SEC

# Exploring the Design Space of Efficient Deep Neural Networks

## -- The Fifth ACM/IEEE Symposium on Edge Computing

**Fuxun Yu[1], Dimitrios Stamoulis[2], Di Wang[2], Dimitrios Lymberopoulos[2], Xiang Chen[1]**

George Mason University[1], Microsoft[2]

GEORGE MASON UNIVERSITY

# Introduction

- Motivation: Efficient Deep Neural Network Design

# Efficiency on Static DNN Architecture Search

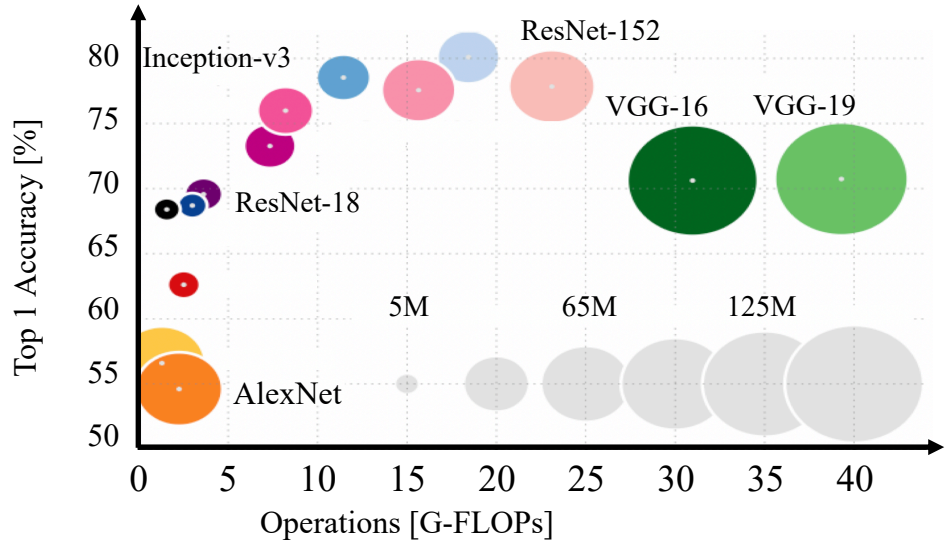
- Hardware-aware Design Space Optimization
- Single-Path Neural Architecture Search

# Efficiency on Dynamic DNN Execution

- Dynamic Feature Map Pruning
- Hardware-oriented Deployment Optimization

# Conclusion

# Motivation: Efficient Deep Neural Network Design



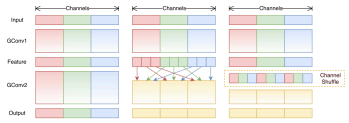

## Current DNN Challenges

- Heavy Computation Cost;
- High Energy Consumption;
- Long Latency, etc.

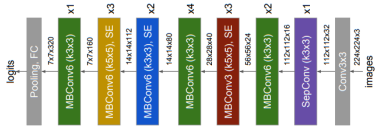
## Current DNN Optimizations

- Neural Architecture Design;
- Filter Pruning;
- Weight Quantization, etc.

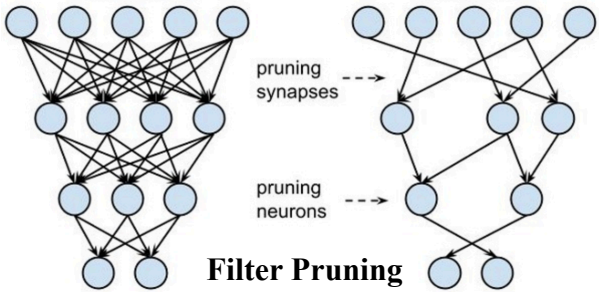
**MobileNet**



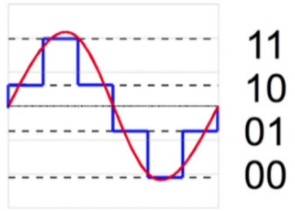
**ShuffleNet**



**MnasNet**



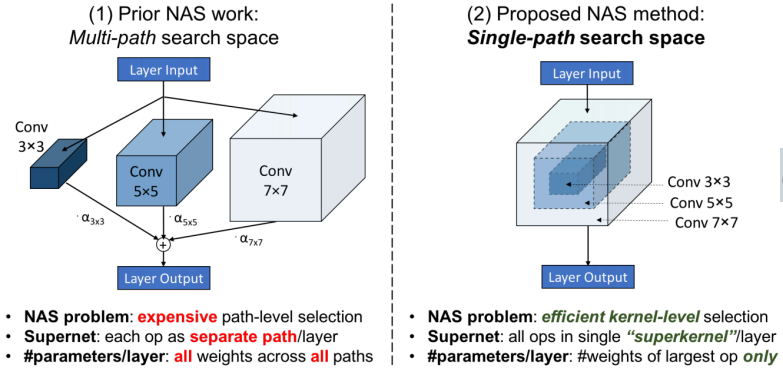

**Filter Pruning**




**Weight Quantization**

# Outline: Efficient Deep Neural Network Design

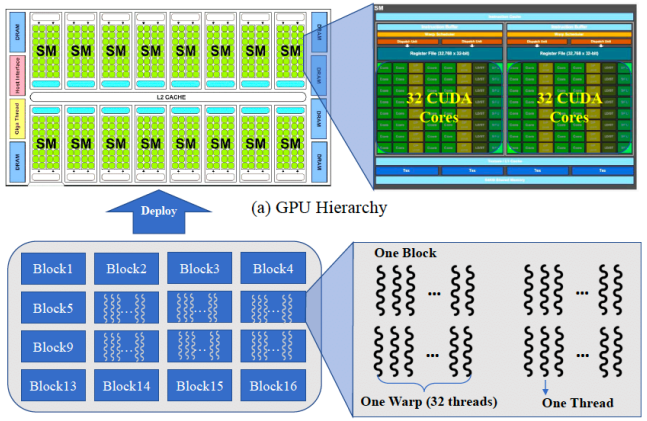
## Efficient Neural Architecture Search




- **NAS problem**: **expensive** path-level selection
- **Supernet**: each op as **separate path**/layer
- **#parameters/layer**: **all** weights across **all** paths

- **NAS problem**: ***efficient kernel-level*** selection
- **Supernet**: all ops in single ***"superkernel"***/layer
- **#parameters/layer**: #weights of largest op ***only***

## Hardware-aware Design Space Optimization

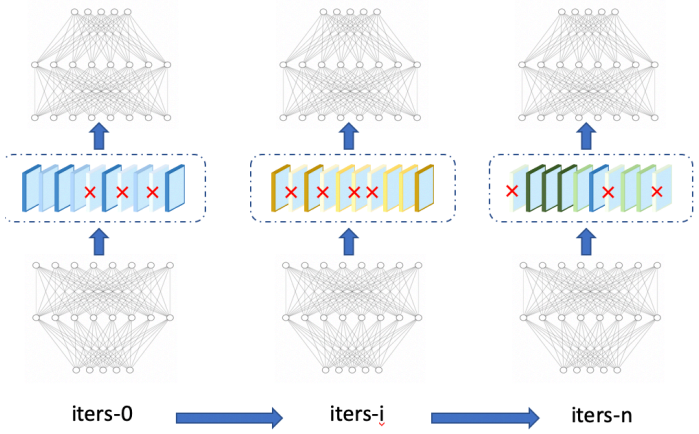



***Static Optimization***

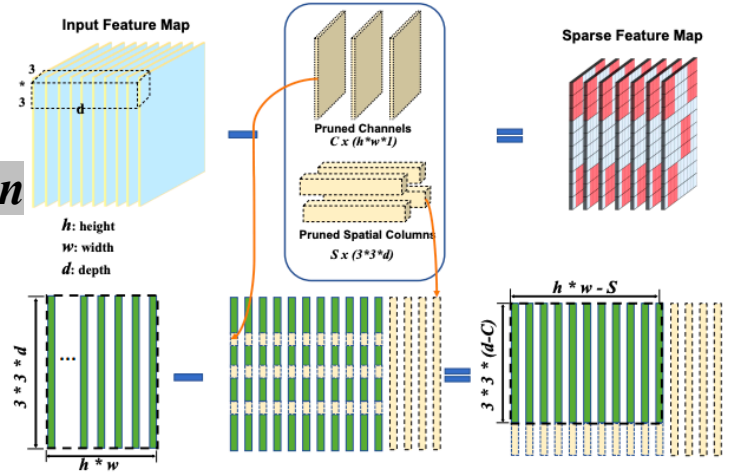
***Algorithm-oriented Optimization***

***Hardware-aware Optimization***

***Dynamic Optimization***




## Dynamic DNN Feature Map Pruning




## Hardware-oriented Deployment Optimization

## Introduction

- Motivation: Efficient Deep Neural Network Design

## Efficiency on Static DNN Architecture Search

- Single-Path Neural Architecture Search
- Hardware-aware Design Space Optimization

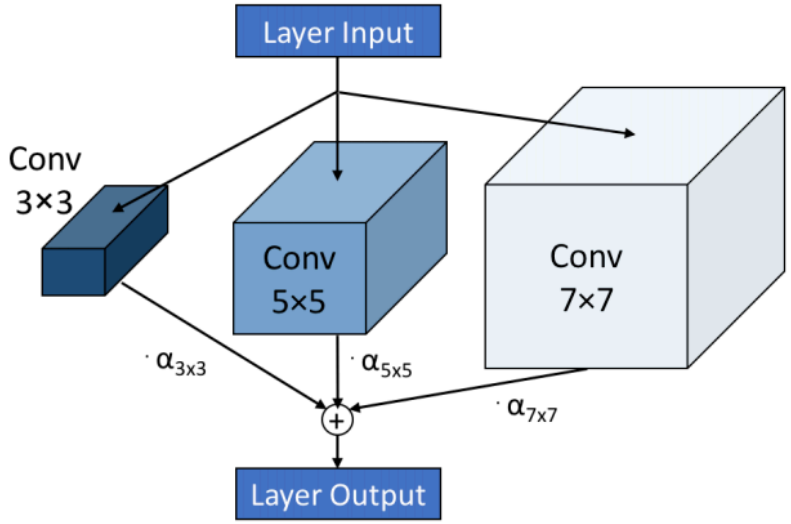
## Efficiency on Dynamic DNN Execution

- Dynamic Feature Map Pruning
- Hardware-oriented Deployment Optimization

## Conclusion

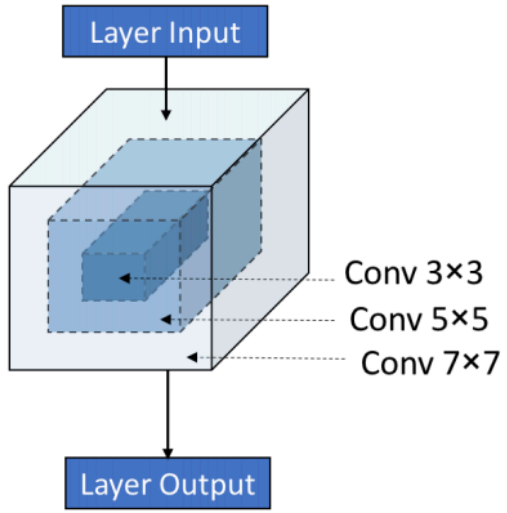
# Motivation: Multi-Path NAS vs. Single-Path NAS

(1) Prior NAS work:
*Multi-path* search space




- **NAS problem**: **expensive** path-level selection
- **Supernet**: each op as **separate path**/layer
- **#parameters/layer**: **all** weights across **all** paths

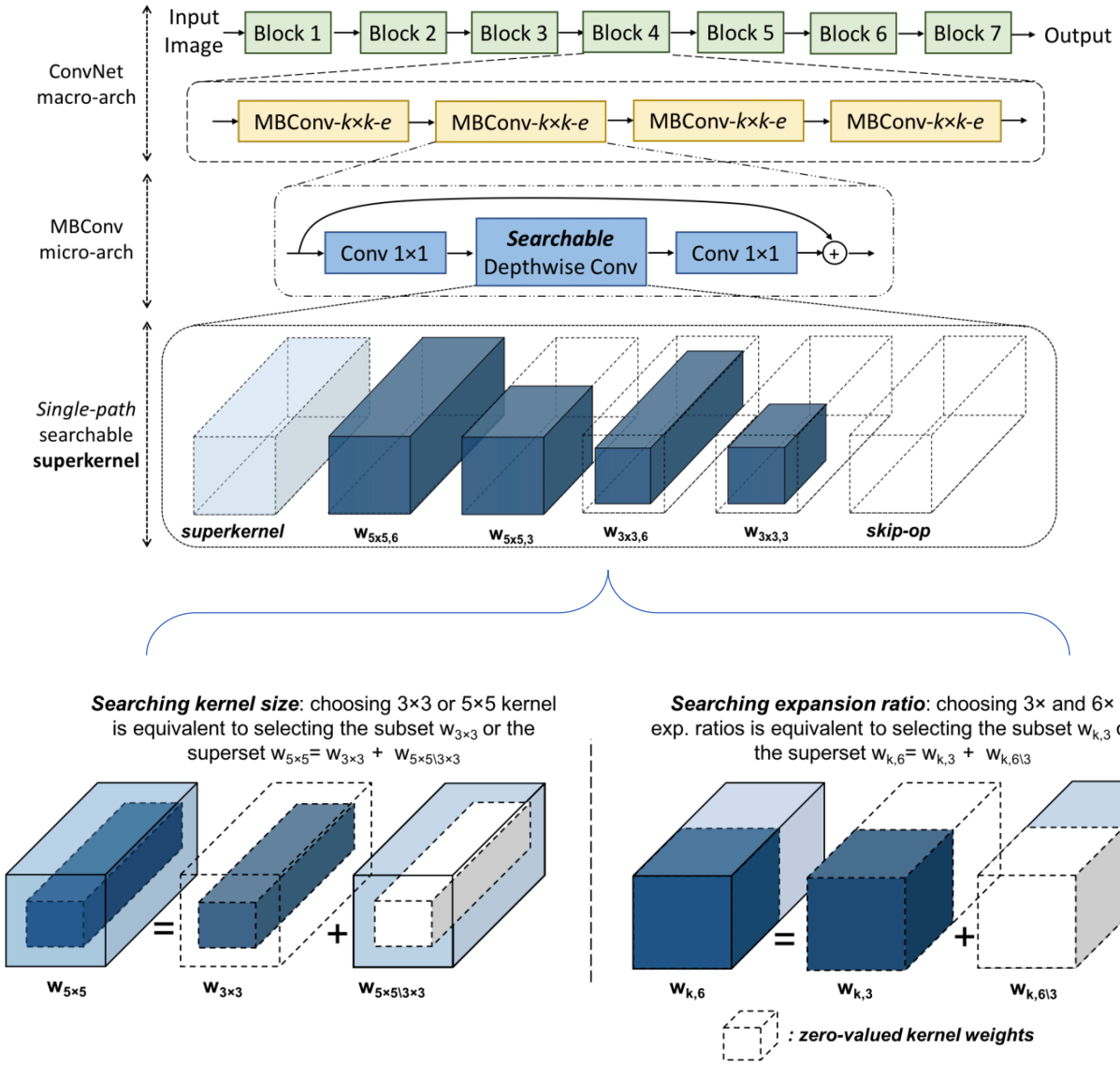
(2) Proposed NAS method:
***Single-path* search space**




- **NAS problem**: ***efficient kernel-level*** selection
- **Supernet**: all ops in single ***"superkernel"***/layer
- **#parameters/layer**: #weights of largest op ***only***

Stamoulis, Dimitrios, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. "Single-path nas: Designing hardware-efficient convnets in less than 4 hours." In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481-497. Springer, Cham, 2019.

# Single-Path Neural Architecture Search Overview




- **Macro-Architecture**: Mobile Conv-Nets;
- **Micro-Architecture**: MB-Conv Block;
- **Universal Super Kernel;**

- **Kernel Size Search;**
- **Expansion Ratio Search, etc..**

# Accuracy and Efficiency Comparison

| Method | Top-1 Acc (%) | Top-5 Acc (%) | Mobile Runtime (ms) | Search Cost (epochs) |
|---|---|---|---|---|
| MobileNetV1 [11] | 70.60 | 89.50 | 113 | - |
| MobileNetV2 1.0x [17] | 72.00 | 91.00 | 75.00 | |
| MobileNetV2 1.0x (our impl.) | 73.59 | 91.41 | 73.57 | |
| Random search | 73.78 ± 0.85 | 91.42 ± 0.56 | 77.31 ± 0.9 ms | - |
| MnasNet 1.0x [20] | 74.00 | 91.80 | 76.00 | 40,000 |
| MnasNet 1.0x (our impl.) | 74.61 | 91.95 | 74.65 | |
| ChamNet-B [6] | 73.80 | – | – | 240‡ |
| ProxylessNAS-R [4] | 74.60 | 92.20 | 78.00 | 200* |
| ProxylessNAS-R (our impl.) | 74.65 | 92.18 | 77.48 | |
| FBNet-B [21] | 74.1 | - | - | 90 |
| FBNet-B (our impl.) | 73.70 | 91.51 | 78.33 | |
| *Single-Path* NAS (**proposed**) | **74.96** | **92.21** | 79.48 | **8 (3.75 hours)** |

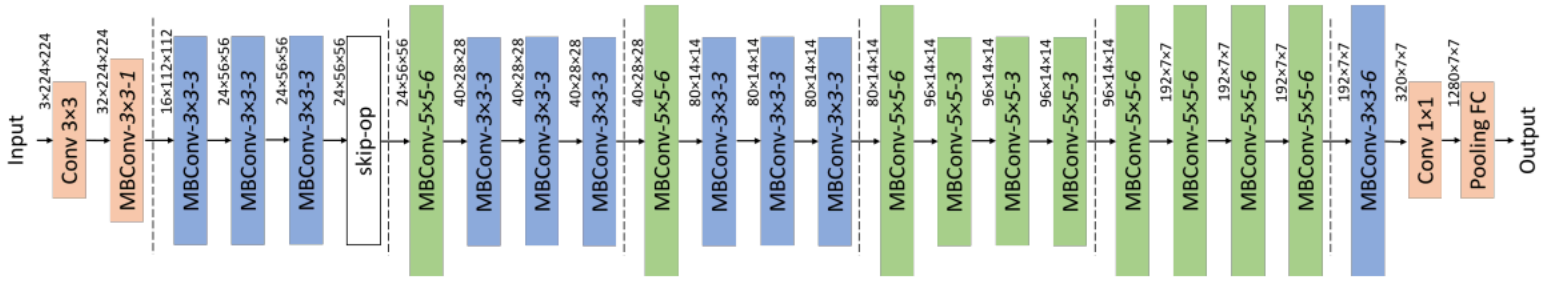



**Fig. 5.** Hardware-efficient ConvNet found by *Single-Path* NAS, with top-1 accuracy of **74.96%** on ImageNet and inference time of 79.48*ms* on Pixel 1 phone.

## Introduction

- Motivation: Efficient Deep Neural Network Design

## Efficiency on Static DNN Architecture Search

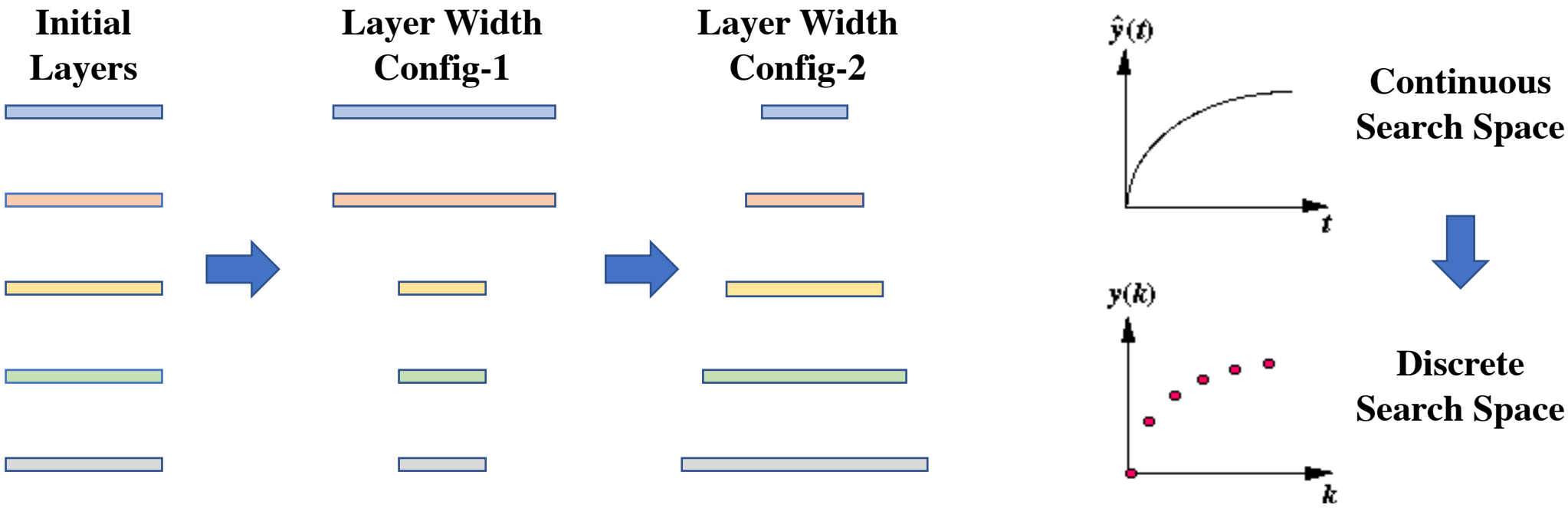
- Single-Path Neural Architecture Search
- Hardware-aware Design Space Optimization

## Efficiency on Dynamic DNN Execution

- Dynamic Feature Map Pruning
- Hardware-oriented Deployment Optimization

## Conclusion

# Motivation: Continuous vs. Discrete Search Space




- Optimal layer width configuration is need in many Efficient DNN Design applications, like NAS, Filter pruning, etc.
- However, the **continuous layer width design space** and **the exponential design space exploration** by multiple layers make it hard to find the optimal model-wise configuration.

# Hardware-aware DNN Search Space Optimization (GPU)

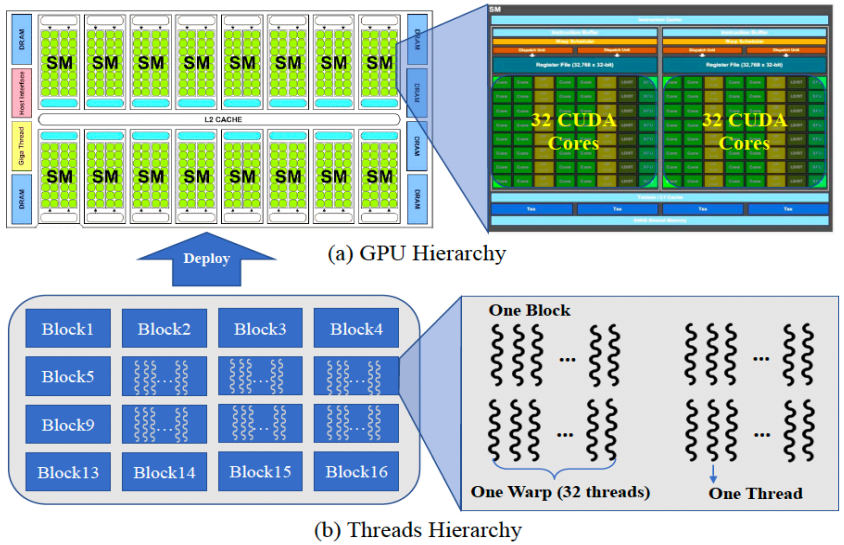



*Figure 2.* An illustration on parallel workload mapping to GPUs. (a) The GPU architecture hierarchy is composed of SMs and CUDA cores. (b) The parallel workload is also logically split into blocks and threads so as to map to the GPU hierarchy.

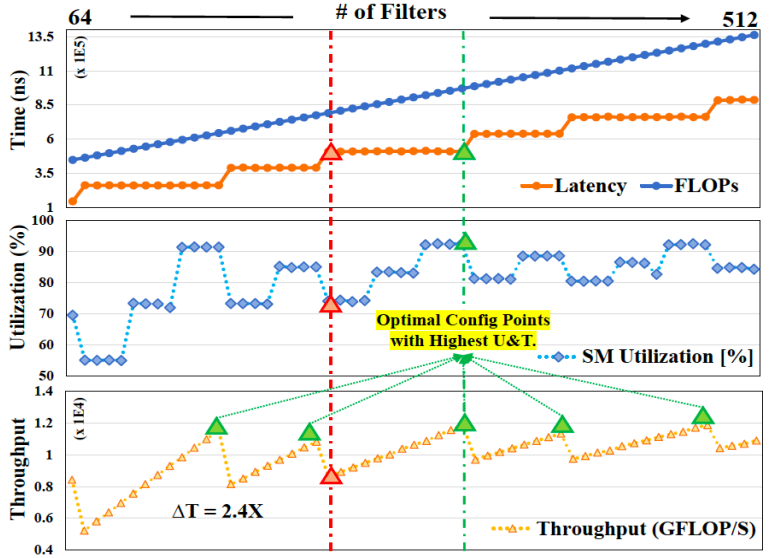



*Figure 6.* Underlying the latency staircase, the GPU's runtime efficiency changes dramatically, which can be reflected by the SM Utilization (U) and Throughput (T). By being aware of such GPU runtime efficiency guidelines, we can identify the optimal configuration candidates for optimization.

- When taking **hardware-aware deployment** into consideration, we could find **optimal discrete layer width** configurations which achieve the optimal utilization/arithmetic throughput;
- Thus, we could significantly reduce the search space from **continuous to discrete** ones, e.g., 512 candidates per layer -> 5 candidates per layer, **100x Less**;

# Hardware-aware DNN Search Space Optimization (GPU)

**Algorithm 2** GPU-Aware Model Optimization Algorithm.

**Input:** Model's initial layer width configs $r[l]$, latency $L[l][n]$, utilization $U[l][n]$, and throughput $T[l][n]$.
**Output:** Optimized model configs $R_{new}[l]$.
Identify candidates $C_l[m]$ for each layer $l$ by Eq. 4.
Initialize latency & parameter gain list $LG[l]$, $PG[l]$.
**for** layers $i = 1$ **to** $l$ **do**
    Get $LG_i$, $PG_i$ estimation by Eq. 5a and 5b.
Sort the layer index list by $LG[l]$ or $PG[l]$.
**while** layer index list is not empty **do**
    Pop out layer j with $\text{Argmax}_j\ LG[l]$.
    Scale down $R_{j,new}$ by Eq. 8a for max latency gain.
    **while** $\Sigma^l PG(R_{new}) \notin (-\tau, \tau)$ **do**
        Pop out layer k with $\text{Argmin}_k\ LG[l]$.
        Scale up $R_{k,new}$ by Eq. 8b to balance param gain.
Get runtime latency evaluation $L_{new}$ of config $R_{new}$.
**if** $L_{new} \leq L_{old} \times \delta$ [← Targeted reduction ratio] **then**
    Train and evaluate the model accuracy.
**else**
    Set $\tau$ *= 2 and repeat the algo. from line 9.
**Return** Optimized config $R_{new}$.

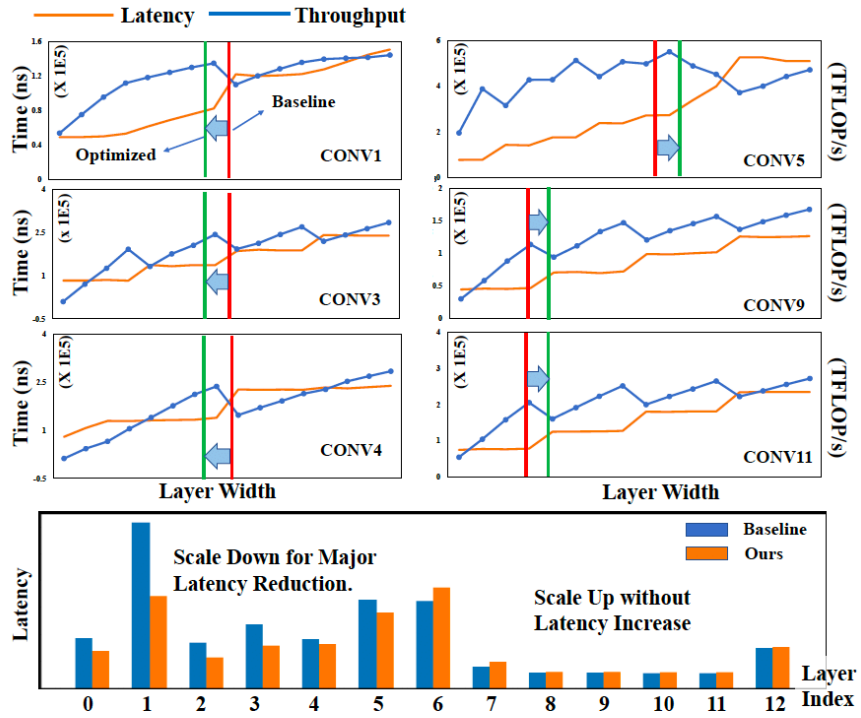



*Figure 7.* VGG pruning configuration optimization on HRank. We slightly scale down Conv1-Conv4 using minimal accuracy drop to trade for major latency reduction, while scale up Conv5-Conv12 to compensate the accuracy drop without obvious latency increment.

- Based on the hardware discrete search space optimization, we could **improve the search efficiency of NAS** methods, as well as **adapt/optimize the existed DNN architectures** on the given hardware.

# Experimental Evaluation

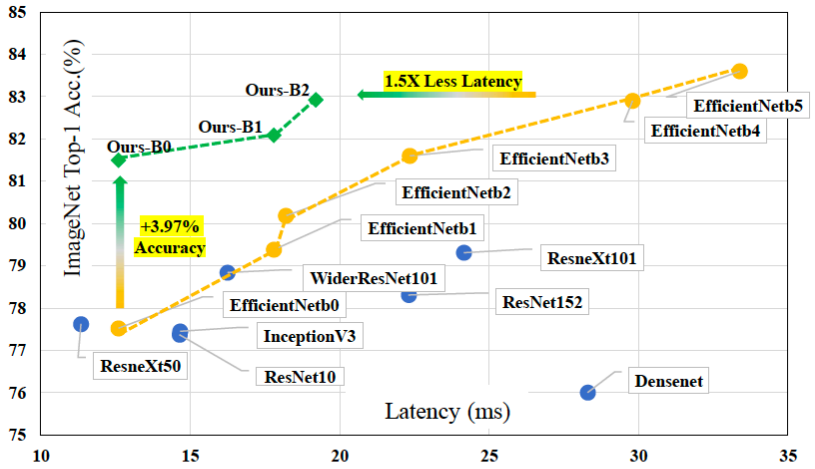



*Figure 8.* ImageNet Accuracy Latency Trade-Off Comparisons. Compared to baseline EfficientNets (yellow), our GPU-aware optimized models achieve better accuracy latency trade-offs. Evaluated Platform: NVIDIA Titan-V GPU.

- Compared to SOTA Efficient Net:
  - **1.5X less latency**;
  - **+3.97% ImageNet accuracy**;

*Table 2.* Latency Optimization on SOTA Pruning Works. HRank: High Rank Pruning [CVPR20]. SOFT: Soft Pruning [IJCAI18].

| [Titan-V] | Method | Params | #FLOPs | FLOP/s | Acc.% | Time (ns) |
|---|---|---|---|---|---|---|
| VGG16 (CIFAR10) | HRank | 1.90M | 67.0M | 2.41T | **93.1** | 2.50E6 |
| | Ours | 2.85M | **104.1M** | **3.90T** | 92.9 | **2.05E6 (-17.7%)** |
| ResNet56 (CIFAR10) | HRank | 0.48M | 65.9M | 0.83T | 93.6 | 4.20E6 |
| | Ours-1 | 0.50M | **79.1M** | **1.00T** | **93.8** | **3.72E6 (-11.3%)** |
| | Ours-2 | 0.50M | **75.1M** | **0.95T** | 93.5 | **3.51E6 (-16.3%)** |
| ResNet56 (CIFAR10) | SOFT-1 | 0.53M | 68.8M | 0.88T | 93.1 | 4.08E6 |
| | Ours | 0.43M | **71.4M** | **0.91T** | **93.2** | **3.52E6 (-13.7%)** |
| | SOFT-2 | 0.45M | 53.1M | 0.68T | 92.3 | 3.64E6 |
| | Ours | 0.43M | **66.0M** | **0.79T** | **92.3** | **3.01E6 (-17.3%)** |

*Note that, SOFT-1 and -2 denotes two configs with different pruning rates from the original paper. The latency is evaluated on Titan-V with batch size = 128.

*Table 4.* Generalizability Evaluation on the Pascal P6000 GPU.

| [P6000] | Method | Params | FLOP/s | Acc.% | Time (ns) |
|---|---|---|---|---|---|
| VGG16 (CIFAR10) | HRank | 1.90M | 1.68T | **93.1** | 4.15E6 |
| | Ours | 2.04M | **1.86T** | 92.9 | **3.02E6 (-27.2%)** |
| ResNet56 (CIFAR10) | HRank | 0.48M | 0.84T | 93.6 | 5.84E6 |
| | Ours | 0.50M | **1.00T** | **93.7** | **5.32E6 (-9.0%)** |

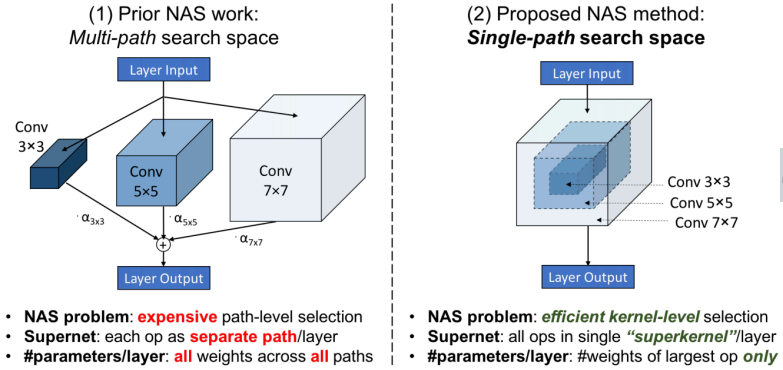
*Table 5.* Generalizability Evaluation on the Jetson Nano GPU.

| [JetsonNano] | Method | Params | FLOP/s | Acc.% | Time (ms) |
|---|---|---|---|---|---|
| VGG16 (CIFAR10) | HRank | 1.90M | 35.5G | **93.1** | 60.5 |
| | Ours | 2.04M | **39.4G** | 92.9 | **48.1 (-20.5%)** |
| ResNet56 (CIFAR10) | HRank-1 | 0.48M | 25.6G | 93.6 | 82.1 |
| | Ours | 0.50M | **28.8G** | **93.7** | **68.0 (-17.1%)** |
| | HRank-2 | 0.24M | 16.7G | 92.3 | 67.2 |
| | Ours | 0.39M | **24.3G** | **92.5** | **58.1 (-13.3%)** |

- **Consistent Latency Reduction;**
- **Generality Across GPUs:**
  - Volta Titan-V
  - Pascal P6000
  - Maxwell Jetson Nano

# Outline: Efficient Deep Neural Network Design

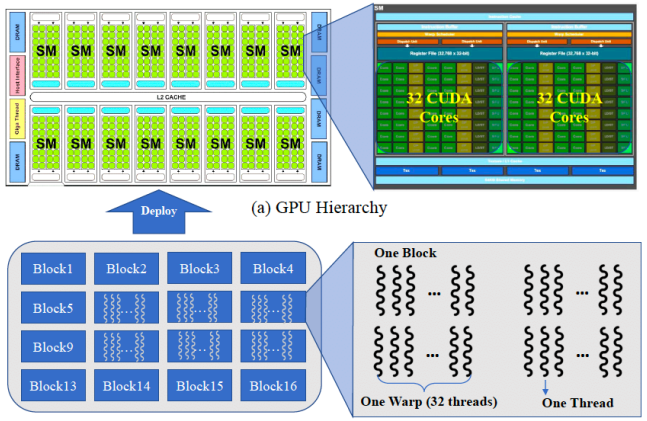
## Efficient Neural Architecture Search




- **NAS problem**: **expensive** path-level selection
- **Supernet**: each op as **separate path**/layer
- **#parameters/layer**: **all** weights across **all** paths

- **NAS problem**: ***efficient kernel-level*** selection
- **Supernet**: all ops in single ***"superkernel"***/layer
- **#parameters/layer**: #weights of largest op ***only***

## Hardware-aware Design Space Optimization

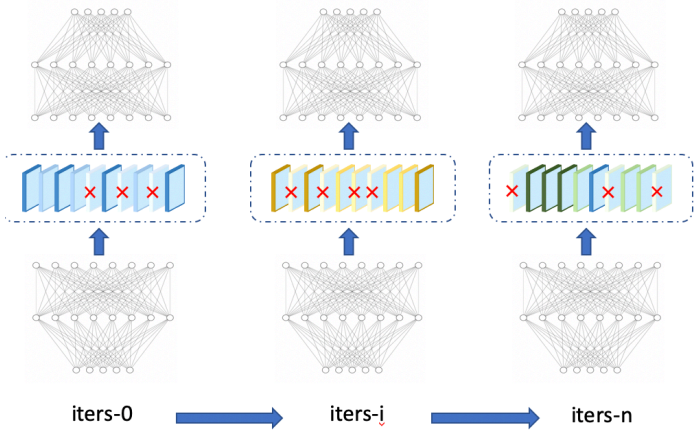



***Static Optimization***

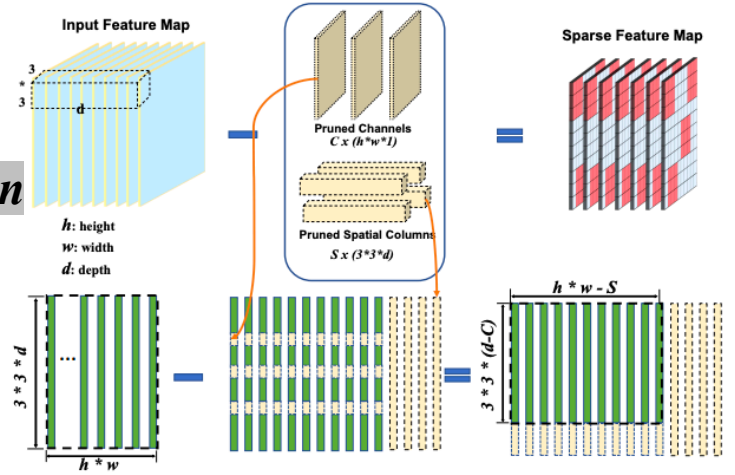
***Algorithm-oriented Optimization***

***Hardware-aware Optimization***

***Dynamic Optimization***




## Dynamic DNN Feature Map Pruning




## Hardware-oriented Deployment Optimization

## Introduction

- Motivation: Efficient Deep Neural Network Design

## Efficiency on Static DNN Architecture Search

- Hardware-aware Design Space Optimization
- Single-Path Neural Architecture Search

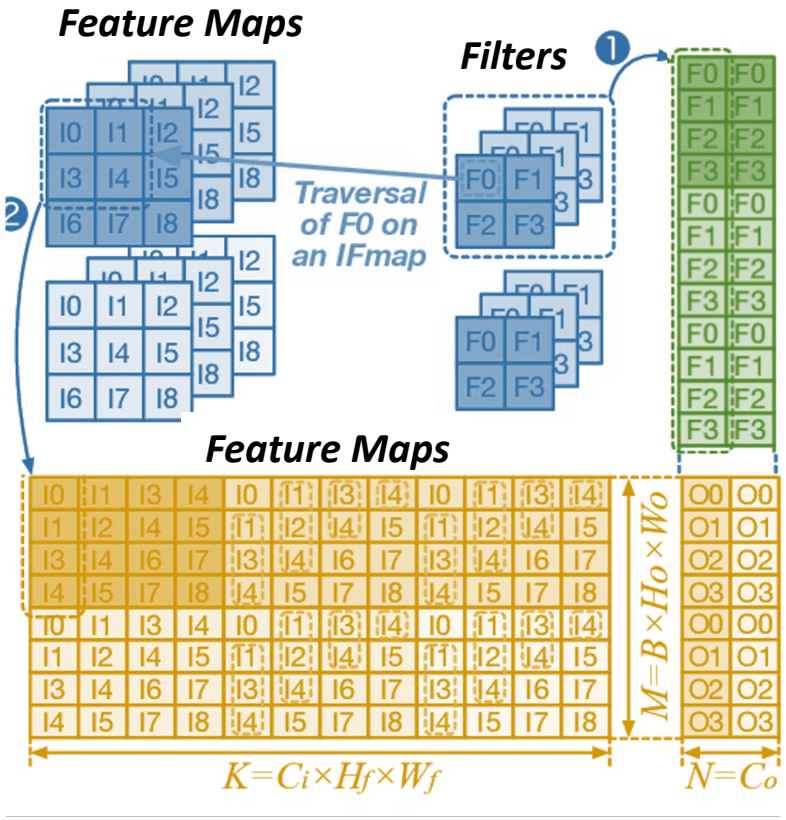
## Efficiency on Dynamic DNN Execution

- Dynamic Feature Map Pruning
- Hardware-oriented Deployment Optimization

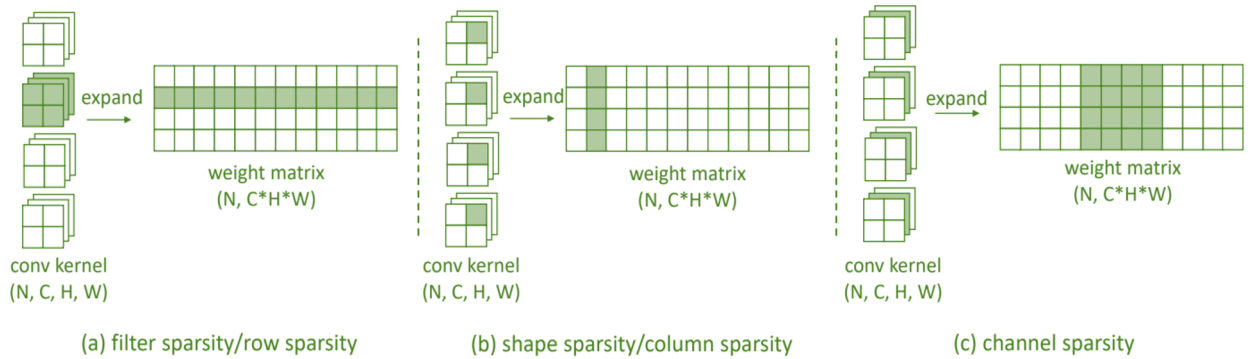
## Conclusion

# Motivation: DNN redundancy exists in both filters & feature maps.

**Convolution *im2col* implementation[1].**




**Conventional Filter Weights Pruning**




Many works are on **filter redundancy** pruning.

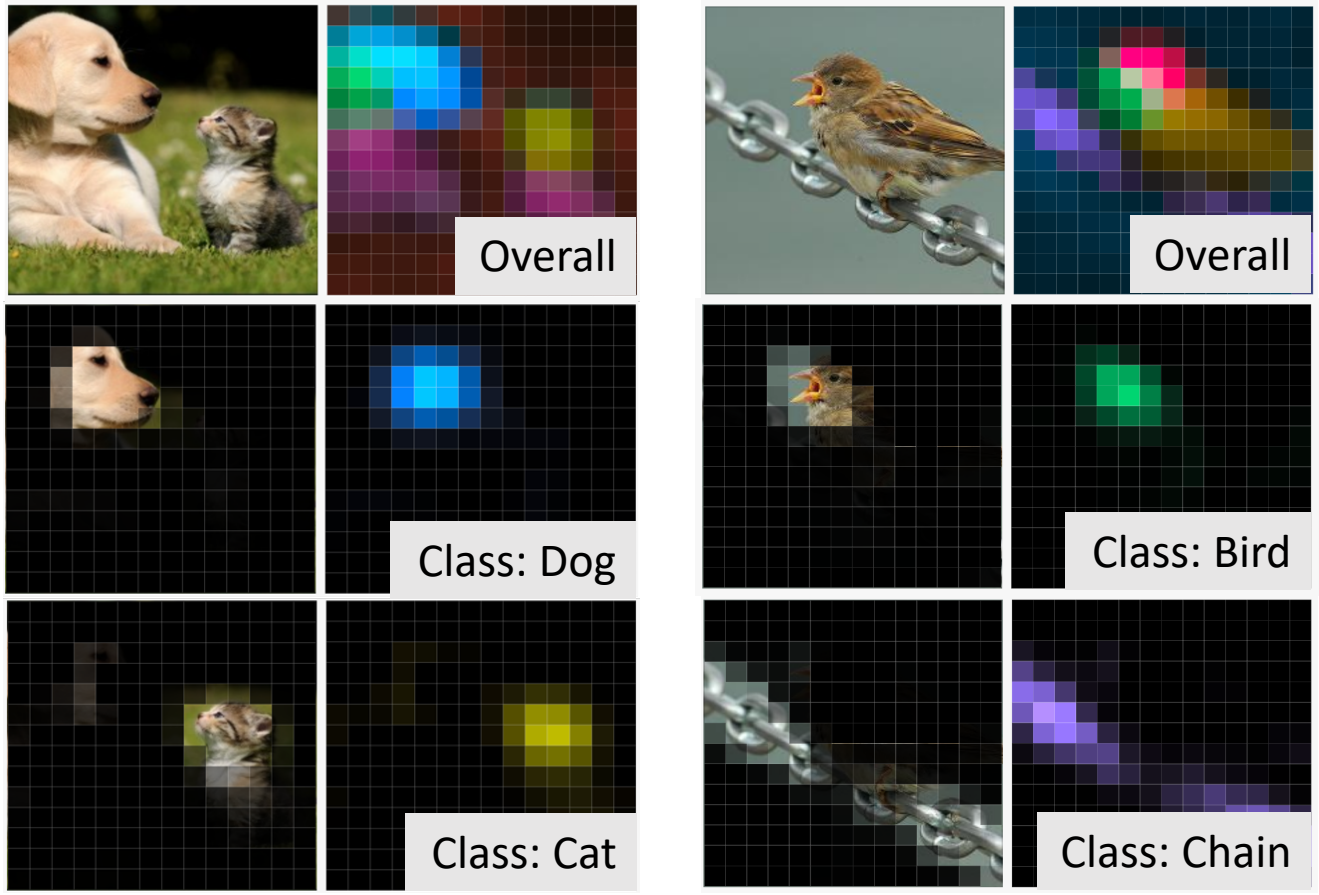
But very few works tackle the **feature map redundancy** pruning.

[1] DeLTA: GPU Performance Model for Deep Learning Applications with In-depth Memory System Traffic Analysis

GEORGE MASON UNIVERSITY

# Motivation: Dynamic Input-based DNN redundancy exists in feature maps.

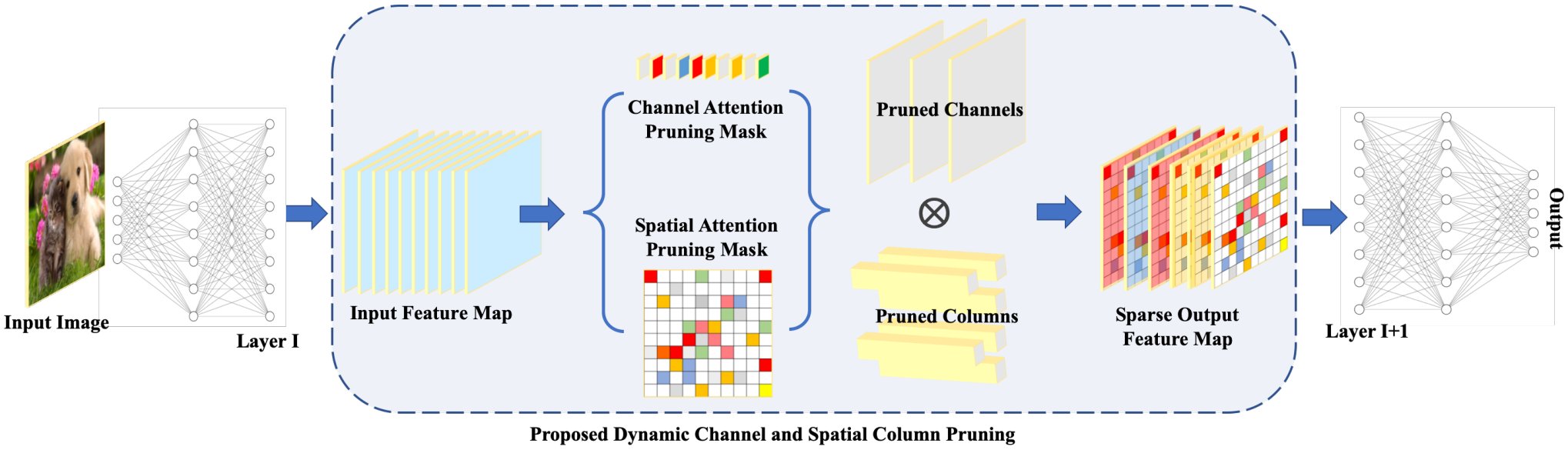
Olah, et al., "The Building Blocks of Interpretability", Distill, 2018.




- **Feature map importance is very Sparse**;
  - Bright components: Useful features;
  - Dark components: Non-useful features;
- For DNN-based classifiers, such **feature map redundancy** can exist a lot.

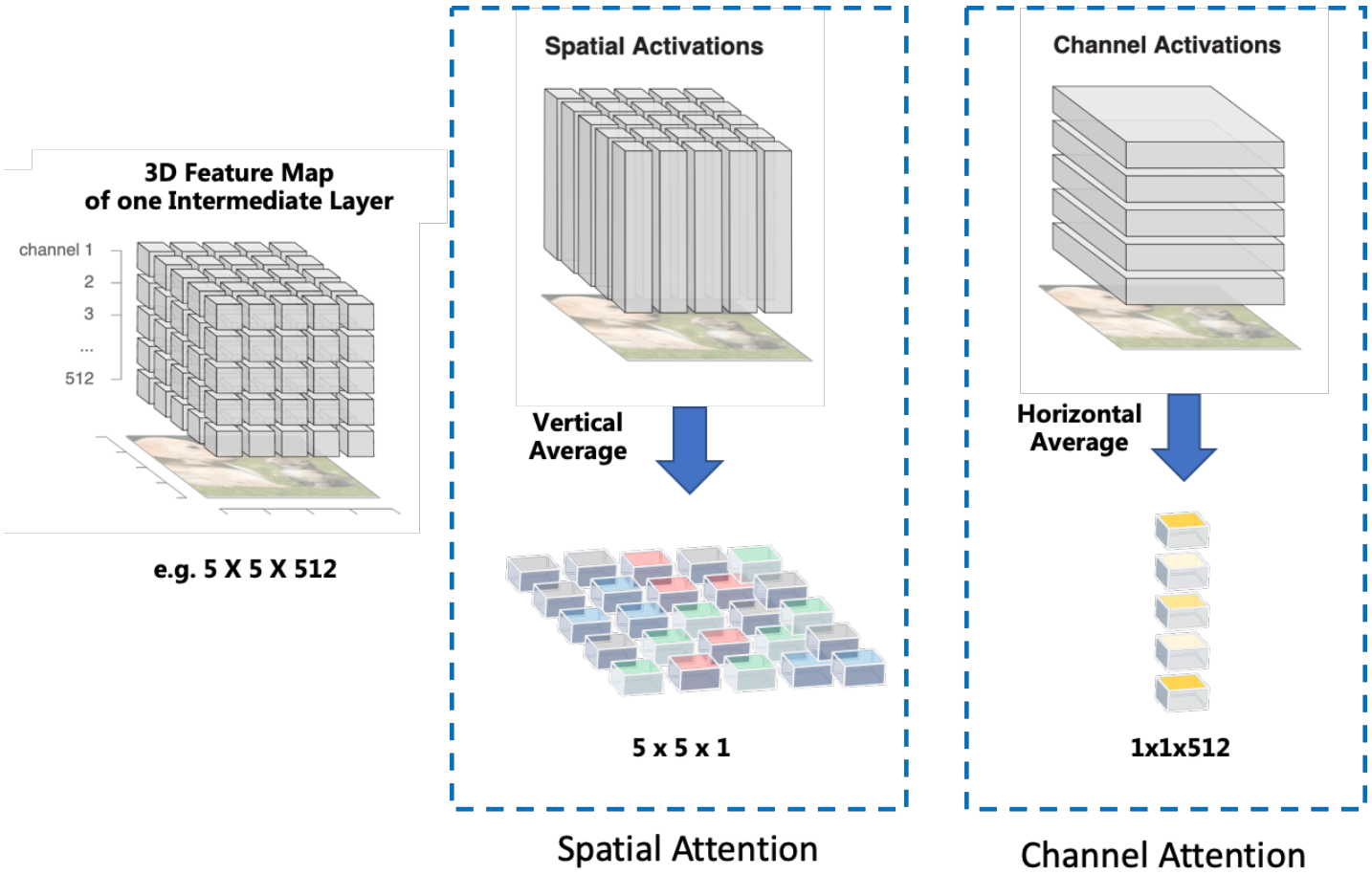
Therefore, we propose to conduct **feature map redundancy elimination**, i.e., **dynamic feature map pruning.**

# Antidote: *channel-wise* & *spatial-wise* feature map pruning




- Between any two conv layers, we use **attention-based mechanism** to prune **non-important channels and spatial columns in feature maps;**
- The output sparse feature map will consist of less channels and spatial columns to save next layer's computation.

# Attention-based feature importance ranking




## Spatial Attention:

$$A_{spatial}(F,h,w) = \frac{1}{C}\sum_{i}^{C} F_{h,w}(i).$$

## Channel Attention:

$$A_{channel}(F,c) = \frac{1}{H*W}\sum_{i}^{H}\sum_{j}^{W} F_c(i,j),$$

## Attention-based Pruning mask:

$$M(F,c) = \begin{cases} True, & \text{If } c \in topk(A_{channel}), \\ & k = int(p*C); \\ False, & \text{Otherwise.} \end{cases}$$

GEORGE MASON UNIVERSITY

# Case Study: Effectiveness of attention-based ranking

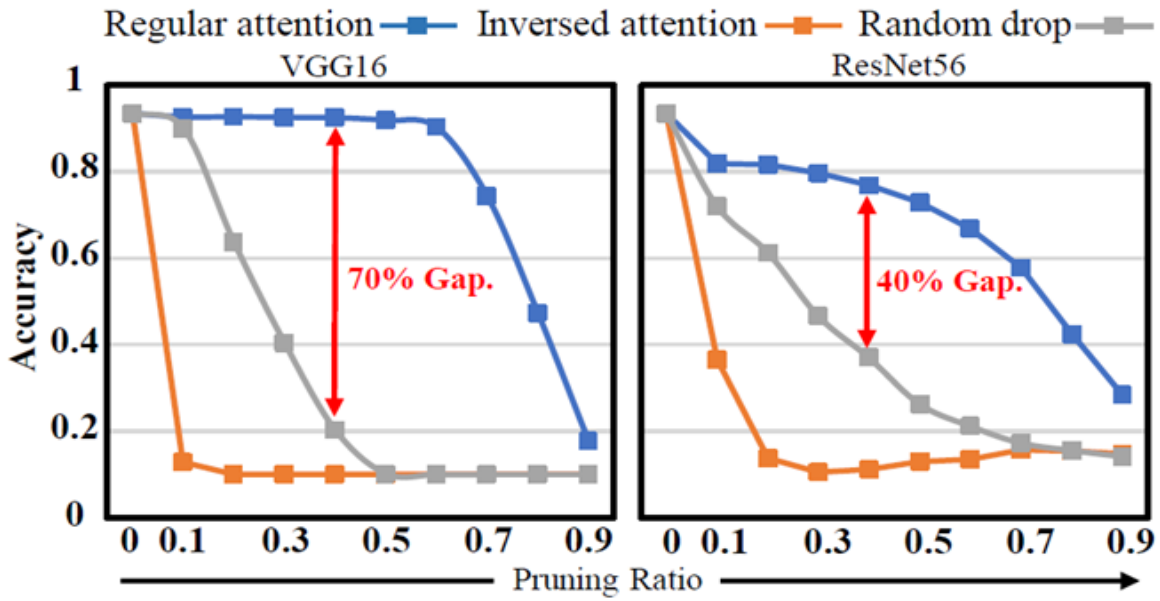
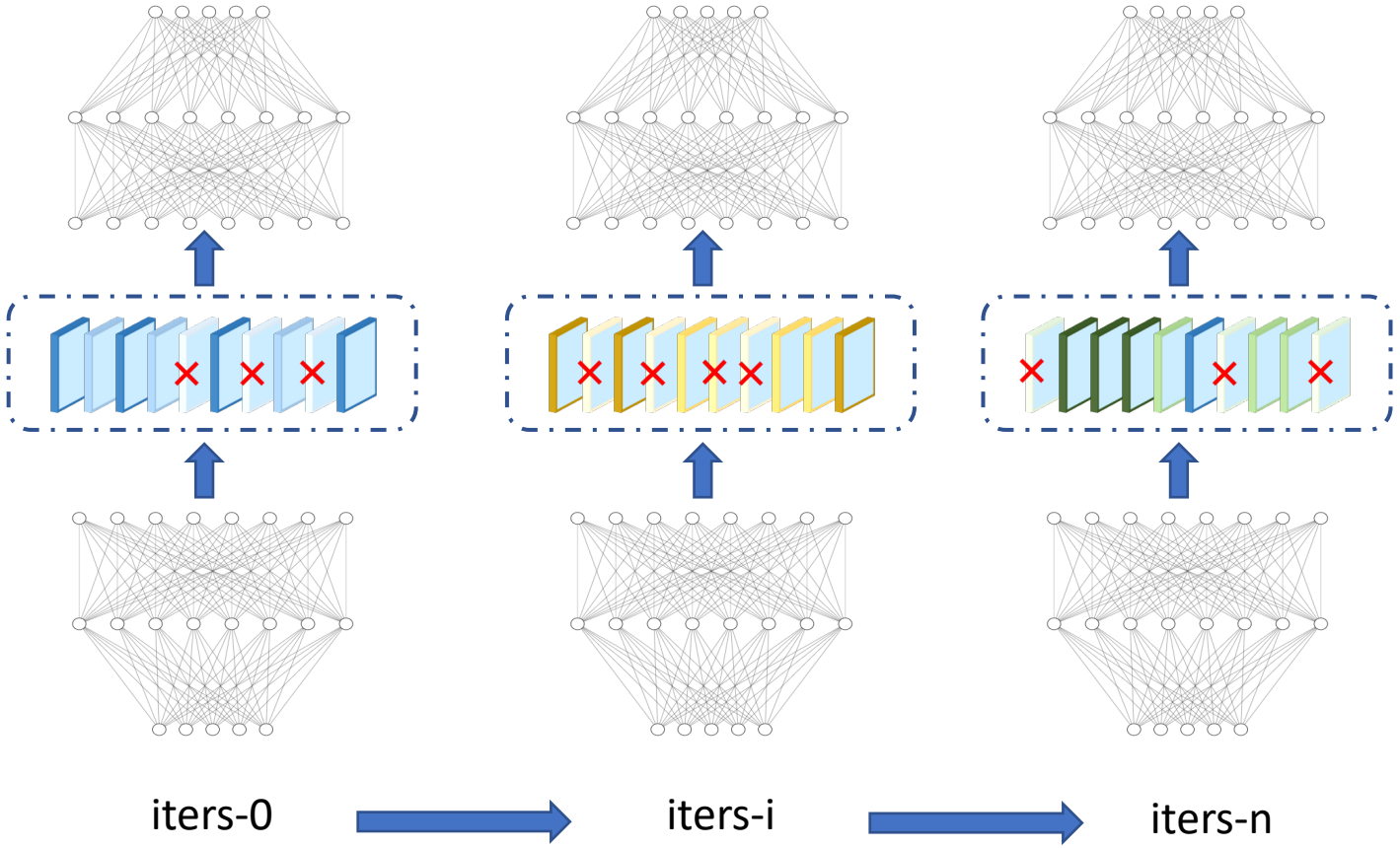



Fig. 2. Attention-based and Random Pruning Accuracy Drop Comparison.

- **Attention-based ranking** is very effective in evaluating feature importance:
  - With same pruning rate, **70% higher accuracy** than random feature selection;
  - **60% feature map redundancy** can be eliminated without accuracy drop in VGG16;

- However, if we reach high feature pruning rate, the **accuracy drop** can still be non-negligible;
- Therefore, we propose **training-phase optimization** as a remedy: **Training with targeted dropout**.

# Training-phase Optimization: Targeted Dropout




**Attention-based Targeted Dropout Training**

- **During training, dropping out least-important features;**

$$F' = F \otimes M_{spatial}(h, w),$$
$$F'' = F' \otimes M_{channel}(c),$$

- **Advantages:**
  - Resistance to accuracy drop;
  - No repetitive retraining.

# Experimental Evaluation

TABLE I
EXPERIMENT RESULTS ON CIFAR AND IMAGENET DATASETS.

| CNN Models | Pruning Methods | Baseline Accuracy(%) | Baseline FLOPs | Final FLOPs | FLOPs Reduction(%) | Final Accuracy(%) | Accuracy Drop(%) |
|---|---|---|---|---|---|---|---|
| VGG16 (CIFAR10) | L1 Pruning* [8] | 93.3 | - | 2.06E+08 | 34.2 | 93.4 | **-0.1** |
| | Taylor Pruning* [19] | 93.3 | - | 1.85E+08 | 44.1 | 92.3 | 1.0 |
| | GM Pruning* [20] | 93.6 | - | 2.11E+08 | 35.9 | 93.2 | 0.4 |
| | FO Pruning* [21] | 93.4 | - | 1.85E+08 | 44.1 | **93.3** | 0.1 |
| | **Proposed** | 93.3 | 3.13E+08 | **1.46E+08** | **53.5** | 93.1 | 0.2 |
| ResNet56 (CIFAR10) | L1 Pruning* [8] | 93.0 | - | 0.91E+08 | 27.6 | 93.1 | -0.1 |
| | Taylor Pruning* [19] | 92.9 | - | 0.71E+08 | 43.0 | 92.0 | 0.9 |
| | FO Pruning* [21] | 92.9 | - | **0.71E+08** | **43.0** | **93.3** | **-0.4** |
| | **Proposed** | 93.0 | 1.28E+08 | 0.80E+08 | 37.4 | 93.2 | -0.2 |
| VGG16 (CIFAR100) | L1 Pruning* [8] | 73.1 | - | 1.96E+08 | 37.3 | 72.3 | 0.8 |
| | Taylor Pruning* [19] | 73.1 | - | 1.96E+08 | 37.3 | 72.5 | 0.6 |
| | FO Pruning* [21] | 73.1 | - | 1.96E+08 | 37.3 | 73.2 | -0.1 |
| | **Proposed: Setting-1** | 73.1 | 3.13E+08 | **1.87E+08** | **40.4** | **73.2** | **-0.1** |
| | **Proposed: Setting-2** | 73.1 | 3.13E+08 | **1.72E+08** | **44.9** | 72.9 | 0.2 |
| VGG16 (ImageNet100) | L1 Pruning* [8] | 78.5 | - | 0.76E+10 | 50.6 | 76.6 | 0.8 |
| | Taylor Pruning* [19] | 78.5 | - | 0.76E+10 | 50.6 | 77.3 | 0.6 |
| | FO Pruning* [21] | 78.5 | - | 0.76E+10 | 50.6 | 79.5 | -1.0 |
| | **Proposed: Setting-1** | 78.5 | 1.52E+10 | **0.74E+10** | **51.2** | **79.6** | **-1.1** |
| | **Proposed: Setting-2** | 78.5 | 1.52E+10 | **0.69E+10** | **54.5** | 79.4 | -0.9 |

* indicates the methods' performance is refered from [20], [21].

Across different datasets and models, our method could generally achieve **37%~54% FLOPs reduction** with **<1% accuracy drop**, significantly outperforming previous state-of-the-art works.

# Advantages of Antidote: Dynamic Feature Pruning

- **Dynamic vs. Static**: targeting at each image, more precise and aggressive.
  - For VGG16 on cifar10, traditional **static pruning** methods can remove about **[17%, 10%, 10%, 45%, 65%]** channels per block (General Redundancy on whole datasets).
  - Our **dynamic method** can remove **[20%, 20%, 60%, 90%, 90%]** channels (Per-input Redundancy).

- **Spatial-wise + Channel-wise:** Flexible & comprehensive redundancy elimination;
  - Filter pruning can remove **channel redundancy only**;
  - But for ImageNet-size input (224x224), **spatial redundancy** is the main redundancy!
  - We can **prune 50~60% spatial feature columns** for every layer for VGG on ImageNet.

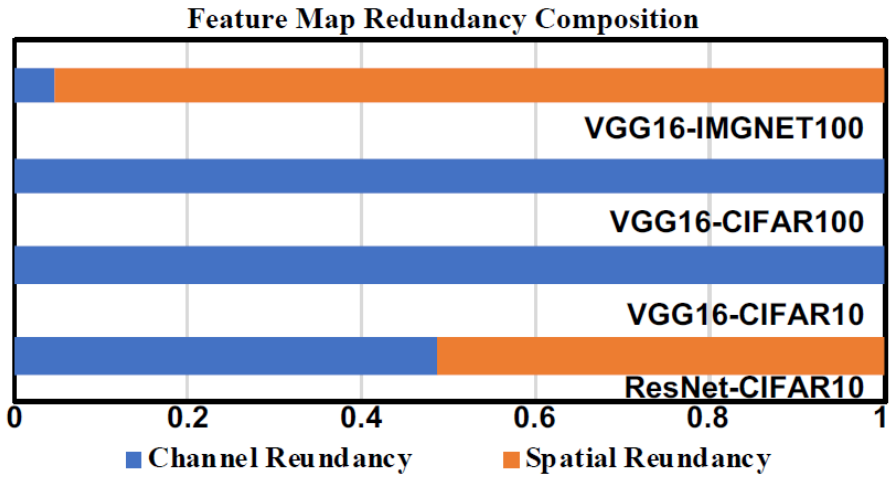



Fig. 4. The redundancy composition varies in channel and spatial dimensions.

# Hardware-oriented Deployment Optimization

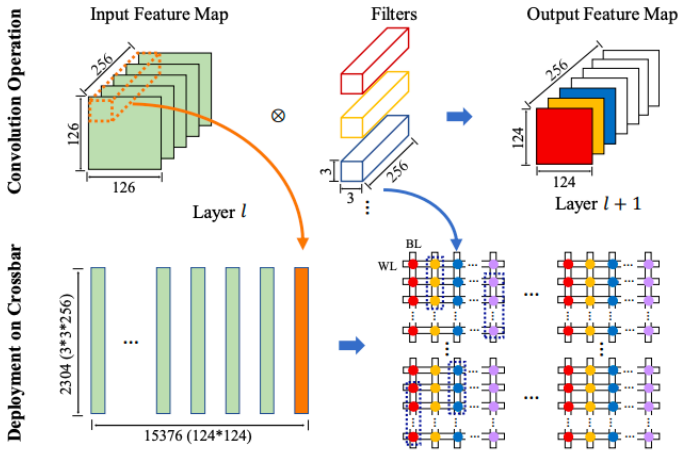



**Figure 1: Deploying convolution layer on ReRAM crossbar.**

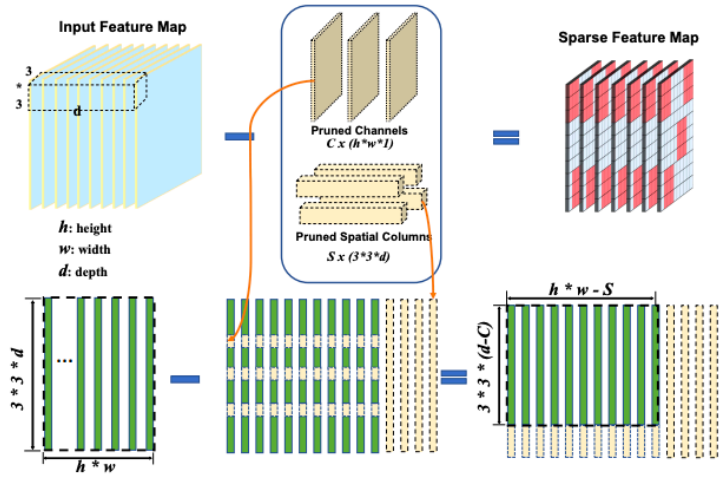



**Figure 2: The structural-spatial and channel feature map pruning bring the structural sparsity on ReRAM crossbar.**

**Table 3: Inference Latencies**

| Network | Original | Ours | Speedup |
|---|---|---|---|
| VGG-16 (ms) | 0.39 | 0.31 | **1.3×** |
| ResNet-56 (ms) | 2.87 | 2.67 | **1.1×** |

**Table 4: Computing Efficiency Evaluation**

| Network | Computational Efficiency (GOPs/s/$mm^2$) | Power Efficiency (GOPs/W) | Area ($mm^2$) | Power (mW) |
|---|---|---|---|---|
| Baseline VGG | 736.30 | 290.3 | 1.09 | 1078.48 |
| Ours | 938.42 | 1262 | 0.25 | 248.88 |
| Relative Gain | **1.3×** | **4.4×** | **4.4×** | **4.3×** |
| Baseline ResNet | 530.94 | 154.4 | 0.084 | 82.96 |
| Ours | 570.71 | 205.8 | 0.063 | 62.22 |
| Relative Gain | **1.1×** | **1.3×** | **1.3×** | **1.3×** |

On ReRAM-based crossbar architecture, such **spatial & column feature map pruning** enables **fine-grid sparsity**, which we could leverage to get for actual speedup.

## Introduction

- Motivation: Efficient Deep Neural Network Design

## Efficiency on Static DNN Architecture Search

- Hardware-aware Design Space Optimization
- Single-Path Neural Architecture Search

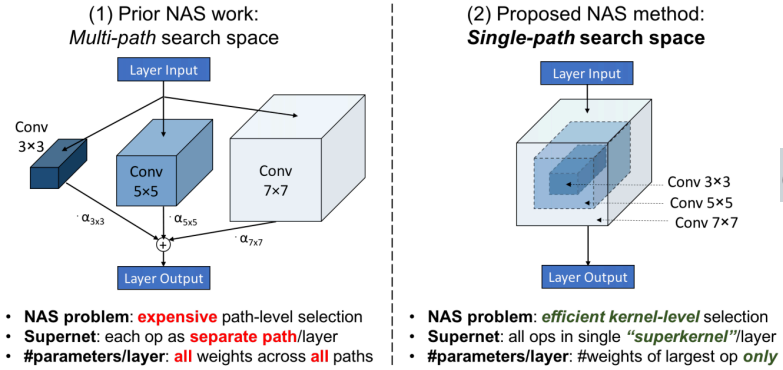
## Efficiency on Dynamic DNN Execution

- Dynamic Feature Map Pruning
- Hardware-oriented Deployment Optimization

## Conclusion

# Conclusion

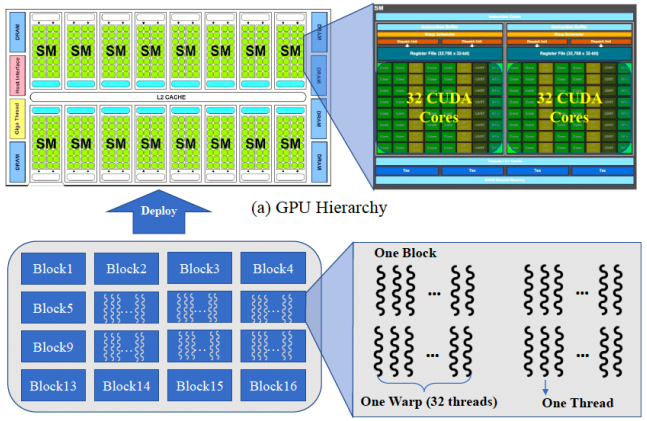
## Efficient Neural Architecture Search




- **NAS problem**: **expensive** path-level selection
- **Supernet**: each op as **separate path**/layer
- **#parameters/layer**: **all** weights across **all** paths

- **NAS problem**: ***efficient kernel-level*** selection
- **Supernet**: all ops in single ***"superkernel"***/layer
- **#parameters/layer**: #weights of largest op ***only***

## Hardware-aware Design Space Optimization

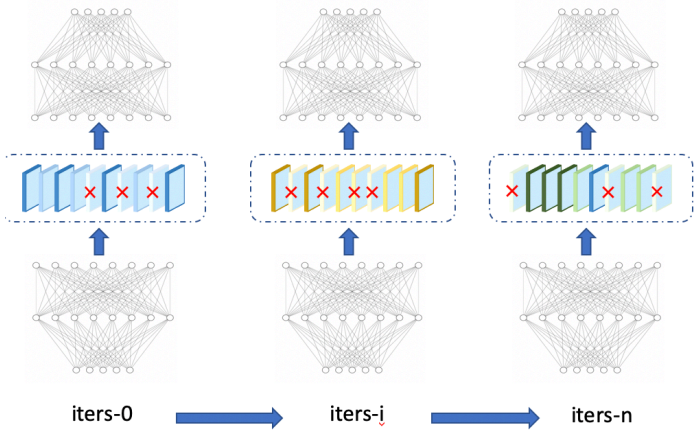



***Static Optimization***

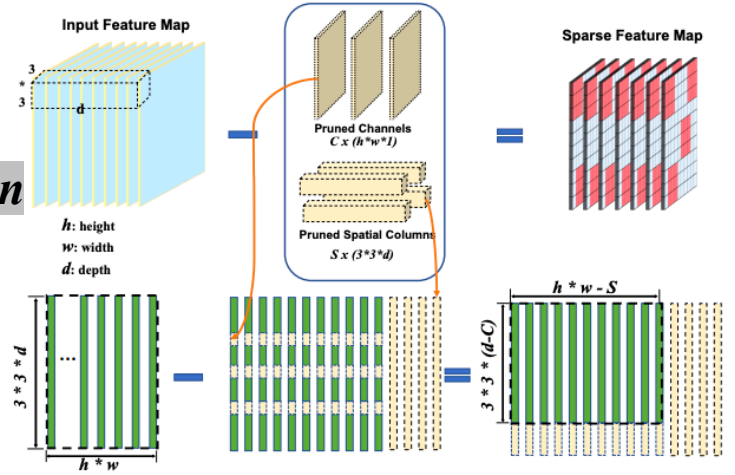
***Algorithm-oriented Optimization***

***Hardware-aware Optimization***

***Dynamic Optimization***




## Dynamic DNN Feature Map Pruning




## Hardware-oriented Deployment Optimization


Thank You!