FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization

Pavan Kumar Anasosalu Vasu† James Gabriel Jeff Zhu Oncel Tuzel Anurag Ranjan†

Apple

Abstract

The recent amalgamation (合并) of transformer and convolutional designs has led to steady improvements in accuracy and efficiency of the models. In this work, we introduce FastViT, a hybrid vision transformer architecture that obtains the state-of-the-art latency-accuracy trade-off. To this end, we introduce a novel token mixing operator, RepMixer, a building block of FastViT, that uses structural reparameterization to lower the memory access cost by removing skip-connections in the network. We further apply train-time overparametrization and large kernel convolutions to boost accuracy and empirically show that these choices have minimal effect on latency. We show that – our model is 3.5× faster than CMT, a recent state-of-the-art hybrid transformer architecture, 4.9× faster than EfficientNet, and 1.9× faster than ConvNeXt on a mobile device for the same accuracy on the ImageNet dataset. At similar latency, our model obtains 4.2% better Top-1 accuracy on ImageNet than MobileOne. Our model consistently outperforms competing architectures across several tasks – image classification, detection, segmentation and 3D mesh regression with significant improvement in latency on both a mobile device and a desktop GPU. Furthermore, our model is highly robust to out-of-distribution samples and corruptions, improving over competing robust models. Code and models are available at https://github.com/apple/ml-fastvit

1. Introduction

Vision Transformers [14] have achieved state-of-the-art performance on several tasks such as image classification, detection and segmentation [35]. However, these models have traditionally been computationally expensive. Recent works [68, 39, 42, 59, 29] have proposed methods to lower the compute and memory requirements of vision transformers. Recent hybrid architectures [42, 17, 10, 64] effectively combine the strengths of convolutional architectures and transformers to build architectures that are highly competitive on a wide range of computer vision tasks. Our goal is to build a model that achieves state-of-the-art latency-accuracy trade-off.

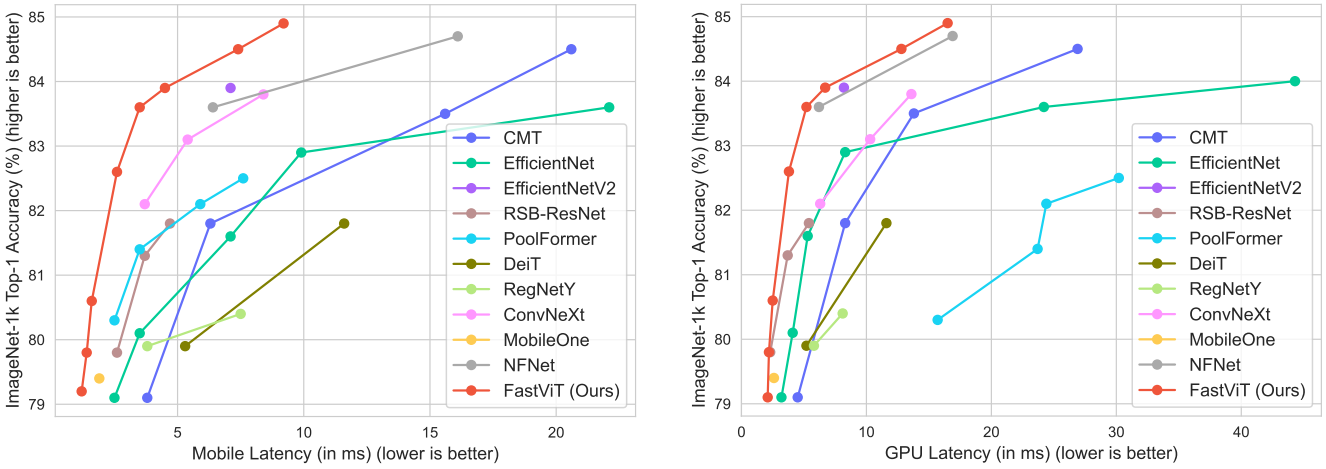
Recent vision and hybrid transformer models [53, 17, 43, 42] follow the Metaformer [67] architecture, which consists of a token mixer with a skip connection followed by Feed Forward Network (FFN) with another skip connection. These skip connections account for a significant overhead in latency due to increased memory access cost [13, 57]. To address this latency overhead, we introduce *RepMixer*, a fully reparameterizable token mixer that uses structural reparameterization to remove the skip-connections. The RepMixer block also uses depthwise convolutions for spatial mixing of information similar to ConvMixer [55]. However, the key difference is that our module can be reparameterized at inference to remove any branches.

To further improve on latency, FLOPs and parameter count, we replace all dense k×k convolutions with their factorized version, i.e. depthwise followed by pointwise convolutions. This is a common approach used by efficient architectures [26, 47, 25] to improve on efficiency metrics, but, naively using this approach hurts performance as seen in Table 1. In order to increase capacity of the these layers, we use linear train-time overparameterization as introduced in [13, 11, 12, 57, 18]. These additional branches are only introduced during training and are reparameterized at inference.

In addition, we use large kernel convolutions in our network. This is because, although self-attention based token mixing is highly effective to attain competitive accuracy, they are inefficient in terms of latency [39]. Therefore, we incorporate large kernel convolutions in Feed Forward Network (FFN) [14] layer and patch embedding layers. These changes have minimal impact on overall latency of the model while improving performance.

Thus, we introduce *FastViT* that is based on three key design principles– i) use of RepMixer block to remove skip connections, ii) use of linear train-time overparameteriza-

corresponding authors: {panasosaluvasu, anuragr}@apple.com



(a) (b)

Figure 1: (a) Accuracy vs. Mobile latency scaling curves of recent methods. The models are benchmarked on an iPhone 12 Pro following [57]. (b) Accuracy vs. GPU latency scaling curves of recent methods. For better readability only models with Top-1 accuracy better than 79% are plotted. See supplementary materials for more plots. Across both compute fabrics, our model has the best accuracy-latency tradeoff.

tion to improve accuracy, iii) use of large convolutional kernels to substitute self-attention layers in early stages.

FastViT achieves significant improvements in latency compared to other hybrid vision transformer architectures while maintaining accuracy on several tasks like – image classification, object detection, semantic segmentation and 3d hand mesh estimation. We perform a comprehensive analysis by deploying recent state-of-the-art architectures on an iPhone 12 Pro device and an NVIDIA RTX-2080Ti desktop GPU.

In Figure 1, we show that, at ImageNet Top-1 accuracy of 83.9%, our model is 4.9× faster than EfficientNet-B5 [50], 1.6× faster than EfficientNetV2-S [51], 3.5× faster than CMT-S [17] and 1.9× faster than ConvNeXt-B [36] on an iPhone 12 Pro mobile device. At ImageNet Top-1 accuracy of 84.9% our model is just as fast as NFNet-F1 [1] on a desktop GPU while being 66.7% smaller, using 50.1% less FLOPs and 42.8% faster on mobile device. At latency of 0.8ms on an iPhone 12 Pro mobile device, our model obtains 4.2% better Top-1 accuracy on ImageNet than MobileOne-S0. For object detection and instance segmentation on MS COCO using Mask-RCNN [20] head, our model attains comparable performance to CMT-S [17] while incurring 4.3× lower backbone latency. For semantic segmentation on ADE20K, our model improves over PoolFormer-M36 [67] by 5.2%, while incurring a 1.5× lower backbone latency on an iPhone 12 Pro mobile device. On 3D hand mesh estimation task, our model is 1.9× faster than MobileHand [16] and 2.8× faster than recent state-of-the-art MobRecon [4] when benchmarked on GPU.

In addition to accuracy metrics, we also study the robustness of our models to corruption and out-of-distribution samples which does not always correlate well with accuracy. For example, PVT [60] achieves highly competitive performance on ImageNet dataset, but has very poor robustness to corruption and out-of-distribution samples as reported in Mao et al. [38]. In real world applications, using a robust model in such a scenario can significantly improve user experience. We demonstrate the robustness of our architecture on popular benchmarks and show that our models are highly robust to corruption and out-of-distribution samples while being significantly faster than competing robust models. In summary, our contributions are as follows:

- We introduce *FastViT*, a hybrid vision transformer that uses structural reparameterization to obtain lower memory access cost and increased capacity, achieving state-of-the-art accuracy-latency trade-off.
- We show that our models are the fastest in terms of latency on two widely used platforms – mobile device and desktop GPU.
- We show that our models generalize to many tasks – image classification, object detection, semantics segmentation, and 3D hand mesh regression.
- We show that our models are robust to corruption and out-of-distribution samples and significantly faster than competing robust models.

2. Related Work

For the past decade, convolutional neural networks have been the standard architecture for vision models [21, 44, 66, 50, 51, 26, 47, 25, 36, 13, 57]. More recently, transformers have shown great success on computer vision tasks [14, 65, 53, 35, 60, 54, 43, 42]. Unlike convolutional

layers, the self-attention layers in vision transformers provide a global context by modeling long-range dependencies. Unfortunately, this global scope often comes at a high computational price [39]. Works like [42, 59, 29, 39] address ways to alleviate computational costs associated with self-attention layers. In our work, we explore an efficient alternative to self-attention layers for lower latency.

Hybrid Vision Transformers In order to design efficient networks while maintaining accuracy, recent works introduce hybrid architectures that combine convolutional and transformer design to effectively capture local and global information. Some designs replace the patchify stem [14] with convolutional layers [65], introduce early convolutional stages [9, 42] or implicitly hybridize through windowed attention [35, 7]. More recent works build explicit hybrid structures for better exchange of information between tokens (or patches) [17, 10, 64]. In majority of the hybrid architectures, token mixers are predominantly self-attention based. Recently, MetaFormer [67] introduced Pooling, a simple and efficient candidate for token mixing.

Structural Reparameterization Recent work [13, 57] shows the benefits of reparameterizing skip connections to lower memory access cost. In our model, we introduce a novel architectural component *RepMixer*, that is reparameterizable at inference. For better efficiency, works like [26, 47, 25, 37, 71] introduce factorized k×k convolutions using depthwise or grouped convolutions followed by 1×1 pointwise convolutions. While this approach is highly effective in improving the overall efficiency of the model, the lower parameter count can lead to reduced capacity. Recently, linear train-time over overparameterization was introduced in [57, 11, 12] to improve capacity of such models. We use factorized k×k convolutions in our model and boost the capacity of these layers using linear train-time overparameterization. To the best of our knowledge, structural reparameterization to remove skip connections and linear overparameterization has not been attempted in any prior hybrid transformer architecture.

3. Architecture

3.1. Overview

FastViT is a hybrid transformer and has four distinct stages which operate at different scales as shown in Figure 2. We detail all the FastViT variants in Table 2.

FastViT uses *RepMixer*, a token mixer that reparameterizes a skip connection, which helps in alleviating memory access cost (see Figure 2d). To further improve efficiency and performance, we replace dense k×k convolutions commonly found in stem and patch embedding layers with its factorized version that uses train-time overparameterization (see Figure 2a).

Architectural Choices	Params (M)	FLOPs (G)	Mobile Latency (ms)	Top-1 (%)
PoolFormer-S12 (Baseline)	11.9	1.8	1.50	77.2
+ 224 → 256	11.9	2.4	2.25	77.6
Section 3.2.1				
+ Pooling → RepMixer	11.9	2.4	1.58	78.5
Section 3.2.2				
+ Factorized dense conv.	8.7	1.7	1.26	78.0
+ Train-Time Overparam.	8.7	1.7	1.26	78.9
Section 3.2.3				
+ LK. conv. FFN	8.7	1.8	1.33	79.4
+ LK. conv. Patch Emb.	8.8	1.8	1.40	79.8

Table 1: Analysis of architectural choices made to obtain FastViT-S12 variant, starting from PoolFormer-S12. "LK." stands for Large Kernel.

Self-attention [14] token mixers are computationally expensive, especially at higher resolutions [43, 39]. While efficient versions of self-attention layers are explored in [17, 42], we use large kernel convolutions as an efficient alternative to improve receptive field in early stages of the network architecture (see Figure 2c).

We analyze various architectural choices made in designing FastViT from a PoolFormer [67] baseline in Table 1 and detail our approach below.

3.2. FastViT

3.2.1 Reparameterizing Skip Connections

RepMixer Convolutional mixing was first introduced in ConvMixer[55]. For an input tensor X, the mixing block in the layer was implemented as,

$$Y = \texttt{BN}(\sigma(\texttt{DWConv}(X))) + X \tag{1}$$

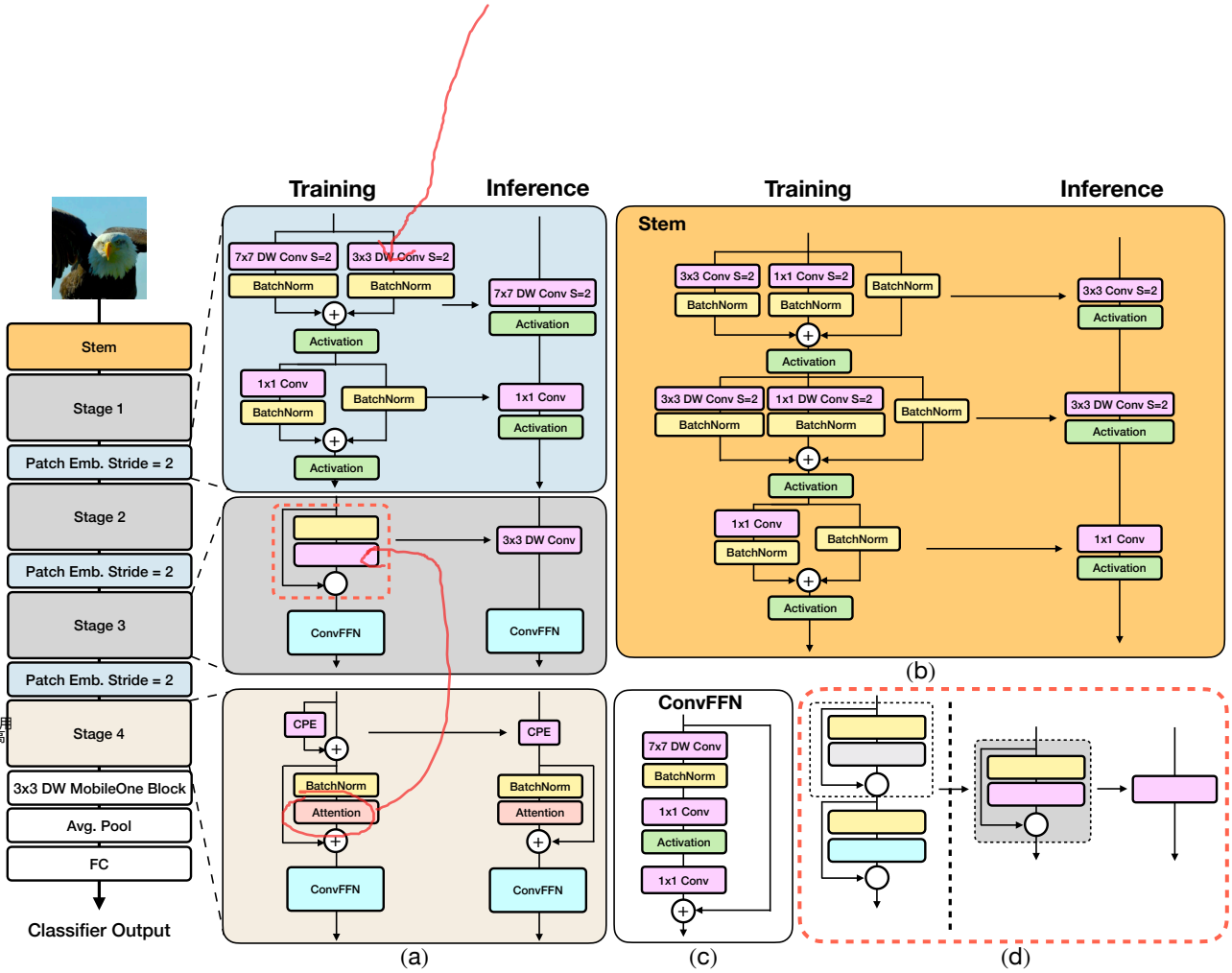
where σ is a non-linear activation function and BN is Batch Normalization [27] layer and DWConv is depthwise convolutional layer. While this block was shown to be effective, in *RepMixer*, we simply rearrange the operations and remove the non-linear activation function as shown below,

$$Y = \texttt{DWConv}(\texttt{BN}(X)) + X \tag{2}$$

The main benefit of our design is that it can be reparameterized at inference time to a single depthwise convolutional layer as shown below and in Figure 2d.

$$Y = \texttt{DWConv}(X) \tag{3}$$

Positional Encodings We use conditional positional encodings [7, 8] that is dynamically generated and conditioned on the local neighborhood of the input tokens. These encodings are generated as a result of a depth-wise convolution operator and are added to the patch embeddings. Note the lack of non-linearities in this group of operations, hence this block is reparameterized as shown in Figure 2a.



前几个stage并没有使用attention，复杂度较高

Figure 2: (a) Overview of FastViT architecture which decouples train-time and inference-time architecture. Stages 1, 2, and 3 have the same architecture and uses RepMixer for token mixing. In stage 4, self attention layers are used for token mixing. (b) Architecture of the convolutional stem. (c) Architecture of convolutional-FFN (d) Overview of *RepMixer* block, which reparameterizes a skip connection at inference.

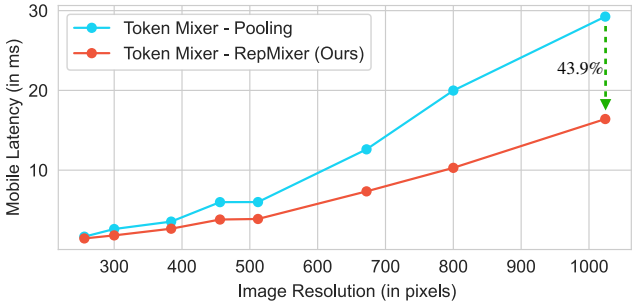


Figure 3: Latency comparison of a MetaFormer (S12) architecture with Pooling and RepMixer as a choice for token mixing; measured on iPhone 12 Pro for various image resolutions. Both models have ∼1.8G FLOPs. Absence of a skip connection in RepMixer lowers the overall memory access cost leading to lower latency.

Empirical Analysis In order to verify the benefits of reparameterizing skip connections, we ablate over the using one of the most efficient (in terms of latency) token mixer, i.e. Pooling and RepMixer in a MetaFormer S12 architecture. Both the models being ablated have ∼1.8G FLOPs. We time the models for various input resolutions starting from 224×224 to 1024×1024 on an iPhone 12 Pro mobile device. From Figure 3, we see that RepMixer significantly improves over Pooling, especially at higher resolutions. At 384×384, using RepMixer will lower the latency by 25.1% and at larger resolutions like 1024×1024, latency is lowered significantly by 43.9%.

3.2.2 Linear Train-time Overparameterization

In order to further improve efficiency (parameter count, FLOPs, and latency), we replace all dense k×k convolutions with its factorized version, i.e. k×k depthwise followed by 1×1 pointwise convolutions. However, the lower parameter count from factorization can diminish the capacity of the model. In order to increase capacity of the factorized layers, we perform linear train-time overparameterization as described in MobileOne [57]. MobileOne-style overparameterization in stem, patch embedding, and projection layers help in boosting performance. From Table 3, we note that train-time overparameterization improves Top-1 accuracy on ImageNet by 0.6% on FastViT-SA12 model. On a smaller FastViT-S12 variant, Top-1 accuracy improves

Stage	#Tokens	Layer Spec.		FastViT						
				T8	T12	S12	SA12	SA24	SA36	MA36
Stem	$H \times W$	Conv.		3×3, stride 2						
				3×3 MobileOne Style, stride 2						
				48	64					76
1	$\frac{H}{4} \times \frac{W}{4}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				48	64					76
		FastViT Block	Mixer	RepMixer						
			Exp.	3		4				
			Blocks	2	2	2	2	4	6	6
2	$\frac{H}{8} \times \frac{W}{8}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				96	128					152
		FastViT Block	Mixer	RepMixer						
			Exp.	3		4				
			Blocks	2	2	2	2	4	6	6
3	$\frac{H}{16} \times \frac{W}{16}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				192	256					304
		FastViT Block	Mixer	RepMixer						
			Exp.	3		4				
			Blocks	4	6	6	6	12	18	18
4	$\frac{H}{32} \times \frac{W}{32}$	Patch Embed.		7×7 MobileOne Style, stride 2						
				384	512					608
		FastViT Block	Mixer	RepMixer			Attention			
			Exp.	3		4				
			Blocks	2	2	2	2	4	6	6
Parameters (M)				3.6	6.8	8.8	10.9	20.6	30.4	42.7
FLOPs (G)				0.7	1.4	1.8	1.9	3.8	5.6	7.9

Table 2: **Architecture details of FastViT variants.** Models with smaller embedding dimensions, i.e. [64, 128, 256, 512] are prefixed with "S" and models that contain Self-Attention layers are prefixed with "SA". Models with bigger embedding dimensions, i.e. [76, 152, 304, 608] are prefixed with "M". Models with MLP expansion ratio less than 4 are prefixed with "T". The number in the notation denotes total number of FastViT blocks. FLOP count is computed by `fvcore`[15] library.

by 0.9% as shown in Table 1.

However, train-time overparameterization results in increased training time due to computational overhead from the added branches. In our architecture, we only overparameterize those layers that replace dense k×k convolutions with its factorized form as described above. These layers are found in the convolutional stem, patch embedding and projection layers. The computational cost incurred in these layers are lower than the rest of the network, hence overparameterizing these layers do not result in significant increases to train time. For example, FastViT-SA12 takes 6.7% longer and FastViT-SA36 takes 4.4% longer to train with train-time overparameterization as opposed to training those variants without it under the same settings described in Section 4.1.

Model	Ablation	Top-1 (%)	Train Time (hrs)
FastViT-SA12	Without Train-Time Overparam.	80.0	31.3
	With Train-Time Overparam.	**80.6**	33.4
FastViT-SA36	Without Train-Time Overparam.	83.3	73.5
	With Train-Time Overparam.	**83.6**	76.7

Table 3: Comparison of FastViT variants with and without linear train-time overparameterization when trained on ImageNet-1k dataset. Train time is wall clock time elapsed at the end of a training run.

3.2.3 Large Kernel Convolutions

The receptive field of RepMixer is local compared to self-attention token mixers. However, self-attention based token mixers are computationally expensive. A computationally efficient approach to improve the receptive field of early stages that do not use self-attention is by incorporating depthwise large kernel convolutions. We introduce depthwise large kernel convolutions in FFN and patch embedding layers. From Table 4, we note that variants using depthwise large kernel convolutions can be highly competitive to variants using self-attention layers while incurring a modest increase in latency. When we compare V5 with V3, model size increases by 11.2%, and latency increases by a factor of 2.3× for a relatively small gain of 0.4% in Top-1 accuracy. V2 is larger than V4 by 20% and has 7.1% higher latency than V4 while attaining similar Top-1 accuracy on ImageNet. Further ablations on kernel sizes and latency is provided in the supplementary materials. In Table 1, we ablate over large kernel convolutions in FFN and patch embedding layers. Overall, large kernel convolutions provide 0.9% improvement in Top-1 accuracy on FastViT-S12.

The architecture of our FFN and patch embedding layer is shown in Figure 2. Our FFN block has a structure similar to ConvNeXt [36] block with a few key differences, see Figure 2c. We use Batch Normalization as opposed to Layer Normalization, as it can be fused with the preceding layer at inference. Also, it does not require additional reshape operations to obtain appropriate tensor layout for LayerNorm as done in the original implementation of ConvNeXt block.

Along with increased receptive field, large kernel convolutions help in improving model robustness as observed in [61] and convolutional-FFN blocks generally tend to be more robust than vanilla-FFN blocks as observed in [38]. Hence, incorporating large kernel convolutions is an efficient way to improve model performance and robustness.

4. Experiments

4.1. Image Classification

We report results on the the widely used ImageNet-1K dataset [46] which contains ∼1.3M training images and 50K validation images. We follow the training recipe prescribed in [53, 67], i.e. the models are trained for 300

Variant	Stages				Params	Top-1	Mobile
	1	2	3	4	(M)	(%)	Latency (ms)
Standard Setting							
V1	RM	RM	RM	RM	8.7	78.9	1.3
V2	RM	RM	RM	SA	10.8	79.9	1.5
V3	RM	RM	SA	SA	12.4	81.0	3.7
Large Kernel Convolutions (7×7)							
V4	RM	RM	RM	RM	8.8	79.8	1.4
V5	RM	RM	RM	SA	10.9	80.6	1.6

Table 4: Ablation on using large kernel convolutions as a substitute for self-attention layers. "RM" indicates [RepMixer-FFN] block is used in the stage. "SA" indicates [Self Attention-FFN] block is used in the stage. Standard setting uses 3x3 factorized convolutions in patch embedding and stem layers and 1×1 convolutions for FFN. In variants V4 and V5, large kernel convolutions (7×7) are used in patch embedding and FFN layers.

epochs using AdamW optimizer with weight decay of 0.05 and peak learning rate 10^{-3} for a total batch size of 1024. The number of warmup epochs is set to 5 and cosine schedule is used to decay the learning rate. Our implementation uses timm library [62] and all the models were trained on 8 NVIDIA A100 GPUs. See supplementary materials for details on hyper parameters used for all the variants. For 384×384 input size, we fine-tune the models for 30 epochs with weight decay of 10^{-8} and learning rate of 5×10^{-5} and batch size of 512 following [36]. To measure latency, we use the input sizes corresponding to the respective methods. For iPhone latency measurements, we export the models using Core ML Tools (v6.0) and run it on iPhone12 Pro Max with iOS 16 and batch size is set to 1 for all the models. We follow the same protocol as [57]. For GPU latency measurements, we export the traced model to TensorRT (v8.0.1.6) format and run it on NVIDIA RTX-2080Ti with batch size of 8. We report the median latency estimate from 100 runs.

Comparison with SOTA Models In Table 5, we compare our models against recent state-of-the-art models on ImageNet-1k dataset. For a fair comparison, we modify ConvNeXt [36] from official implementation by avoiding costly reshape operations, see supplementary materials for more details. We were unable to reliably export LITv2 [42] due to poor support for deformable convolutions in either library. Our model obtains the best accuracy-latency trade-off when compared to recent state-of-the-art models on two different compute fabrics, i.e. desktop-grade GPU and mobile device. Our models improve over LITv2 [42] on both parameter count and FLOPs, at Top-1 accuracy of 84.9%, FastViT-MA36 is 49.3% smaller and consumes 55.4% less FLOPs than LITv2-B. FastViT-S12 is 26.3% faster than MobileOne-S4 [57] on iPhone 12 Pro and 26.9% faster on

Model	Eval Image Size	Param (M)	FLOPs (G)	GPU Latency (ms)	Mobile Latency (ms)	Top-1 Acc. (%)
MobileOne-S0[57]	224	2.1	0.3	**1.0**	**0.8**	71.4
FastViT-T8	256	3.6	0.7	1.7	**0.8**	**75.6**
MobileOne-S3[57]	224	10.1	1.8	**2.1**	1.5	78.1
CMT-T*[17]	160	9.5	0.6	4.5	3.8	**79.1**
EfficientNet-B1[50]	256	7.8	0.7	3.2	2.5	**79.1**
FastViT-T12	256	6.8	1.4	**2.1**	**1.2**	**79.1**
MobileOne-S4[57]	224	14.8	2.9	2.6	1.9	79.4
RSB-ResNet50[63]	224	25.6	4.1	2.3	2.6	**79.8**
DeiT-S[53]	224	22.0	4.6	5.2	5.3	**79.8**
FastViT-S12	256	8.8	1.8	**2.2**	**1.4**	**79.8**
RegNetY-8G[44]	224	39.2	8.0	5.8	3.8	79.9
EfficientNet-B2[50]	288	9.2	1.0	4.1	3.5	80.1
PoolFormer-S24[67]	224	21.0	3.4	15.7	2.5	80.3
RegNetY-16G [44]	224	83.6	16.0	8.1	7.5	80.4
FastViT-SA12	256	10.9	1.9	**2.5**	**1.6**	**80.6**
ResNeSt50[69]	224	27.5	5.4	51.5	29.9	81.1
Swin-T[35]	224	29.0	4.5	-	9.3	81.3
PoolFormer-S36[67]	224	31.0	5.0	23.7	3.5	81.4
EfficientNet-B3[50]	300	12.0	1.8	5.3	7.1	81.6
CvT-13[64]	224	20.1	4.5	18.1	62.6	81.6
CoAtNet-0[9]	224	25.0	4.2	-	6.7	81.6
CMT-XS*[17]	192	15.2	1.5	8.3	6.3	81.8
DeiT-B [53]	224	86.0	17.5	11.6	11.6	81.8
RSB-ResNet152[63]	224	60.2	11.6	5.4	4.7	81.8
LITv2-S[42]	224	28.0	3.7	-	-	82.0
ConvNeXt-T†[36]	224	29.0	4.5	6.3	3.7	82.1
PoolFormer-M48[67]	224	73.0	11.6	30.2	7.6	82.5
FastViT-SA24	256	20.6	3.8	**3.8**	**2.6**	**82.6**
ResNeSt101[69]	224	48.2	10.2	75.5	37.7	83.0
Swin-S[35]	224	50.0	8.7	-	13.0	83.0
ConvNeXt-S†[36]	224	50.0	8.7	10.3	5.4	83.1
MOAT-0[5]	224	27.8	5.7	-	4.9	83.3
CoAtNet-1[9]	224	42.0	8.4	-	13.4	83.3
Swin-B [35]	224	88.0	15.4	-	18.5	83.5
CMT-S*[17]	224	25.1	4.0	13.8	15.6	83.5
MaxViT-T[56]	224	31.0	5.6	-	31.5	**83.6**
LITv2-B[42]	224	87.0	13.2	-	-	**83.6**
EfficientNet-B5[50]	456	30.0	9.9	24.2	22.1	**83.6**
NFNet-F0[1]	256	71.5	12.4	6.2	6.4	**83.6**
FastViT-SA36	256	30.4	5.6	**5.2**	**3.5**	**83.6**
ConvNeXt-B†[36]	224	89.0	15.4	13.6	8.4	83.8
CoAtNet-0	384	25.0	13.4	-	20.1	**83.9**
EfficientNetV2-S[51]	384	22.0	8.8	8.2	7.1	**83.9**
FastViT-MA36	256	42.7	7.9	**6.7**	**4.5**	**83.9**
EfficientNet-B6[50]	528	43.0	19.0	44.3	51.7	84.0
CoAtNet-2[9]	224	75.0	15.7	-	19.6	84.1
MOAT-1[5]	224	41.6	9.1	-	8.3	84.2
EfficientNet-B7[50]	600	66.0	37.0	72.6	85.5	84.3
CMT-B*[17]	256	45.7	9.3	26.9	20.6	**84.5**
Swin-B[35]	384	88.0	47.0	-	57.2	**84.5**
MaxViT-S[56]	224	69.0	11.7	-	48.7	**84.5**
FastViT-SA36	384	30.4	12.6	**12.8**	**7.4**	**84.5**
MOAT-0[5]	384	27.8	18.2	-	24.3	84.6
NFNet-F1[1]	320	132.6	35.5	16.9	16.1	84.7
MOAT-2[5]	224	73.4	17.2	-	12.6	84.7
LITv2-B[42]	384	87.0	39.7	-	-	84.7
FastViT-MA36	384	42.7	17.7	**16.5**	**9.2**	**84.9**

Table 5: Comparison of different state-of-the-art methods on ImageNet-1k classification. HardSwish is not well supported by Core ML, $*$ denotes we replace it with GELU for fair comparison. "†" denotes that model has been modified from original implementation for efficient deployment. Models which could not be reliably exported either by TensorRT or Core ML Tools are annotated by "-".

Model	Eval Image Size	Param (M)	FLOPs (G)	GPU Latency (ms)	Mobile Latency (ms)	Top-1 Acc. (%)
FastViT-T8	256	3.6	0.7	1.7	**0.8**	**76.7**
FastViT-T12	256	6.8	1.4	2.1	**1.2**	**80.3**
CaiT XXS-36[54]	224	17.3	3.8	15.8	-	79.7
FastViT-S12	256	8.8	1.8	**2.2**	**1.4**	**80.9**
DeiT-S[53]	224	22	4.6	5.2	6.7	81.2
FastViT-SA12	256	10.9	1.9	**2.5**	**1.6**	**81.9**
EfficientFormer-L3[31]	224	31.3	3.9	**3.6**	3.0	82.4
EfficientFormer-L7[31]	224	82.1	10.2	7.5	7.0	83.3
DeiT-B[53]	224	87	17.6	11.6	11.6	**83.4**
FastViT-SA24	256	20.6	3.8	3.8	**2.6**	**83.4**
CaiT S-24[54]	224	46.9	9.4	19.1	-	83.5
CaiT XS-24[54]	384	26.6	19.3	77.7	-	84.1
FastViT-SA36	256	30.4	5.6	**5.2**	**3.5**	**84.2**
FastViT-MA36	256	42.7	7.9	**6.7**	**4.5**	**84.6**

Table 6: Comparison of different state-of-the-art methods on ImageNet-1k classification when trained using distillation objective specified by [53]. Models which could not be reliably exported either by TensorRT or Core ML Tools are annotated by "-".

GPU. At Top-1 accuracy of 83.9%, FastViT-MA36 is 1.9× faster than an optimized ConvNeXt-B model on iPhone 12 Pro and 2.0× faster on GPU. At Top-1 accuracy of 84.9%, FastViT-MA36 is just as fast as NFNet-F1 [1] on GPU while being 66.7% smaller and using 50.1% less FLOPs and 42.8% faster on mobile device.

Knowledge distillation We report performance of our models when trained with distillation objective in Table 6. We follow the setting described in DeiT [53], with RegNet-16GF [44] as the teacher model. Following DeiT [53], we use hard distillation where hard decision of the teacher is set as true label. Our models are trained for 300 epochs. Unlike [53, 54, 31], we do not introduce an additional classification head for distillation. FastViT outperforms recent state-of-the-art model EfficientFormer [31]. FastViT-SA24 attains similar performance as EfficientFormer-L7 while having 3.8× less parameters, 2.7× less FLOPs and 2.7× lower latency.

4.2. Robustness Evaluation

We evaluate our models for out-of-distribution robustness on the following benchmarks – (i) ImageNet-A [24], a dataset that contains naturally occurring examples that are misclassified by ResNets; (ii) ImageNet-R [22], a dataset that contains natural renditions of ImageNet object classes with different textures and local image statistics; (iii) ImageNet-Sketch [58], a dataset that contains black and white sketches of all ImageNet classes, obtained using google image queries; and (iv) ImageNet-C [23], a dataset that consists of algorithmically generated corruptions (blur, noise) applied to the ImageNet test-set. We evaluate our models using the implementation provided by [38]. Our models are more robust than recent vision transformers while being faster than pure-CNN based models which exhibit low robustness as seen Table 7. Architectural choices like using a large kernel convolutions in FFN and patch-embedding layers in combination with self-attention layers helps in improving the robustness of our model as discussed in Section 3.2.3. All the models compared in Table 7 are trained only on ImageNet-1k dataset using similar training recipes. From Table 7, our model is highly competitive to RVT and ConvNeXt, in fact FastViT-M36 has better clean accuracy, better robustness to corruptions and similar out-of-distribution robustness as ConvNeXt-S which has 6.1M more parameters and has 10% more FLOPs than our model.

Model	#Params	#FLOPs	Clean	IN-C (↓)	IN-A	IN-R	IN-SK
PVT-Tiny	13.2	1.9	75.0	79.6	7.9	33.9	21.5
RVT-Ti	8.6	1.3	78.4	**58.2**	13.3	**43.7**	**30.0**
FastViT-SA12	10.9	1.9	**80.6**	62.2	**17.2**	42.6	29.7
PVT-Small	24.5	3.8	79.9	66.9	18.0	40.1	27.2
ResNet-50*	25.6	4.1	79.0	65.5	5.9	42.5	31.5
ResNeXt50-32x4d	25.0	4.3	79.8	64.7	10.7	41.5	29.3
FastViT-SA24	20.6	3.8	**82.6**	**55.3**	**26.0**	**46.5**	**34.0**
Swin-T	28.3	4.5	81.2	62.0	21.6	41.3	29.1
ConViT-S	27.8	5.4	81.5	**49.8**	24.5	45.4	33.1
RVT-S	22.1	4.7	81.7	50.1	24.1	46.9	35.0
ConvNeXt-T	28.6	4.5	82.1	53.2	24.2	47.2	33.8
EfficientNet-B4	19.3	4.2	83.0	71.1	26.3	47.1	34.1
FastViT-SA36	30.4	5.6	**83.6**	51.8	**32.3**	**48.1**	**35.8**
ConvNeXt-S	50.0	8.7	83.1	51.2	31.2	**49.5**	**37.1**
FastViT-MA36	42.7	7.9	**83.9**	**49.9**	**34.6**	**49.5**	36.6

Table 7: Results on robustness benchmark datasets. Models are grouped based on FLOPs. Performance of competing models is reported by [38] and [36]. *ResNet-50 model is trained with AugMix to improve robustness as reported in [38]. For ImageNet-C mean corruption error is reported (lower is better) and for other datasets Top-1 accuracy is reported (higher is better). The best results are in bold and second best results are underlined.

4.3. 3D Hand mesh estimation

Recent works on real-time 3D hand mesh estimation introduce complex mesh regression layers over CNN based backbones. The backbones usually belong to ResNet or MobileNet family of architectures with the exception of METRO and MeshGraphormer which use HRNets [48] for feature extraction. While most hardware devices are highly optimized for feature extraction from 2D CNNs, the same is not true for the complex mesh regression heads used in these methods. In our method, we replace the complex mesh regression head with a simple regression module which regresses weak perspective camera, pose and shape parameters of the MANO model [45]. We argue that using a feature

Method	Backbone	PJ ↓	PV ↓	F@5 ↑	F@15 ↑	FPS ↑
Non Real-Time methods						
HIU-DMTL [70]	Customized	7.1	7.3	0.699	0.974	5
METRO [32]	HRNet	6.3	6.5	0.731	0.984	4.2
MeshGraphormer [33]	HRNet	5.9	6.0	0.764	0.986	2.6
Real-Time methods						
Hasson *et al*. [19]	ResNet18	-	13.2	0.436	0.908	20
MobileHand [16]	MobileNetV3	-	13.1	0.439	0.902	110
YoutubeHand [30]	ResNet50	8.4	8.6	0.614	0.966	60
I2L-MeshNet [40]	ResNet50	7.4	7.6	0.681	0.973	33.3
Pose2Mesh [6]	Customized	7.4	7.6	0.683	0.973	20
I2UV-HandNet [2]	ResNet50	7.2	7.4	0.682	0.973	-
Tang *et al*. [52]	ResNet50	7.1	7.1	0.706	0.977	39.1
MobRecon[4]	DenseStack	6.9	7.2	0.694	0.979	77
CMR [3]	ResNet50*	6.9	7.0	0.715	0.977	-
Ours	**FastViT-MA36**	**6.6**	**6.7**	**0.722**	**0.981**	**218**

Table 8: Results on the FreiHAND test dataset. FPS is computed on NVIDIA RTX-2080Ti under similar setting as MobRecon [4]. Performance of competing methods obtained from training exclusively on FreiHand dataset.

extraction backbone that learns a good representation for underlying images can alleviate learning challenges in mesh regression. While other real-time methods compensate for weak feature extraction backbone with complex mesh regression layers, we use a better feature extraction backbone with a simple mesh regression layer. We compare our approach with other published methods on the FreiHand [75] dataset. For a fair comparison, we cite results of methods that only used FreiHand dataset for training as some methods either pre-train, train, or fine-tune with additional pose datasets. We only use ImageNet-1k dataset for pretraining and then train exclusively on the FreiHand dataset using the experimental setup described in [32]. For more details, please see supplementary materials. From Table 8, amongst real-time methods, our method outperforms other methods on all joint and vertex error related metrics while being 1.9× faster than MobileHand [16] and 2.8× faster than recent state-of-art MobRecon.

4.4. Semantic Segmentation and Object Detection

For semantic segmentation, we validate the performance of our models on ADE20k [72]. The dataset contains 20K training images and 2K validation images with 150 semantic categories. We train semantic segmentation models with Semantic FPN [28] decoder. Models with Semantic FPN head use the same settings as [67]. All models are initialized with pretrained weights from their corresponding image classification models. FLOPs and backbone latency are estimated on image crops of 512×512. Due to higher resolution of input image, GPU latencies are estimated over a batch size of 2 in both Table 9 and Table 10. In Table 9, we compare our models with recent works. FastViT-MA36 model obtains 5.2% higher mIoU than PoolFormer-M36 which has higher FLOPs, parameter count and latency on both desktop GPU and mobile device.

Backbone	GPU Latency	Mobile Latency	Semantic FPN 80k		
			Param(M)	FLOPs(G)	mIoU(%)
ResNet-50	2.7	8.7	28.5	46	36.7
PoolFormer-S12[67]	9.7	6.2	15.7	31	37.2
FastViT-SA12	**2.5**	**5.6**	14.1	29	**38.0**
ResNet-101	4.6	12.4	47.5	65	38.8
PVT-Small[60]	-	-	28.2	41	39.8
PoolFormer-S24[67]	18.9	10.7	23.2	48	40.3
FastViT-SA24	**4.4**	**9.3**	23.8	37	**41.0**
PoolFormer-S36[67]	28.0	17.0	34.6	48	42.0
FastViT-SA36	**6.1**	**12.9**	33.6	44	**42.9**
PVT-Medium[60]	-	-	48.0	55	41.6
PoolFormer-M36[67]	41.4	24.8	59.8	68	42.4
FastViT-MA36	**8.2**	**16.3**	45.7	53	**44.6**

Table 9: Performance of different backbones on ADE20K semantic segmentation task. Following common convention, FLOPs and backbone latencies are measured on crops of 512×512.

Backbone	GPU Latency	Mobile Latency	AP^b	AP^b_{50}	AP^b_{75}	AP^m	AP^m_{50}	AP^m_{75}
Poolfomer-S12[67]	9.7	6.2	37.3	59.0	40.1	34.6	55.8	36.9
ResNet-50 [21]	2.7	8.7	38.0	58.6	41.4	34.4	55.1	36.7
FastViT-SA12	**2.5**	**5.6**	**38.9**	**60.5**	**42.2**	**35.9**	**57.6**	**38.1**
ResNet-101 [21]	4.6	12.4	40.0	60.5	44.0	36.1	57.5	38.6
Poolfomer-S24[67]	18.9	10.7	40.1	62.2	43.4	37.0	59.1	39.6
PVT-S [60]	-	-	40.4	62.9	43.8	37.8	60.1	40.3
FastViT-SA24	**4.4**	**9.3**	**42.0**	**63.5**	**45.8**	**38.0**	**60.5**	**40.5**
Swin-T [35]	-	-	42.2	64.6	46.2	39.1	61.6	42.0
Twins-SVT-S [7]	-	-	42.7	65.6	46.7	39.6	62.5	42.6
Twins-PCPVT-S [7]	-	52.1	42.9	65.8	47.1	40.0	62.7	42.9
FastViT-SA36	**6.1**	**12.9**	**43.8**	**65.1**	**47.9**	**39.4**	**62.0**	**42.3**
CMT-S [17]	19.9	70.9	44.6	**66.8**	48.9	**40.7**	**63.9**	**43.4**
Poolfomer-S36[67]	28.0	17.0	41.0	63.1	44.8	37.7	60.1	40.0
FastViT-MA36	**8.2**	**16.3**	**45.1**	**66.8**	**49.5**	40.5	63.8	**43.4**

Table 10: Results for object detection and instance segmentation on MS-COCO `val2017` split using Mask-RCNN [20] framework using 1x training schedule, i.e. 12 epochs used for training the models. Backbone latencies are measured on crops of 512×512.

We train object detection on the MS-COCO [34] dataset with 80 classes containing 118K training and 5K validation images. In Table 10, we compare our models with recent works. All the models are trained with 1x schedule following [67] using a Mask-RCNN [20] head. All models are initialized with pretrained weights from their corresponding image classification models. We show that our models achieve state-of-the-art performance under multiple latency regimes. FastViT-MA36 model has similar performance as CMT-S, while being 2.4× and 4.3× faster on desktop GPU and mobile device respectively.

5. Conclusion

We have proposed a general purpose hybrid vision transformer that is highly efficient on multiple compute fabrics: mobile devices and desktop grade GPUs. Through structural reparameterization, our model incurs reduced memory access cost. This leads to significant improvements in runtime especially at higher resolutions. In addition, we propose further architectural changes that boosts performance on ImageNet classification task and other downstream tasks like object detection, semantic segmentation and 3D hand mesh estimation. We empirically show that our backbone is highly robust to out-of-distribution samples, while being significantly faster than competing robust models.

Acknowledgements We thank Jen-Hao Rick Chang, Skyker Seto and Hadi Pouransari for helpful discussions and reviewing the manuscript.

References

[1] Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. 2021.

[2] Ping Chen, Yujin Chen, Dong Yang, Fangyin Wu, Qin Li, Qingpei Xia, and Yong Tan. I2uv-handnet: Image-to-uv prediction network for accurate and high-fidelity 3d hand mesh modeling. In *International Conference on Computer Vision (ICCV)*, 2021.

[3] Xingyu Chen, Yufeng Liu, Chongyang Ma, Jianlong Chang, Huayan Wang, Tian Chen, Xiaoyan Guo, Pengfei Wan, and Wen Zheng. Camera-space hand mesh recovery via semantic aggregationand adaptive 2d-1d registration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[4] Xingyu Chen, Yufeng Liu, Dong Yajiao, Xiong Zhang, Chongyang Ma, Yanmin Xiong, Yuan Zhang, and Xiaoyan Guo. Mobrecon: Mobile-friendly hand mesh reconstruction from monocular image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[5] Chenglin Yang et al. MOAT: Alternating mobile convolution and attention brings strong vision models. In *ICLR*, 2023.

[6] Hongsuk Choi, Gyeongsik Moon, and Kyoung Mu Lee. Pose2mesh: Graph convolutional network for 3d human pose and mesh recovery from a 2d human pose. In *European Conference on Computer Vision (ECCV)*, 2020.

[7] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. *arXiv preprint arXiv:2104.13840*, 2021.

[8] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Conditional positional encodings for vision transformers. 2021.

[9] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

[10] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[11] Xiaohan Ding, Yuchen Guo, Guiguang Ding, and Jungong Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[12] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Diverse branch block: Building a convolution as an inception-like unit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[13] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[15] fvcore. Light-weight core library that provides the most common and essential functionality shared in various computer vision frameworks developed in fair. https://github.com/facebookresearch/fvcore, 2019.

[16] Lim Guan Ming, Jatesiktat Prayook, and Ang Wei Tech. Mobilehand: Real-time 3d hand shape and pose estimation from color image. In *27th International Conference on Neural Information Processing (ICONIP)*, 2020.

[17] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. Cmt: Convolutional neural networks meet vision transformers. *arXiv preprint arXiv:2107.06263*, 2021.

[18] Shuxuan Guo, Jose M. Alvarez, and Mathieu Salzmann. Expandnets: Linear over-parameterization to train compact convolutional networks. In *Advances in Neural Information Processing Systems*, 2020.

[19] Yana Hasson, Gül Varol, Dimitris Tzionas, Igor Kalevatykh, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning joint reconstruction of hands and manipulated objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu,

Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. 2021.

[23] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[24] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. 2021.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

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

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[28] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019.

[29] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020.

[30] Dominik Kulon, Riza Alp Guler, Iasonas Kokkinos, Michael M. Bronstein, and Stefanos Zafeiriou. Weakly-supervised mesh-convolutional hand reconstruction in the wild. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[31] Yanyu Li, Geng Yuan, Yang Wen, Eric Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. Efficientformer: Vision transformers at mobilenet speed. *arXiv preprint arXiv:2206.01191*, 2022.

[32] Kevin Lin, Lijuan Wang, and Zicheng Liu. End-to-end human pose and mesh reconstruction with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[33] Kevin Lin, Lijuan Wang, and Zicheng Liu. Mesh graphormer. In *International Conference on Computer Vision (ICCV)*, 2021.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, 2014.

[35] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[36] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[37] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision*, pages 116–131, 2018.

[38] Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[39] Dmitrii Marin, Jen-Hao Rick Chang, Anurag Ranjan, Anish Prabhu, Mohammad Rastegari, and Oncel Tuzel. Token pooling in vision transformers. *arXiv preprint arXiv:2110.03860*, 2021.

[40] Gyeongsik Moon and Kyoung Mu Lee. I2l-meshnet: Image-to-lixel prediction network for accurate 3d human pose and mesh estimation from a single rgb image. In *European Conference on Computer Vision (ECCV)*, 2020.

[41] Junting Pan, Adrian Bulat, Fuwen Tan, Xiatian Zhu, Lukasz Dudziak, Hongsheng Li, Georgios Tzimiropoulos, and Brais Martinez. Edgevits: Competing light-weight cnns on mobile devices with vision transformers. In *European Conference on Computer Vision*, 2022.

[42] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Fast vision transformers with hilo attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[43] Zizheng Pan, Bohan Zhuang, Haoyu He, Jing Liu, and Jianfei Cai. Less is more: Pay less attention in vision transformers. In *AAAI*, 2022.

[44] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[45] Javier Romero, Dimitrios Tzionas, and Michael J. Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 2017.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[47] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[48] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[49] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[50] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[51] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[52] Xiao Tang, Tianyu Wang, and Chi-Wing Fu. Towards accurate alignment in real-time 3d hand-mesh reconstruction. In *International Conference on Computer Vision (ICCV)*, 2021.

[53] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357, 2021.

[54] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[55] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[56] Zhengzhong Tu, Hossein Talebi, Han Zhang, Feng Yang, Peyman Milanfar, Alan Bovik, and Yinxiao Li. Maxvit: Multi-axis vision transformer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[57] Pavan Kumar Anasosalu Vasu, James Gabriel, Jeff Zhu, Oncel Tuzel, and Anurag Ranjan. An improved one millisecond mobile backbone. *arXiv preprint arXiv:2206.04040*, 2022.

[58] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, 2019.

[59] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[60] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.

[61] Zeyu Wang, Yutong Bai, Yuyin Zhou, and Cihang Xie. Can cnns be more robust than transformers? *arXiv preprint arXiv:2206.03452*, 2022.

[62] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[63] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm, 2021.

[64] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. 2021.

[65] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross B. Girshick. Early convolutions help transformers see better. *CoRR*, abs/2106.14881, 2021.

[66] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[67] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10819–10829, 2022.

[68] Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

[69] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2736–2746, 2022.

[70] Xiong Zhang, Hongsheng Huang, Jianchao Tan, Hongmin Xu, Cheng Yang, Guozhu Peng, Lei Wang, and Ji Liu. Hand image understanding via deep multi-task learning. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[71] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[72] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 633–641, 2017.

[73] Daquan Zhou, Qibin Hou, Yunpeng Chen, Jiashi Feng, and Shuicheng Yan. Rethinking bottleneck structure for efficient mobile network design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[74] Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. On the continuity of rotation representations in neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[75] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russell, Max Argus, and Thomas Brox. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 813–822, 2019.

A. Ablations

A.1. Architectural Choices

The primary motivation behind the design choices made for FastViT is efficient mobile deployment. The cost of self-attention blocks is very high, especially when there are many tokens. In early stages (when the number of tokens are high), self attention can be replaced with efficient alternatives with small accuracy degradation but significantly lower latency. In Table 4 of main paper, we analyzed this for last two stages. In Table 11, we present the full analysis for S12 architecture.

Ablation	Description	Top-1 Acc.	Mobile Latency (ms)
-	Baseline	79.8	1.4
Normalization	BatchNorm → LayerNorm	79.7	1.7
Activation	GELU → ReLU	79.4	1.3
	GELU → SiLU	79.7	1.4
Hybrid Stages	[RepMix, RepMix, RepMix, SelfAttn.]	80.6	1.6
	[RepMix, RepMix, SelfAttn., SelfAttn.]	81.2	3.7
	[RepMix, SelfAttn., SelfAttn., SelfAttn.]	81.5	5.0
	[SelfAttn., SelfAttn., SelfAttn., SelfAttn.]	82.0	11.7

Table 11: Ablation for FastViT-S12 on ImageNet-1K. All models are trained and benchmarked using the same settings described in main paper.

A.2. Large Kernel Convolutions

In Table 12, we ablate over various kernel sizes in FFN and patch embedding layers and report their Top-1 accuracy on ImageNet-1k along with mobile latency on iPhone 12 Pro. We observe that performance of the model stagnates beyond the kernel size of 7×7, while the overall FLOPs, latency and parameter count increases. Hence, we use 7×7 kernel size in our network architecture.

Kernel Size	Params (M)	FLOPs (G)	Mobile Latency(ms)	Top-1 (%)
3×3	8.7	1.7	1.3	78.9
5×5	8.7	1.8	1.4	79.5
7×7	8.8	1.8	1.4	79.8
9×9	8.8	1.8	1.5	79.6
11×11	8.8	1.9	1.5	79.8

Table 12: Top-1 accuracy on ImageNet-1k dataset for FastViT-S12 model with varied kernel sizes in FFN and patch embedding layers.

A.3. Training Time

We provide a coarse ablation of this in Table 3 of main paper. In our model, we do not overparameterize every element of the architecture, especially the parameter dense blocks like ConvFFN. In fact, we do not obtain any improvement by overparameterizing ConvFFN layers, verified empirically in Table 13. Since our model is partially overparameterized (only convolutional stem and patch embedding layers) during training, we do not see a significant degradation in train time as opposed to other train-time overparameterized models in literature which overparameterize all layers in a network. Note, all models were trained using the same hardware as described in Section 4.1 of main paper.

Ablation	Description	Top-1 Acc.	Train Time (hrs)
No OverParam.	-	80.0	31.3
Train-Time OverParam.	Stem + Patch Emb.	80.6	33.4
	Stem + Patch Emb. + ConvFFN	80.6	40.1

Table 13: Train-time overparameterization ablation for FastViT-SA12 on ImageNet-1K. Extension to Table 3 in main paper. Train time is wall clock time elapsed at the end of a training run.

B. Experiments

B.1. Benchmarking

We follow the same protocol prescribed in [57] while benchmarking the model on an iPhone 12 Pro mobile device. To benchmark the models on desktop-grade GPU NVIDIA RTX-2080Ti, we first warmup the device by running the forward pass of TensorRT model for 100 iterations and then benchmark the model over a 100 iterations. We report the median latency value from 100 estimates. For image classification models we use a batchsize of 8 (similar approach was adopted in [37]) and a batchsize of 2 (due to GPU memory limits) for semantic segmentation and object detection models. Both mobile and GPU latency estimates can have a standard deviation of ±0.1ms.

While benchmarking ConvNeXt [36] models on mobile device, we noticed the inefficiencies introduced by reshape ops causing increase in latency. While majority of hybrid models that use self-attention based token mixers require explicit reshaping of tensors. We can avoid this operation in ConvNeXt basic block by simply using a channel first implementation of LayerNorm and replacing `nn.Linear` layers with 1×1 convolutions. This simple change results in significant improvement in runtime as shown in Table 14.

Model	Mobile Latency (ms)	
	Before	After
ConvNeXt-T	33.5	3.7
ConvNeXt-S	66.4	5.4
ConvNeXt-B	89.1	8.4

Table 14: Benchmarking ConvNeXt before and after modifications.

B.2. Image Classification

We provide hyperparameters used for training models on ImageNet-1k dataset reported in Table 5 in main paper. Models are trained at resolution 256×256 and fine-tuned for 384×384. We follow the same training setup as [67, 53]. The hyperparameters used for all FastViT variants are listed in Table 15. For distillation experiments, we use RegNetY-16GF [44] as the teacher model similar to [53]. Additional

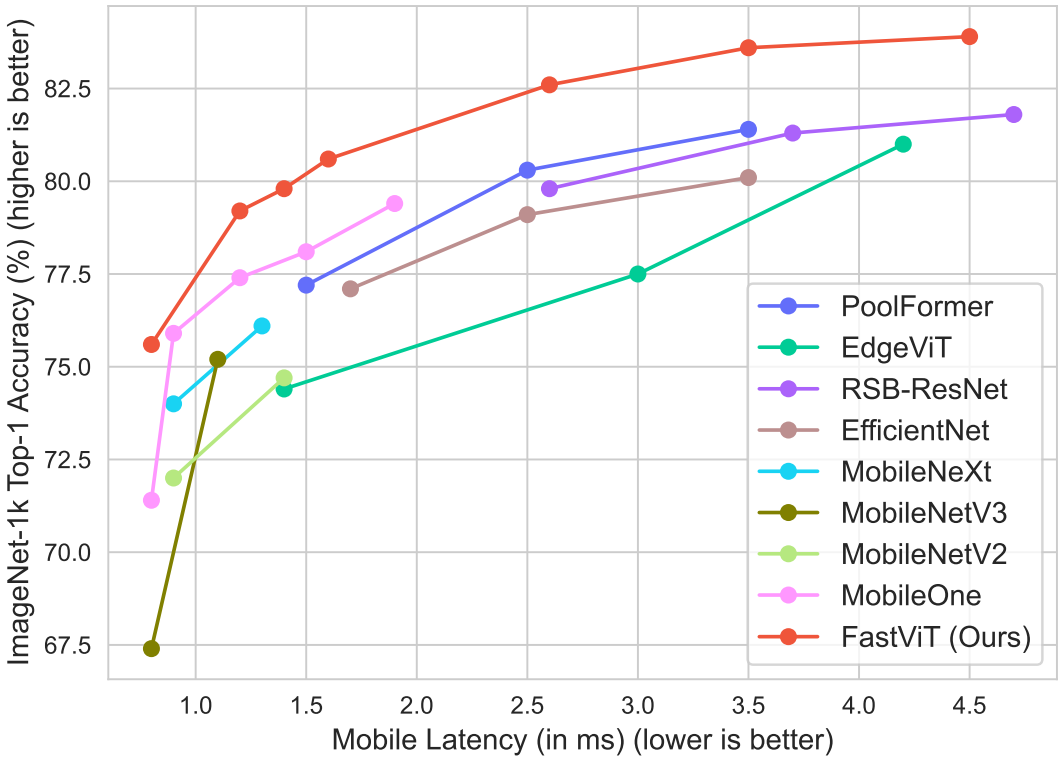


Figure 4: Accuracy vs. Mobile latency scaling curves of recent state-of-the-art Mobile Architectures and FastViT variants. The models are benchmarked on iPhone 12 Pro using the appropriate image sizes described in Table 16.

Hyperparameter	Training T8, T12, S12, SA12, SA24, SA36, MA36	Fine-tuning SA36, MA36
Stochastic depth rate	[0.0, 0.0, 0.0, 0.1, 0.1, 0.2, 0.35]	[0.2, 0.4]
Input resolution	256×256	384×384
Data augmentation	RandAugment	RandAugment
Mixup α	0.8	0.8
CutMix α	1.0	1.0
Random erase prob.	0.25	0.25
Label smoothing	0.1	0.1
Train epochs	300	30
Warmup epochs	5	None
Batch size	1024	1024
Optimizer	AdamW	AdamW
Peak learning rate	1e-3	5e-6
LR. decay schedule	cosine	None
Weight decay rate	0.05	1e-8
Gradient clipping	None	None
EMA decay rate	0.9995	0.9995

Table 15: Training hyperparameters for ImageNet-1k experiments.

hyperparameters are same as our image classification training procedure and are listed in Table 15. When trained using different seeds, results are within $\pm 0.2\%$ in Top-1 accuracy.

B.3. Comparison with Mobile Architectures

We compare our model against highly efficient mobile architectures in Table 16 and in Figure 4. Our model outperforms recent state-of-the-art MobileOne [57] architec-

Model	Eval Image Size	Param (M)	FLOPs (G)	Mobile Latency (ms)	Top-1 Acc. (%)
MobileNetV3-S*[25]	224	2.5	0.06	**0.8**	67.4
MobileOne-S0[57]	224	2.1	0.3	**0.8**	71.4
MobileNetV2-x1.0[47]	224	3.4	0.3	0.9	72.0
DeiT-Ti[53]	224	5.7	1.3	4.8	72.2
MobileNeXt-x1.0[73]	224	3.4	0.3	0.9	74.0
EdgeViT-XXS[41]	224	4.1	0.6	1.4	74.4
MNASNet-A1[49]	224	3.9	0.3	1.0	75.2
FastViT-T8	256	3.6	0.7	**0.8**	**75.6**
MobileNetV2-x1.4[47]	224	6.9	0.6	1.4	74.7
MobileNetV3-L[25]	224	5.4	0.2	1.1	75.2
MobileNeXt-x1.4[73]	224	6.1	0.6	1.3	76.1
EfficientNet-B0[50]	224	5.3	0.4	1.7	77.1
EdgeViT-XS[41]	224	6.7	1.1	3.0	77.5
MobileOne-S3[57]	224	10.1	1.8	1.5	78.1
CMT-T*[17]	160	9.5	0.6	3.8	**79.1**
EfficientNet-B1[50]	256	7.8	0.7	2.5	**79.1**
FastViT-T12	256	6.8	1.4	**1.2**	**79.1**
MobileOne-S4[57]	224	14.8	2.9	1.9	79.4
DeiT-S[53]	224	22.0	4.6	5.3	**79.8**
FastViT-S12	256	8.8	1.8	**1.4**	**79.8**

Table 16: Comparison of different state-of-the-art Mobile architectures on ImageNet-1k classification. HardSwish is not well supported by Core ML, $*$ denotes we replace it with GELU for fair comparison.

ture which is purely convolutional. Our model also outperforms EdgeViT [41], which is a recent state-of-the-art light-weight ViT architecture.

B.4. Model Scaling

In this work, we sample architectures that are smaller than 50M parameters for efficient deployment. Similar to the Swin-T [35] variant, we use a stage compute ratio of 1:1:3:1 most of our variants and for the smallest variant we use 1:1:2:1. For our models, width doubles at each new stage. We use configurations of [48, 96, 192, 384], [64, 128, 256, 512] and [76, 152, 304, 608] in this paper. The tiny("T") variants use MLP expansion ratio of 3. Rest of the variants use an MLP expansion ratio of 4.

B.5. 3D Hand mesh estimation

Architecture As shown in Figure 5, our model uses simple regression layers to regress weak perspective camera, pose and shape parameters of the MANO model [45]. These layers are single fully connected layers unlike deep regression layers used in [16]. We regress 6D rotations [74] for all joints in MANO model. There are 3 losses to minimize in our framework, L_{V3D}, 3D vertex loss between predicted mesh and ground truth mesh. L_{J3D}, 3D joint loss between predicted 3D joints and ground truth 3D joints. L_{J2D}, 2D joint loss between projected 3D joints and ground 2D keypoints.

Setup We train our model on FreiHand [75] dataset, that contains 130,240 training images and 3,960 test images. Following METRO [32], we train our model on 224$\times$224 images from the dataset using Adam optimizer. The models are trained for 200 epochs, with an initial learning rate of 1e-4 and decayed by a factor of 10 after 100 epochs. We initialize the backbone with weights obtained by pretraining on ImageNet-1k dataset. The weighting for all the losses in our framework is set to 1.0.

Results Figure 6 shows qualitative results from our framework on FreiHand test set. Even though our model is simple, it can model complicated gestures. Our model predicts reliable poses even in the presence of occlusion from hand-held objects.

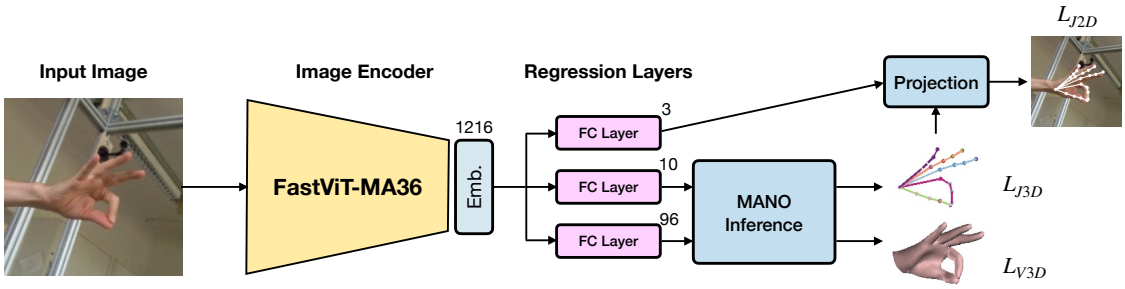


Figure 5: Overview of 3d hand mesh estimation framework.

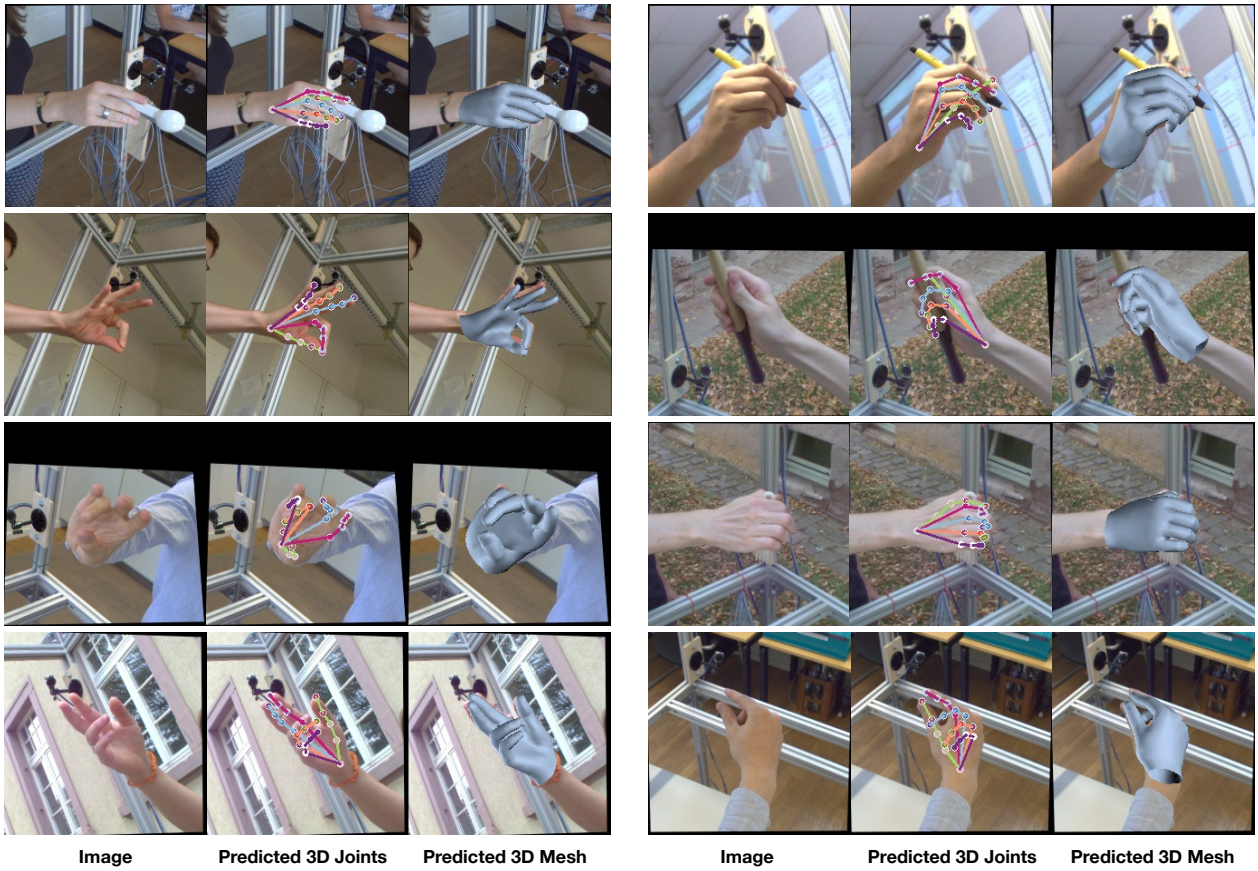


Figure 6: Qualitative results from our framework on FreiHand test set. 3D predictions are projected on to the image using weak perspective camera model, parameters for this camera model is also predicted by the model.