EdgeViTs: Competing Light-weight CNNs on Mobile Devices with Vision Transformers *Supplementary Materials*

Junting Pan[1⋆], Adrian Bulat[2], Fuwen Tan[2], Xiatian Zhu[2], Lukasz Dudziak[2], Hongsheng Li[1], Georgios Tzimiropoulos[2,3] and Brais Martinez[2]

[1] The Chinese University of Hong Kong
[2] Samsung AI Cambridge
[3] Queen Mary University of London

A Computing Complexity

We calculate the computational cost of spatial context modeling involved in our proposed LGL bottleneck. We omit point-wise operations for simplicity as the key difference is on the spatial modeling part. Let us assume an input $X \in \mathcal{R}^{h \times w \times c}$ where h, w, c denotes the height, the width, and the channel dimension, respectively. The cost of the local aggregation is $\mathcal{O}(k^2hwc)$, where k^2 is the local group size. By selecting one delegate out of r^2 tokens with r the sub-sampling rate, the complexity of our Sparse Global Self-Attention is then $\mathcal{O}(\frac{h^2w^2}{r^4}c)$. Finally, the local propagation step takes a cost of $\mathcal{O}(r^2hwc)$. Putting all these together we have a total cost of LGL is $\mathcal{O}(k^2hwc + \frac{h^2w^2}{r^4}c + r^2hwc)$. When comparing with the cost of a standard multi-head self-attention $\mathcal{O}(h^2w^2c)$, we can see that our LGL significantly reduces the computation overhead when $k \ll h, w$; and $r > 1$. In our experiments, for simplicity we set $k = 3$, and r to (4,2,2,1) for the four stages.

B Implementation Details

In Algorithm 1, we provide Pytorch Style pseudo-code for all the build blocks of our EdgeViTs. All variants of EdgeViTs can be built upon these components according to the schematic overview (Fig. 2a of the main paper), and the model configuration parameters (Table 1a. of the main paper).For more details with the ablation studies in the main paper, we have replaced or removed one of these blocks with the details given below.

(1) In Table 4a, for the case of w/o LA, we aim to test the importance of separate local and global context modeling. Thus we remove both `LocalAgg` and `GlobalSparseAttn`, and instead use the Spatial-Reduced Self-Attention[4] introduced in PVT [7], resulting in a single Self-Attention Block for both local and

⋆ Work done during an internship at Samsung AI Cambridge.
[4] https://github.com/whai362/PVT/blob/v2/classification/pvt_v2.py#L54-L126

global context modeling. For the case of LA(LSA) we simply replace `LocalAgg` with Local-grouped Self-Attention[5] introduced in [1].

(2) In Table 4b, we replace the default sampler (**Center**) with **Avg** and **Max** functions which can be implemented with `AvgPool2d()` and `MaxPool2d()` in Pytorch [4], respectively. Note, for both cases the kernel size is set to `sample_rate`.

(3) In Table 4c, in the case of w/o LP, we replace our `GlobalSparseAttn` with Spatial-Reduce Self-Attention from PVT [7], but different from w/o LA, we keep the `LocalAgg`. For the case of LP(Bilinear), the `LocalProp` is instantiated as a bilinear interpolation function (`Upsample(mode='bilinear')` in Pytorch).

Note, the number of layers for each of these variants is down-scaled to have 0.5GFLOPs for fair comparison.

C Accuracy-Speed Pareto-Optimal Models

In order to facilitate the Accuracy vs. Speed interpretation. We identify pareto optimal models when comparing trade-off between accuracy and latency [2]. In our context, the accuracy-latency pareto-optimal models are defined as those upon which no other models can improve in either accuracy or latency without degrading other metrics. As shown in Fig. **??**, our EdgeViTs are well comparable with best efficient CNNs [5,3,6], whilst significantly dominating over all prior ViT counterparts. Specifically, EdgeViTs are all pareto-optimal in both trade-offs.

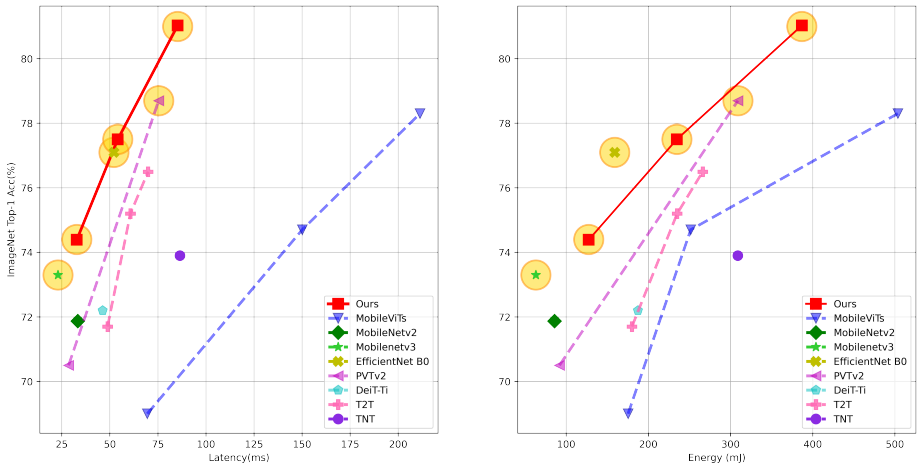

Fig. 1. Accuracy/Latency and Accuracy/Energy trade off on ImageNet-1K. Note that, all three variants of our EdgeViTs are pareto-optimal, which are highlighted with amber cercle. *Testing device*: Samsung Galaxy S21 (latency), Snapdragon 888 Hardware Development Kit (energy).

[5] https://github.com/Meituan-AutoML/Twins/blob/main/gvt.py#L32-L71

Model	AP (%)	CPU (s)	Energy (J)	Efficiency (%/msW)
PVTv2-B0	37.2	$1.34_{\pm 0.05}$	$3.50_{\pm 0.77}$	0.011
ResNet18	31.8	$0.58_{\pm 0.02}$	$2.22_{\pm 0.35}$	0.014
PVTv1-Tiny	36.7	$1.91_{\pm 0.18}$	$4.40_{\pm 0.94}$	0.008
PVTv2-B1	41.2	$2.81_{\pm 0.26}$	$5.29_{\pm 1.49}$	0.008
EdgeViT-XXS	38.7	$0.59_{\pm 0.02}$	$2.02_{\pm 0.58}$	0.019
EdgeViT-XS	40.6	$0.90_{\pm 0.03}$	$2.89_{\pm 0.66}$	0.014
EdgeViT-S	43.4	$1.88_{\pm 0.05}$	$4.36_{\pm 1.06}$	0.010

Table 1. Detection Efficiency. Input size: 800×800

Model	mIoU (%)	CPU (s)	Energy (J)	Efficiency (%/msW)
PVTv2-B0	37.2	$0.35_{\pm 0.01}$	$1.10_{\pm 0.30}$	0.034
ResNet18	32.9	$0.23_{\pm 0.01}$	$1.03_{\pm 0.20}$	0.032
PVTv1-Tiny	35.7	$0.49_{\pm 0.02}$	$1.63_{\pm 0.32}$	0.022
PVTv2-B1	42.5	$0.75_{\pm 0.03}$	$2.13_{\pm 0.82}$	0.020
EdgeViT-XXS	39.7	$0.19_{\pm 0.01}$	$0.71_{\pm 0.11}$	0.056
EdgeViT-XS	41.4	$0.31_{\pm 0.01}$	$1.11_{\pm 0.28}$	0.037
EdgeViT-S	45.9	$0.52_{\pm 0.02}$	$1.73_{\pm 0.36}$	0.027

Table 2. Segmentation Efficiency. Input size: 512×512

D Efficiency in detection/segmentation

This work proposes a genetic transformer-based network and demonstrate its efficacy when used as the backbone in detection/segmentation. We provide a evaluation by measuring only the inference time, energy and efficiency of the backbones for detection/segmentation. As shown in Tab. 1 and 2, EdgeViTs demonstrate higher efficiency compared to the baselines.

Algorithm 1 EdgeViTs Building Blocks, PyTorch-like Code

```
class LocalAgg():
    def __init__(self, dim):
        self.conv1 = Conv2d(dim, dim, 1)
        self.conv2 = Conv2d(im, dim, 3, padding=1, groups=dim)
        self.conv3 = Conv2d(dim, dim, 1)
        self.norm1 = BatchNorm2d(dim)
        self.norm2 = BatchNorm2d(dim)

    def forward(self, x):
        """
        [B, C, H, W] = x.shape
        """
        x = self.conv1(self.norm1(x))
        x = self.conv2(x)
        x = self.conv3(self.norm2(x))
        return x

class GlobalSparseAttn():
    def __init__(self, dim, sample_rate, scale):
        self.scale = scale
        self.qkv = Linear(dim, dim * 3)
        self.sampler = AvgPool2d(1, stride=sample_rate)
        kernel_size=sr_ratio
        self.LocalProp = ConvTranspose2d(dim, dim, kernel_size, stride=sample_rate, groups=dim
            )
        self.norm = LayerNorm(dim)
        self.proj = Linear(dim, dim)

    def forward(self, x):
        """
        [B, C, H, W] = x.shape
        """
        x = self.sampler(x)
        q, k, v = self.qkv(x)

        attn = q @ k * self.scale
        attn = attn.softmax(dim=-1)
        x = attn @ v

        x = self.LocalProp(x)
        x = self.proj(self.norm(x))
        return x

class DownSampleLayer():
    def __init__(self, dim_in, dim_out, downsample_rate):
        self.downsample = Conv2d(dim_in, dim_out, kernel_size=downsample_rate, stride=
            downsample_rate)
        self.norm = LayerNorm(dim_out)

    def forward(self, x):
        x = self.downsample(x)
        x = self.norm(x)
        return x

class PatchEmbed():
    def __init__(self, dim):
        self.embed = Conv2d(dim, dim, 3, padding=1, groups=dim)

    def forward(self, x):
        return x + self.embed(x)

class FFN():
    def __init__(self, dim):
        self.fc1 = nn.Linear(dim, dim*4)
        self.fc2 = nn.Linear(dim*4, dim)

    def forward(self, x):
        x = self.fc1(x)
        x = GELU(x)
        x = self.fc2(x)
        return x
```

References

1. Chu, X., Tian, Z., Wang, Y., Zhang, B., Ren, H., Wei, X., Xia, H., Shen, C.: Twins: Revisiting the design of spatial attention in vision transformers. Advances on Neural Information Processing Systems (2021)
2. Deb, K.: Multi-objective optimization. In: Search methodologies, pp. 403–449. Springer (2014)
3. Howard, A., Pang, R., Adam, H., Le, Q.V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., Zhu, Y.: Searching for MobileNetV3. In: IEEE International Conference on Computer Vision (2019)
4. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. In: Advances on Neural Information Processing Systems (2019)
5. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: IEEE Conference on Computer Vision and Pattern Recognition (2018)
6. Tan, M., Le, Q.: EfficientNet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning (2019)
7. Wang, W., Xie, E., Li, X., Fan, D.P., Song, K., Liang, D., Lu, T., Luo, P., Shao, L.: PVTv2: Improved baselines with pyramid vision transformer. Computational Visual Media (2022)