Latency-aware Spatial-wise Dynamic Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Spatial-wise dynamic convolution has become a promising approach to improving the inference efficiency of deep networks. By allocating more computation to the most informative feature pixels, such an *adaptive* inference paradigm alleviates the spatial redundancy in image features and reduces a considerable amount of unnecessary computation. However, the *theoretical* efficiency achieved by previous methods can hardly translate into the *realistic* speedup, especially on the multi-core processors (e.g. GPUs). The key challenge is that the existing literature has only focused on designing algorithms with minimal *computation*, ignoring the fact that the practical latency can also be influenced by *scheduling strategies* and *hardware properties*. To bridge the gap between the theoretical computation and the practical efficiency, we propose a *latency-aware* spatial-wise dynamic network (LASNet), which performs *coarse-grained* spatially adaptive inference under the guidance of a novel *latency prediction model*. This latency prediction model can efficiently estimate the inference latency of dynamic networks by simultaneously considering the algorithms, the scheduling strategies, and the hardware properties. We use the latency predictor to guide both the algorithm design and the scheduling optimization on various hardware platforms. Experiments on image classification demonstrate that the proposed framework significantly improves the trade-off between the accuracy and the inference efficiency of deep networks. For example, the average latency of a ResNet-101 on the ImageNet validation set could be reduced by 23% and 45% on a server GPU (Nvidia Tesla-V100) and an IoT device (Nvidia Jetson TX2 GPU) respectively without sacrificing the accuracy.

1 Introduction

Dynamic neural networks [6] have attracted great research interests in recent years. Compared to static models [8, 14, 10, 20] which treat different inputs equally during inference, dynamic networks can allocate the computation in a *data-dependent* manner. For example, they can conditionally skip the computation of network layers [12, 29, 27] and convolutional channels [16, 1], or perform *spatially* adaptive inference on the most informative image regions (e.g. the foreground areas) [5, 4, 28, 32, 30, 7]. Spatial-wise dynamic networks, which typically decide whether to compute each feature pixel with plug-in *masker* modules [4, 28, 32, 7] (see Figure 1 (a)), have shown very promising results in improving the inference efficiency of convolution neural networks (CNNs).

Despite the remarkable *theoretical* efficiency achieved by spatial-wise dynamic networks [4, 28, 32], researchers have found it challenging to translate the theoretical results into *realistic* speedup, especially on some multi-core processors, e.g., GPUs [32, 2, 7]. The challenges are two-fold: 1) most previous approaches [4, 28, 32] perform spatially adaptive inference at the finest *granularity*: every *pixel* is flexibly decided whether to be computed or not. Such flexibility induces non-contiguous *memory access* [32] and requires specialized *scheduling strategies* (Figure 1 (b)); 2) the existing literature has only adopted the *hardware-agnostic* FLOPs (floating-point operations) as an inaccurate proxy for the efficiency, lacking latency-aware guidance on the algorithm design. For dynamic

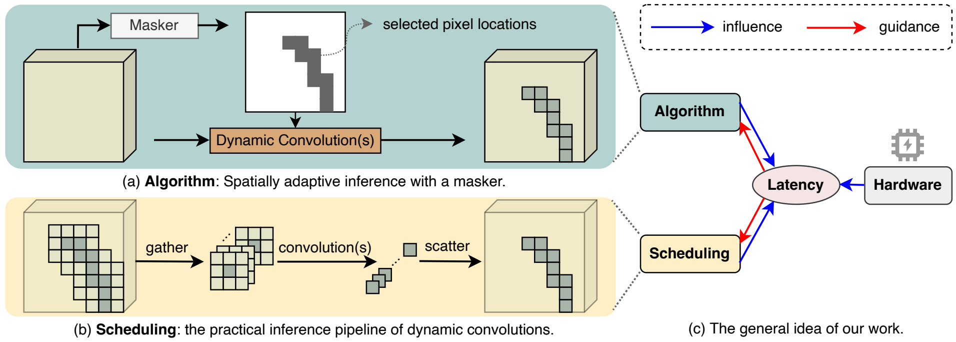


Figure 1: An overview of our method. (a) illustrates the spatially adaptive inference *algorithm*; (b) is the *scheduling* strategy; and (c) presents the three key factors to the practical latency. For a given hardware, the latency is used to *guide* our algorithm design and scheduling optimization.

networks, the adaptive computation with sub-optimal scheduling strategies further enlarges the discrepancy between the theoretical FLOPs and the practical latency. Note that it has been validated by previous works that the latency on CPUs has a strong correlation with FLOPs [7, 32]. Therefore, we mainly focus on the GPU platform in this paper, which is more challenging and less explored.

We address the above challenges by proposing a *latency-aware* spatial-wise dynamic network (LASNet). Three key factors to the inference latency are considered: the *algorithm*, the *scheduling strategy*, and the *hardware properties*. Given a target hardware device, we directly use the latency, rather than the FLOPs, to *guide* our algorithm design and scheduling optimization (see Figure 1 (c)).

Because the memory access pattern and the scheduling strategies in our dynamic operators differ from those in static networks, the libraries developed for static models (e.g. cuDNN) are sub-optimal for the acceleration of dynamic models. Without the support of libraries, each dynamic operator requires scheduling optimization, code optimization, compiling, and deployment for each device. Therefore, it is laborious to evaluate the network latency on different hardware platforms. To this end, we propose a novel *latency prediction model* to efficiently estimate the realistic latency of a network by simultaneously considering the aforementioned three factors. Compared to the hardware-agnostic FLOPs, our predicted latency can better reflect the practical efficiency of dynamic models.

Guided by this latency prediction model, we establish our latency-aware spatial-wise dynamic network (LASNet), which adaptively decides whether to allocate computation on feature *patches* instead of *pixels* [4, 28, 32] (Figure 2 top). We name this behavior as spatially adaptive inference at a *coarse granularity*. While less flexible than the pixel-level adaptive computation in previous works [4, 28, 32], it facilitates more contiguous memory access, benefiting the realistic speedup on hardware. The scheduling strategy and the implementation are further ameliorated for faster inference.

It is worth noting that LASNet is designed as a general framework in two aspects: 1) the coarse-grained spatially adaptive inference paradigm can be conveniently implemented in various CNN backbones, e.g., RegNets [22] and ResNets [8]; and 2) the latency prediction model is an off-the-shell tool which can be directly used for various computing platforms (e.g. server GPUs and IoT devices).

We evaluate the performance of LASNet on multiple CNN architectures on image classification, object detection, and instance segmentation tasks. Experiment results show that our LASNet improves the efficiency of deep CNNs both theoretically and practically. For example, the inference latency of ResNet-101 is reduced by 23% and 45% on an Nvidia Tesla V100 GPU and an Nvidia Jetson TX2 GPU, respectively, without sacrificing the accuracy on the ImageNet [3] validation set. Moreover, the proposed method outperforms various lightweight networks in a low-FLOPs regime.

Our main contributions are summarized as follows:

1) We propose LASNet, which performs coarse-grained spatially adaptive infee.g., nce guided by the practical latency instead of the theoretical FLOPs. To the best of our knowledge, LASNet is the first framework that directly considers the real latency in the design phase of dynamic neural networks;

2) We propose a latency prediction model, which efficiently estimates the latency of dynamic operators by simultaneously considering the algorithm, the scheduling strategy, and the hardware properties;

3) Experiments on image classification, object detection, and instance segmentation tasks verify that our proposed LASNet can effectively improve the practical efficiency of different CNN architectures.

2 Related works

Spatial-wise dynamic network is a common type of dynamic neural networks [6]. Compared to static models which treat different feature locations evenly during inference, these networks perform spatially adaptive inference on the most informative regions (e.g., foregrounds), and reduce the unnecessary computation on less important areas (e.g., backgrounds). Existing works mainly include three levels of dynamic computation: resolution level [33, 34], region level [30] and pixel level [4, 28, 32]. The former two generally manipulate the network inputs [30, 34] or require special architecture design [33]. In contrast, pixel-level dynamic networks can flexibly skip the convolutions on certain feature pixels in arbitrary CNN backbones [4, 28, 32]. Despite its remarkable *theoretical* efficiency, the pixel-wise dynamic computation brings considerable difficulty to achieving *realistic* speedup on multi-core processors, e.g., GPUs. Compared to the previous approaches [4, 28, 32] which only focus on reducing the theoretical computation, we propose to directly use the latency to guide our algorithm design and scheduling optimization.

Hardware-aware network design. To bridge the gap between theoretical and practical efficiency of deep models, researchers have started to consider the real latency in the network design phase. There are two lines of works in this direction. One directly performs speed tests on targeted devices, and summarizes some guidelines to facilitate *hand-designing* lightweight models [20]. The other line of work *searches* for fast models using the neural architecture search (NAS) technique [26, 31]. However, all existing works try to build *static* models, which have intrinsic computational redundancy by treating different inputs in the same way. However, speed tests for dynamic operators on different hardware devices can be very laborious and impractical. In contrast, our proposed latency prediction model can efficiently estimate the inference latency on any given computing platforms by simultaneously considering algorithm design, scheduling strategies and hardware properties.

3 Methodology

In this section, we first introduce the preliminaries of spatially adaptive inference, and then demonstrate the architecture design of our LASNet. The latency prediction model is then explained, which guides the granularity settings and the scheduling optimization for LASNet. We further present the implementation improvements for faster inference, followed by the training strategies.

3.1 Preliminaries

Spatially adaptive inference. The existing spatial-wise dynamic networks are usually established by attaching a masker $\mathcal{M}$ in each convolutional block of a CNN backbone (see Figure 1 (a)). Specifically, let $\mathbf{x}\in\mathbb{R}^{H\times W\times C}$ denote the input of a block, where H and W are the feature height and width, and C is the channel number. The masker $\mathcal{M}$ takes $\mathbf{x}$ as input, and generates a binary-valued spatial mask $\mathbf{M}=\mathcal{M}(\mathbf{x})\in\{0,1\}^{H\times W}$. Each element of $\mathbf{M}$ determines whether to perform convolution operations on the corresponding location of the output feature. The unselected regions will be filled with the values from the input [4, 28] or obtained via interpolation [32]. We define the *activation rate* of a block as $r=\frac{\sum_{i,j}\mathbf{M}_{i,j}}{H\times W}$, representing the ratio of the calculated pixels.

Scheduling strategy. During inference, the current scheduling strategy for spatial-wise dynamic convolutions generally involve three steps [23] (see Figure 1 (b)): 1) *gathering*, which re-organizes the selected pixels (if the convolution kernel size is greater than 1×1, the neighbors are also required) along the *batch* dimension; 2) *computation*, which performs convolution on the gathered input; and 3) *scattering*, which fills the computed pixels on their corresponding locations of the output feature.

Limitations. Compared to performing convolutions on the entire feature map, the aforementioned scheduling strategy reduces the computation while bringing considerable overhead to *memory access* due to the mask generation and the non-contiguous memory access. Such overhead would increase the overall latency, especially when the *granularity* of dynamic convolution is the finest pixel level.

3.2 Architecture design

Spatial granularity. As mentioned above, *pixel*-level dynamic convolutions [4, 28, 32] raise substantial challenges to achieving realistic speedup on multi-core processors due to the non-contiguous memory access. To this end, we propose to optimize the *granularity* of spatially adaptive inference. Specifically, take the commonly used bottleneck structure in [8] as an example, our coarse-grained spatial-wise dynamic convolutional block is illustrated in Figure 2. Instead of directly producing

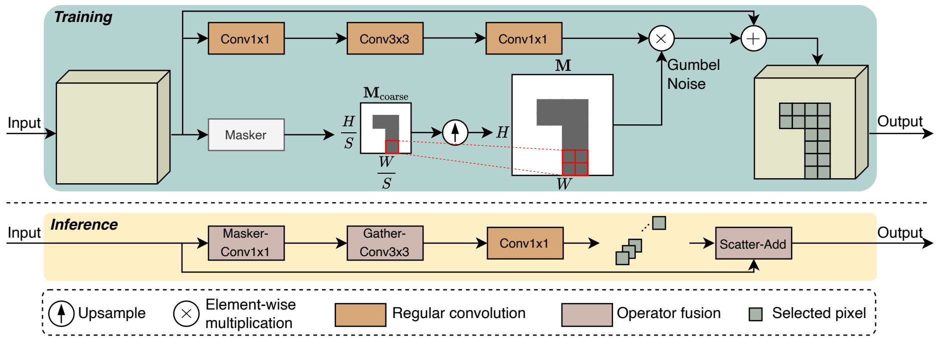


Figure 2: Our proposed LASNet block. Top: we first generate a low-resolution spatial mask $\mathbf{M}_{\text{coarse}}$, which is then upsampled to obtain the mask $\mathbf{M}$ with the same size as the output feature. Gumbel Softmax [15, 21] is used for end-to-end training (Sec. 3.5). Bottom: the scheduling optimimzation is performed to decrease the memory access for faster inference (Sec. 3.4).

a mask with the shape of $H \times W$, we first generate a low-resolution mask $\mathbf{M}_{\text{coarse}} \in \{0, 1\}^{\frac{H}{S} \times \frac{W}{S}}$, where S is named as the *spatial granularity*. Each element in $\mathbf{M}_{\text{coarse}}$ determines the computation of an $S \times S$-sized feature patch. For instance, the feature size in the first ResNet stage[1] is 56×56. Then the possible choices for S are $\{1, 2, 4, 7, 8, 14, 28, 56\}$. The mask $\mathbf{M}_{\text{coarse}}$ is then upsampled to the size of $H \times W$. Notably, $S = 1$ means that the granularity is still at the pixel level as previous methods [4, 28, 32]. In this paper, the other extreme situation ($S = 56$) is not considered, when the masker directly determines whether to skip the whole block (i.e. layer skipping [27, 29]). The masker is composed of a pooling layer followed by a 1×1 convolution.

Differences to existing works. Without using the interpolation operation [32] or the carefully designed two-branch structure [7], the proposed block architecture is simple and sufficiently general to be plugged into most backbones with minimal modification. Our formulation is mostly similar to that in [28], which could be viewed as a variant of our method with the spatial granularity S=1 for all blocks. Instead of performing spatially adaptive inference at the finest pixel level, our granularity S is optimized under the guidance of our *latency prediction model* (details are presented in the following Sec. 4.2) to achieve *realistic speedup* on target computing platforms.

3.3 Latency prediction model

As stated before, it is laborious to evaluate the latency of dynamic operators on different hardware platforms. To efficiently seek preferable granularity settings on arbitrary hardware devices, we propose a latency prediction model $\mathcal{G}$, which can directly *predict* the delay of executing dynamic operators on any target devices. For a spatial-wise dynamic convolutional block, the latency predictor $\mathcal{G}$ takes the hardware properties $\mathbf{H}$, the layer parameters $\mathbf{P}$, the spatial granularity S, and the activation rate r as input and predicts the latency ℓ of a dynamic convolutional block: $\ell = \mathcal{G}(\mathbf{H}, \mathbf{P}, S, r)$.

Hardware modeling. We model a hardware device as multiple processing engines (PEs), and parallel computation can be executed on these PEs. As shown in Figure 3, we model the memory system as a three-level structure [9]: 1) off-chip memory, 2) on-chip global memory, and 3) memory in PE. Such a hardware model enables us to accurately predict the cost on both *data movement* and *computation*.

Latency prediction. When simulating the data movement procedure, the efficiency of non-contiguous memory accesses under different granularity S settings is considered. As for the computation latency, it is important to adopt a proper scheduling strategy to increase the parallelism of computation. Therefore, we search for the optimal scheduling (the configuration of tiling and in-PE parallel computing) of dynamic operations to maximize the utilization of hardware resources. A more detailed description of our latency prediction model is presented in the supplementary material.

Empirical validation. We take the first block in ResNet-101 as an example and vary the activation rate r to evaluate the performance of our prediction model. The comparison between our predictions and the real testing latency on the Nvidia V100 GPU is illustrated in Figure 4, from which we can observe that our predictor can accurately estimate the real latency in a wide range of activation rates.

[1] Here we refer to a stage as the cascading of multiple blocks which process features with the same resolution.

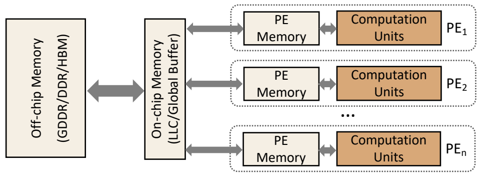


Figure 3: Our hardware model.

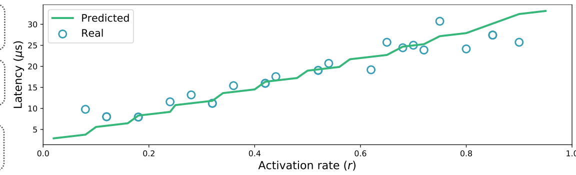


Figure 4: Latency prediction results.

3.4 Implementation details

We use general optimization methods like fusing activation functions and batch normalization layers into convolution layers. We also optimize the specific operators in our spatial-wise dynamic convolutional blocks as follows (see also Figure 2 for an overview).

Fusing the masker and the first convolution. As mentioned in Sec. 3.1, the masker in each block consumes very little computation, but it takes the whole feature map as input. Therefore, it is a *memory-bounded* operation (the inference time is mainly spent on memory access). Since the masker and the first convolution in the block share the same input, there is an opportunity to fuse these two operations to avoid the repeated access of the input data. Note that a spatial-wise dynamic convolution requires the output of the masker. If we fuse the two layers, the first convolution will be changed to a static operation, which may increase the inference latency. There exists a threshold of activation rate r_{th}, when $r > r_{\text{th}}$, the overall latency can be reduced. We decide whether to fuse them according to the average activation rate. See more details in the supplementary material.

Fusing the gather operation and the dynamic convolution. Traditional approaches first gather the input pixels of the first dynamic convolution in a block (see Figure 1 (b)). The gather operation is also a *memory-bounded* operation. Furthermore, when the size of the convolution kernel exceeds 1×1, the area of input patches may overlap, resulting in repeated memory load/store. We fuse the gather operation into the dynamic convolution to reduce the memory access.

Fusing the scatter operation and the add operation. Traditional approaches scatter the output pixels of the last dynamic convolution, and then execute the element-wise addition (see Figure 1 (b)). We fuse these two operators to reduce the memory access. The ablation study in Sec. 4.4 validates the effectiveness of the proposed fusing methods.

3.5 Training

Optimization of non-differentiable maskers. The masker modules are required to produce binary-valued spatial masks for making discrete decisions, and cannot be directly optimized with back propagation. Following [32, 28, 7], we adopt straight-through Gumbel Softmax [15, 21] to train the network in an end-to-end fashion. Specifically, let $\tilde{\mathbf{M}} \in \mathbb{R}^{H\times W\times 2}$ denote the output of the mask generator. The decisions are obtained with the argmax function during inference. In the training phase, a differentiable approximation is defined by replacing the argmax operation with a Softmax:

$$\hat{\mathbf{M}} = \frac{\exp\left\{\left(\log\left(\tilde{\mathbf{M}}_{:,:,0}\right) + \mathbf{G}_{:,:,0}\right)/\tau\right\}}{\sum_{k=0}^{1}\exp\left\{\left(\log\left(\tilde{\mathbf{M}}_{:,:,k}\right) + \mathbf{G}_{:,:,k}\right)/\tau\right\}} \in [0,1]^{H\times W}, \tag{1}$$

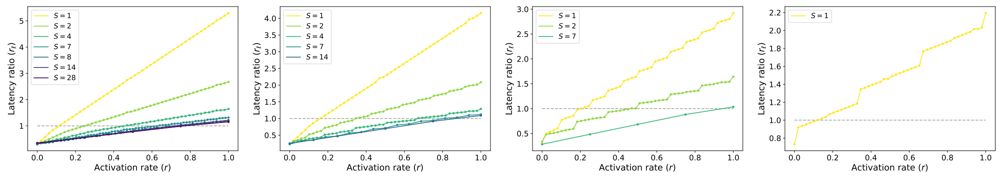
where τ is the Softmax temperature. Following the common practice [28, 7], we let τ decrease exponentially from 5.0 to 0.1 in training to facilitate the optimization of maskers.

Training objective. The FLOPs of each spatial-wise dynamic convolutional block can be calculated based on our defined activation rate r [28]. Then we can obtain the FLOPs of the overall dynamic network F_{dyn}. Let F_{stat} denotes the FLOPs of its static counterpart. We optimize their ratio to approximate a target $0 < t < 1$:$L_{\text{FLOPs}} = (\frac{F_{\text{dyn}}}{F_{\text{stat}}} - t)^2$. In addition, we define loss item L_{bounds} as in [28] to constrain the upper bound and the lower bound of activation rates in early training epochs.

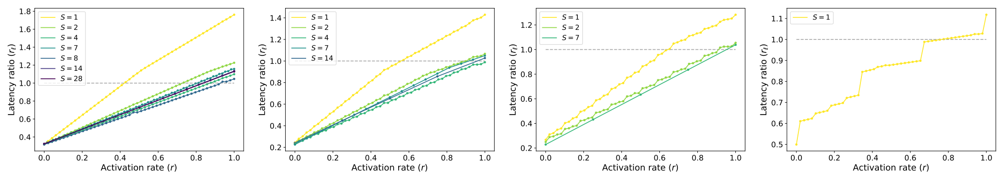
We further propose to leverage the static counterparts of our dynamic networks as "teachers" to guide the optimization procedure. Let $\mathbf{y}$ and $\mathbf{y}'$ denote the output logits of a dynamic model ("student") and its "teacher", respectively. Our final loss can be written as

$$L = L_{\text{task}} + \alpha(L_{\text{FLOPs}} + L_{\text{bounds}}) + \beta T^2 \cdot \mathrm{KL}(\sigma(\mathbf{y}/T)||\sigma(\mathbf{y}'/T)), \tag{2}$$

where L_{task} represents the task-related loss, e.g., cross-entropy loss in image classification. $\mathrm{KL}(\cdot||\cdot)$ denotes the Kullback–Leibler divergence, and α, β are the coefficients balancing these items. We use σ to denote the log-Softmax function, and T is the temperature for computing KL-divergence.



(a) Relationship between r_ℓ and r for LAS-ResNet blocks on the Nvidia Tesla V100 GPU.



(b) Relationship between r_ℓ and r for LAS-RegNetY-800MF blocks on the Nvidia Jetson TX2 GPU.

Figure 5: Latency prediction results of LAS-ResNet blocks on V100 (a) and LAS-RegNet blocks on TX2 (b). For both networks, we plot the relationship between the latency ratio r_ℓ and the activation rate r for the blocks in 4 stages with the convolutional stride 1. The practical efficiency is only improved when $r_\ell < 1$. Note that $S = 1$ can harm the practical latency even for a small r (reduced computation), while a larger S will alleviate this problem. See detailed analysis in Sec. 4.2.

4 Experiments

In this section, we first introduce the experiment settings in Sec. 4.1. Then the latency of different granularity settings are analyzed in Sec. 4.2. The performance of our LASNet on ImageNet is further evaluated in Sec. 4.3, followed by the ablation studies in Sec. 4.4. Visualization results are illustrated in Sec. 4.5, and we finally validate our method on the object detection task (Sec. 4.6). The results on the instance segmentation task are presented in the supplementary material. For simplicity, we add "LAS-" as a prefix before model names to denote our LASNet, e.g., LAS-ResNet-50.

4.1 Experiment settings

Latency prediction. Various types of hardware platforms are tested, including a server GPU (Tesla V100), a desktop GPU (GTX1080) and IoT devices (e.g., Nvidia Nano and Jetson TX2). The major properties considered by our latency prediction model include the number of processing engines (#PE), the floating-point computation in a processing engine (#FP32), the frequency and the bandwidth. It can be observed that the server GPUs generally have a larger #PE than the IoT devices. The batch size is set as 1 for all dynamic models and computing platforms.

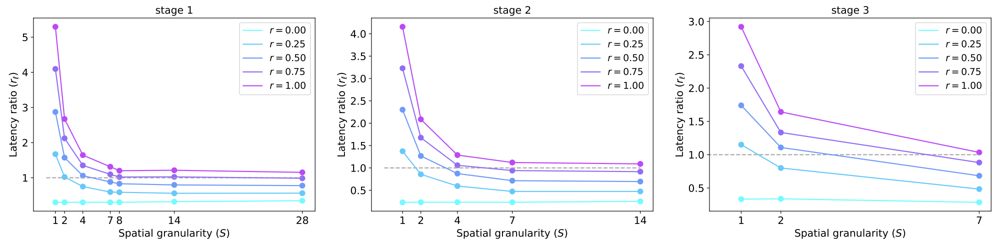
Image classification. The image classification experiments are conducted on the ImageNet [3] dataset. Following [28], we initialize the backbone parameter from a pre-trained checkpoint[2], and finetune the whole network for 100 epochs with the loss function in Eq. (2). We fix $\alpha = 10, \beta = 0.5$ and $T = 4.0$ for all dynamic models. More details are provided in the supplementary material.

4.2 Latency prediction results

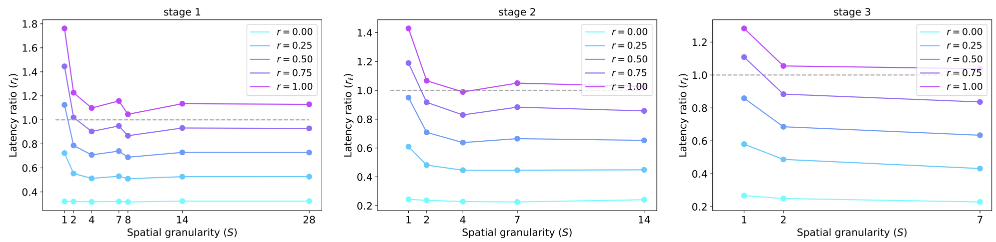
In this subsection, we present the latency prediction results of the spatial-wise dynamic convolutional blocks in two different models: LAS-ResNet-101 [8] (on V100) and LAS-RegNetY-800MF [22] (on TX2). All the blocks have the bottleneck structure with different channel numbers and convolution groups, and the RegNetY is equipped with Squeeze-and-Excitation (SE) [11] modules.

We first define ℓ_{dyn} as the latency of a spatial-wise dynamic convolutional block, and ℓ_{stat} as that of a static block without a masker. Their ratio is denoted as $r_\ell = \frac{\ell_{\text{dyn}}}{\ell_{\text{stat}}}$. We investigate the relationship between r_ℓ and the activation rate r (cf. Sec. 3.5) for different *granularity* settings. The results in Figure 5 demonstrate that: 1) pixel-level spatially adaptive inference (S=1) *cannot always* improve the practical efficiency. Such fine-grained adaptive inference is adopted by most previous works [28, 32], and our result can explain the reason why they can only achieve realistic speedup on less powerful CPUs [32] or specialized devices [2]; 2) a proper granularity $S > 1$ effectively alleviates this problem; 3) the advantage of coarse-grained spatially adaptive inference ($S > 1$) is more significant

[2]We use the torchvision pre-trained models at `https://pytorch.org/vision/stable/models.html`.



(a) Relationship between r_ℓ and S for LAS-ResNet blocks on V100.



(b) Relationship between r_ℓ and S for LAS-RegNetY-800MF blocks on Nvidia Jetson TX2 GPU.

Figure 6: The relationship between the latency ratio r_ℓ and the spatial granularity S.

on the server GPU with a larger #PE, as the finest granularity ($S = 1$) brings considerable overhead on the memory access and is not friendly to parallel computation.

The latency prediction results are further used to seek for a preferable granularity setting for the first 3 stages (we fix $S = 1$ for the last stage, where the feature resolution is 7×7). Therefore, we plot the relationship between r_ℓ and S in Figure 6. It can be observed that: 1) r_ℓ generally decreases with S increasing for a given r on V100; 2) an overly large S brings insignificant improvement on V100, and even harms the practical efficiency on the less powerful TX2. Therefore, we can simply set S_{net}=4-4-2-1 for the 4 stages in a network on TX2. As for the server GPU V100, S_{net}=8-4-7-1 will be appropriate for realistic speedup. The accuracy-latency plots in Figure 7 also validate this observation. More results of our latency prediction model are presented in the supplementary material.

4.3 ImageNet classification results

We now empirically evaluate our proposed LASNet on the ImageNet dataset. The network performance is measured in terms of the trade-off between classification accuracy and inference efficiency. Both theoretical (i.e. FLOPs) and practical efficiency (i.e. latency) are tested in our experiments.

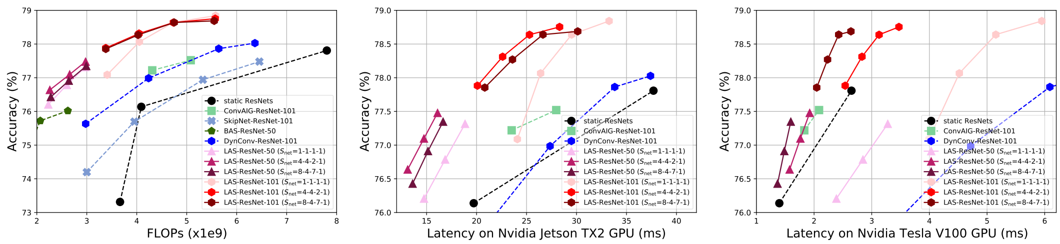
4.3.1 Standard baseline comparison: ResNets

We first establish our LASNet based on the standard ResNets [8]. Specifically, we build LAS-ResNet-50 and LAS-ResNet-101 by plugging our maskers in the two common ResNet structures.

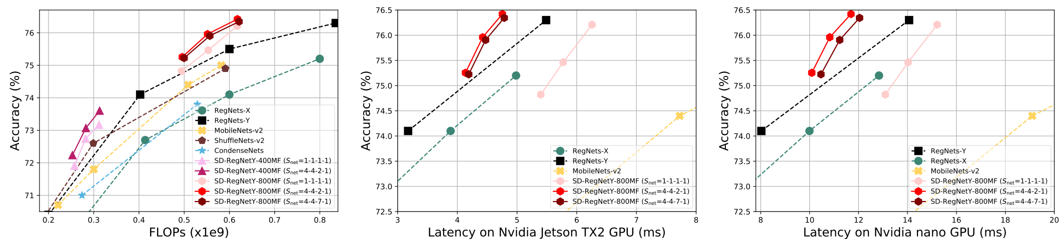
Compared baselines include various types of dynamic inference approaches: 1) layer skipping (SkipNet [29] and Conv-AIG [27]); 2) channel skipping (BAS [1]); and 3) pixel-level spatial-wise dynamic network (DynConv [28]). For our LASNet, we compare various settings of the spatial granularity S_{net}. We set training targets (cf. Sec. 3.5) $t \in \{0, 4, 0.5, 0.6, 0.7\}$ for our dynamic models to evaluate their performance in different sparsity regimes. We apply the same operator fusion (Sec. 3.4) for both our models and the compared baselines [27, 28] for fair comparison.

Results are presented in Figure 7a. On the left we plot the relationship of accuracy *v.s.* FLOPs. It can be observed that our LAS-ResNets with different granularity settings significantly outperform the competing dynamic neural networks. Surprisingly, coarse-grained spatially adaptive inference (S_{net}=4-4-2-1 and S_{net}=8-4-7-1 for the 4 stages) can achieve even higher accuracy when consuming similar FLOPs on ResNets, despite the sacrificed flexibility compared to S_{net}=1-1-1-1.

We compare the practical latency of three granularity settings in Figure 7a (middle on TX2 and right on V100) based on our latency prediction model. We can witness that although they achieve comparable theoretical efficiency (Figure 7a left), larger S is more hardware-friendly compared to the finest granularity. For example, the inference latency of LAS-ResNet-101 (S_{net}=1-1-1-1) is significantly higher than the ResNet-101 baseline on V100 (Figure 7a right), even though its theoretical computation is much smaller than that of the static model. However, larger granularities



(a) LAS-ResNet results.



(b) LAS-RegNetY results.

Figure 7: Experimental results on the ImageNet classification task. The proposed coarse-grained spatially adaptive inference is tested on standard ResNets (a) and lightweight RegNets (b).

(S_{net}=4-4-2-1 and S_{net}=8-4-7-1) can effectively improve the inference latency due to its lower burden on the memory access. The superiority of coarse-grained spatially adaptive inference diminishes (but still exists) on the less powerful IoT device, Nvidia Jetson TX2 (Figure 7a right). The advantage is more significant on the Tesla V100 GPU, because of its large number of processing engines (#PE). Moreover, the realistic speedup ratio r_ℓ is more close to the theoretical FLOPs ratio target t on TX2, because the latency is *computation-bounded* on such IoT devices. Interestingly, the optimal settings for the two different hardware platforms are different (S_{net}=4-4-2-1 is superior on TX2, while S_{net}=8-4-7-1 leads to higher efficiency on V100). Remarkably, the latency of ResNet-101 could be reduced by 23% and 45% on V100 and TX2 respectively without sacrificing the accuracy. The classification accuracy is increased by 1.9% with similar inference efficiency.

4.3.2 Lightweight baseline comparison: RegNets

We further evaluate our LASNet in lightweight CNN architectures, i.e. RegNets-Y [22]. Two different sized models are tested: RegNetY-400MF and RegNetY-800MF. Compared baselines include other types of efficient models, e.g., MobileNets-v2 [25], ShuffletNets-v2 [20] and CondenseNets [13].

The results are presented in Figure 7b. The x-axis for the three sub-figures are the FLOPs, the latency on TX2, and the latency on the Nvidia nano GPU, respectively. We can observe that our method outperforms various types of static models in terms of the trade-off between accuracy and efficiency.

More results on image classification are provided in the supplementary material.

4.4 Ablation studies

Operator fusion. We first conduct ablation studies to investigate the effect of our operator fusion introduced in Sec. 4.2. One convolutional block in the first stage of a LAS-ResNet-101 (S=4, r=0.6) is tested. We summarize the results in Table 1. It can be observed that every step of operator fusion benefits the practical latency of a block, as the overhead on memory access is effectively reduced.

Table 1: Ablation studies on operator fusion.

Masker-Conv1x1	Gather-Conv3x3	Scatter-Add	Latency (µs)
✗	✗	✗	103.52
✓	✗	✗	99.84
✓	✓	✗	95.76
✓	✓	✓	86.56

More granularities settings. We test various granularity settings on LAS-ResNet-101 to examine the effects of S in different stages. The results on the Tesla-V100 GPU are presented in Figure 8. It can be found that the finest granularity (S_{net}=1-1-1-1) leads to substantial inefficiency despite the reduced FLOPs (cf. Figure 7a left). Coarse-grained spatially adaptive inference in the first two stages (S_{net}=4-4-1-1) effectively reduces the inference latency. We further increase S in the third stage to 2 and 7, and this procedure consistently improves the realistic efficiency on the V100 GPU. It is worth noting that increasing the granularity S does

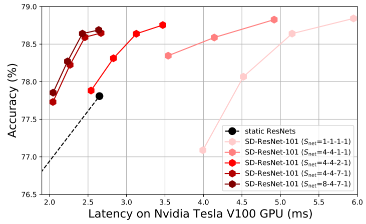


Figure 8: Ablation studies on S.

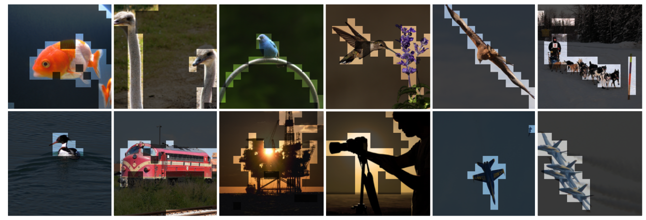


Figure 9: Visualization results.

Table 2: Object detection results on the COCO dataset.

Detection Framework	Backbone	Backbone FLOPs (G)	Backbone Latency (ms) V100	GTX1080	TX2	mAP (%)
Faster R-CNN	ResNet-101 (Baseline)	141.2	39.2	118.0	720.7	39.4
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.5)	**79.3**	47.8	91.6	**398.5**	39.8
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.5)	79.5	**37.3**	**86.0**	444.2	**40.0**
RetinaNet	ResNet-101 (Baseline)	141.2	39.2	118.0	720.7	38.5
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.5)	**77.8**	47.1	90.2	**392.1**	**39.3**
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.5)	79.4	**37.4**	**86.1**	443.8	**39.3**

not always improve the inference efficiency on other computing devices. For example, S_{net}=4-4-2-1 achieves a sweetspot on Nvidia Jetson TX2 (see Figure 7a middle).

4.5 Visualization

We visualize the masks generated by our masker in the third block of a LAS-ResNet-101 (S_{net}=4-4-2-1) in Figure 9. The brilliant areas correspond to the locations of 1 elements in a mask, and the computation on the dimmed regions is skipped by our dynamic model. It can be found that the masker is trained to accurately locate the most task-related regions (even the tiny aircraft at the corner), which helps reduce the unnecessary computation on background areas. Moreover, these results suggest that for the first stage, the granularity S=4 is sufficiently flexible to recognize the important regions, and a *win-win* can be achieved between accuracy and efficiency. Interestingly, the masker could select some objects that are *not labeled* for the sample, e.g., the flower beside the hummingbird and the human holding the camera. This suggests that our spatial-wise dynamic networks can automatically recognize the regions with semantics, and their capability is not limited by the classification labels. This property is helpful in some downstream tasks, such as object detection (Sec. 4.6), which requires detecting multiple classes and objects in an image.

4.6 Object detection results

We further evaluate our LASNet on the COCO [19] object detection task. The mean average precision (mAP), the average backbone FLOPs, and the average backbone latency on the validation set are used to measure the network performance. We test two commonly used detection frameworks: Faster R-CNN [24] with Feature Pyramid Network [17] and RetinaNet [18]. Thanks to the generality of our method, we can conveniently replace the backbones with ours pre-trained on ImageNet, and the whole models are finetuned on COCO with the standard setting for 12 epochs (see detailed setup in the supplementary material). The input images are resized to a short side of 800 and a long side not exceeding 1333. The results of our LAS-ResNet-101 with different S_{net} settings are presented in Table 2. It could be observed that our LASNet can reduce the practical latency on GTX1080 and TX2 by 27% and 45% respectively while improving the mAP of both detection frameworks.

5 Conclusion

In this paper, we propose to build *latency-aware* spatial-wise dynamic networks (LASNet) under the guidance of a *latency prediction model*. By simultaneously considering the algorithm, the scheduling strategy and the hardware properties, we can efficiently estimate the practical latency of spatial-wise dynamic operators on arbitrary computing platforms. Based on the empirical analysis on the relationship between the latency and the *granularity* of spatially adaptive inference, we optimize both the algorithm and the scheduling strategies to achieve realistic speedup on many multi-core processors, e.g., the Tesla V100 GPU and the Jetson TX2 GPU. Experiments on image classification, object detection and instance segmentation tasks validate that the proposed method significantly improves the practical efficiency of deep CNNs, and outperforms various competing approaches.

References

[1] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. In *ICLR*, 2020.

[2] Steven Colleman, Thomas Verelst, Linyan Mei, Tinne Tuytelaars, and Marian Verhelst. Processor architecture optimization for spatially dynamic neural networks. In *VLSI-SoC*, 2021.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[4] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *CVPR*, 2017.

[5] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017.

[6] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *TPAMI*, 2021.

[7] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Yitian Zhang, and Haojun Jiang. Spatially adaptive feature refinement for efficient inference. *TIP*, 2021.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[9] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[10] 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.

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[12] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018.

[13] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[15] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[16] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NeurIPS*, 2017.

[17] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[18] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[19] 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 *ECCV*, 2014.

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[21] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

[22] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020.

[23] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. SBNet: Sparse Blocks Network for Fast Inference. *CVPR*, 2018.

[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[26] 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*, pages 2820–2828, 2019.

[27] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

[28] Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting Spatial Sparsity for Faster Inference. In *CVPR*, 2020.

[29] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

[30] Yulin Wang, Kangchen Lv, Rui Huang, Shiji Song, Le Yang, and Gao Huang. Glance and focus: a dynamic approach to reducing spatial redundancy in image classification. In *NeurIPS*, 2020.

[31] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[32] Zhenda Xie, Zheng Zhang, Xizhou Zhu, Gao Huang, and Stephen Lin. Spatially adaptive inference with stochastic feature sampling and interpolation. In *ECCV*, 2020.

[33] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *CVPR*, 2020.

[34] Mingjian Zhu, Kai Han, Enhua Wu, Qiulin Zhang, Ying Nie, Zhenzhong Lan, and Yunhe Wang. Dynamic resolution network. *NeurIPS*, 2021.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 4.1 and Section 4.6, the public ImageNet and COCO datasets are cited.

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See the supplementary material.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code will be released if the paper is accepted.

   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Most important hyperparameters are given in the main paper, and more details are presented in the supplementary material.

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Since the experiment results on the large-scale dataset, ImageNet and COCO, are generally stable, we perform several repeat experiments in the early exploring phase, and dismiss the error bars in our main results as in most previous works.

   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The details are presented in the supplementary material.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes] See Section 4.1. We use the torchvision pre-trained models.

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]