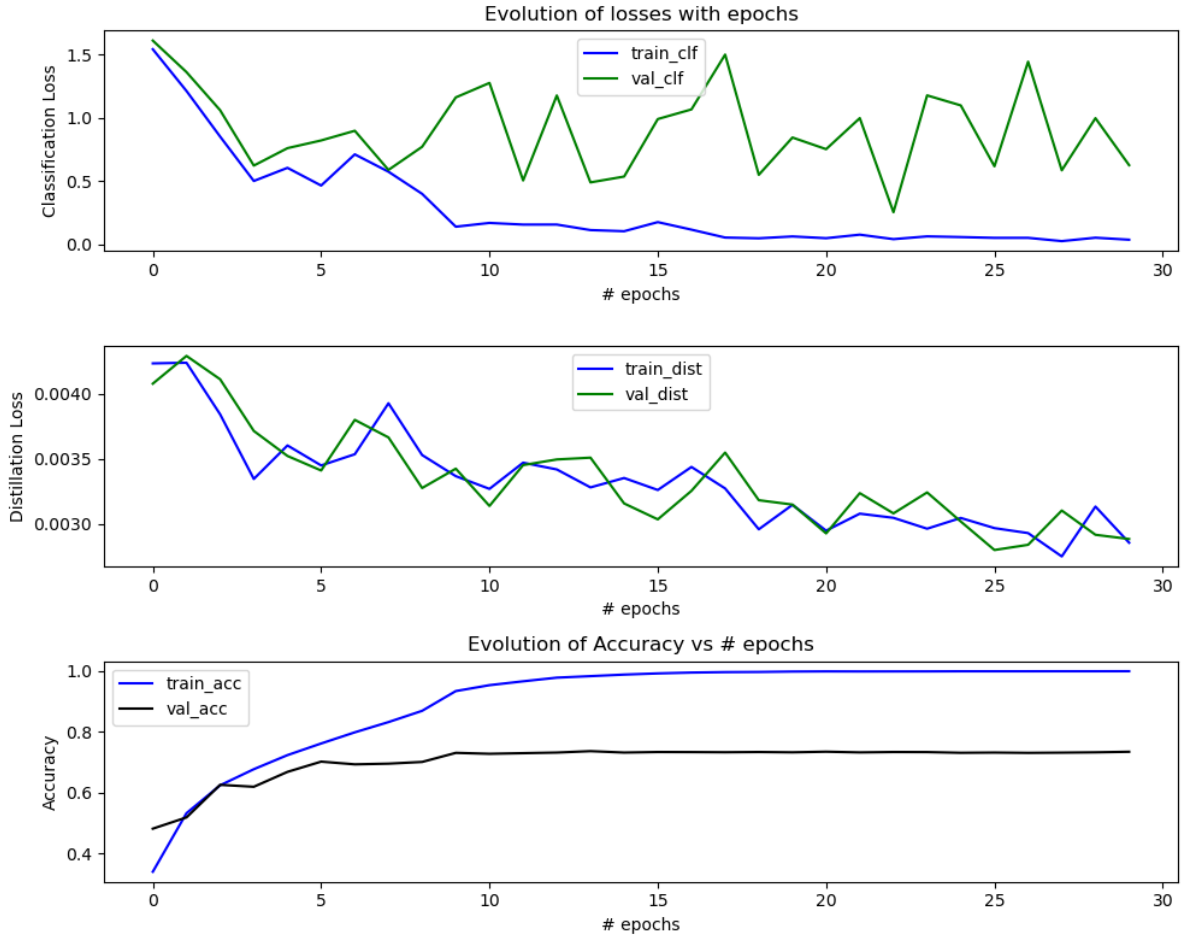



Figure 3.14: **Evolution of Classification, Distillation Losses and Accuracy with the number of epochs when trained with both Classification and Distillation Objectives**

Finally, the evaluation of losses and accuracies when trained with a distillation loss alone is plotted in figure 3.15. Here, we see constant improvements in the distillation loss (sans overfitting), without any meaningful impact on the classification loss or accuracy.

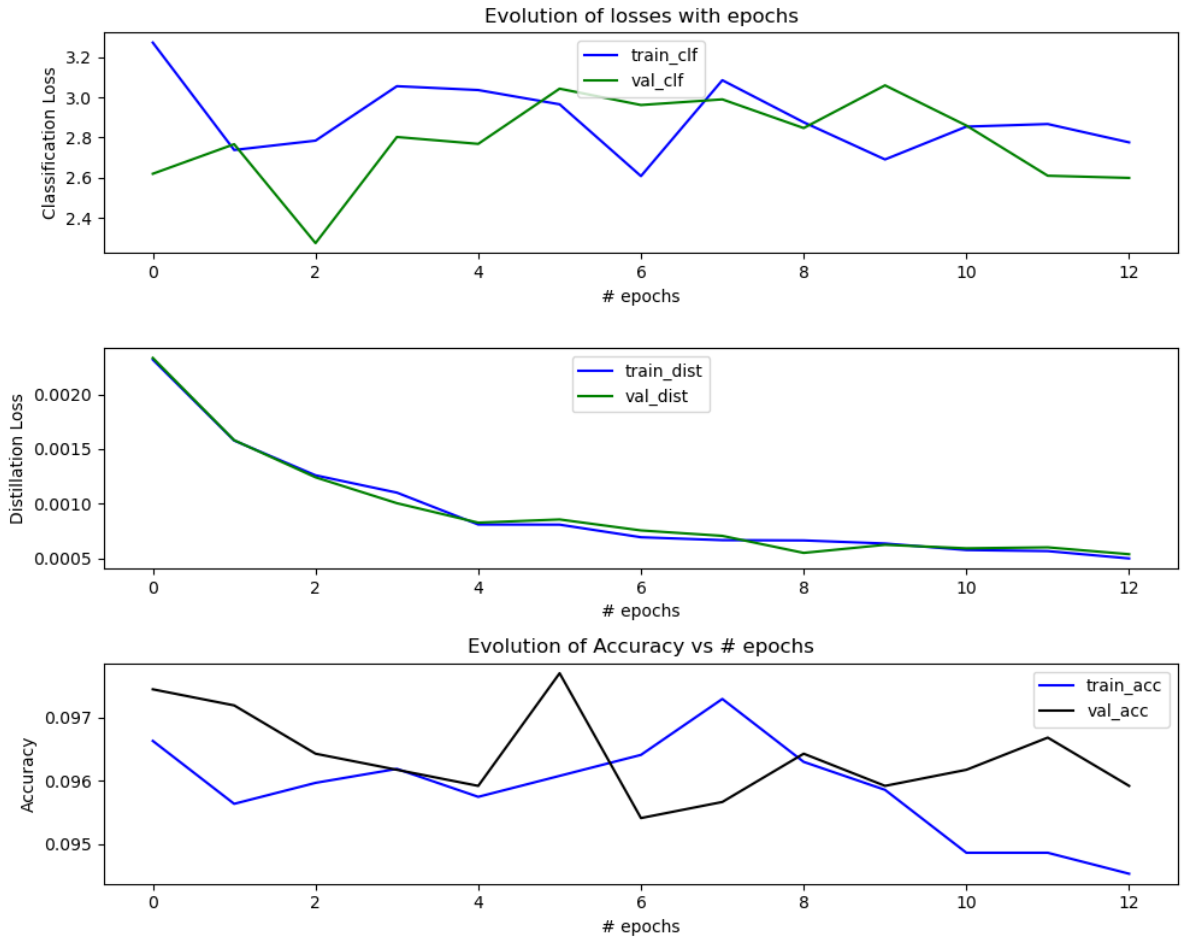



Figure 3.15: **Evolution of Classification, Distillation Losses and Accuracy with the number of epochs when trained with a Distillation Objective alone**

The above results lead us to the following insights:

1. The improvement in accuracies of the MetaFormer models are driven entirely by the classification loss between the output logits and the ground truth. The results in figure 3.12 prove that MetaFormer models can attain decent accuracy even if their intermediate features are drastically different from that of the parent ViT for the same input image. We believe this is because the intermediate features belong to a huge vector space- it is likely that multiple unrelated points in this vector space correspond to features that result in good accuracy.
2. Bringing a knowledge distillation loss into the training mix helps bring the intermediate

features of the MetaFormer model and the parent ViT closer but isn't sufficient for the model to mimic the parent. We see that the distillation loss does not go all the way to 0, regardless of the number of epochs the model is trained. This stagnation in the distillation loss indicates that there is always a finite non-zero distance between the teacher and student, which is significant enough to hurt student-teacher fidelity.

3. The results in figure 3.12 show that models using the same token mixer operation can end up with high accuracy and high distillation loss, high accuracy and low distillation loss, or low accuracy and low distillation loss depending on the loss functions used. This highlights the lack of correlation between the classification and distillation losses, which seems to be the Achilles' heel of this training methodology. One of the core hypotheses of the Knowledge Distillation hypothesis (section 3.4.1) is that reducing the distillation loss would invariably lead to models with higher classification accuracy. However, this does not appear to be true so long as the distillation loss assumes finite non-zero values.

### 3.5.3 Self-distillation using random dataset

In this set of experiments, we attempt to transfer the parameters of a layer to an untrained copy of itself using knowledge distillation. We spawn two copies of a layer, assign one the role of a teacher and another the role of a student, and train the student using a distillation loss on a dataset generated randomly. We conduct this experiment on Self-Attention layers and small Dense networks, and compare the distillation loss attained at the end of 30 epochs for both sub-networks. The training and validation losses, in the `log` scale, are plotted in figure 3.16.

We see that the distillation loss attained by the Dense networks are orders of magnitude lower than that attained by the Self-Attention operation. This is seen despite some of the dense networks tested having more trainable parameters than the self-attention student model (table 3.2). This suggests that the nature of the operation being trained also impacts the effectiveness of the distillation scheme. That is, since the class of functions that can be approximated by dense networks is limited in comparison to those that can be approximated by self-attention, owing to the versatility of the attention mechanism, the distillation task is possibly simpler. Thus, while the number of trainable parameters may function as a suitable proxy of the complexity of the

distillation task when comparing two networks employing the same set of functions, it may not extend when comparing networks employing different functions.

| Model | # params (1000s) | # params (w.r.t. Attention) |
|---|---|---|
| Self Attention | 148.224 | 1 |
| Dense-1L-192 | 37.056 | 0.25 |
| Dense-5L-192 | 185.28 | 1.25 |
| Dense-5L-768 | 2067.648 | 13.95 |

Table 3.2: **Number of parameters of networks tested in Self-Distillation Experiments**

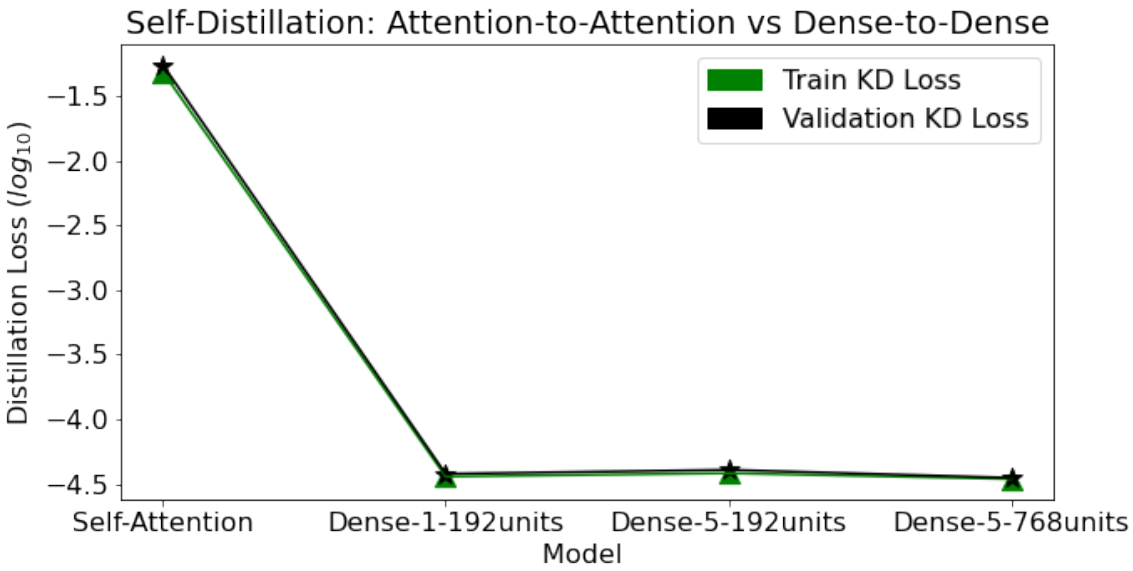
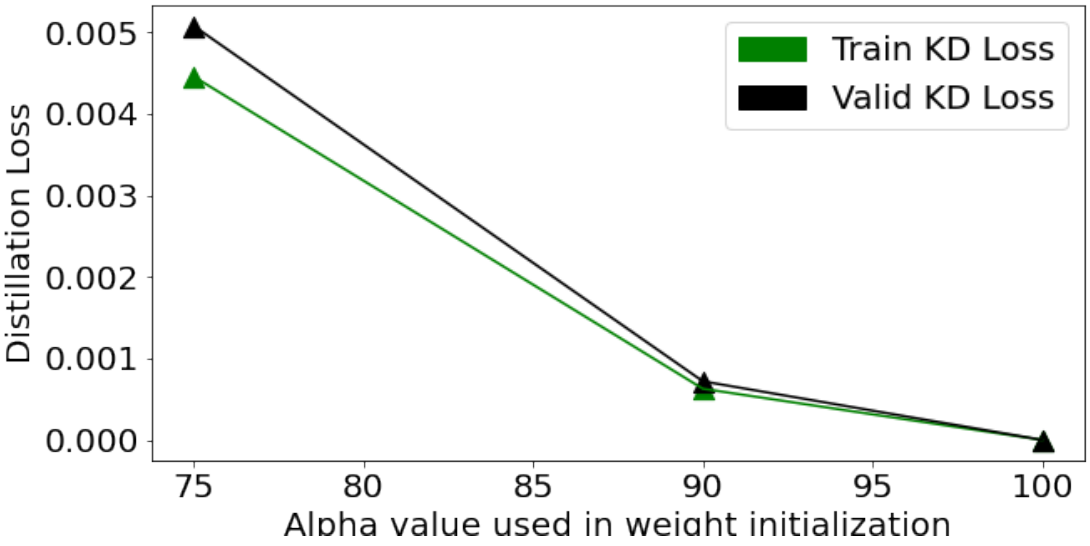
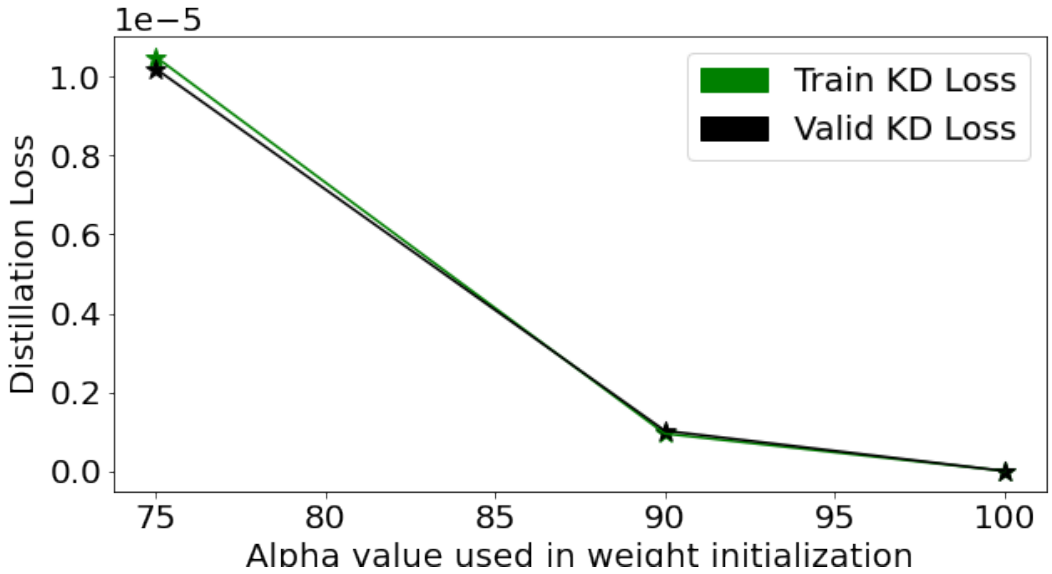



Figure 3.16: **Self-Distillation: Distil from a subnetwork to an untrained copy of itself**

So far, the student models were initialized with a random set of weights that were uncorrelated with the teacher's weights. In the following experiment, we evaluate if initializing the student with weights that are correlated with the teacher's trained weights simplifies the distillation process. Given weights $\theta_{rand}$ that are uncorrelated with the teacher's weights $\theta_T$, we pick the weights corresponding to a point on the line connecting the two sets of points and assign it to the student




(a) **Self-Distillation on Attention: initialize student with teacher's weights + random noise**




(b) **Self-Distillation on Conv2D: initialize student with teacher's weights + random noise**

Figure 3.17: **Initialize Student's Weights to aid the Self-Distillation process; the proximity of student's initial weights to the teacher's is controlled by $\alpha$**

model [34]. Thus, the student's weights $\theta_S$ equals $(1-\alpha)\cdot\theta_{rand} + \alpha\cdot\theta_T$ for $\alpha \in [0,1]$. The value of $\alpha$ is used to vary the proximity of the student's weights to the teacher's weights. An $\alpha$ value of 1 corresponds to initializing the student model with the trained weights of the teacher, while a value of 0 corresponds to initializing the student with random uncorrelated weights.

In figure 3.17, we compare the distillation losses of Attention and Conv2D student networks when trained through self-distillation, with the weight initialization scheme mentioned above. As expected, for both networks, initializing the student with the teacher's weights ($\alpha = 100\%$) results in a distillation loss of 0 (across all epochs). Reducing the $\alpha$ value causes the distillation loss to increase accordingly. However, it is worth noting that the distillation loss of the Conv2D student model is significantly lower than the attention model. This is in accordance with the result from fig. 3.16- performing distillation on "simpler" (less versatile) networks is perhaps easier than on more versatile networks.

Can student attention layers trained through self-distillation replace the teacher layers in the parent ViT model? We pick attention layer #6 of the parent ViT as the teacher layer, initialize the student's weights as a mix of the teacher's weights and random weights, and substitute the student in the teacher's place in the parent ViT post-distillation. The distillation loss and the ViT accuracy on substitution for different values of $\alpha$ is plotted in figure 3.18.

When the student is initialized far away from the teacher's weights ($\alpha < 25\%$), the model optimizer struggles to reduce the loss between the teacher and the student. Consequently, the accuracy upon substituting the student layer into the ViT is low. However, when the student is initialized closer to the teacher's weights, the student is able to more or less match the accuracy of the parent ViT upon substitution. Further, we see some correlation between the distillation loss and the final model accuracy in this figure, unlike the trend in figure 3.12. The additional context provided by correlating the student's initial weights with the teacher's trained weights possibly helps provide some coupling between the distillation loss and the classification accuracy- the weight initialization perhaps incentivizes the model optimizer to reduce the distillation loss along a path that results in better classification accuracy. Unfortunately, since the weight initialization experiment can be carried out only when the teacher and the student networks are the same,

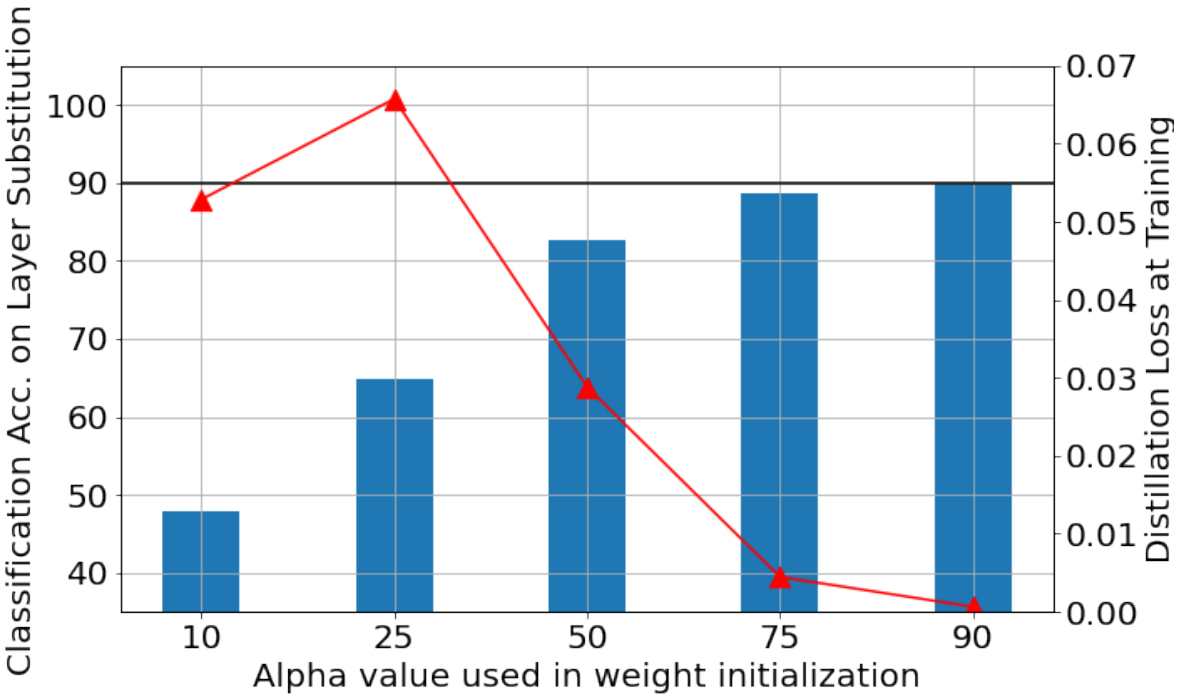



Figure 3.18: **Self-Distillation: Loss values for varying $\alpha$ (scatter plot) and Model Accuracy upon substituting the student layer back into the parent ViT**

we could not train other MetaFormer models using this technique, and thus could not validate if initializing MetaFormers with weights correlated to the parent transformer is an antidote to the pitfalls of the distillation methodology.

### 3.5.4 Tweaking the Token Mixer Stage Architecture

The token mixer stage architecture described in section 3.3 includes two optional layers that were proposed to aid the token mixer operations in the stage. The first layer, `clsMixer`, is a Dense layer that is devised to accumulate some information from the image tokens to the classifier (`cls`) token, without altering the image tokens. This layer, if used, is appended at the end of the token mixer stage. The second layer that was proposed is a `patch-embed` layer. This is a Conv2D layer added at the beginning of %the stage and is aimed at aiding the token mixer layers by accumulating some features between neighboring patches in the input image.

Figure 3.19 plots the accuracy of some MetaFormer models with and without a `clsMixer` layer in the stage. We see that when the replaced token mixer stage is followed by an attention layer from the original ViT, using a `clsMixer` is detrimental to the model's accuracy. This suggests

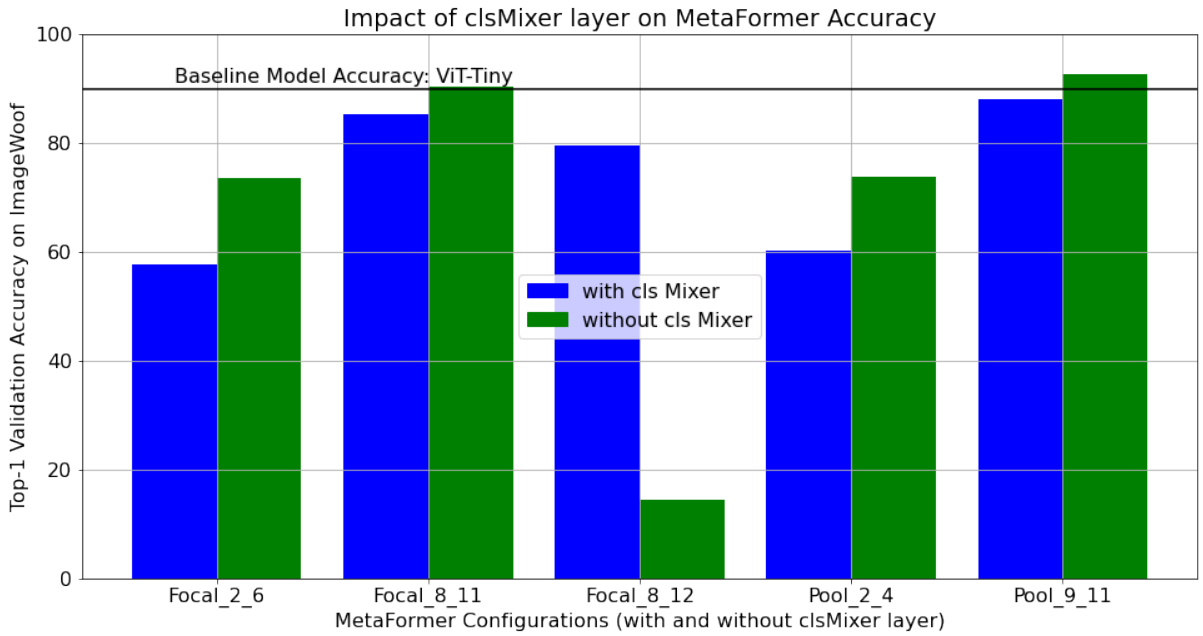



Figure 3.19: **Comparing the accuracies of MetaFormer models with and without a `clsMixer` layer at the end of the Token Mixer Stage**

that the attention layer that follows the token mixer stage is better at accumulating information from the image tokens to the `cls` token than the `clsMixer`. Using a `clsMixer` may cause some of the information in the `cls` token from the previous layers to get corrupted, leading to a drop in accuracy.

The configuration "Focal_8_12" corresponds to replacing layers 8-12 of the ViT with a Focal Modulation stage. Since the last attention layer is removed in this configuration, the features from the image tokens are not accumulated into the `cls` token, when no `clsMixer` is used. Naturally, this causes this model to have a poor accuracy, as the classification head uses the `cls` token to produce the output logits. Adding a `clsMixer` in the configuration allows for some information accumulation into the `cls` token pushing the accuracy to 79%. Thus, in the absence of an attention layer, the `clsMixer` can be used to do a decent job of gathering information from the image tokens. However, attention does a better job than a Dense layer at this, and hence it makes sense to have at least one attention layer following a Token Mixer stage.

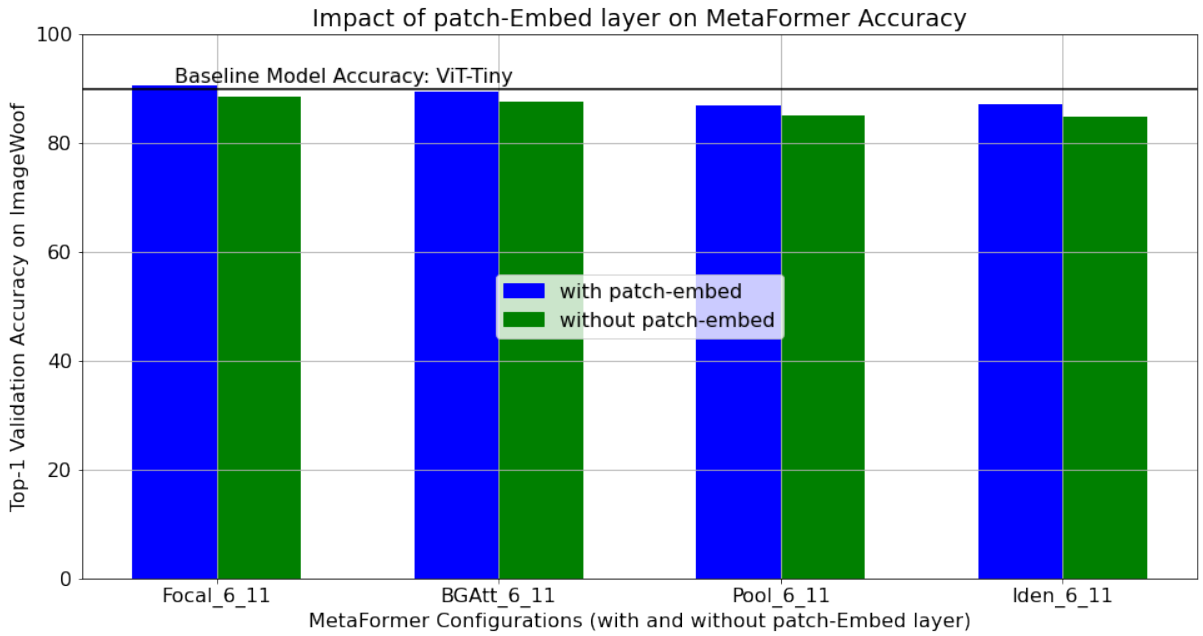



Figure 3.20: **Comparing the accuracies of MetaFormer models with and without a `patch-embed` layer at the beginning of the Token Mixer Stage**

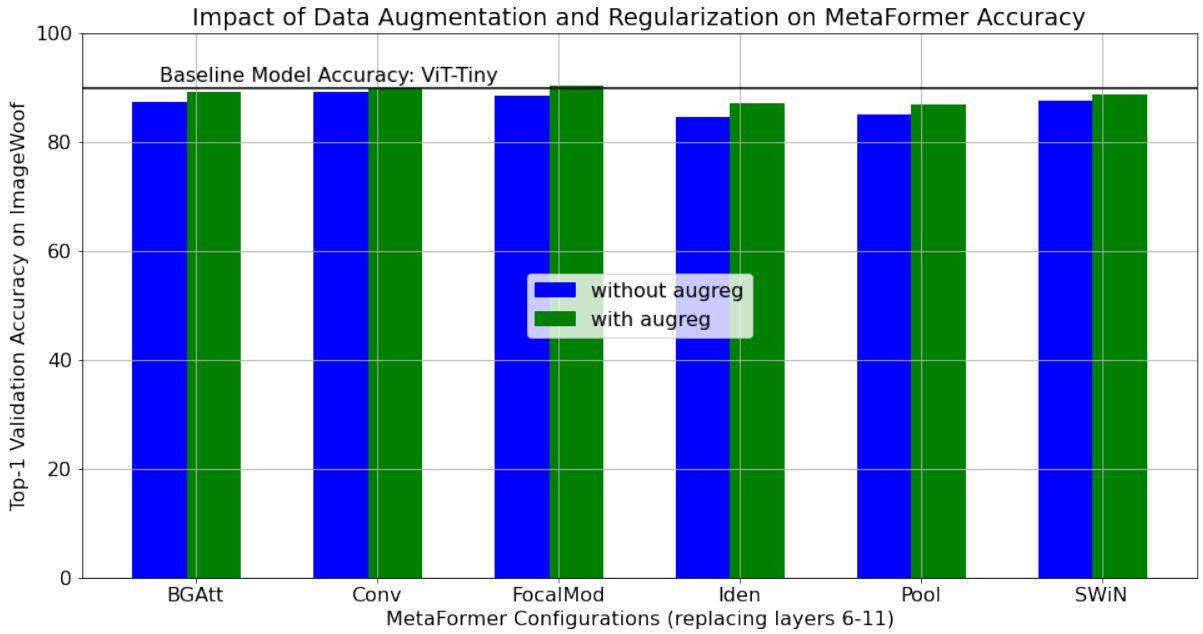
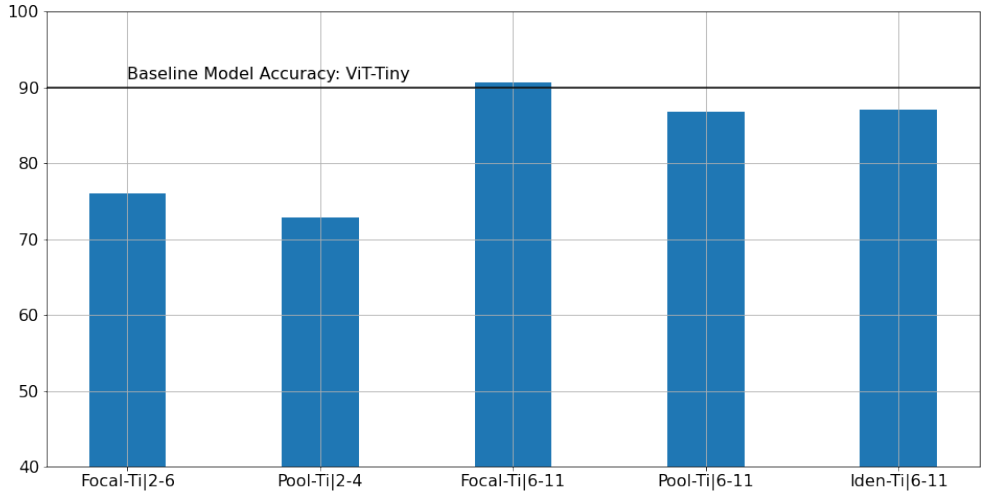
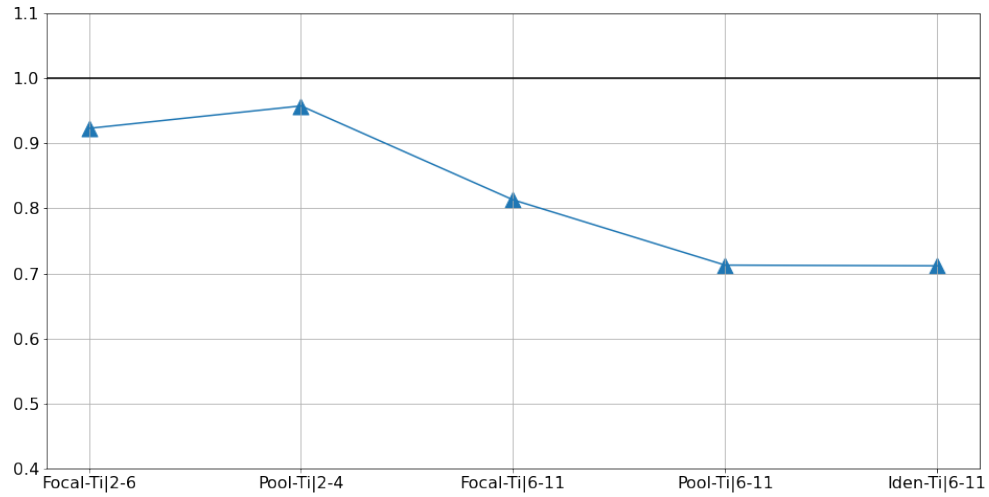


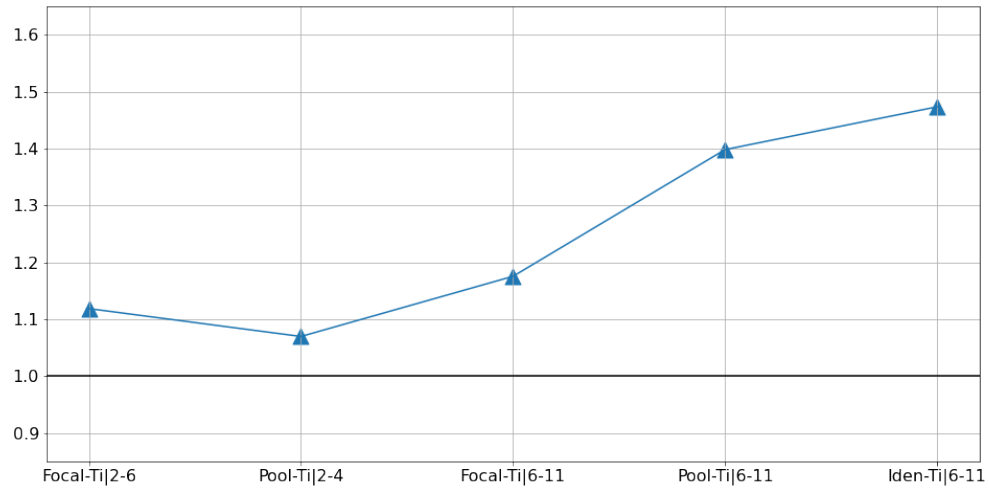

Figure 3.21: **Enhancing MetaFormer accuracies using MixUp Data Augmentation and Dropout Regularization- using AugReg leads to a 1-2% improvement in accuracy**




(a) **Accuracies of MetaFormer models picked for comparison**




(b) **Theoretical FLOPs per inference pass for the MetaFormer models relative to parent ViT**




(c) **Measured Latency on Tesla-T4 GPU for the MetaFormer models relative to the parent ViT**

Figure 3.22: **Accuracy, FLOPs and Latency of MetaFormer models relative to parent ViT-Tiny**

In figure 3.20, we compare the accuracies of models with and without a `patch-embed` layer. We see that using a `patch-embed` layer enhances the accuracy by around 1-2%. This is as expected, since the convolution operation in the `patch-embed` layer accumulates features from neighbouring patches, aiding the token mixer operations in the stage.

Finally, we evaluate the impact of using data augmentation and regularization in the training process. We use mixup data augmentation and dropout regularization, as suggested in [35]. Using augmentation and regularization (augreg), leads to an increase in accuracy by 1-2%, as is evident from figure 3.21. This is in line with the trends observed for Vision Transformers in general.

### 3.5.5 Complexity-Accuracy Trade-off

In this subsection, we analyze how the MetaFormer models fare on reducing the model complexity. We evaluate the theoretical FLOPs per inference pass for the models, and measure their average latency over 100 inference passes. The inference tests are conducted on an NVidia Tesla-T4 Tensor Core GPU (Turing Architecture) [36] using the XLA compiler [37].

Figure 3.22 depicts how some MetaFormer models built on the ViT-Tiny parent model fare on the accuracy, FLOPs and latency trade-off. The theoretical FLOPs plotted in fig. 3.22b are relative to the FLOPs of the ViT-Tiny model. The Pooling and Identity Former variants manage to reduce the FLOPs to $0.7\times$ of the parent model, while limiting the accuracy drop to 3%.

The measured latency of the models are plotted in figure 3.22c. The numbers reported here are mostly consistent with the numbers for theoretical FLOPs. A reduction in FLOPs to around $0.9\times$ of the parent for "Focal-Ti|2-6" and "Pool-Ti|2-4" variants lead to a speedup of around $1.1\times$ ($\approx \frac{1}{0.9}$). Similarly, the "Pool-Ti|6-11" and "Iden-Ti|6-11" variants see a speedup upwards of $1.4\times$ ($\approx \frac{1}{0.7}$).

Figure 3.23 visualizes where MetaFormer models sit on the Accuracy-FLOPs trade-off landscape. The parent ViT models (Tiny, Small and Base variants) are marked in blue, and their MetaFormer clones are marked in green. We see that while the Tiny MetaFormer models manage to match the accuracy of ViT-Tiny, the larger MetaFormers struggle to match their parent models.

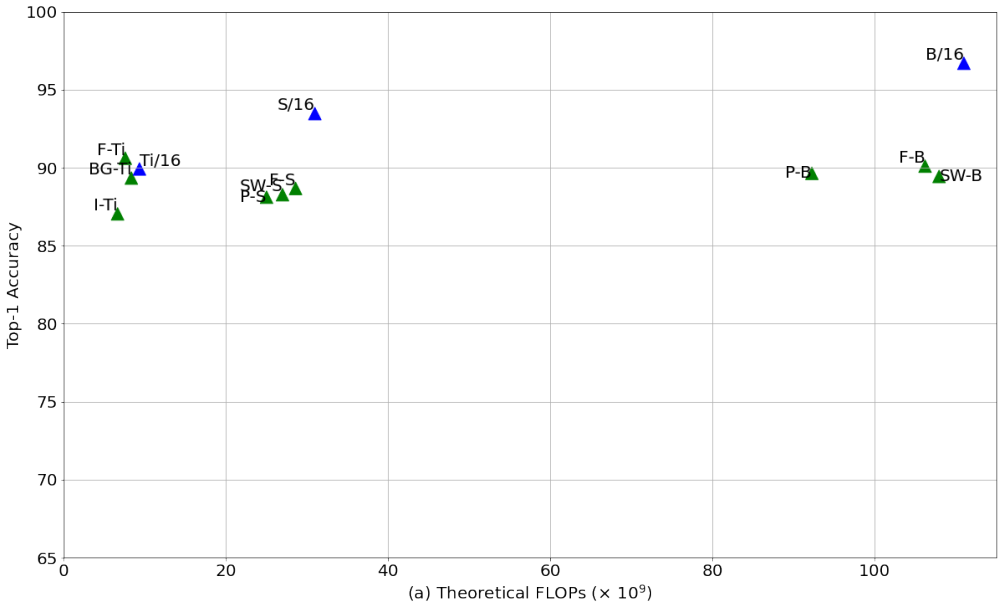



Figure 3.23: **Accuracy v/s FLOPs for MetaFormer Models and their parent ViTs**

### 3.6 CONCLUSION

The goal of this set of experiments was to build lightweight MetaFormer models from pre-trained Vision Transformers. The model training methodology was devised with the aim of distilling the parent transformer's learning (from its pre-training phase) into the MetaFormer clones without having to train each MetaFormer on a massive dataset. There were three key components in this training methodology:

- Train only the layers replaced- use weights trained through pre-training for all the remaining layers.
- Train the new layers in the MetaFormer on a classification loss objective- train the layers such that the model gets most of its output predictions correct.
- Distillation objective- train the new layers such that its output features resemble the output features of the attention layers that were replaced.

From the results, we see that the distillation objective *failed* in achieving its intended role. That is, while the optimizer did reduce the gap between the feature maps of the MetaFormer and the parent ViT, as defined by the distillation loss, the MetaFormer model failed to resemble the parent ViT. Thus, the optimizer was unable to transfer the parent ViT's learnings from its pre-training phase to the MetaFormer.

The improvements in accuracy made by the MetaFormer models were due to the classification loss objective. Interestingly, these experiments show that MetaFormers employing untrained token mixer layers can be made to achieve an accuracy of around 85%-90% on a dataset such as ImageWoof without resorting to pre-training on larger datasets. Standard training techniques employed while training transformers such as data augmentation and regularization, help improve the accuracy of the models further without overfitting on the training dataset. Plus, the distillation loss behaved like a regularization term, since it was included as an additive term in the loss function, restricting the models from overfitting.

The accuracy results for the "Small" and "Base" variants of MetaForfmers in figure 3.23 highlight the limit of this training methodology. The MetaFormer models are unable to attain an accuracy beyond 90% despite the parent ViTs scoring north of 93% and 96% respectively. This indicates that further improvements in accuracy cannot be got without performing pre-training on larger datasets. Although the token mixer stages of the MetaFormers do not use attention layers, the MLP layers in the stage start off with a poor inductive bias and would benefit from pre-training.

### 3.6.1 Why did Knowledge Distillation fail?

Now that we understand the role played by the classification objective and its gaps in building MetaFormer models, we move on to understand why Knowledge Distillation failed in its objective. First, we see that the optimization task at hand in the distillation objective is more difficult than the optimization task for classification [34]. L. Beyer, et. al., [32] suggest training models over a large number of epochs (of the order of 1000s) in order to circumvent this issue and attain low distillation losses. This however would lead to long training schedules, hurting the prospects of building MetaFormer models quickly at a low cost.

The knowledge distillation hypothesis was based on three assumptions. To recap, these were:

1. *Attention layers in a ViT can be replaced by some simpler token mixing operations without a considerable drop in accuracy.* This assumption is validated by the fact that some MetaFormer models outperform the parent transformer in accuracy.
2. *Attaining a low distillation loss on the dataset implies the student and teacher models approximate similar functions.* We see that the distillation loss between the student and teacher models is consistent post-training, regardless of the input images or the dataset it was trained on. In this sense the functions approximated by the two models are similar. However, the distillation loss isn't low enough for the student model to be considered as a functional replica of the teacher model.
3. *Reducing the distillation loss between the student and teacher models invariably leads to higher classification accuracy on the dataset.* This isn't true, since we see in figure 3.12 examples of models with low distillation loss and low classification accuracy, and high distillation loss and high classification accuracy. The underlying reason behind this is that the classification and distillation losses were uncoupled. The feature matching is performed in a vector space with a large dimension, and attaining a distillation MSE loss of $\epsilon$ here implies the student model is at a point on a sphere of radius $\epsilon$ that is centered at the teacher model. Now, all the points on this sphere need not correspond to good classification accuracies on the dataset. The trick here is to communicate to the model optimizer to tread along a path that improved classification accuracy while reducing the distance between the two models. One way to achieve this would have been to devise the two loss functions such that they were correlated. This is an open problem for us at the moment and is a possible direction to pursue in the future.

# CHAPTER 4

# SPARSITY EXPERIMENTS ON LINEAR LAYERS

As observed in section 2.2, a significant chunk of a Vision Transformer's complexity comes from the dense linear projection and MLP layers. These dense layers are often over-parameterized mappings from one latent feature space to another within the transformer. During model training, it helps to have such over-parameterized function mappings as constituent layers since it aids in making the architecture more versatile. However, once trained, these layers can be skimmed down, by removing redundant parameters, to just the required amount of sophistication.

The dense layers within a ViT are likely to be sparse due to their nature to be over-parameterized. This means that a substantial number of weights within each layer are zero or have an absolute value very close to 0. This presents an opportunity to skip the computations with the 0 elements in the weights. In addition to reducing the number of FLOPs required for a forward pass, this also leads to fewer memory accesses since the 0 elements in the weight matrices are not required to be fetched.

A trained network is *sparsified* through a series of steps. First, the layers undergo pruning. In this step, the elements in the weights that are 0 or close to 0 are rounded off to 0, and the remaining non-zero elements are left as is. Then, the layers go through a few iterations of re-training. During this step, the zeroed-out weights are frozen at 0 while the non-zero elements are set to be trainable. This re-training step is required to recover any degradation in the layer's learning arising from pruning. Following this, the weight matrices are compressed to a sparse format that stores the non-zero elements along with some metadata indices. The metadata indices contain information about the locations of the non-zero elements; these are used during inference for selecting the input elements that are multiplied with the non-zero elements.

Naturally occurring sparsity patterns in the weights tend to be irregular. Irregular sparsity patterns are challenging to speed up since they are not cache or SIMD-execution friendly. In contrast, if the sparsity pattern is regular and known ahead of time, it is easier to achieve a speed-up in

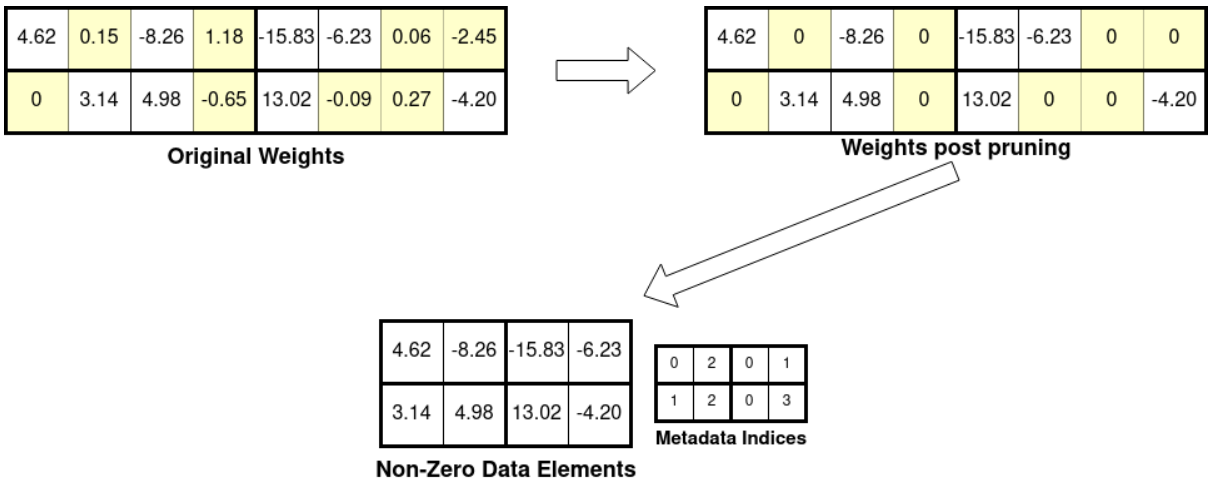



Figure 4.1: **Illustration of pruning data to the 2:4 sparse format**

computations by making modifications to the hardware unit and/or by writing code that optimizes cache latency and hardware utilization. In this chapter, our focus is on the 2:4 sparsity pattern that is supported by the latest generations of NVidia GPUs, starting with the Ampere architecture [38; 39].

### 4.1 2:4 STRUCTURED SPARSITY

The 2:4 structured sparsity pattern constrains 2 elements out of every 4 contiguous elements in a row to be 0. This ensures that the matrix is locally sparse while maintaining 50% sparsity globally. This simplifies the sparse storage format for the weights. The metadata indices now have to only store the index of the non-zero elements within the block of 4 elements in the row, which requires only 2 bits per non-zero element in the block. An example of a matrix pruned to the 2:4 sparsity pattern is shown in figure 4.1. Further, ensuring the matrix is locally sparse allows the hardware unit to speed up Sparse Matrix Multiplication with minimal hardware overhead.

The NVidia Ampere GPU architecture has a Tensor Core block in every streaming multiprocessor (SM) for accelerated Matrix-Matrix Multiplication (GEMM) operations. Tensor Cores are systolic array based hardware accelerators that are designed to handle dense matrices. In the Ampere architecture (and subsequent GPU architectures), the Tensor Cores are modified to additionally handle 2:4 sparse weight matrices. The non-zero data elements from the weight matrices are propagated to the Tensor Core, while the input activations are routed through a multiplexer (figure

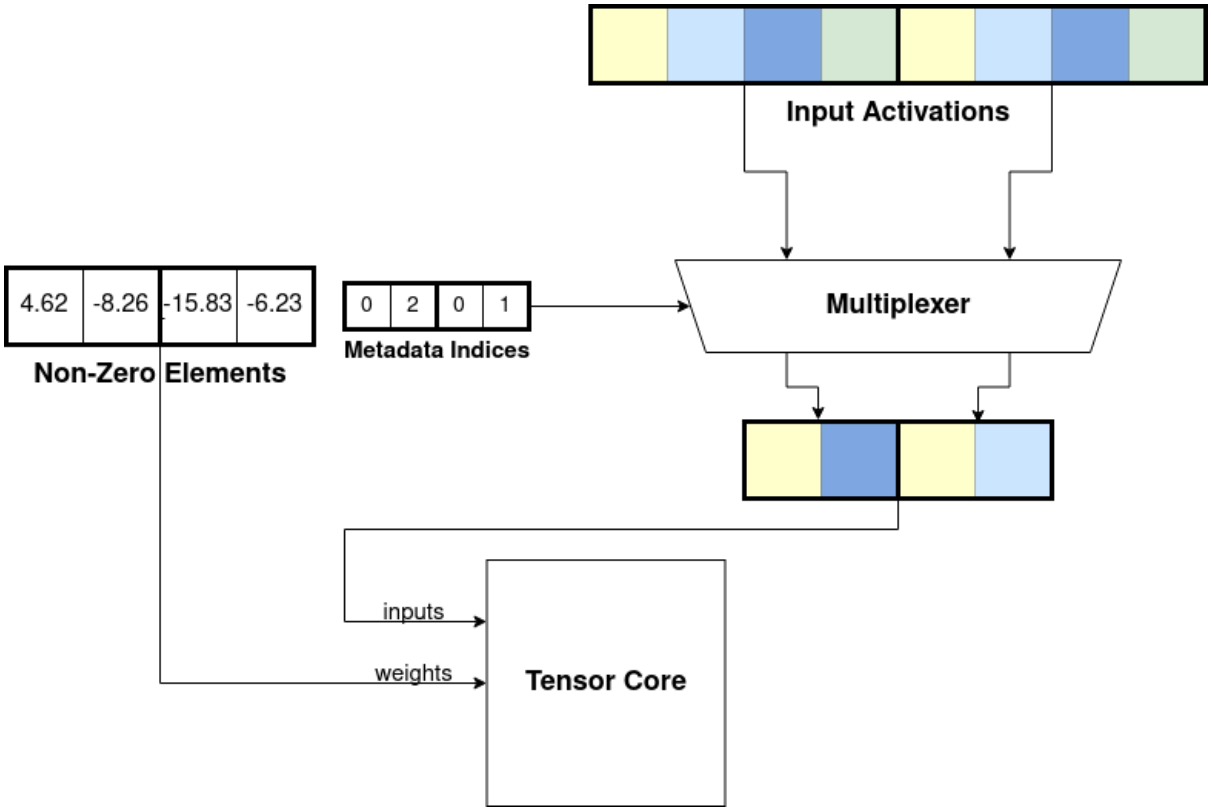



Figure 4.2: **Modifications made to Tensor Core Hardware for supporting 2:4 sparse weights**

4.2). The metadata indices of the weights are used as select bits of the multiplexer to pick the elements from the input matrix that are multiplied with the non-zero weight elements. The 50% sparsity in the weights allows the GPU to populate the Tensor Cores with input activations that are $2\times$ larger than the inputs supported for Dense-Dense matrix multiplications. Hence, 2:4 sparsity enables a theoretical speedup of $2\times$ without making major overhauls to the Tensor Core hardware. The peak FLOPS rates of Ampere GPUs with and without 2:4 sparsity enabled is reported in table 4.1.

| GPU | Tensor Core Peak FLOPS (FP16) | Sparse Tensor Core Peak FLOPS (FP16; 2:4 sparse) |
|---|---|---|
| A10 | 125 TFLOPS | 250 TFLOPS |
| A30 | 165 TFLOPS | 330 TFLOPS |
| A40 | 150 TFLOPS | 300 TFLOPS |
| A100 | 312 TFLOPS | 624 TFLOPS |

Table 4.1: **Peak Tensor Core FLOPS rates of NVidia Ampere GPUs**

### 4.1.1 Software Support for 2:4 Sparsity

Software libraries for pruning trained Neural Network models to 2:4 sparse format, and re-training them post-pruning are available in PyTorch [40] and TensorFlow. These libraries follow a process flow similar to the one described at the beginning of this chapter. These libraries additionally perform column permutations, if necessary, before pruning in order to distribute the low-magnitude elements along the rows. This reduces the chance of relatively larger values from being pruned away due to them being the smallest within a block of 4 elements in the row [41]. Published works from NVidia [39; 41] have demonstrated that pruning Convolution and Dense layers in Neural Networks to 2:4 sparsity does not significantly hurt the model accuracy; some models also outperform the original model post-sparsification as the exercise helps them generalize the data better.

During inference, the sparsity feature of the Tensor Cores can be enabled using NVidia's TensorRT compiler [42]. The compiler handles converting the sparse weights to the 2:4 format, and invoking appropriate CUDA kernels for performing sparse GEMM computations on the Tensor Cores.

## 4.2 ANALYSIS OF LAYERS IN THE VISION TRANSFORMER

The peak speedup that can be got from enabling 2:4 sparsity on the Ampere Tensor Cores is $2\times$, over the dense compute mode. Under what circumstances is a model likely to achieve this speedup? And, which layers in the transformer should be sparsified for optimizing the speedup-accuracy trade-off? Layers that are unlikely to provide a speedup post-sparsification are skipped to minimize the degradation in accuracy. We analyze the dense layers in the transformer using the roofline model [43].

### 4.2.1 Roofline Model

The roofline model is a simple and intuitive performance model that relates the compute performance to DRAM bandwidth. This model can be used to estimate when programs are likely to hit the peak FLOPS rates of a processor. While it does not account for cache latency and bandwidth, and synchronization costs for parallel processors, it provides bounds to a first-order

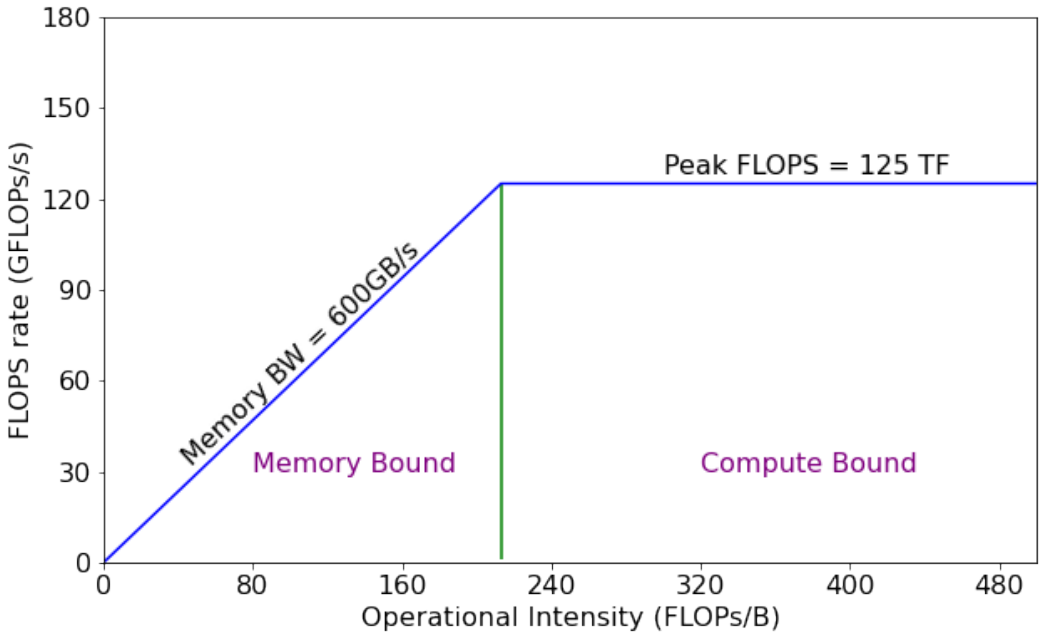



Figure 4.3: **Roofline Model Plot for Ampere A10 GPU**

for the maximum FLOPS rates that can be achieved.

The roofline model plots the maximum attainable FLOPS rate versus the operational intensity of a program. The operational intensity is defined as the number of compute operations (typically FLOPs) that are performed for every byte of data accessed from RAM. Data accesses from DRAM are limited by the DRAM bandwidth and often becomes the bottleneck of programs with low operational intensity. Thus, such programs operate in the Memory Bound regime, where the peak FLOPS rate that can be achieved is linearly dependent on the operational intensity. For a processor with a DRAM bandwidth $BW$, the maximum FLOPS rate achieved by a program with an operational intensity $x$ is $BW \times x$. At high enough operational intensities, however, the execution is limited by the peak FLOPS rate of the processor. Such programs, operating in the compute-bound regime, perform enough operations for every byte accessed that the DRAM latency is effectively hidden. The knee point of this plot is the point where the program is both compute and memory bound. This occurs when the operational intensity $x = \frac{FLOPS_{peak}}{BW}$. The roofline model for the Ampere A10 GPU's specifications is plotted in figure 4.3.

### 4.2.2 When to prune a layer?

The speedup of 2:4 sparse models comes from the fact that the Tensor Cores can now be packed with input activations that are $2\times$ larger than when performing dense computations. The Tensor Cores need only perform half the number of operations as the original model to complete the matrix multiplication task for a given input activation matrix. Thus, when operating at the peak FLOPS rate, the Tensor Cores complete computations with a given input matrix at twice the speed when operating on sparse weights, effectively attaining a $2\times$ speedup in the FLOPS rate over the dense compute mode. However, this speedup is attained only when the sparse model operates in the compute-bound regime. If the sparse model falls in the memory-bound regime, the maximum attainable FLOPS rate is lower than the peak FLOPS rate and is dictated by the memory bandwidth and the operational intensity of the program.

For a linear layer, let $I_B$ and $W_B$ be the number of bytes occupied by the input activation and (dense) weight matrices. The number of DRAM accesses required is approximately equal to $I_B + W_B$, assuming a word once accessed from DRAM is available in the cache throughout its lifetime in the program. Let the number of floating point operations to perform a forward pass be $F$. Then, to a first-order approximation, the operational intensity for a dense implementation of the linear layer is:

$$OpIntensity = \frac{F}{I_B + W_B} = OI_{dense}$$

Pruning the linear layer to the 2:4 sparse format leads to the following changes. The number of float operations required reduces to $\frac{F}{2}$ and the number of DRAM accesses required is approximately $\frac{W_B}{2} + \frac{W_B}{8}$ (the number of bytes occupied by the metadata indices is one-fourth the number of bytes occupied by the non-zero elements since each index uses 2 bits). The input activations are streamed to the Tensor Cores, and the entire matrix is fetched from the DRAM- the multiplexing step happens right at the tensor core. Therefore, the number of DRAM accesses for fetching the input activations remains the same at $I_B$. The operational intensity of the sparse implementation of the linear layer is thus:

$$OpIntensity = \frac{\frac{F}{2}}{I_B + \frac{W_B}{2} + \frac{W_B}{8}} = OI_{sparse}$$

| Dense Model Regime | Sparse Model Regime | FLOPS rate of Sparse Model (rel. Dense) | Theoretical Speedup |
|---|---|---|---|
| Memory-Bound | Memory-Bound | $0.5\times$ | $1\times$ |
| Compute-Bound | Memory-Bound | $\in (0.5, 1)\times$ | $\in (1, 2)\times$ |
| Compute-Bound | Compute-Bound | $1\times$ | $2\times$ |

Table 4.2: **Theoretical Speedup of Sparse Model w.r.t Dense Model based on the region of operation**

Typically, $I_B >> W_B$ since the input activations have multiple channels that share the same set of weights. This means that

$$OI_{sparse} \approx \frac{\frac{F}{2}}{I_B} \approx \frac{OI_{dense}}{2}$$

Due to the monotonous non-decreasing nature of the roofline plot, the above equation mandates that the FLOPS rate attained by the sparse model is less than or equal to the FLOPS rate of the dense model. The FLOPS rates are equal only if both the models operate in the compute-bound regime.

Table 4.2 summarizes the theoretical speedup of the sparse model with respect to the dense model for different operating regions of the models. Since the operational intensity of the sparse model is approximately half that of the dense model, the FLOPS rate attained by the sparse model in this condition is half of that of the dense model. This leads to a speedup of $1\times$ as the number of operations performed by the sparse model is also half that of the dense model. When the dense model is compute-bound and the sparse model is memory-bound, the relative FLOPS rate of the sparse model lies in the range $(0.5, 1)$, resulting in a speedup in the range $(1, 2)$. Finally, when both models are compute-bound, they operate at the peak FLOPS rate, and the sparse model attains a speedup of 2. It is worth noting, however, that even when both models are compute-bound it is difficult to attain the theoretical speedup of $2\times$ due to all the second-order cache and synchronization delays ignored by this model.

### 4.2.3 Region of Operation of Linear Layers in a ViT

We estimate the number of FLOPs and memory accesses in bytes per forward pass for linear layers in a vision transformer, and compare the theoretical operational intensity against the bandwidth and peak FLOPS rate metrics of the Ampere A10 GPU. Within a layer in the encoder,

| Model Variant | Layer | Dense Implementation Regime | Sparse Implementation Regime |
|---|---|---|---|
| **Tiny-16** | MLP-Dense0 | compute | compute |
| | MLP-Dense1 | *memory* | *memory* |
| | Attn-InProject | compute | compute |
| | Attn-OutProject | *memory* | *memory* |
| | Outro-MLP | *memory* | *memory* |
| **Small-16** | MLP-Dense0 | compute | compute |
| | MLP-Dense1 | compute | *memory* |
| | Attn-InProject | compute | compute |
| | Attn-OutProject | compute | *memory* |
| | Outro-MLP | *memory* | *memory* |
| **Base-16** | MLP-Dense0 | compute | compute |
| | MLP-Dense1 | compute | compute |
| | Attn-InProject | compute | compute |
| | Attn-OutProject | compute | compute |
| | Outro-MLP | *memory* | *memory* |

Table 4.3: **Operating Regimes of Linear Layers in different Vision Transformers, on an Ampere A10 GPU**

the attention block contains a qkv-projection layer ("Attn-InProject") and an output projection layer ("Attn-OutProject"). The MLP block consists of two dense layers, ("MLP-Dense0" and "MLP-Dense1"). All the layers in the encoder are identical with regard to the number of parameters and float operations; hence we can expect the impact of pruning on the throughput to be independent of the layer where it is performed. Table 4.3 lists the regimes of operation for these layers in the tiny, small and base variants of a ViT. The outro-MLP in the classifier head of the network is also considered in this analysis, however, this layer turns out to have too small an operational intensity for all variants of the transformer and hence is not suitable for pruning.

For the tiny variant, we see that the "MLP-Dense0" and "Attn-InProject" layers are compute-bound under both implementation scenarios. In contrast, the "MLP-Dense1" and "Attn-OutProject" layers are memory-bound for both dense and sparse implementations, implying that pruning would not lead to a speedup for these layers. This difference arises from the fact that the input activations to the former set of layers ("MLP-Dense0" and "Attn-InProject") are smaller than that for the latter set of layers. As the variants get larger (small and base), the number of float operations performed grow faster than the size of the input activations. Therefore, the "MLP-Dense1" and "Attn-OutProject" layers move towards the compute-bound regime as the variants get larger.

### 4.3 ACCURACY RESULTS

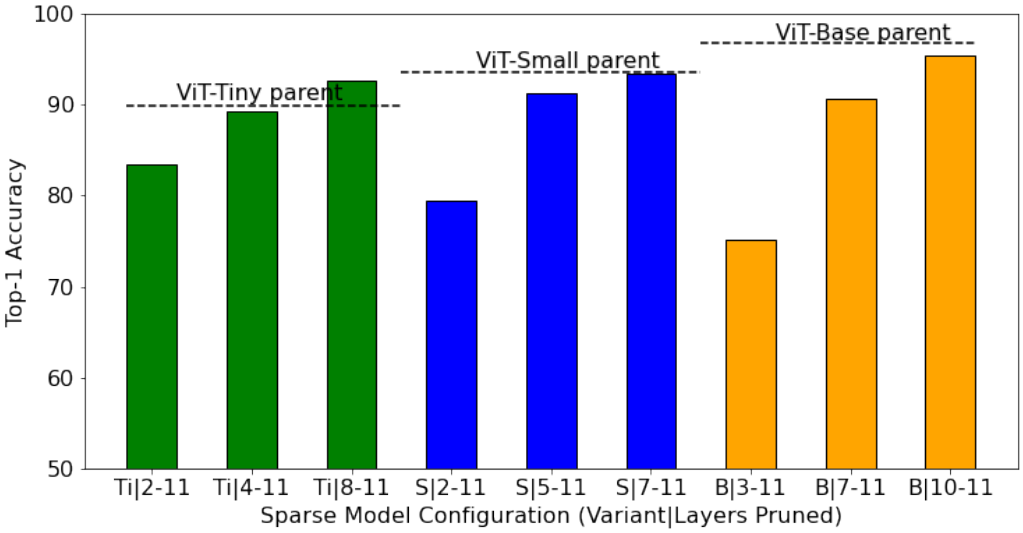
Figure 4.4 compares some sparse ViT model configurations against their corresponding parent models on accuracy, theoretical FLOPs and number of trainable parameters. In general we see that the ViT linear layers are fairly immune towards 2:4 sparsity pruning. For instance, the ViT-Tiny variant maintains an accuracy of 89% (a drop of < 1%) despite pruning linear layers in the last 8 encoder layers (config. "Ti|4-11"). Similarly, the "S|7-11" configuration preserves the accuracy of the ViT-Small variant while using sparse linear projections in the last 5 layers of the encoder network (fig. 4.4a). The "Ti|8-11" configuration prunes layers 8-11 in the transformer and manages to outperform the parent model- this is in line with the trends reported by NVidia in their whitepapers [39; 41]. Pruning layers to 2:4 sparsity help reduce overfitting in over-parameterized networks, which can lead to the models learning better generalization in the process. However, as was observed in the experiments from the previous chapter, preserving the accuracies of larger ViT models is challenging. The sparse configurations corresponding to the base variant perform rather poorly compared to the parent model- pruning just the last encoder layer alone leads to an accuracy drop $> 1\%$. We reckon this can be addressed by re-training the sparse models on larger datasets and by using more sophisticated data augmentation techniques.

The reduction in theoretical FLOPs per forward pass, relative to the respective parent models, is plotted in figure 4.4b. "Ti|4-11" and "S|7-11" position themselves as the best-performing models as they reduce the FLOPs by around 20% while preserving the accuracy. The reduction in the number of trainable parameters is plotted in figure 4.4c.

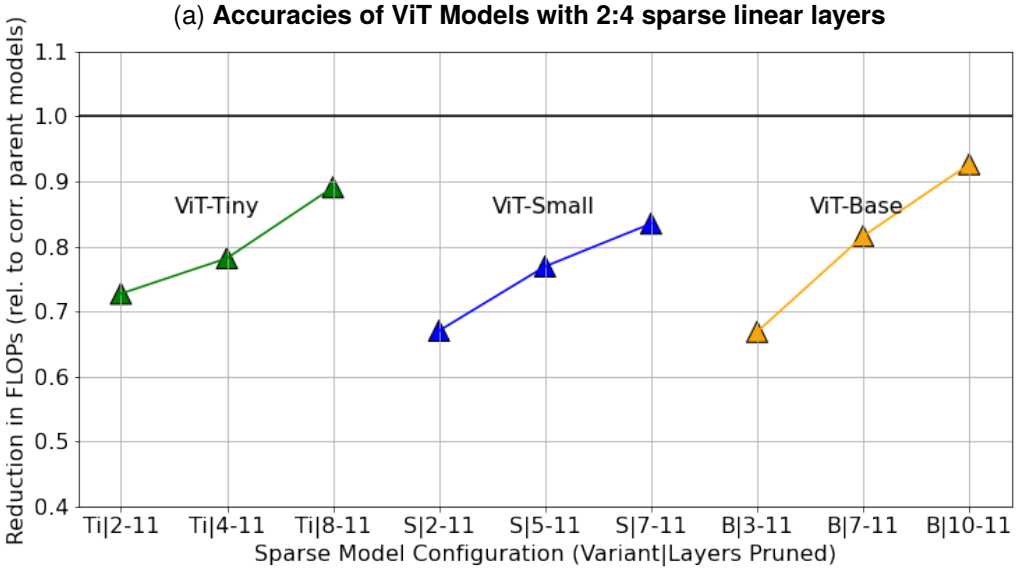
### 4.4 CONCLUSION

Applying 2:4 sparsity to the linear projection layers of the transformer targets to bring down the number of float operations required for a forward pass in order to reduce the computational cost of the model. The results suggest that this optimization method is quite effective at preserving the accuracy of the model. In fact, some of the sparse model configurations outperform the original model due to better generalization by the model post-pruning.

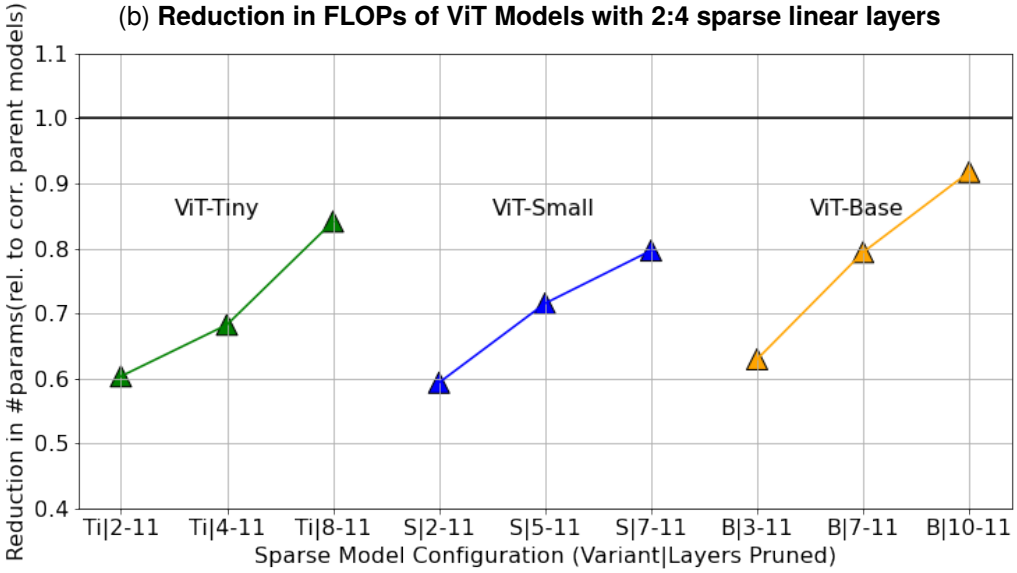
Sparsity, however, is not free lunch- it results in a speedup only under certain conditions where the




(a) **Accuracies of ViT Models with 2:4 sparse linear layers**




(b) **Reduction in FLOPs of ViT Models with 2:4 sparse linear layers**




(c) **Reduction in #params of ViT Models with 2:4 sparse linear layers**

Figure 4.4: **Accuracy Results of 2:4 sparse ViT models**

compute units are populated with sufficient workload. We analyze which layers of the transformer are best suited for a speedup upon sparsification by using the roofline performance model. Finally, this is an optimization technique that is orthogonal to the experiments in chapter 3. While the metaformer models replaced the attention operation with simpler token mixers, sparsity can be applied to all linear projection and convolutional layers in the network. In metaformer models, sparsity may be applied to layers within the token mixer, in addition to the MLP layers. Thus, sparsity can be thought of as a second step of optimization that can be applied on top of model architecture or function-level modifications. This also indicates that applying sparsity on metaformer models would lead to additive reductions in model complexity, which would invariably also lead to additive degradations in accuracy.

# REFERENCES


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” *arXiv e-prints*, p. arXiv:1706.03762, Jun. 2017.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language Models are Few-Shot Learners,” *arXiv e-prints*, p. arXiv:2005.14165, May 2020.

[3] OpenAI, “GPT-4 Technical Report,” *arXiv e-prints*, p. arXiv:2303.08774, Mar. 2023.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv e-prints*, p. arXiv:1810.04805, Oct. 2018.

[5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. Sankaranarayana Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, “PaLM: Scaling Language Modeling with Pathways,” *arXiv e-prints*, p. arXiv:2204.02311, Apr. 2022.

[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,” *arXiv e-prints*, p. arXiv:2010.11929, Oct. 2020.

[7] P. W. Code. (2023) State-of-the-Art Image Classification on ImageNet. [Online]. Available: https://paperswithcode.com/sota/image-classification-on-imagenet

[8] 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*, 2009, pp. 248–255.

[9] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 843–852.

[10] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, “Scaling Vision Transformers,” *arXiv*

*e-prints*, p. arXiv:2106.04560, Jun. 2021.

[11] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," *arXiv e-prints*, p. arXiv:2103.14030, Mar. 2021.

[12] Z. Tu, H. Talebi, H. Zhang, F. Yang, P. Milanfar, A. Bovik, and Y. Li, "MaxViT: Multi-Axis Vision Transformer," *arXiv e-prints*, p. arXiv:2204.01697, Apr. 2022.

[13] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin Transformer V2: Scaling Up Capacity and Resolution," *arXiv e-prints*, p. arXiv:2111.09883, Nov. 2021.

[14] J. Yang, C. Li, X. Dai, L. Yuan, and J. Gao, "Focal Modulation Networks," *arXiv e-prints*, p. arXiv:2203.11926, Mar. 2022.

[15] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-Mixer: An all-MLP Architecture for Vision," *arXiv e-prints*, p. arXiv:2105.01601, May 2021.

[16] W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng, and S. Yan, "MetaFormer Is Actually What You Need for Vision," *arXiv e-prints*, p. arXiv:2111.11418, Nov. 2021.

[17] W. Yu, C. Si, P. Zhou, M. Luo, Y. Zhou, J. Feng, S. Yan, and X. Wang, "MetaFormer Baselines for Vision," *arXiv e-prints*, p. arXiv:2210.13452, Oct. 2022.

[18] L. Liu, Z. Qu, Z. Chen, F. Tu, Y. Ding, and Y. Xie, "Dynamic sparse attention for scalable transformer acceleration," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3165–3178, 2022.

[19] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: Detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 14–26. [Online]. Available: https://doi.org/10.1145/3503222.3507738

[20] Z. Li, S. Ghodrati, A. Yazdanbakhsh, H. Esmaeilzadeh, and M. Kang, "Accelerating Attention through Gradient-Based Learned Runtime Pruning," *arXiv e-prints*, p. arXiv:2204.03227, Apr. 2022.

[21] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, "DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification," *arXiv e-prints*, p. arXiv:2106.02034, Jun. 2021.

[22] NVidia. (2022) NVIDIA H100 Tensor Core GPU Architecture. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[23] J.-B. Cordonnier, A. Loukas, and M. Jaggi, “On the Relationship between Self-Attention and Convolutional Layers,” *arXiv e-prints*, p. arXiv:1911.03584, Nov. 2019.

[24] J. Howard, “Imagewoof.” [Online]. Available: https://github.com/fastai/imagenette/

[25] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *arXiv e-prints*, p. arXiv:1905.11946, May 2019.

[26] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv e-prints*, p. arXiv:1409.1556, Sep. 2014.

[27] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” *arXiv e-prints*, p. arXiv:1503.02531, Mar. 2015.

[28] S. Han, “MIT 6.S965, Lecture 10 - Knowledge Distillation.” [Online]. Available: https://youtu.be/IIqf-oUTHe0

[29] Z. Huang and N. Wang, “Like What You Like: Knowledge Distill via Neuron Selectivity Transfer,” *arXiv e-prints*, p. arXiv:1707.01219, Jul. 2017.

[30] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, “A kernel two-sample test,” *J. Mach. Learn. Res.*, vol. 13, p. 723–773, Mar. 2012.

[31] Y. Abu-Mostafa, “CS 156, Learning From Data, Caltech, Lecture 15- Kernel Methods.” [Online]. Available: https://youtu.be/XUj5JbQihlU

[32] L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil, and A. Kolesnikov, “Knowledge distillation: A good teacher is patient and consistent,” *arXiv e-prints*, p. arXiv:2106.05237, Jun. 2021.

[33] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond Empirical Risk Minimization,” *arXiv e-prints*, p. arXiv:1710.09412, Oct. 2017.

[34] S. Stanton, P. Izmailov, P. Kirichenko, A. A. Alemi, and A. G. Wilson, “Does Knowledge Distillation Really Work?” *arXiv e-prints*, p. arXiv:2106.05945, Jun. 2021.

[35] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, “How to train your ViT? Data, Augmentation, and Regularization in Vision Transformers,” *arXiv e-prints*, p. arXiv:2106.10270, Jun. 2021.

[36] NVidia. (2018) NVIDIA Turing GPU Architecture. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[37] Tensorflow. (2021) XLA: Optimizing Compiler for Machine Learning. [Online]. Available: https://www.tensorflow.org/xla

[38] NVidia. (2020) NVIDIA Ampere GA102 GPU Architecture. [Online].

Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

[39] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[40] NVidia. (2021) NVidia APEX Automatic Sparsity (ASP). [Online]. Available: https://github.com/NVIDIA/apex/tree/master/apex/contrib/sparsity

[41] J. Pool and C. Yu, "Channel permutations for N:M sparsity," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/6e8404c3b93a9527c8db241a1846599a-Paper.pdf

[42] NVidia. (2023) NVIDIA Deep Learning TensorRT Documentation . [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html

[43] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, apr 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785