Deep Learning Optimization
- Selective/Conditional Execution

March 27, 2023

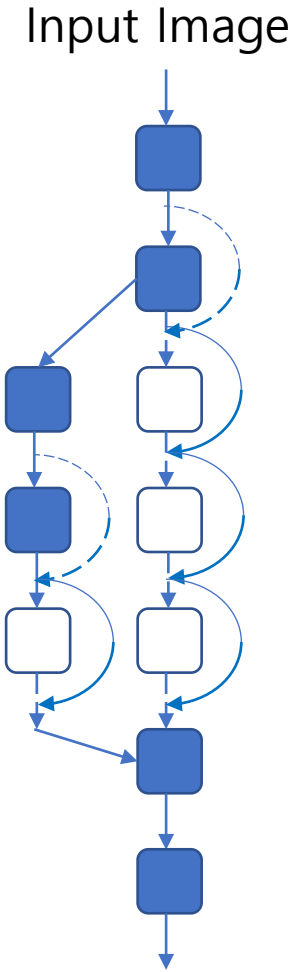
Eunhyeok Park

Selective/Conditional Execution

- Selective/Conditional execution of neural networks?
  - Portion of the network is executed
    - Based on the pattern/difficulty of input data
    - Redundancy of the trained weight

- For the improvement of performance, energy efficiency, etc.

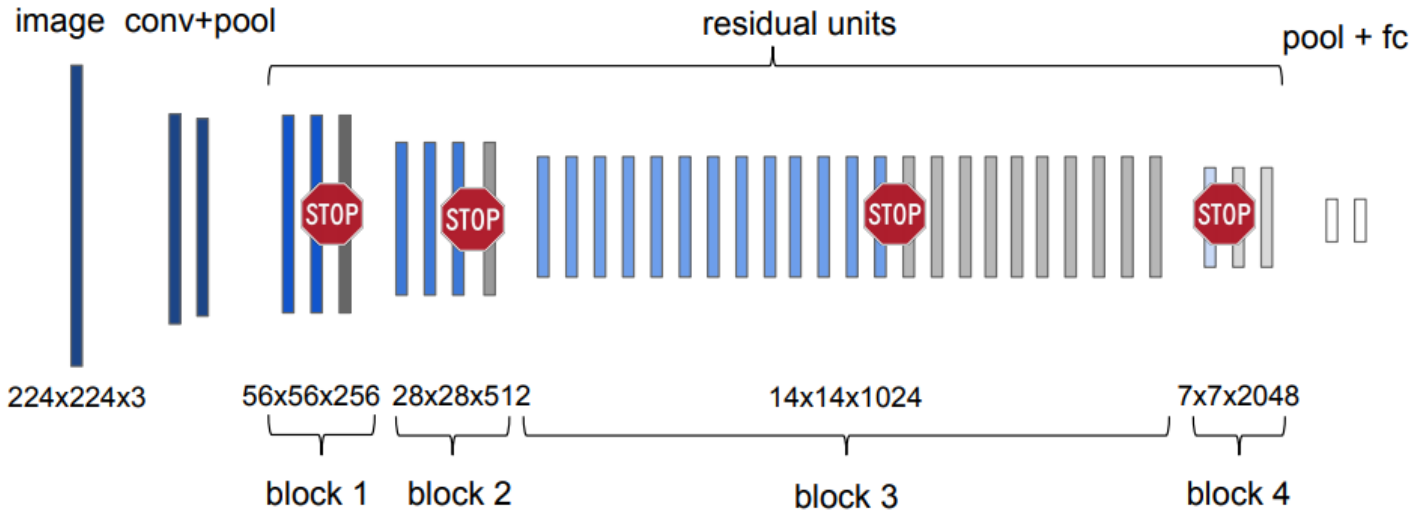
- Agenda
  - Skipping region/layer/channel
  - Early-stopping
  - Skipping early layers and exploiting prev. results
  - Slow/fast model



Whole Network May Not Be Necessary

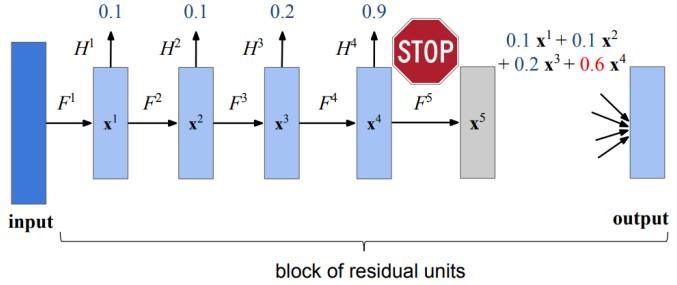
- Recent CNNs have modular design with residual learning
  - Convolutional feature extractor + identity path
  - Feature for easy and clear input can be extracted at early stage
    - Or feature quality become "good enough" with shallow network

- Adaptive Computation Time (ACT) mechanism



[Figurnov et al, 2017]

ACT Training



- Add a branch to the output of each residual unit which predicts a halting score
  - A scalar value in range [0, 1]

$$h^l = H^l(\mathbf{x}^l) = \sigma(W^l \,\mathrm{pool}(\mathbf{x}^l) + b^l)$$

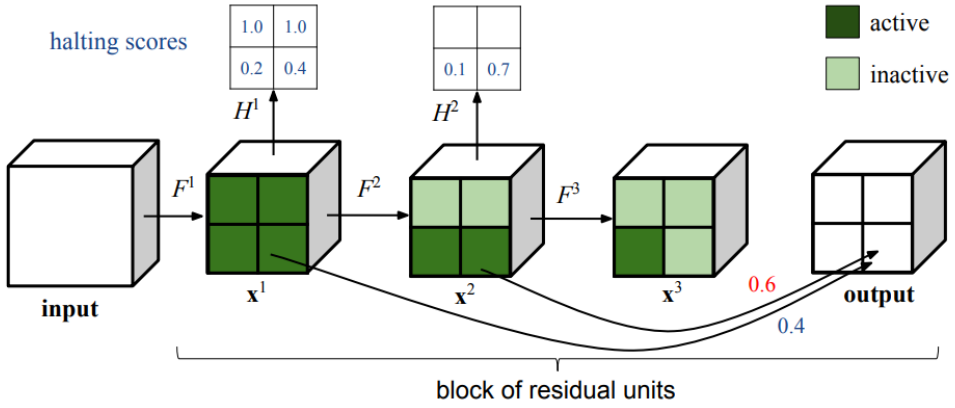
  - Halting score is accumulated sequentially
    - When the accumulated values reach to 1, the remaining residual blocks are skipped
  - Definition of halting distribution & weight output

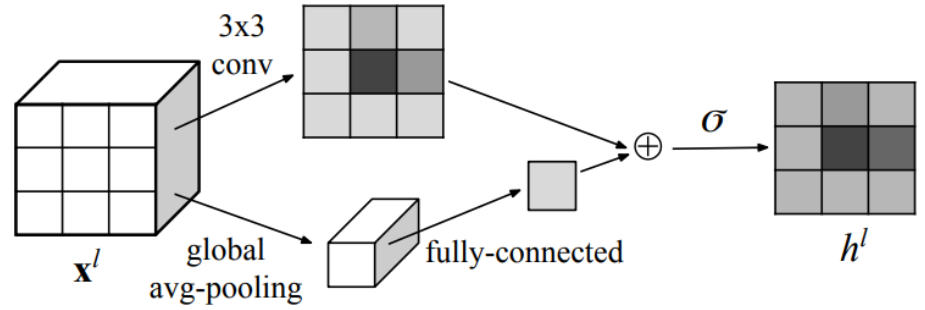
$$p^l = \begin{cases} h^l & \text{if } l < N, \\ R & \text{if } l = N, \\ 0 & \text{if } l > N. \end{cases} \qquad \textbf{output} = \sum_{l=1}^{L} p^l \mathbf{x}^l = \sum_{l=1}^{N} p^l \mathbf{x}^l. \qquad \frac{\partial \rho}{\partial h^l} = \begin{cases} -1 & \text{if } l < N, \\ 0 & \text{if } l \geq N. \end{cases}$$

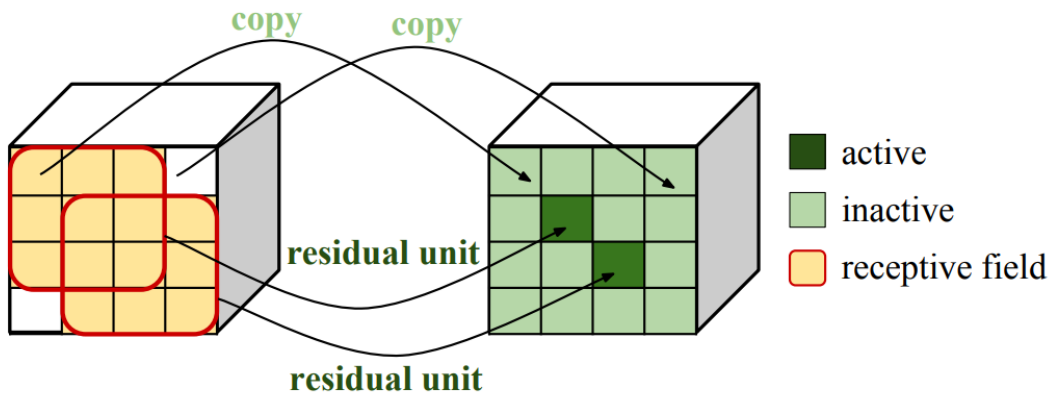
[Figurnov et al, 2017]

Spatial ACT

- Extending ACT toward block-wise spatial dimension
  - Generating global & spatial halting score for predefined tile size



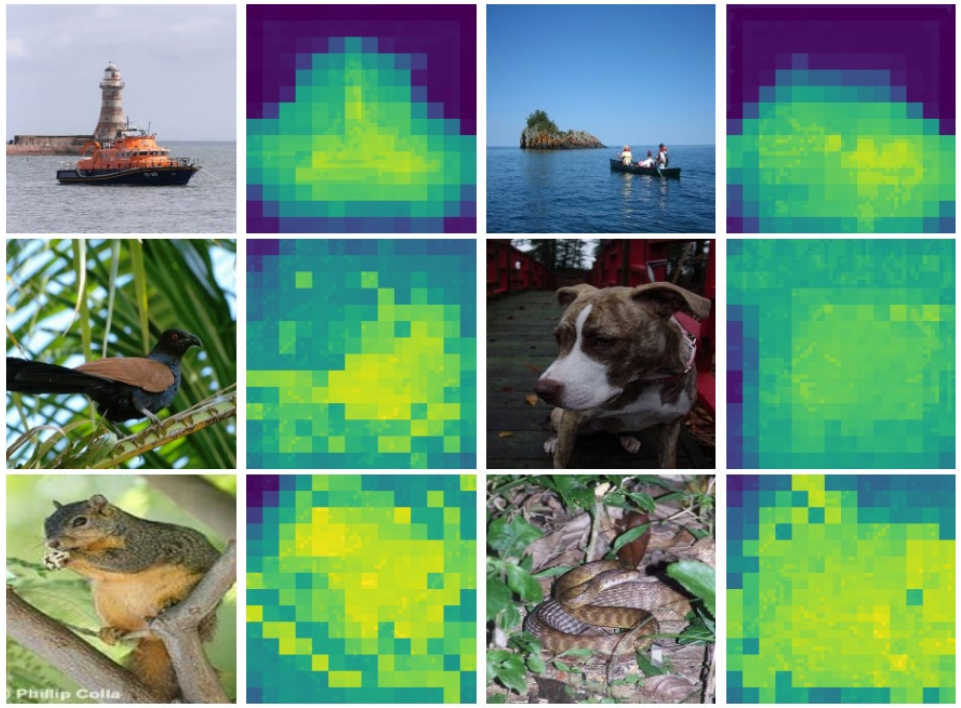


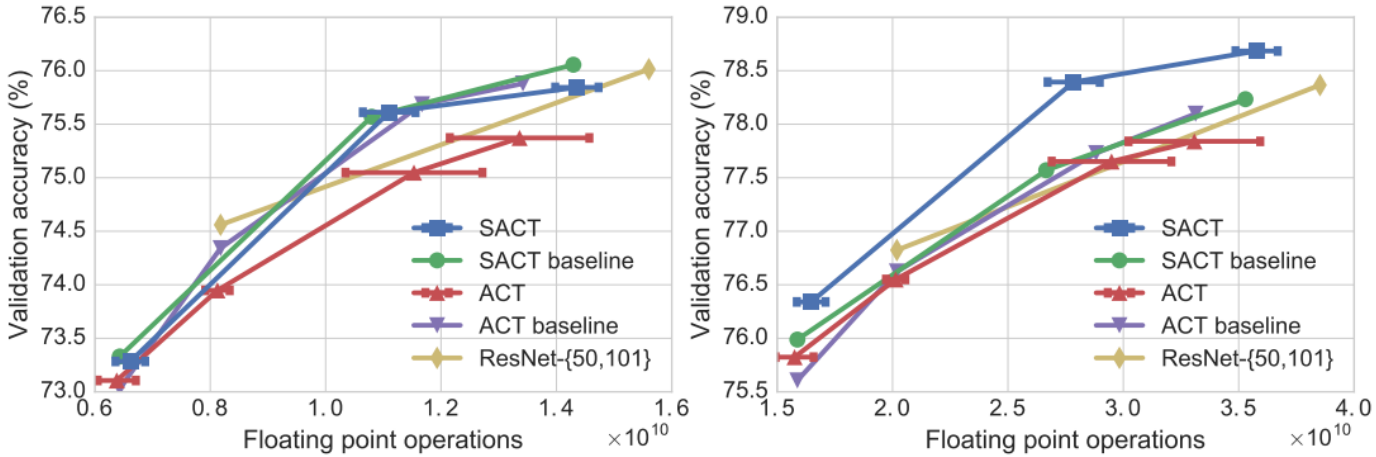


[Figurnov et al, 2017]

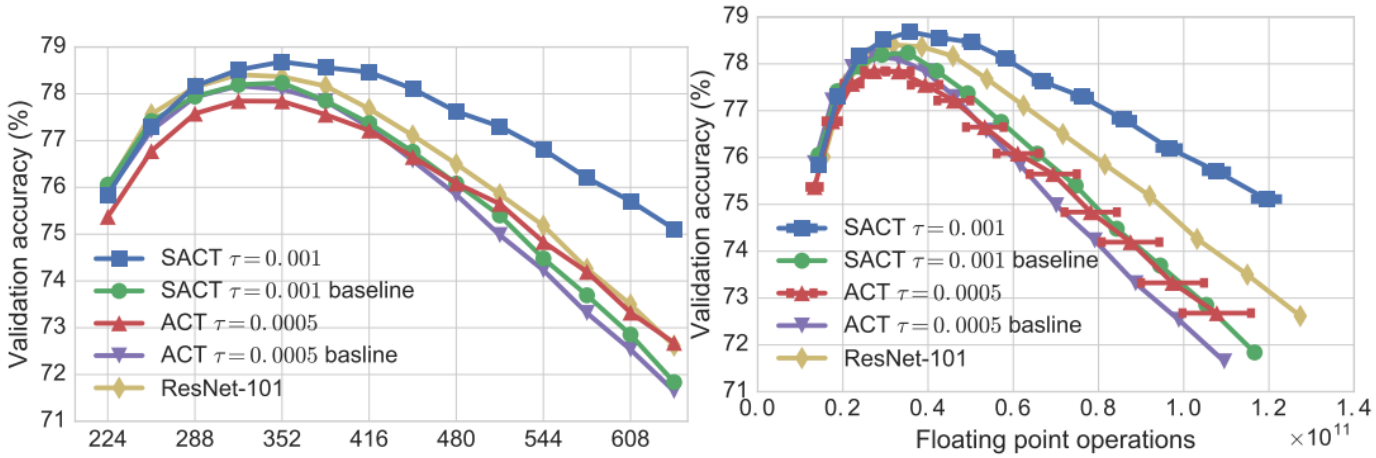
ImageNet Results

- Baseline: ResNet-101/50
- ACT/SACT backbone: ResNet-101





(a) Test resolution 224×224 (b) Test resolution 352×352

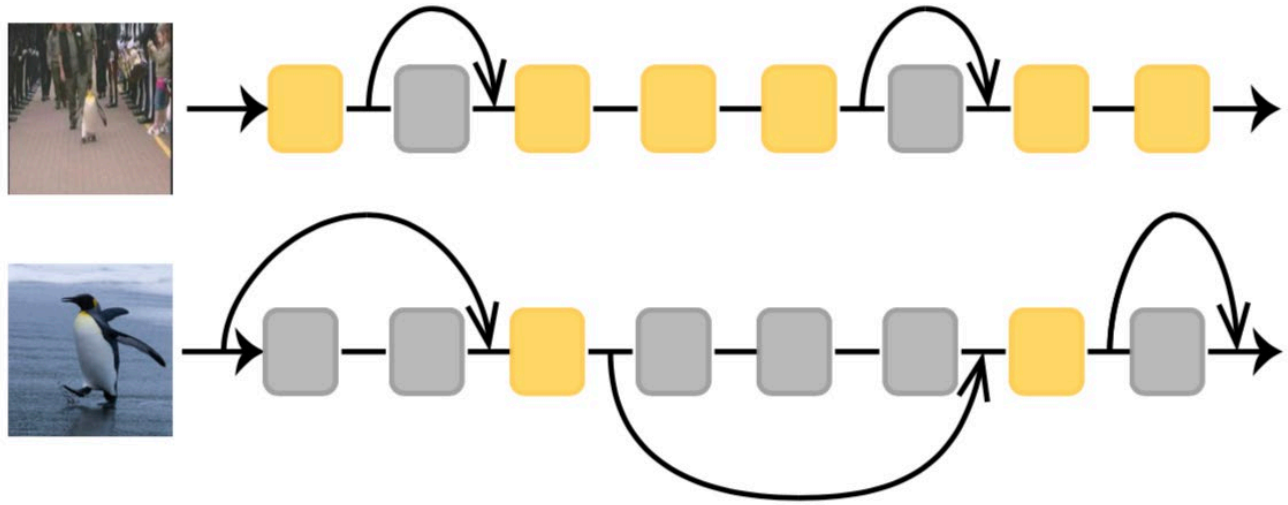


[Figurnov et al, 2017]

SkipNet

- Problem of (S)ACT
  - Early termination loses the potential of intermediate layers
  - Due to early termination, the layers closed to the output are slowly trained

  - Layer-wise skip based on gating units
  - We could easily skip some layers for easy images

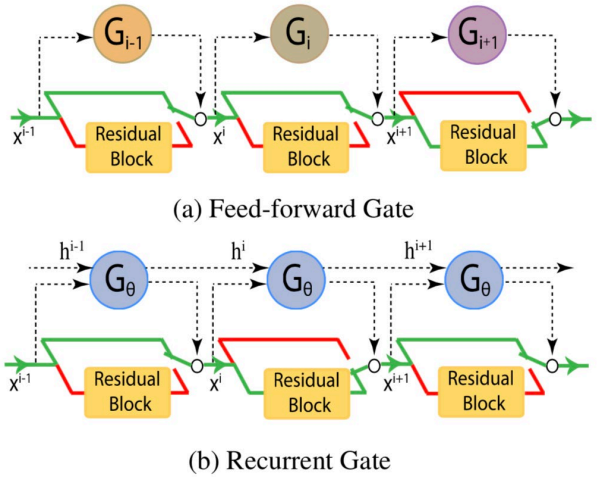


[Wang et al, 2018]

Gate Designs

- Gate takes input activations of a layer and gives a binary output, skip or not based on gating function $G^i(x^i) \in \{0,1\}$

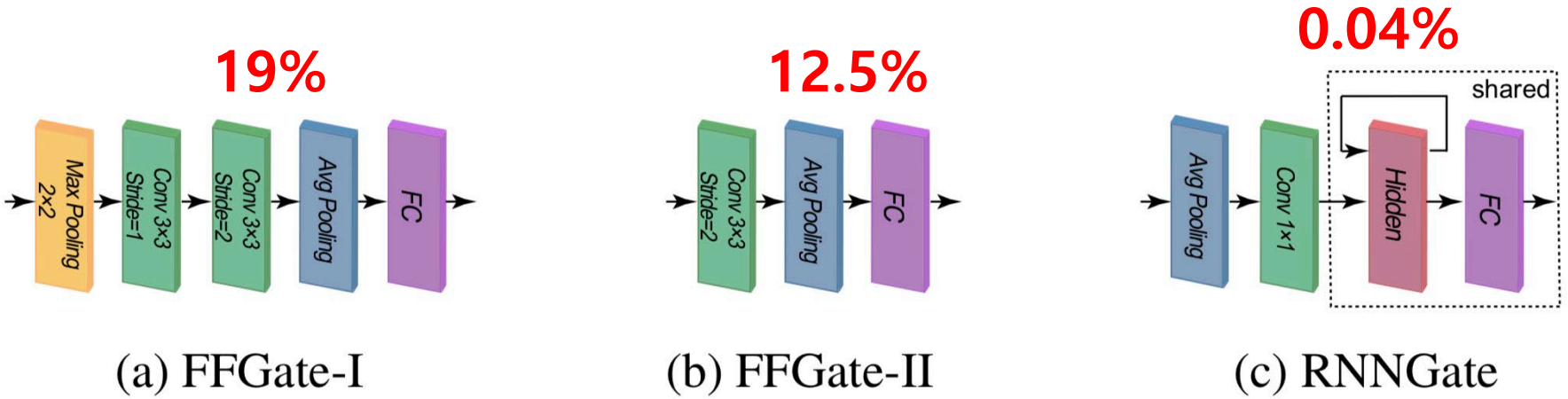
$$\mathbf{x}^{i+1} = G^i(\mathbf{x}^i)F^i(\mathbf{x}^i) + (1 - G^i(\mathbf{x}^i))\mathbf{x}^i$$



(a) Feed-forward Gate

(b) Recurrent Gate

- The computation overhead of gate operation needs to be small
- RNNGate gives good results at a very small computation cost, 0.04% of residual block cost



(a) FFGate-I (b) FFGate-II (c) RNNGate

[Wang et al, 2018]

How to Train SkipNet?

- Reinforcement learning (RL)
  - Loss = accuracy loss + (-sum of rewards)

- Reward, R_i = C_i if layer i is skipped
  - C_i = computation cost of layer i

Reward of gate i

$$R_i = (1 - g_i)C_i$$

$$\min \mathcal{J}(\theta) = \min \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} L_\theta(\mathbf{g}, \mathbf{x})$$

N gates in the network

$$= \min \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} \left[\mathcal{L}(\hat{y}(\mathbf{x}, F_\theta, \mathbf{g}), y) - \frac{\alpha}{N} \sum_{i=1}^{N} R_i \right]$$

Accuracy loss

[Wang et al, 2018]

Gradient, Pre-Training and Fine-Tuning

- Based on REINFORCEMENT rule

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\mathbf{x}} \nabla_\theta \sum p_\theta(\mathbf{g}|\mathbf{x}) L_\theta(\mathbf{g}, \mathbf{x})$$

$$= \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} \nabla_\theta \mathcal{L} - \mathbb{E}_{\mathbf{x}} \mathbb{E}_{\mathbf{g}} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(g_i|\mathbf{x}) r_i$$

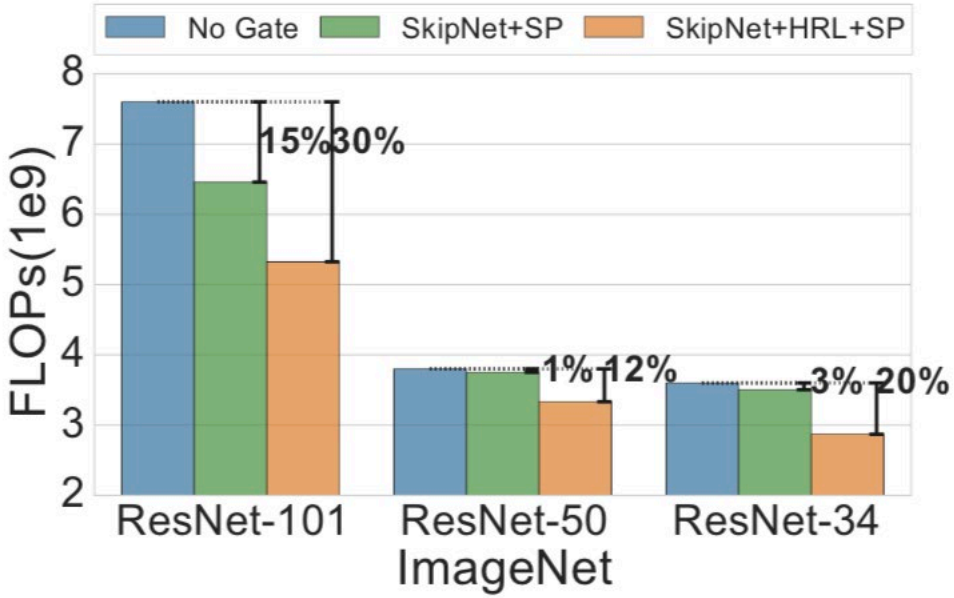
Loss in Policy Gradient

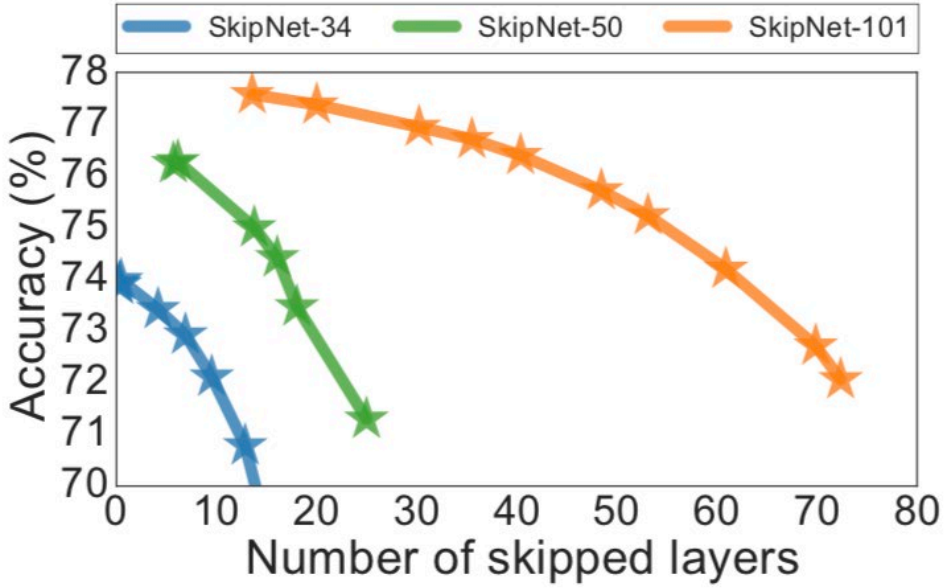
- Supervised pre-training
  - Based on straight-through estimator(STE)
  - Hard-gating during the forward pass with *soft-gating* during backpropagation
  - Handles g_i as a binary value (by rounding) in forward pass and as a continuous value during back propagation
- Fine-tuning
  - Hard-gating during both forward and backward passes

[Wang et al, 2018]

Experimental Results

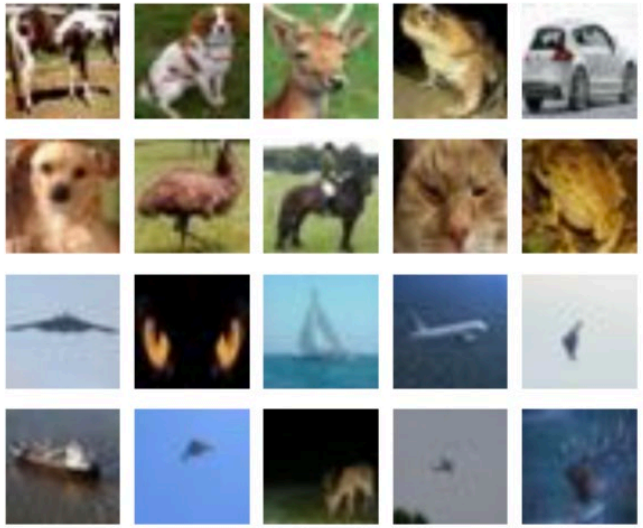
- ImageNet: 30% reduction in computation
- Accuracy vs. # skipped layers
- Easy vs. hard images
  - Easy examples are brighter and clearer while hard ones tend to be dark and blurry





Easy images
skipping >15 layers
in a 74-layer network

Hard images
skipping <8 layers
in a 74-layer network

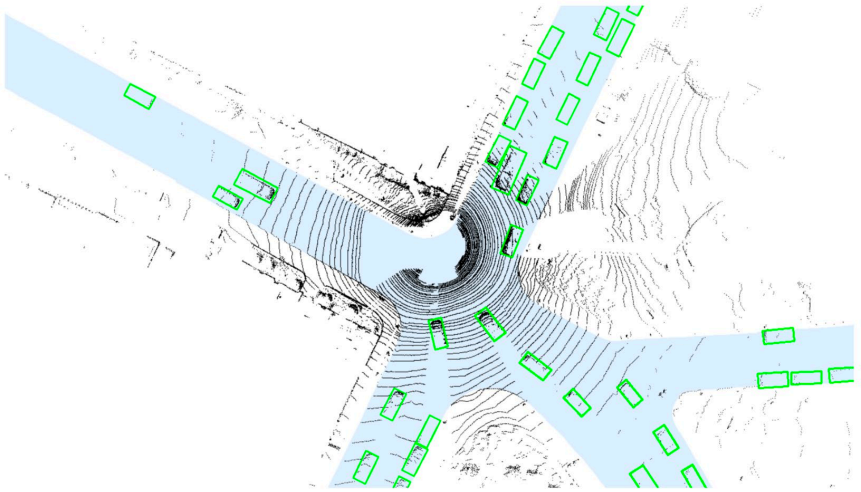


[Wang et al, 2018]

Sparse Convolution?

- In real life, most visual applications only need to be applied on top of the specific area
  - Only the areas on the road for object detection in autonomous driving
  - Occluded and fast-moving pixels in video segmentation
  - Voxel occupancy in 3D object classification

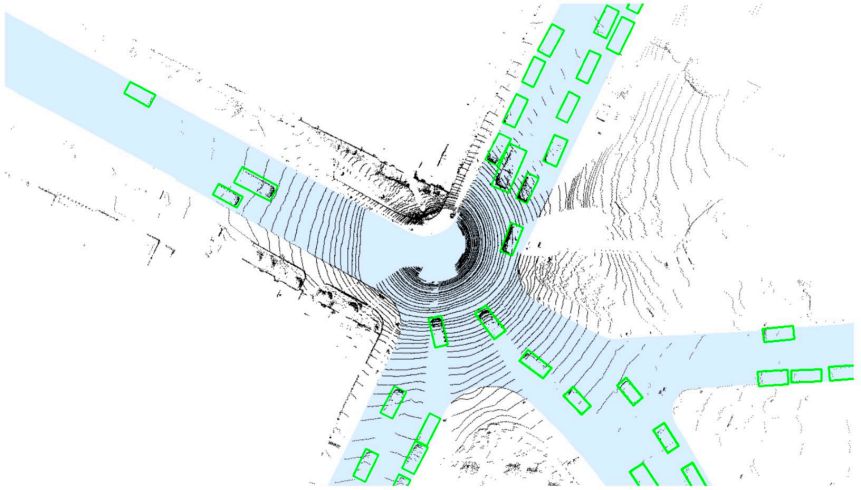
- How to focus the visual application on the interesting area?
  - Do computation only on the **mask** region



SBNet:
Sparse Blocks Network for Fast Inference

- In this paper, given a bird's eye view based on 3D LiDAR, perform object (e.g., car) detection on the road

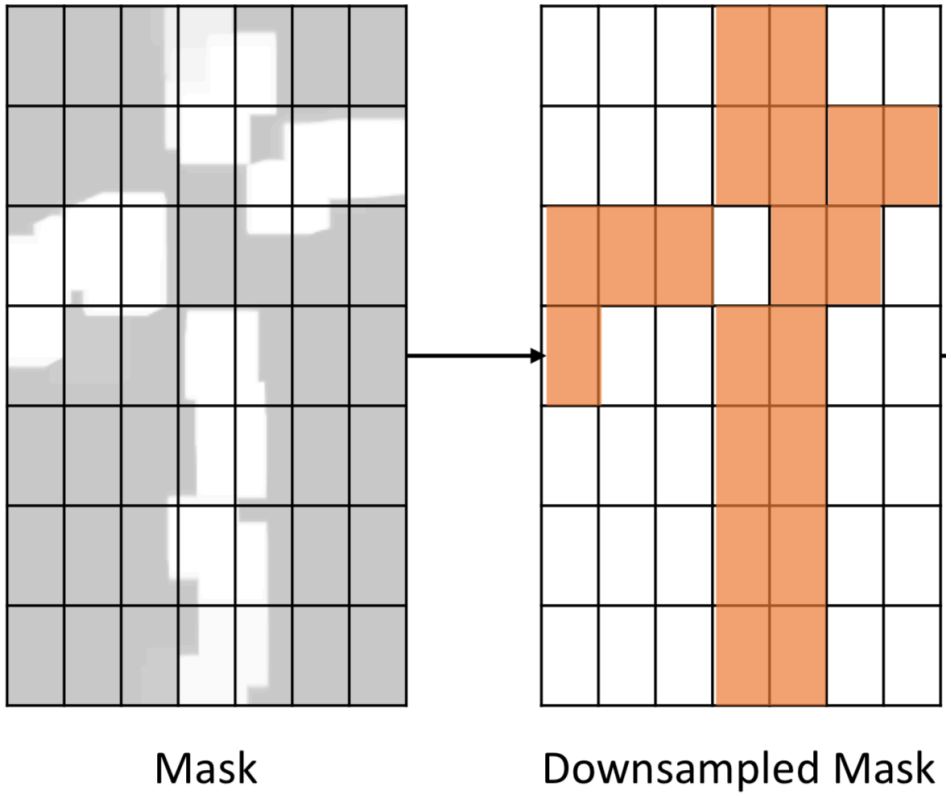
- How to obtain a mask?
  - based on road map & GPS or
  - image segmentation at (8X) coarser granularity
    - Minimize computation overhead
      while achieving accurate mask for the following task



[Ren et al, 2018]

How to Perform Convolution Selectively?

- How to perform convolution, i.e., dense computation for mask area?
  - Most conventional libraries are optimized for the dense convolution computation
- Implementing sparse convolution?
  - Hard to achieve maximum performance regardless of the GPU and input size
  - Laborious and time-consuming
- Different approach?
  - Based on gather / scatter with dense convolution kernel

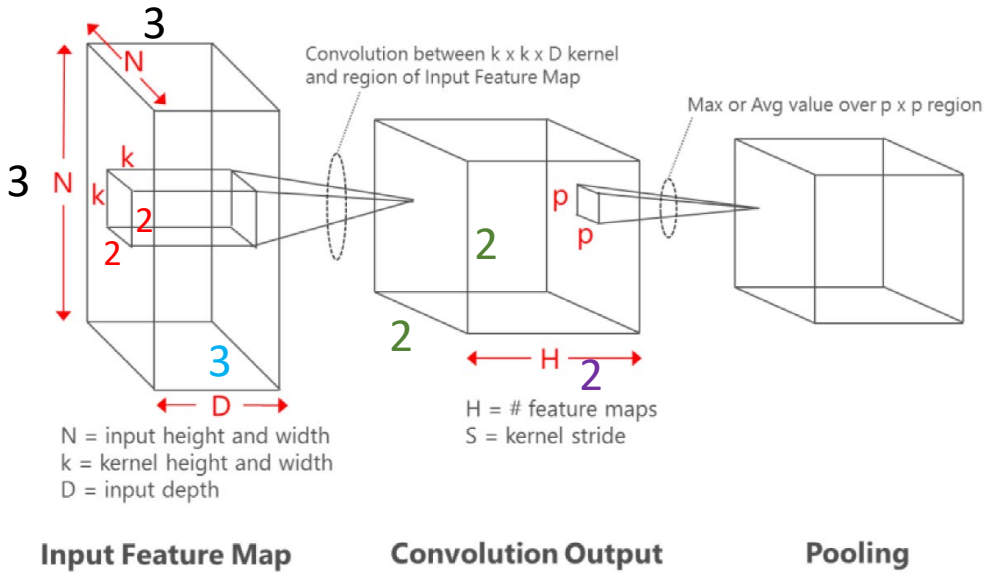


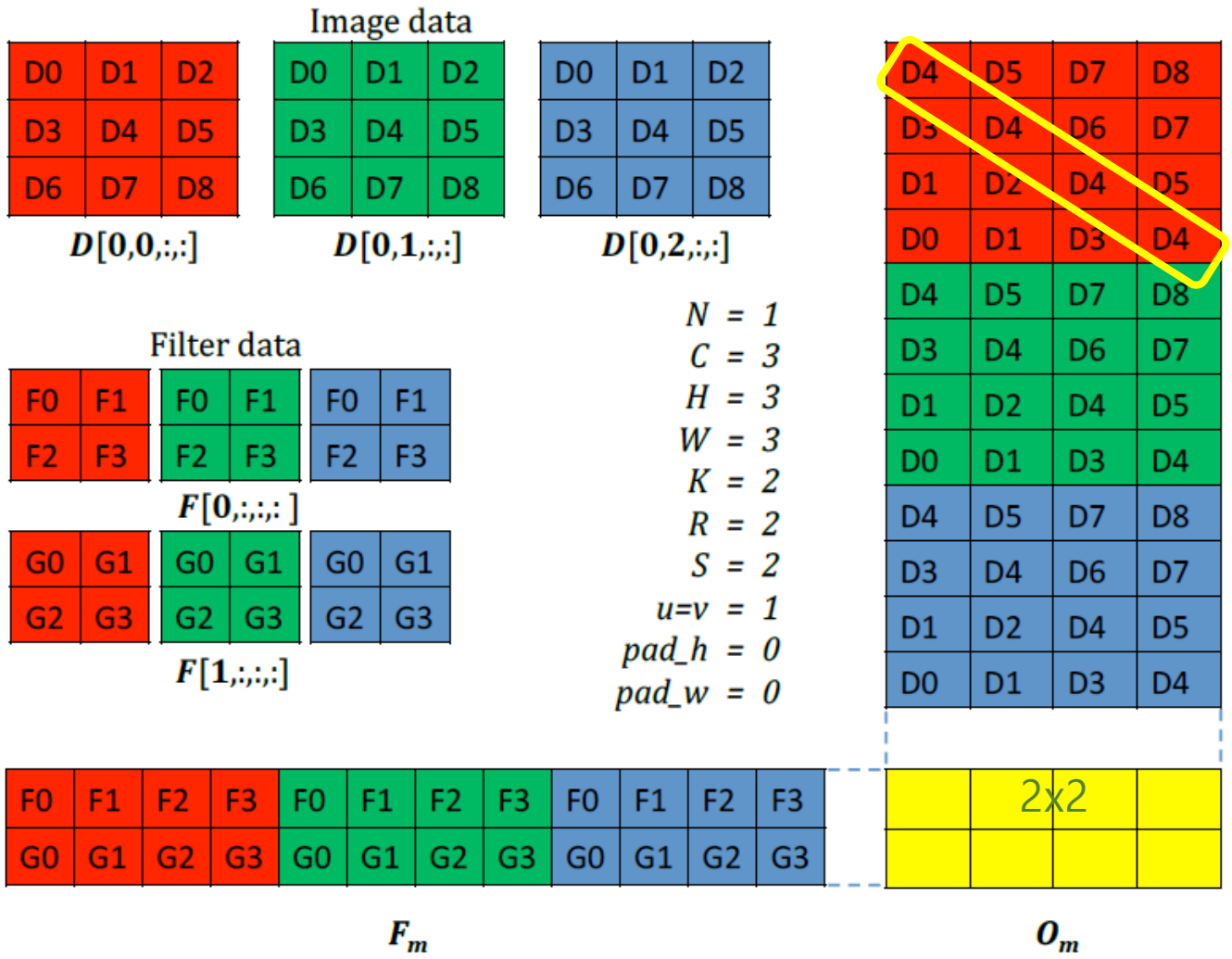
[Ren et al, 2018]

[Chetlur, 2014]
[S.Yoo]

Review: Convolution Lowering

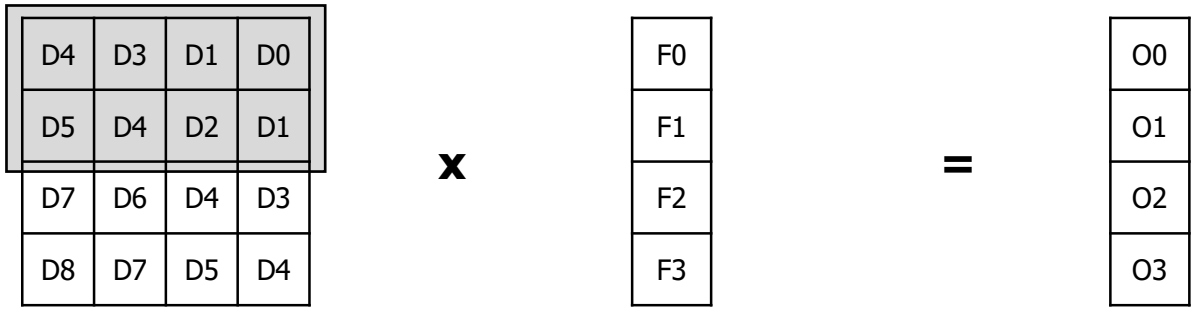
- Input: 3x3x3
- Output: 2x2x2
- Convolutional kernel: 3x2x2

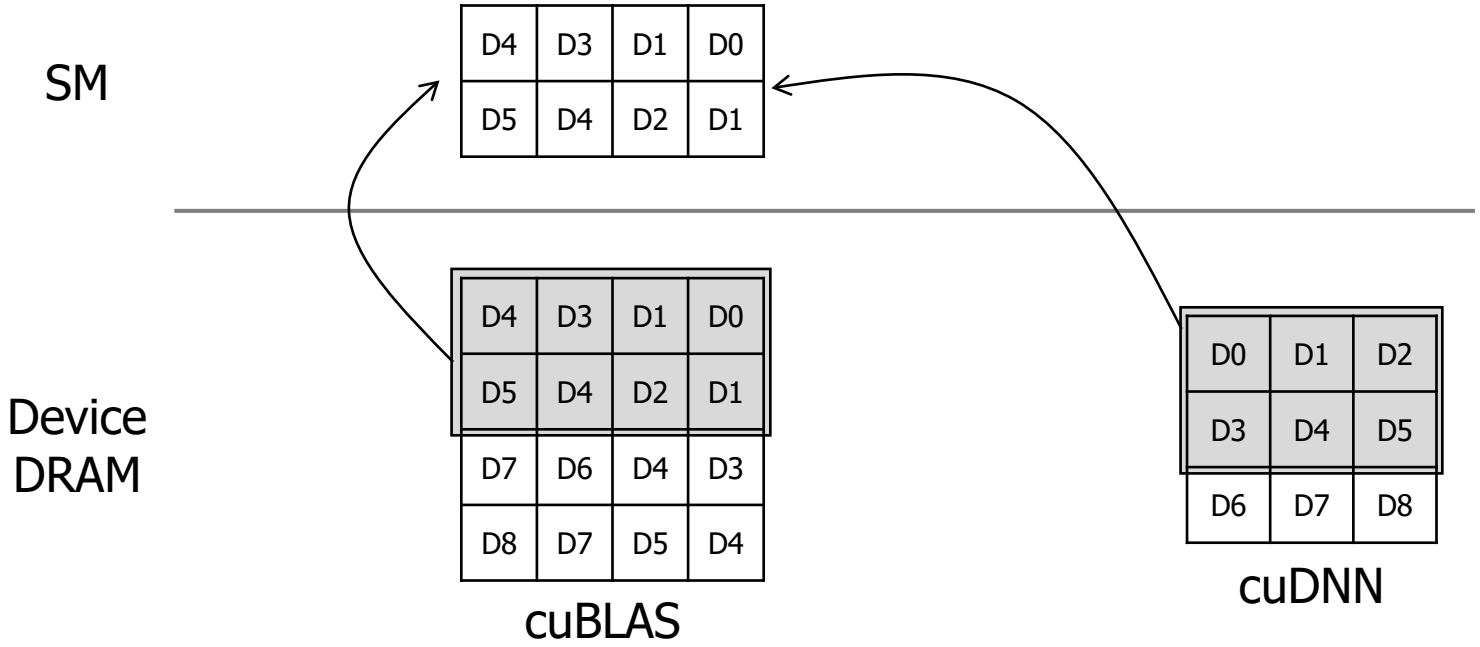


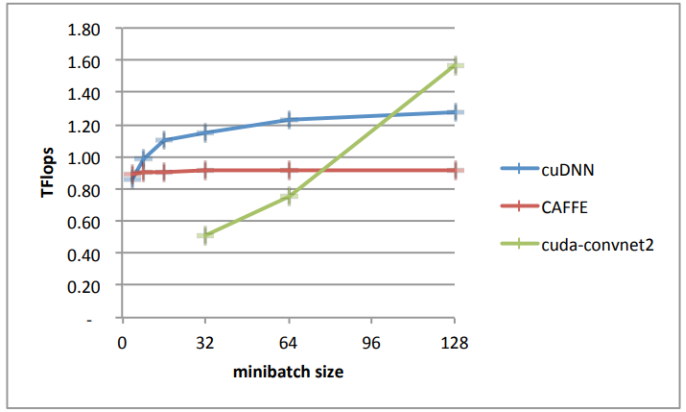


Review: cuBLAS vs. cuDNN

- In the case of cuDNN, it duplicates the data internally with indexing
  - Trade off between index computation and memory resource(capacity & bandwidth)







[cuDNN: Efficient Primitives for Deep Learning]

Review: Winograd Convolution – 2D

- 2D convolution is an extension version of 1D Winograd convolution

$$\begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_0 + m_1 + m_2 \\ m_1 - m_2 - m_3 \end{bmatrix}$$

$$m_0 = (d_0 - d_2)g_0 \qquad m_1 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$

$$m_2 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2} \qquad m_3 = (d_1 - d_3)g_2$$

1D $Y = A^{\mathrm{T}}[U \odot V] = A^{\mathrm{T}}[(Gg) \odot (B^{\mathrm{T}}d)]$

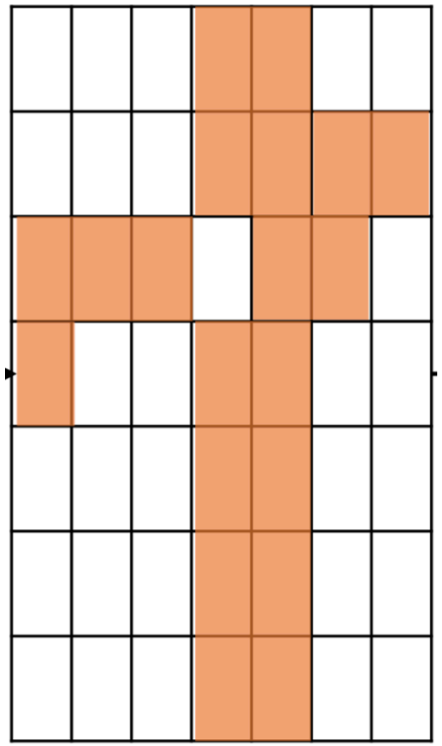
2D $Y = A^{\mathrm{T}}[(GgG^{\mathrm{T}}) \odot (B^{\mathrm{T}}dB)]A$

$$A^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

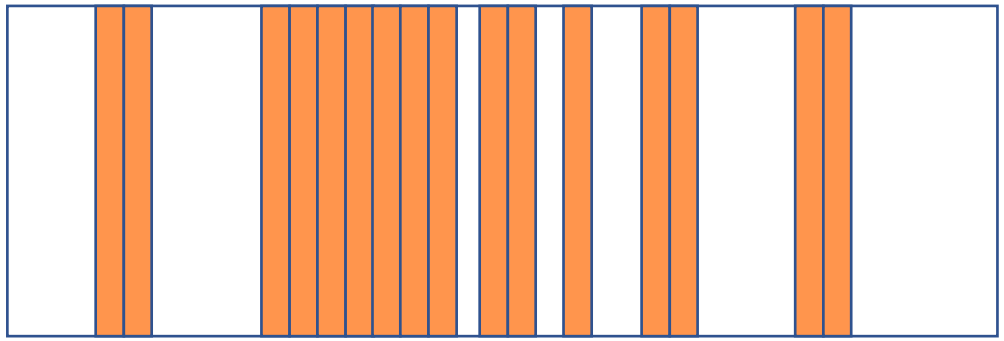
$$B^{\mathrm{T}} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

[Fast Algorithms for Convolutional Neural Networks]

Gathering Tiles to Form a Smaller Dense Matrix



Input feature matrix

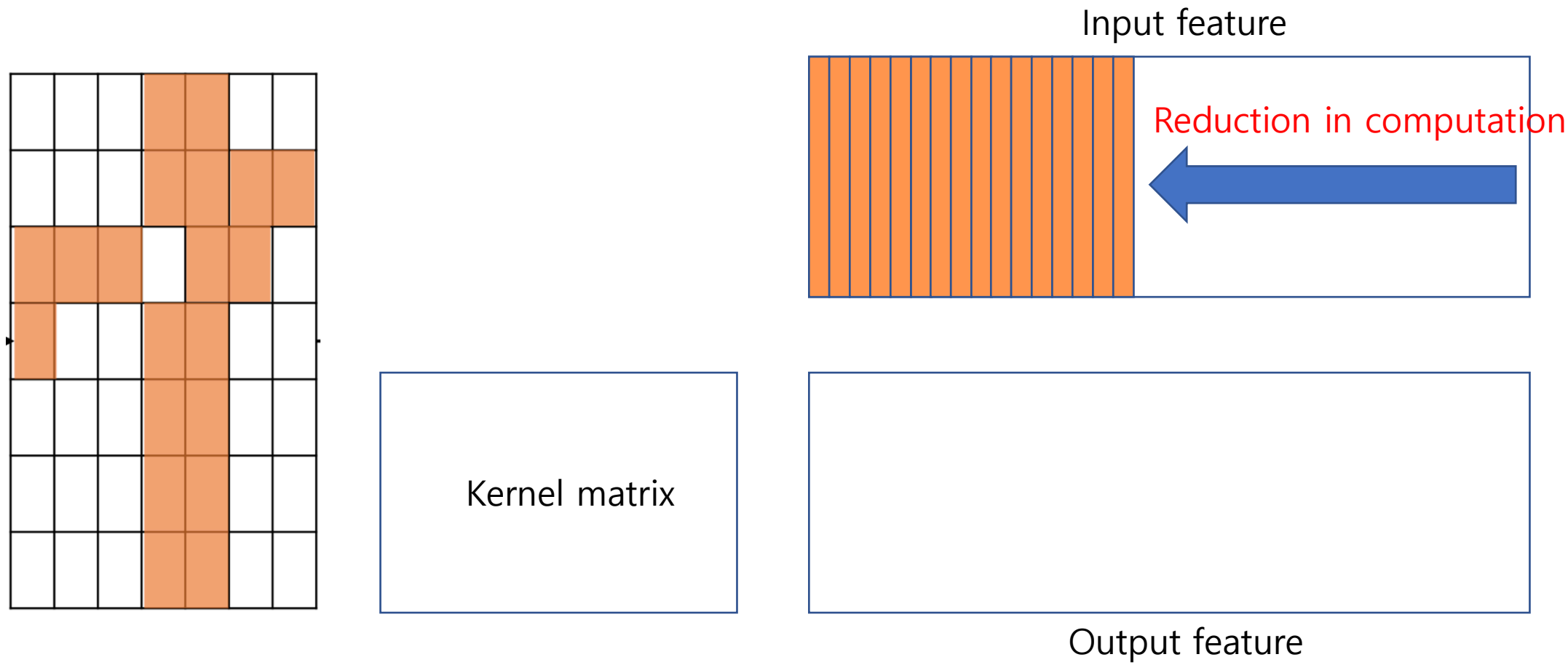


Kernel matrix

Output feature matrix

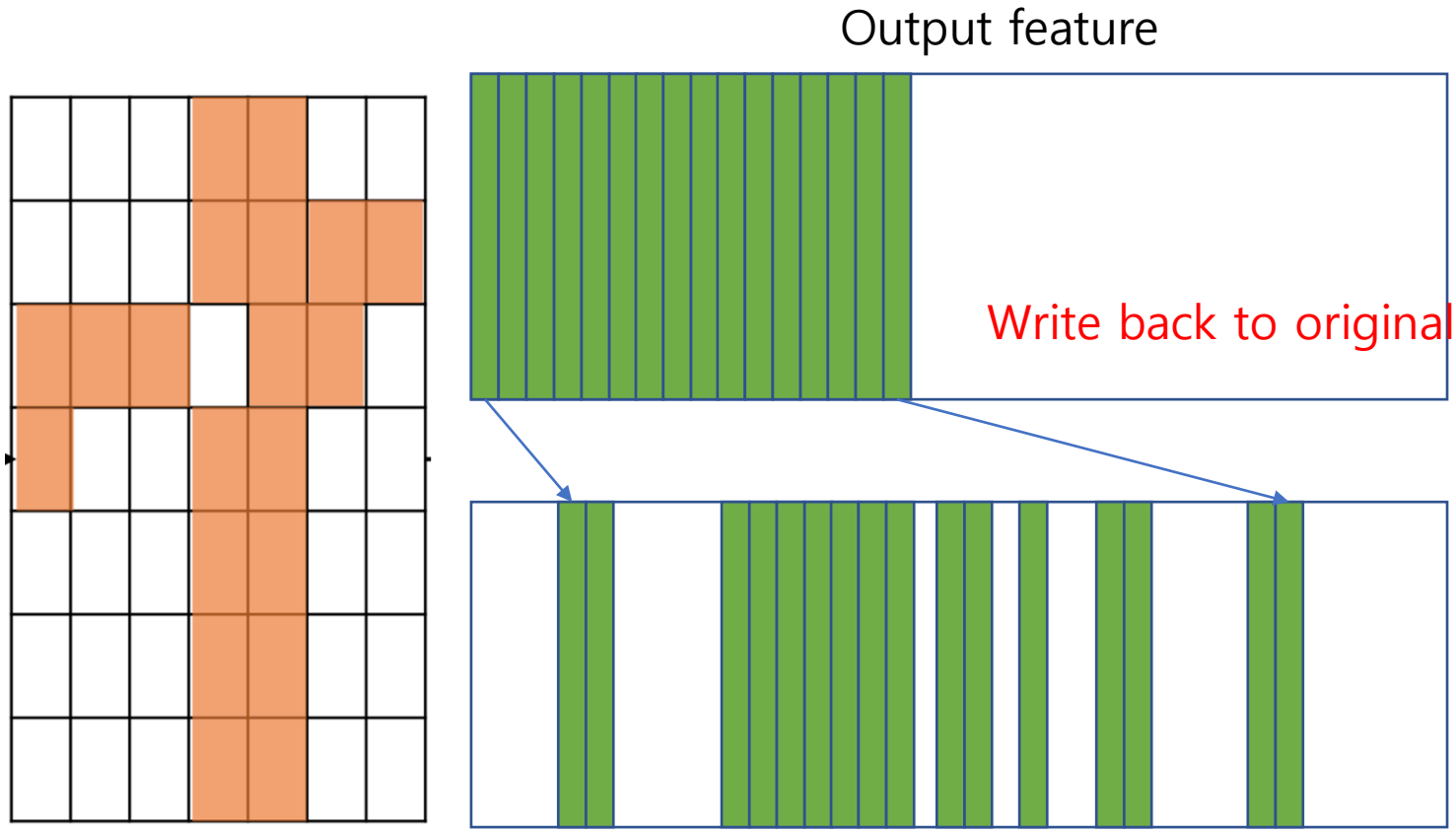
[Ren et al, 2018]
[S.Yoo]

Gathering Tiles to Form a Smaller Dense Matrix



[Ren et al, 2018]
[S.Yoo]

Scatter Outputs to Form a Smaller Dense Matrix



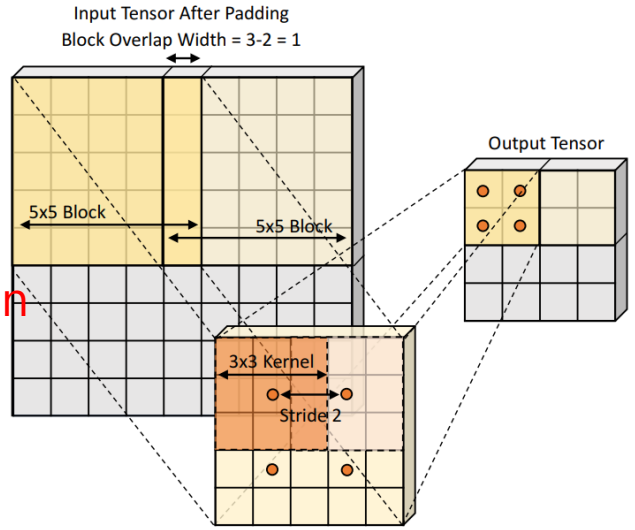


Figure 3: A toy example with block size=5, kernel size=3×3, kernel strides=2×2. Block strides are computed as $k - s = 3 - 2 = 1$.

Output blocks are non-overlapped to avoid atomic operation

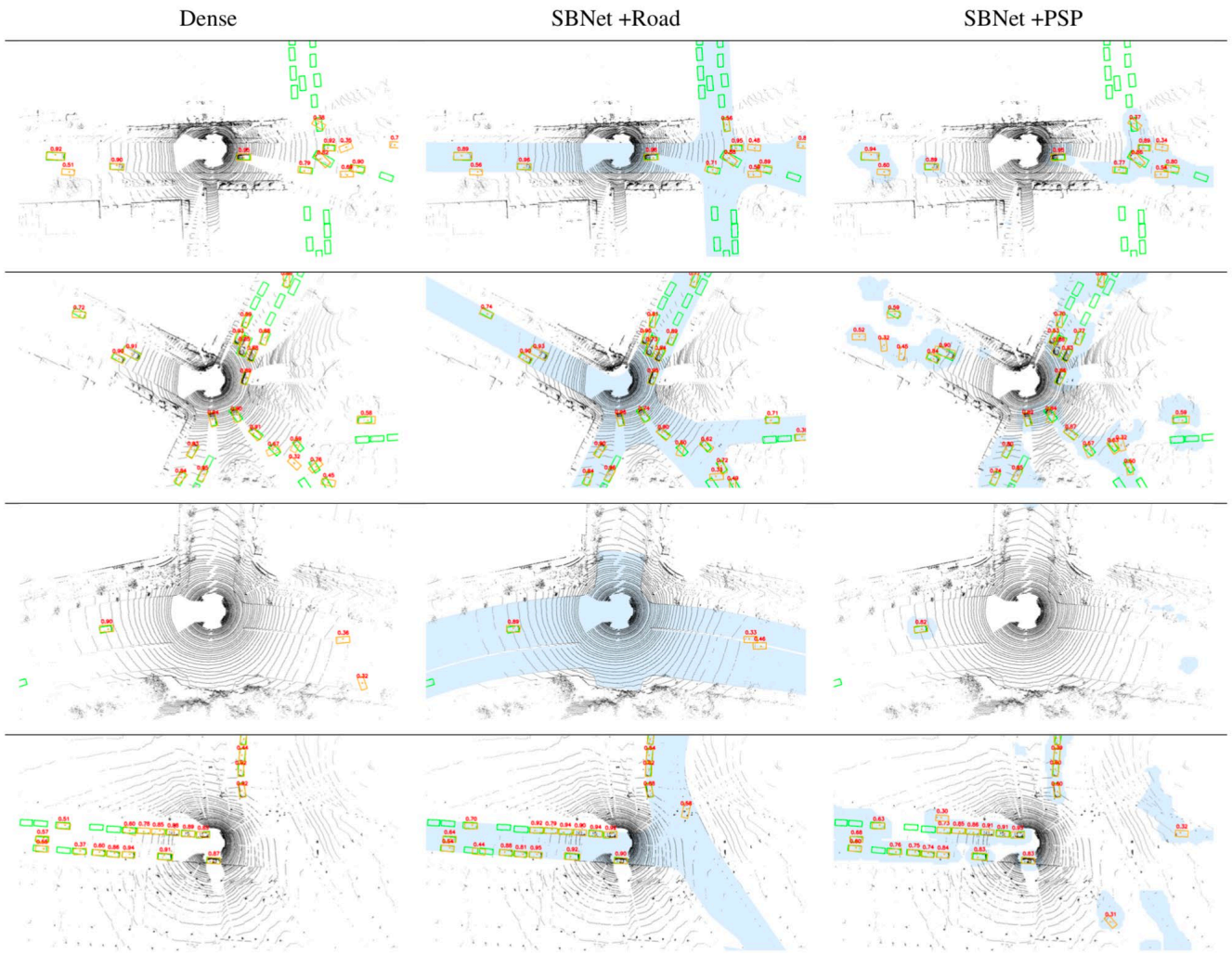
[Ren et al, 2018]

Results

- 2X speedup with
  even better accuracy with road mask

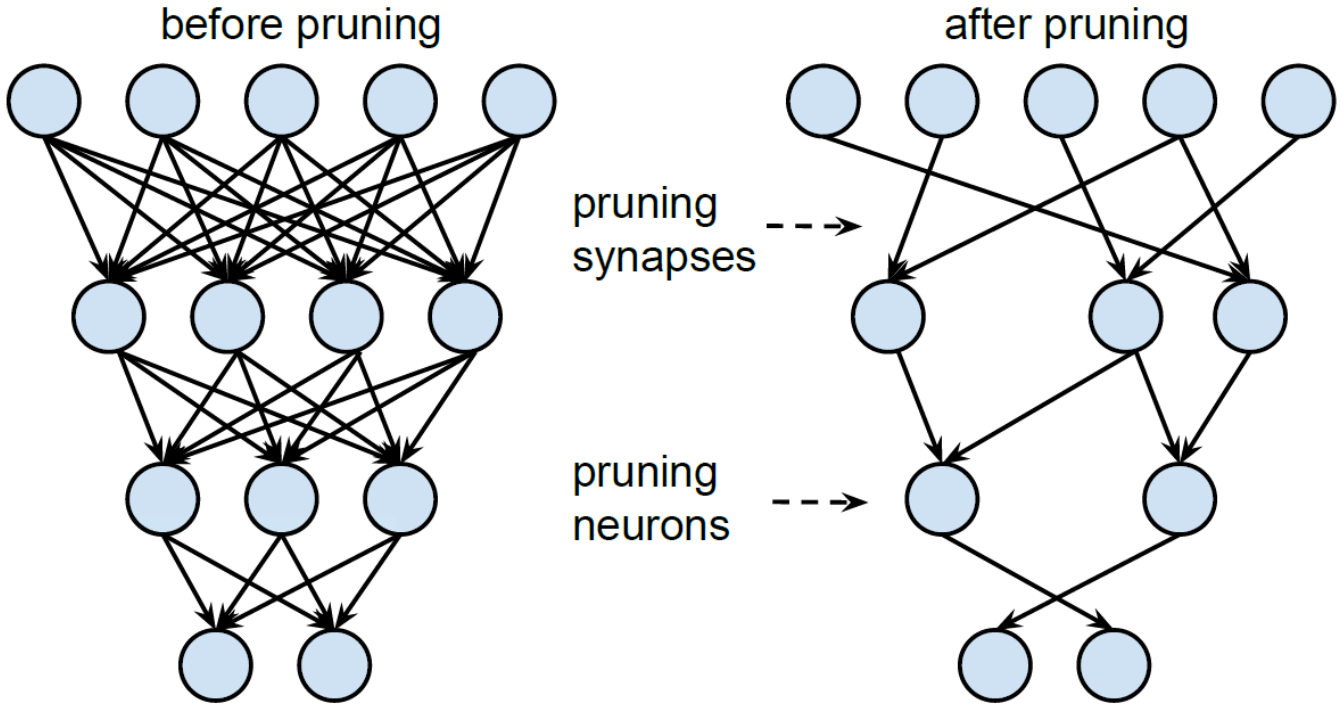
Model	Train Loss	Sparsity	Avg. Speed-up	mAP
Dense	No Mask	0%	1.0×	74.1
Dense	Road Mask	0%	1.0×	75.2
SBNet +Road	Road Mask	75%	2.03×	**77.0**
SBNet +PSP	PSP Mask	90%	**3.08**×	73.8

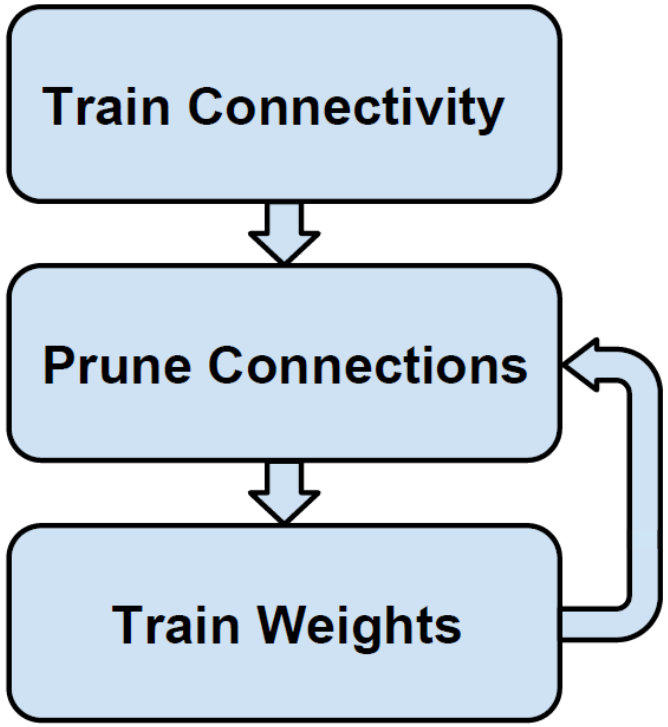
Network	Resolution	Time (ms)
Dense	800 × 1408	83.9
SBNet +Road	800 × 1408	41.3
SBNet +PSP	800 × 1408	27.2
PSPNet	100 × 176	3.2



[Ren et al, 2018]

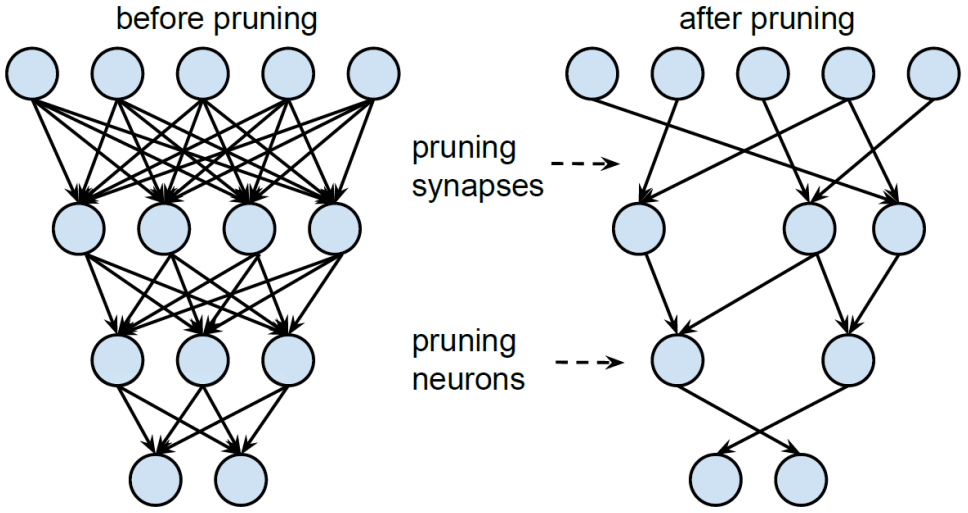
Pruning CNN: NIPS 2015

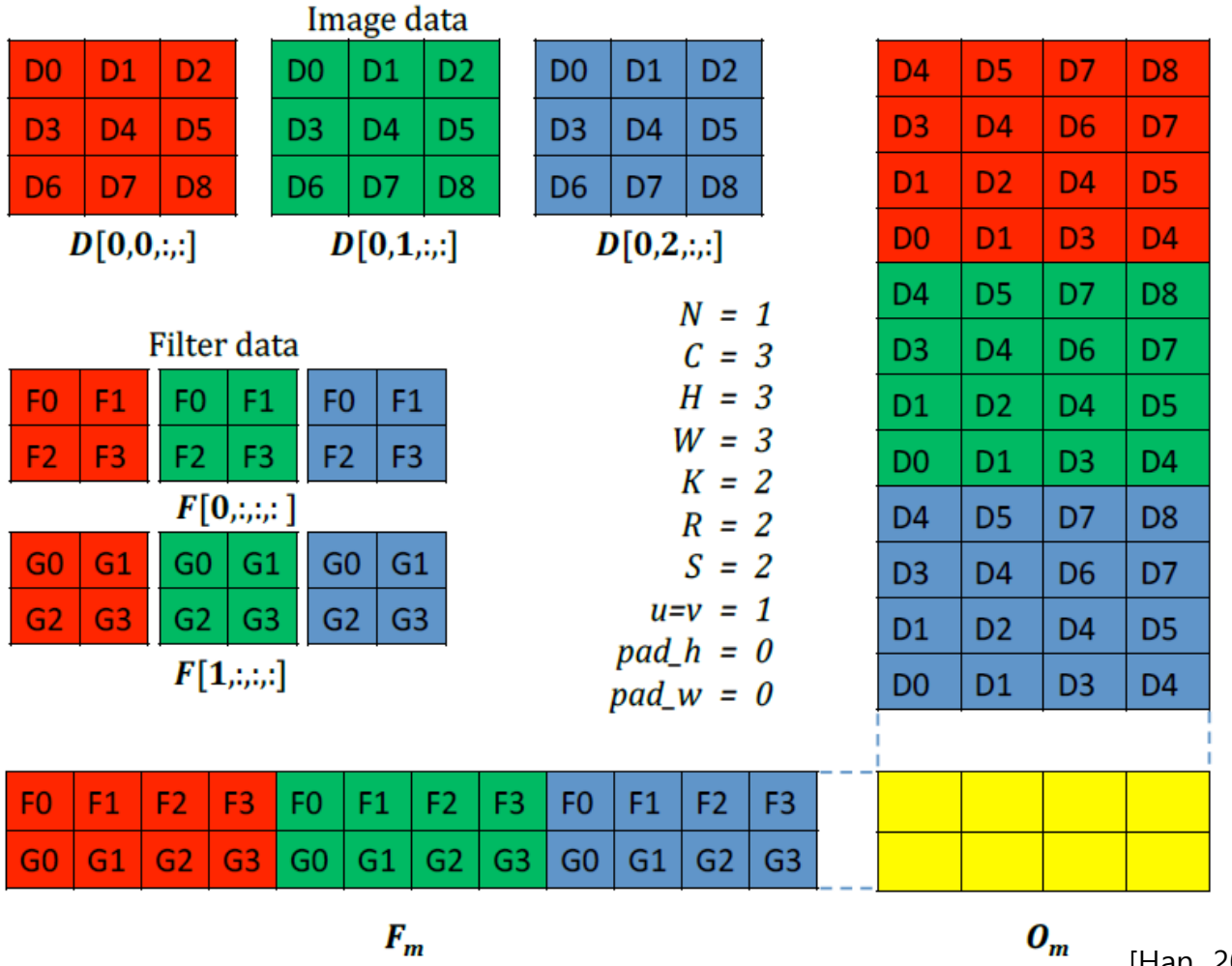




[Han, 2015]

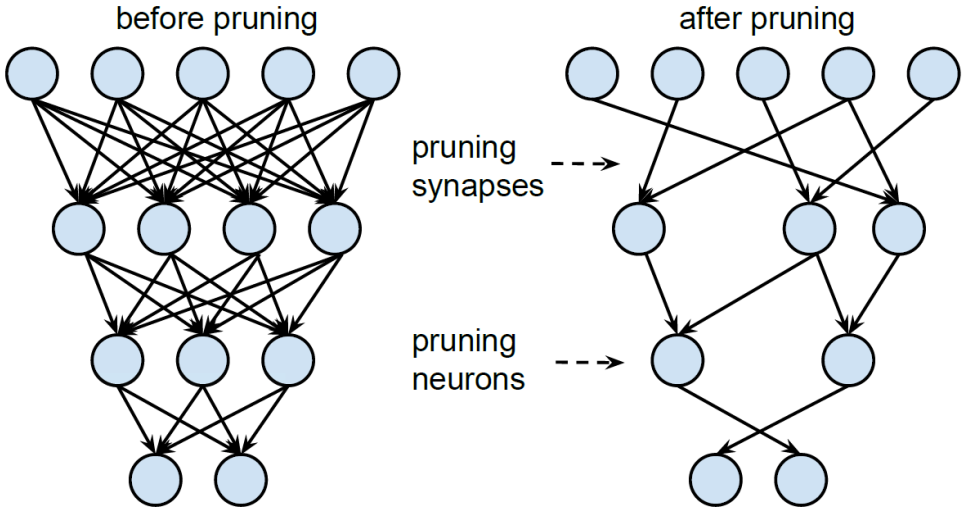
Pruning ~ Acceleration?

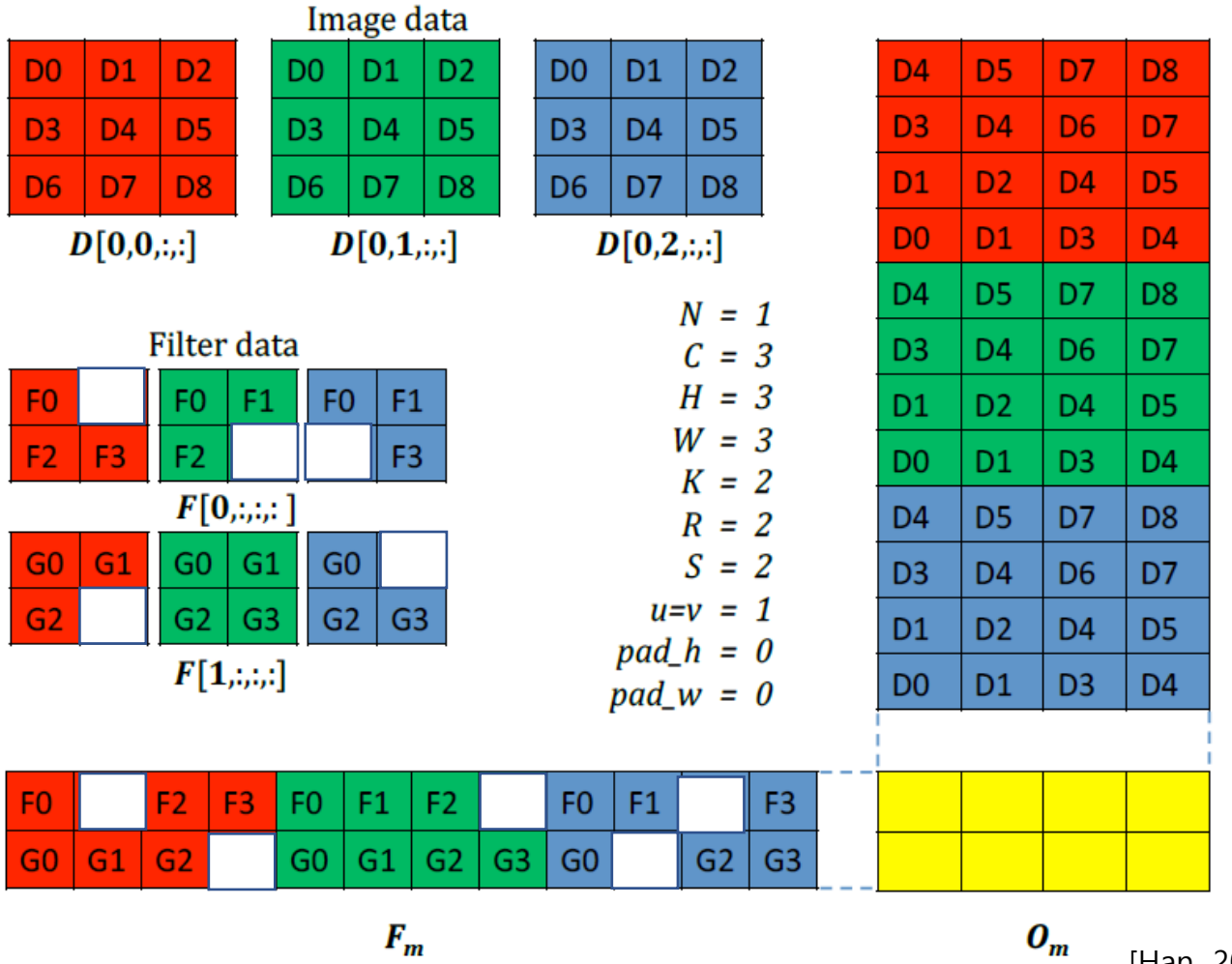




[Han, 2015]
[Chetlur, 2014]

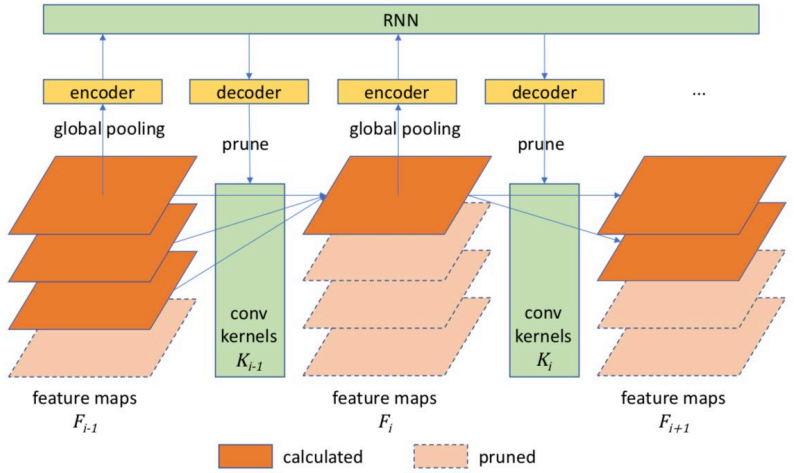
Pruning ~ Acceleration?





[Han, 2015]
[Chetlur, 2014]

Runtime Neural Pruning for Dynamic Channel Selection

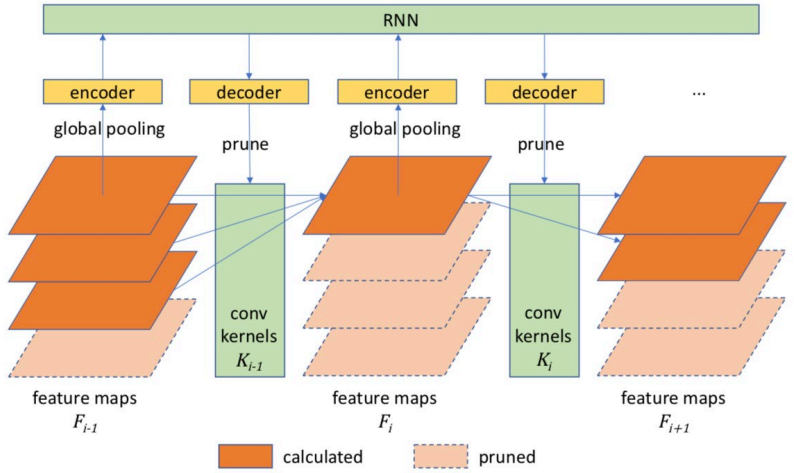


- Backbone CNN + decision network

- Design time
  - Group channels into four groups
  - Given a trained network, train a decision network based on reinforcement learning (RL)
    - State and reward?
- Runtime
  - The decision network selects an action, i.e., some channels for computation while pruning, i.e., zeroing out the other channels

[Lin et al, 2017]

RL-based Strategy Learning



- Based on Markov decision model + Q-Learning
  - Determine the next action based on previous status
  - Q value prediction
    - Given a state, make a prediction on the future cumulative reward of actions
    - State: encoder does global average pooling of feature maps
    - Action: Decoder gives four outputs to determine which channel groups to use
  - Action selection
    - Select the action having maximum reward
    - ε-greedy strategy
      - An action is randomly selected at the probability of ε
        - Typically, 1 → 0.01 during training

$$Q(s_t, a_i) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|\pi], \qquad \min_{\theta} L_{re} = \mathbb{E}[r(s_t, a_i) + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_i)]^2,$$

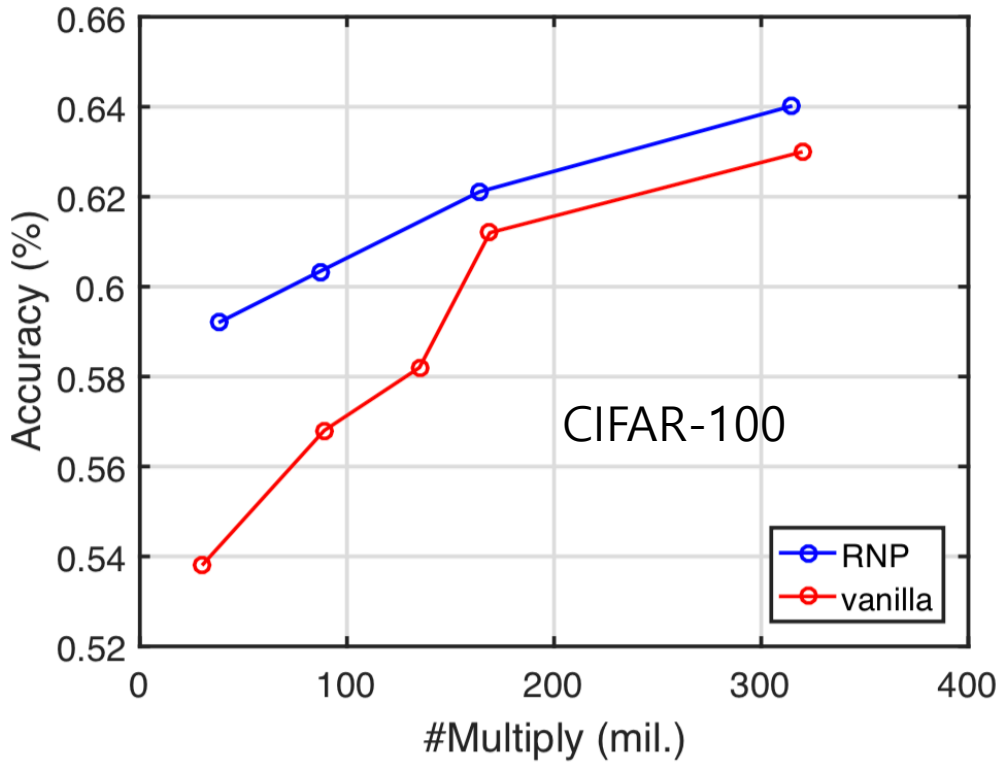
$Q(s_{t+1}, a_i)$

[Lin et al, 2017]

Results

- Vanilla solution
  - Each layer has a reduced # of channels to meet the given computation demand

- Runtime neural pruning
  - Smaller α
  - → more pruning
  - → less computation



$$r_t(a_i) = \begin{cases} -\alpha L_{cls} + (i-1) \times p, & \text{if inference terminates } (t = m-1), \\ (i-1) \times p, & \text{otherwise } (t < m-1) \end{cases}$$

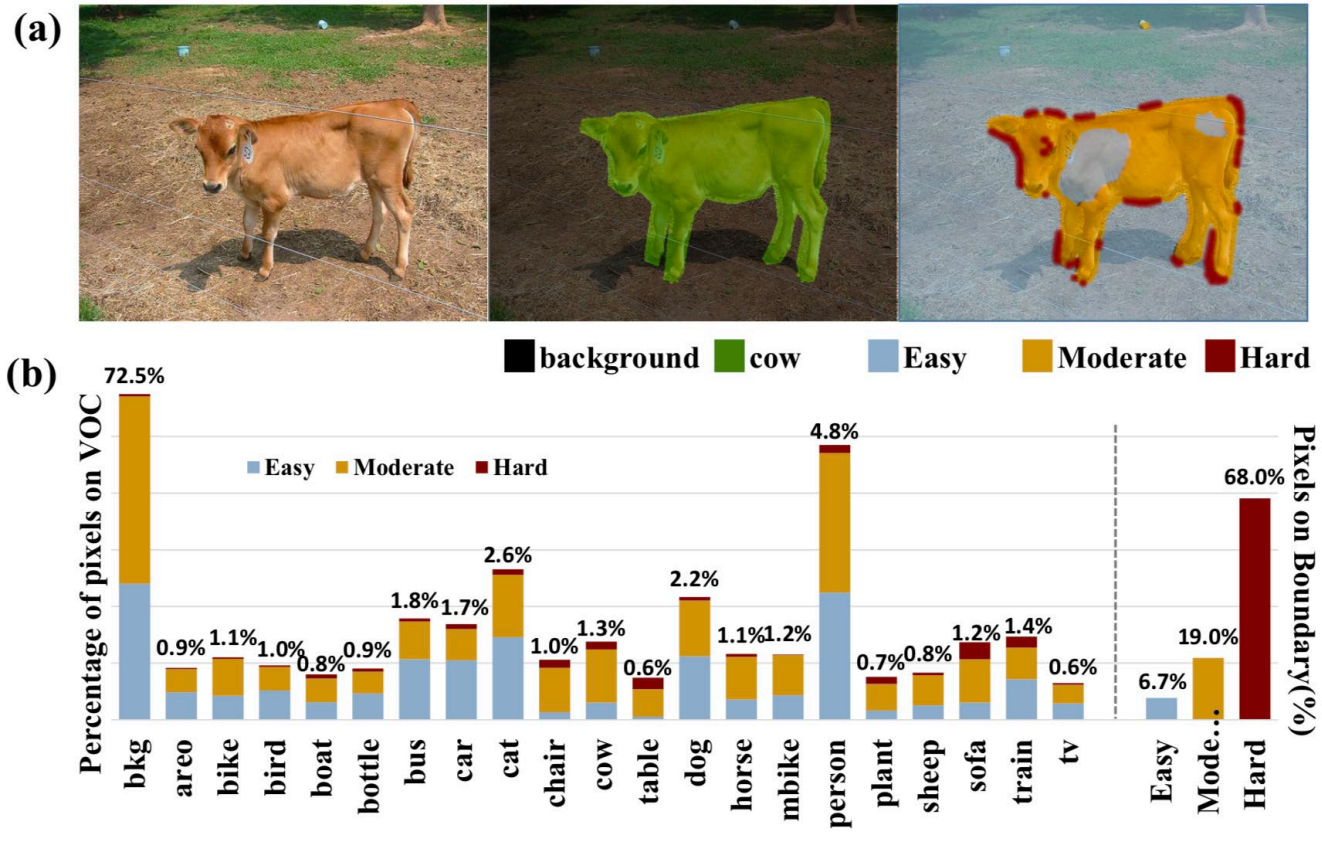
Additional top5 accuracy loss in ImageNet

Speed-up	3×	4×	5×	10×
Jaderberg *et al.* [19] ([46]'s implementation)	**2.3**	9.7	29.7	
Asymmetric [46]	-	3.84	-	
Filter pruning [27] (our implementation)	3.2	8.6	14.6	
Ours	2.32	**3.23**	**3.58**	**4.89**

[Lin et al, 2017]

Layer Cascade for Semantic Segmentation

- The majority of pixels is easy to classify
- We can reduce computation for easy/moderate pixels



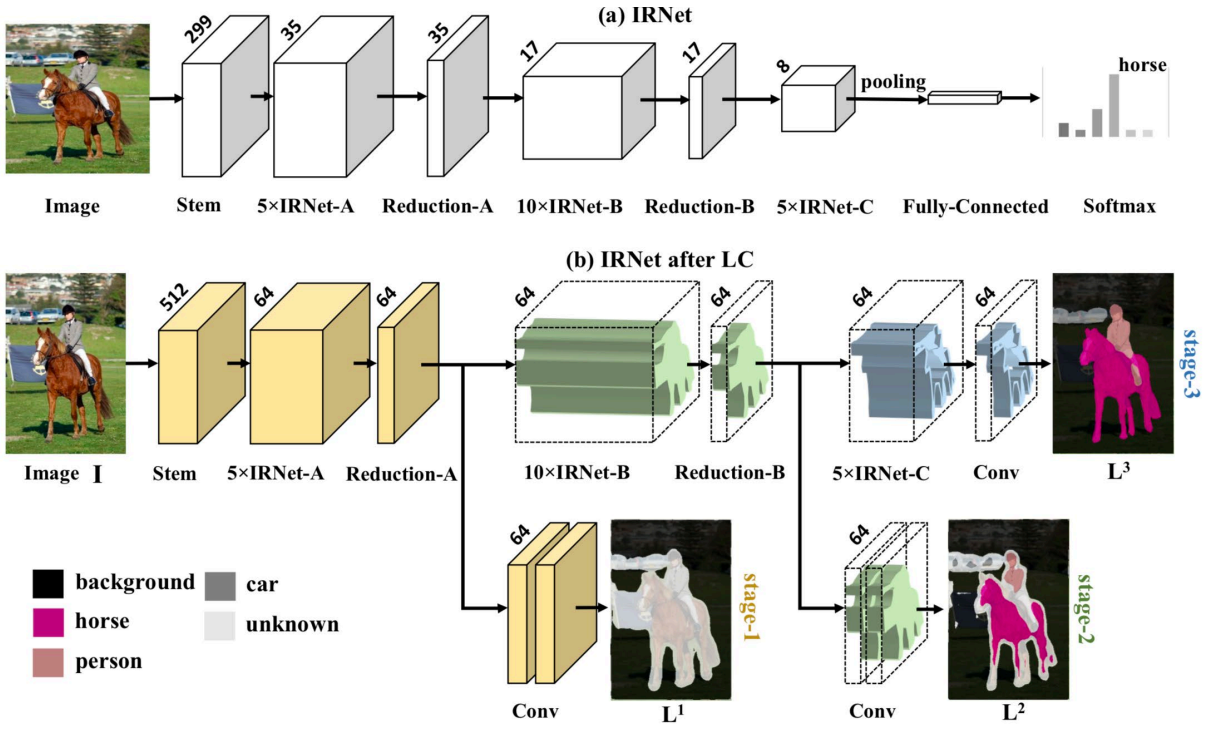
[Li et al, 2017]

Three-Stage Network

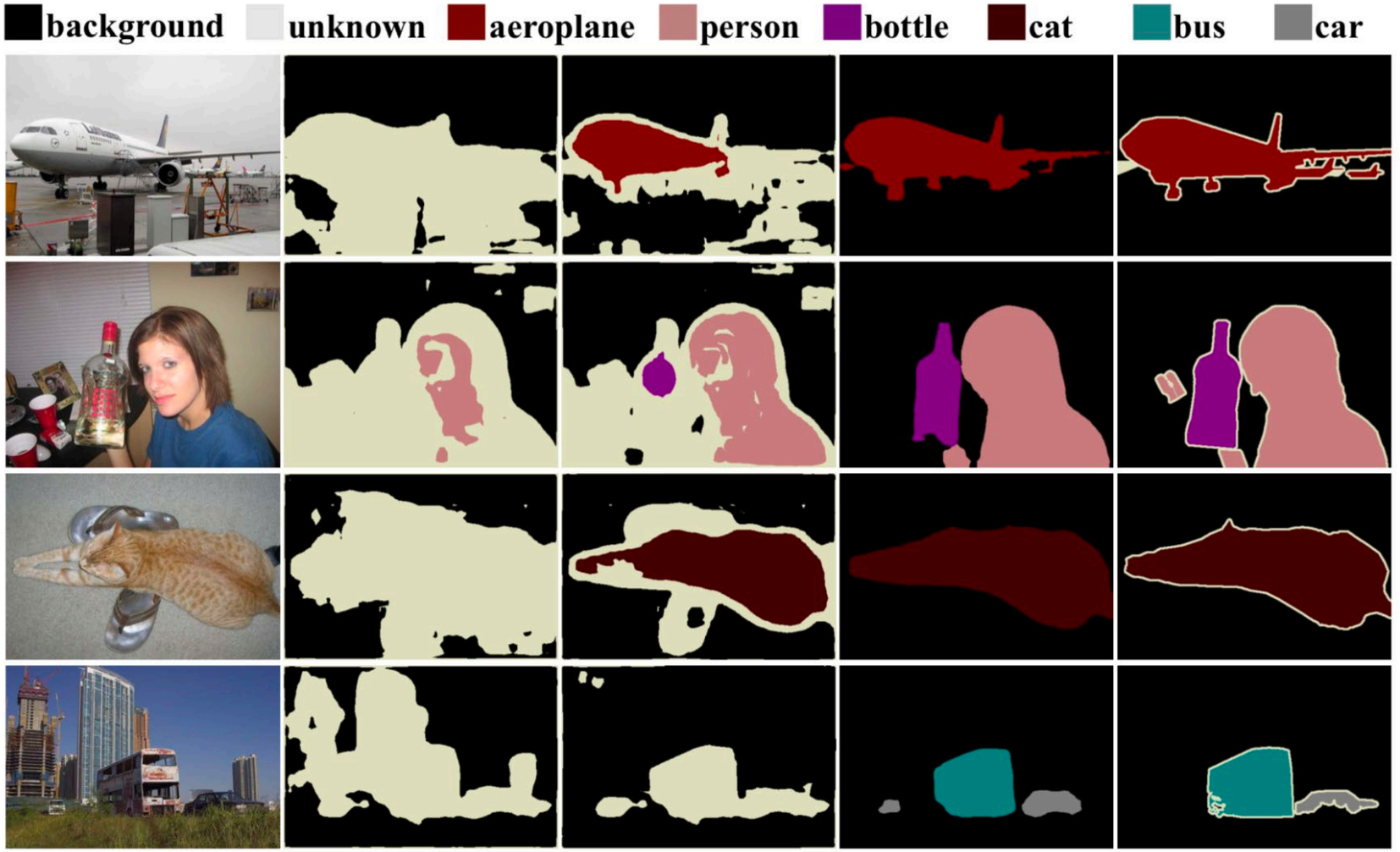
- Intermediate classifiers are inserted
  - Estimate the difficulty of the segmentation based on output confidence
  - If confidence > r, stop at the current layer

- Note: pixel-level decision
  - Sparse computation is needed

- Training
  - Initial stage
    - Training with a combined loss of all classifiers
  - Cascade stage
    - Training with corresponding pixels



Result Examples



[Li et al, 2017]

Results

- VOC12 results
- Threshold ρ is robust, 0.9~1
- Due to sparse nature of remaining pixels, it is difficult to reduce runtime with dense computation

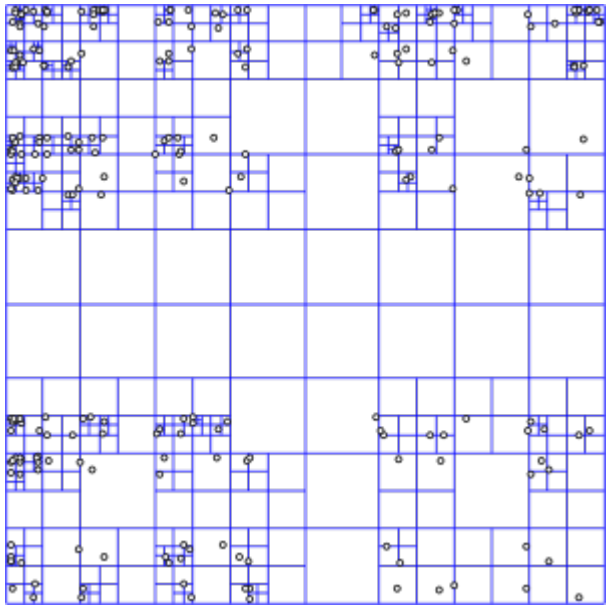
	mIoU(%)
IRNet [32]	72.22
DSN [17]	72.70
DSN [17] + Dropout [30]	72.63
Model Cascade (MC)	44.20
Layer Cascade (LC)	**73.91**

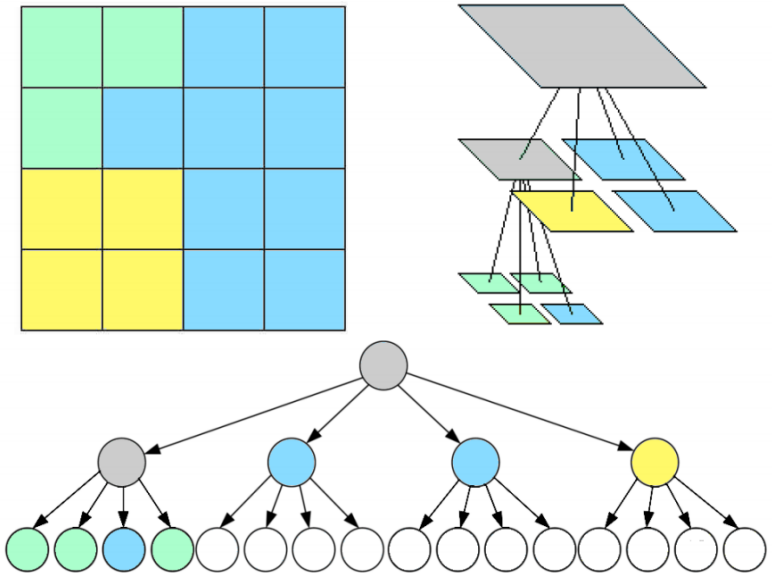
ρ	1	0.995	0.985	0.970	0.950	0.930	0.900	0.800
stage-1 (%)	0	15	23	30	35	35	44	56
stage-2 (%)	0	14	29	31	30	41	31	29
mIoU (%)	72.70	73.56	**73.91**	73.63	73.03	72.53	71.20	66.95

[Li et al, 2017]

Advanced Method for Difficulty-aware Segmentation

- Key idea
  - Decompose the target segmentation mask into a linear quadtree representation
- Quadtree: tree data structure with for children
  - Often used to compress or load balancing for 2-D data





Training QGN

- Ground truth generation by predict T-Pyramids recursively

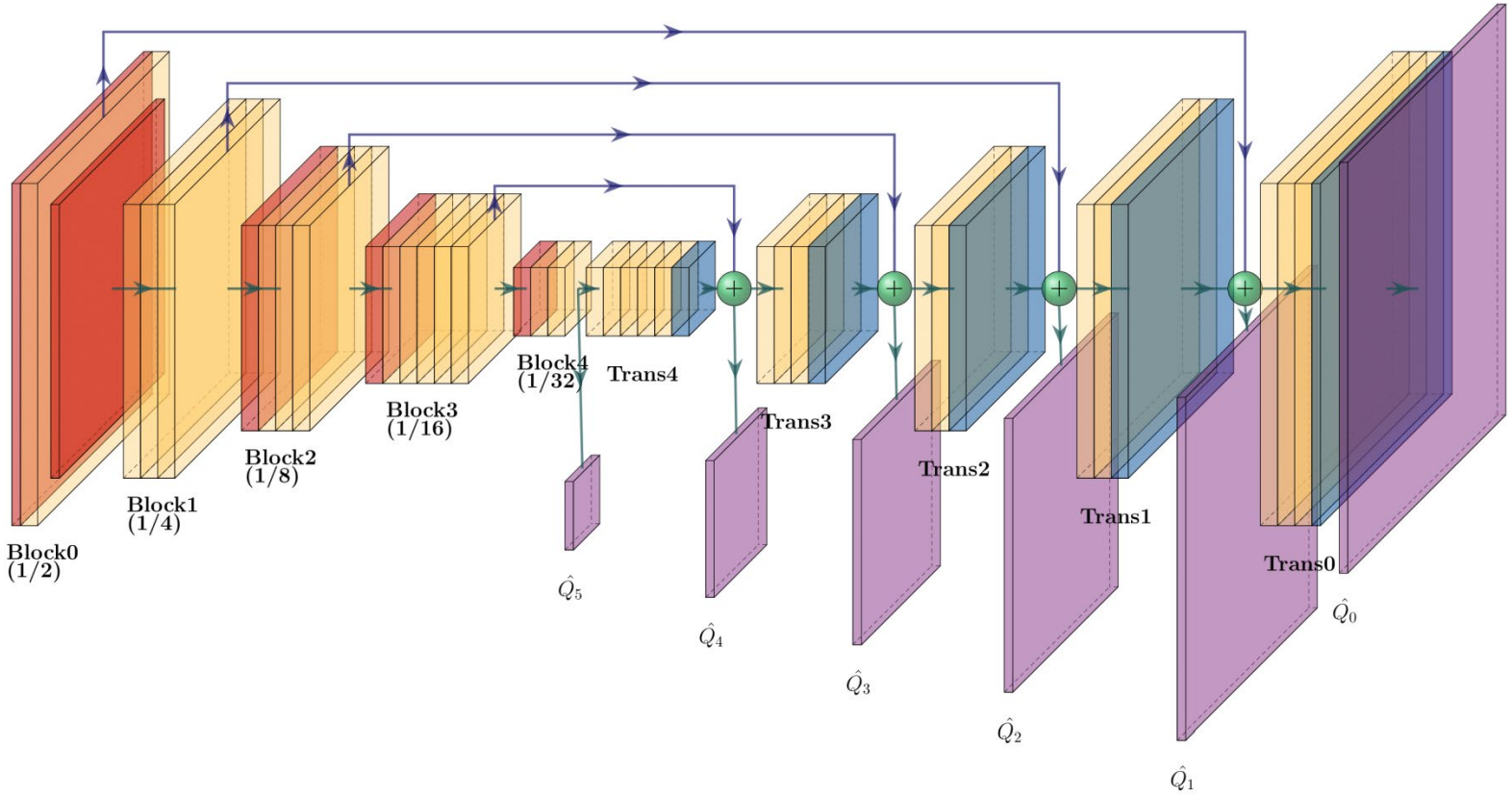
$$\mathcal{M}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = \begin{cases} a, & \text{if } a = b = c = d \\ 0, & \text{otherwise,} \end{cases}$$

$$T_{l+1} = \mathcal{M}(T_l).$$

- Train quad-tree prediction network with supervision

$$\mathcal{L}_l = \frac{1}{N} \sum_{i=1}^{N} \mathcal{H}(v^i, \mathcal{T}(l, x^i, y^i))$$

$$\mathcal{L} = \sum_{l \in Q} \beta_l \mathcal{L}_l.$$



Results and Limitations

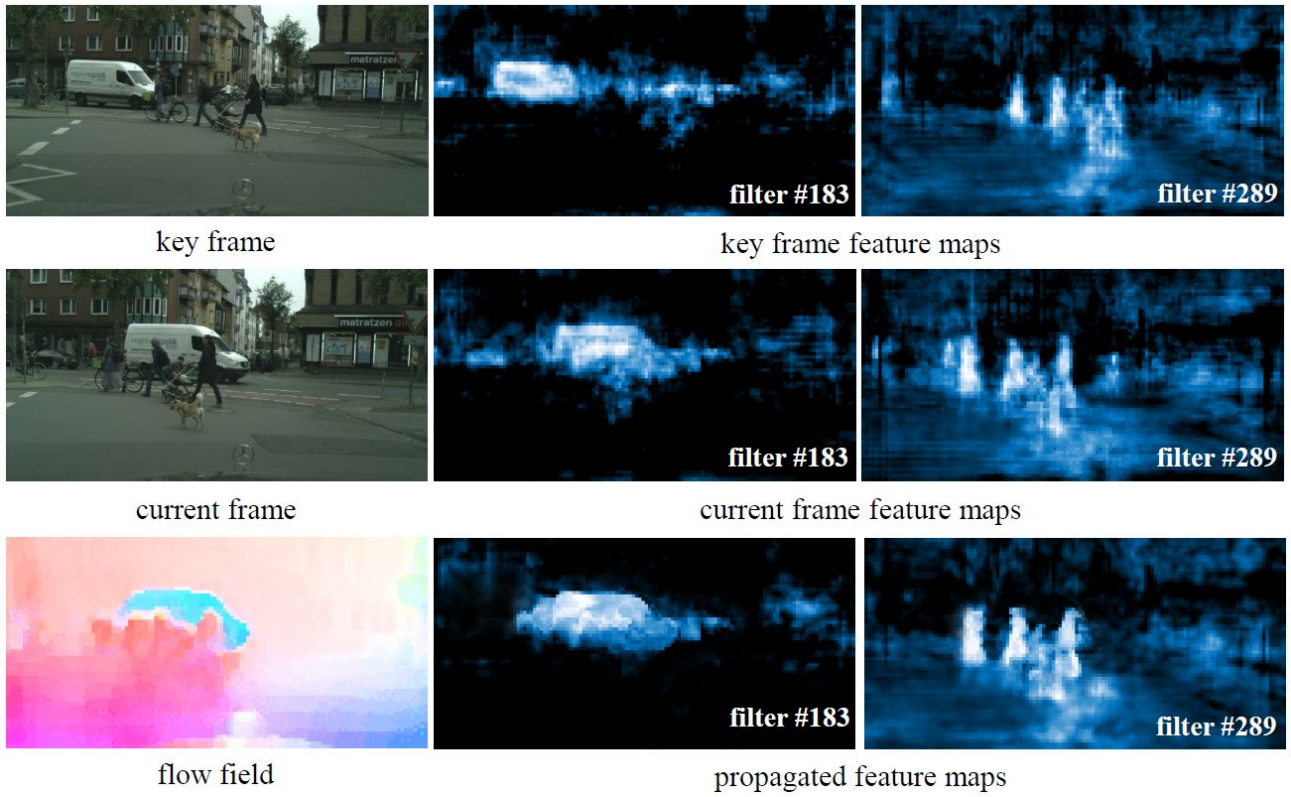
- Better performance with ground truth
- However, severe accuracy drop with predicted trees
  - We need the more accurate method for interesting region proposal

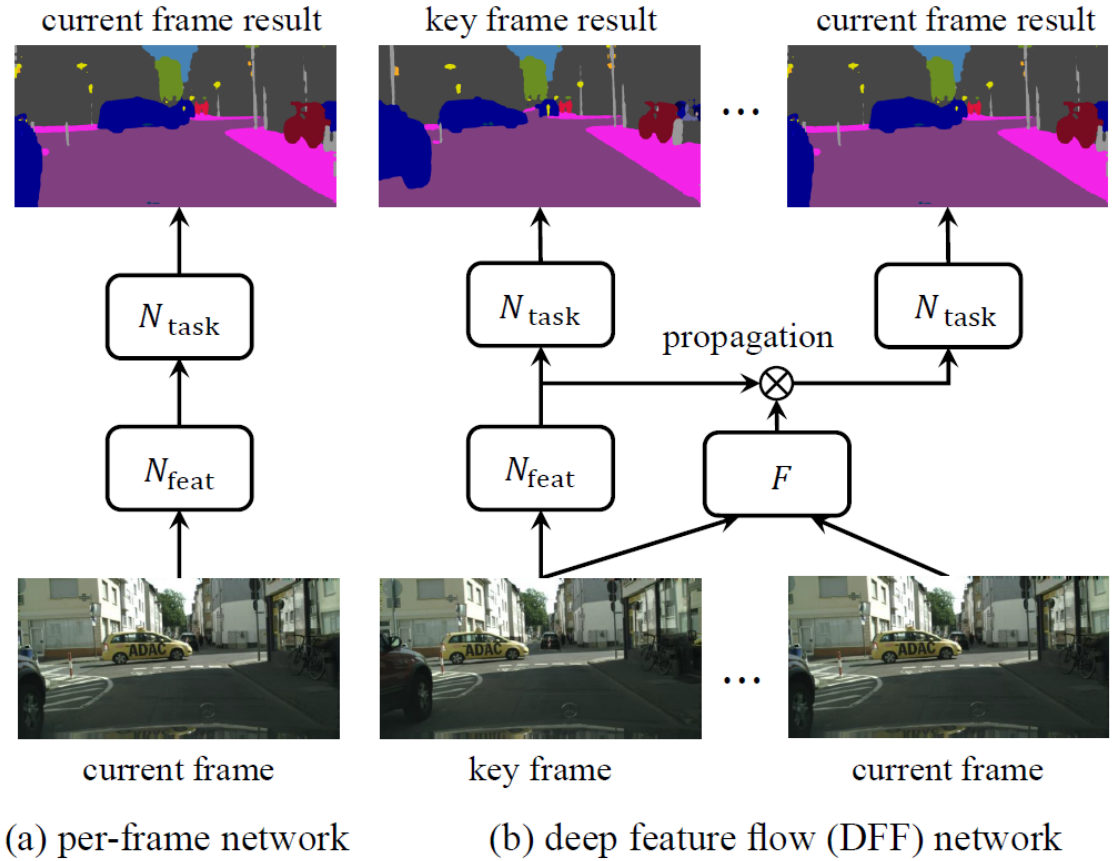
Train Mode	Eval Mode					
	All		GTC		PC	
	Acc	mIoU	Acc	mIoU	Acc	mIoU
SUN-RGBD-13						
All	**85.32**	**62.70**	81.92	55.76	82.59	56.65
GTC	66.91	36.03	**87.12**	**64.57**	**84.08**	**59.19**
SUN-RGBD-37						
All	**80.69**	**44.11**	77.26	37.88	78.05	39.13
GTC	65.24	28.10	**82.36**	**45.36**	**79.26**	**41.10**

Method	Model	Memory	TFLOPs	mIoU
Dilation	DRN-C-42 [43]	3.77	1.07	70.9
	DRN-D-105 [43]	15.15	1.91	75.6
	DeepLabv3 [4]	14.27*	1.97*	79.3
	CCNet [20]	14.33*	1.55*	79.8
QGN (Ours)	Train-All-Eval-All	5.85	0.48	78.2
	Train-GTC-Eval-PC	3.66	0.25	73.0

Deep Feature Flow (DFF)

- Skipping CONV layers by generating CONV features with that of key frame and optical flow between key and current frame

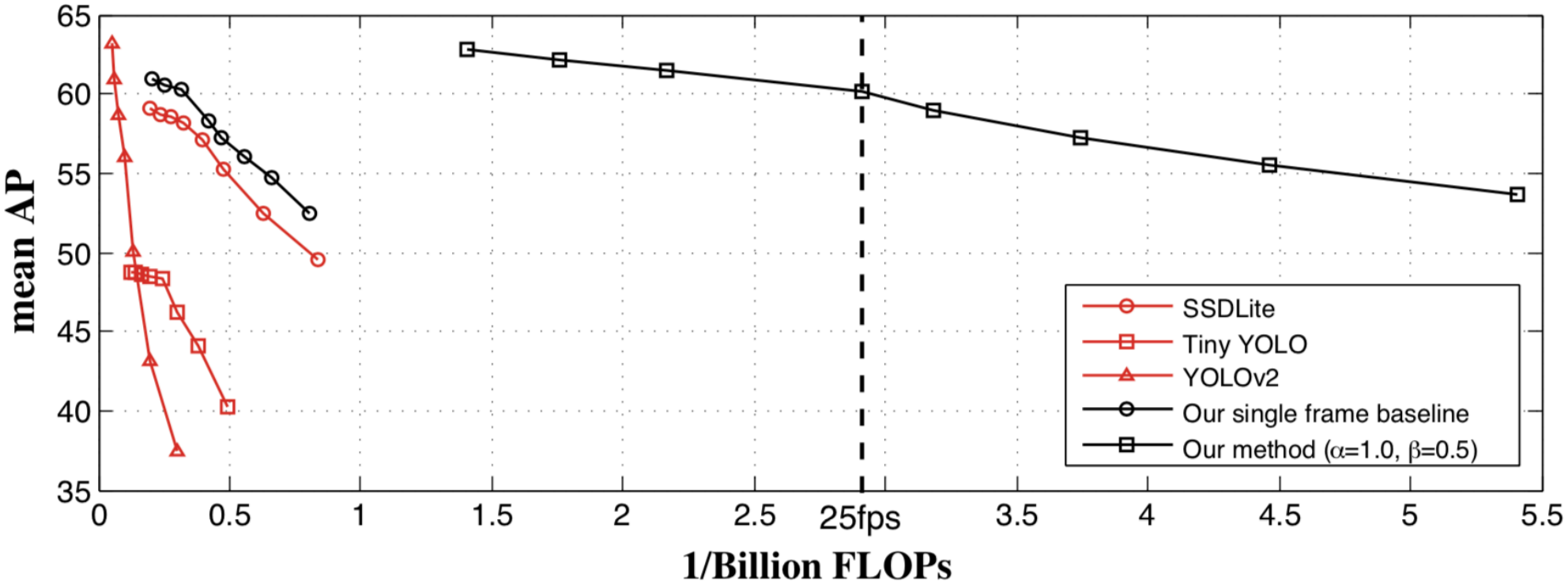




[Zhu et al, 2017]

Comparison with Other Methods

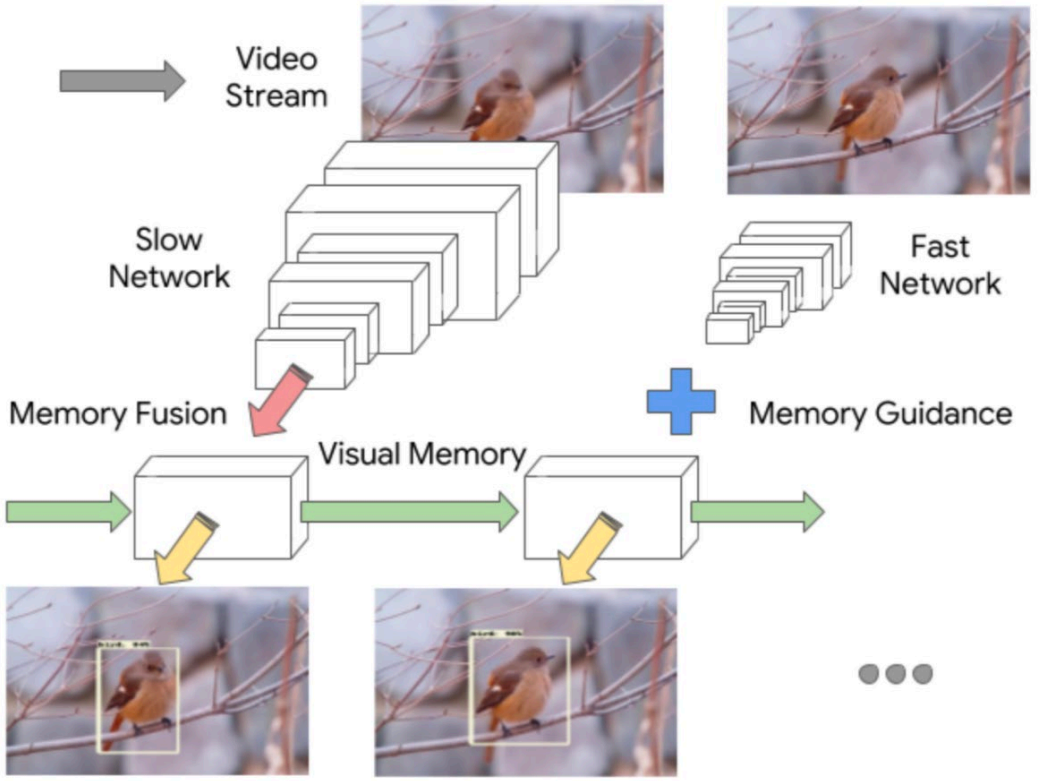
- >10X speedup vs. per-frame object detection methods

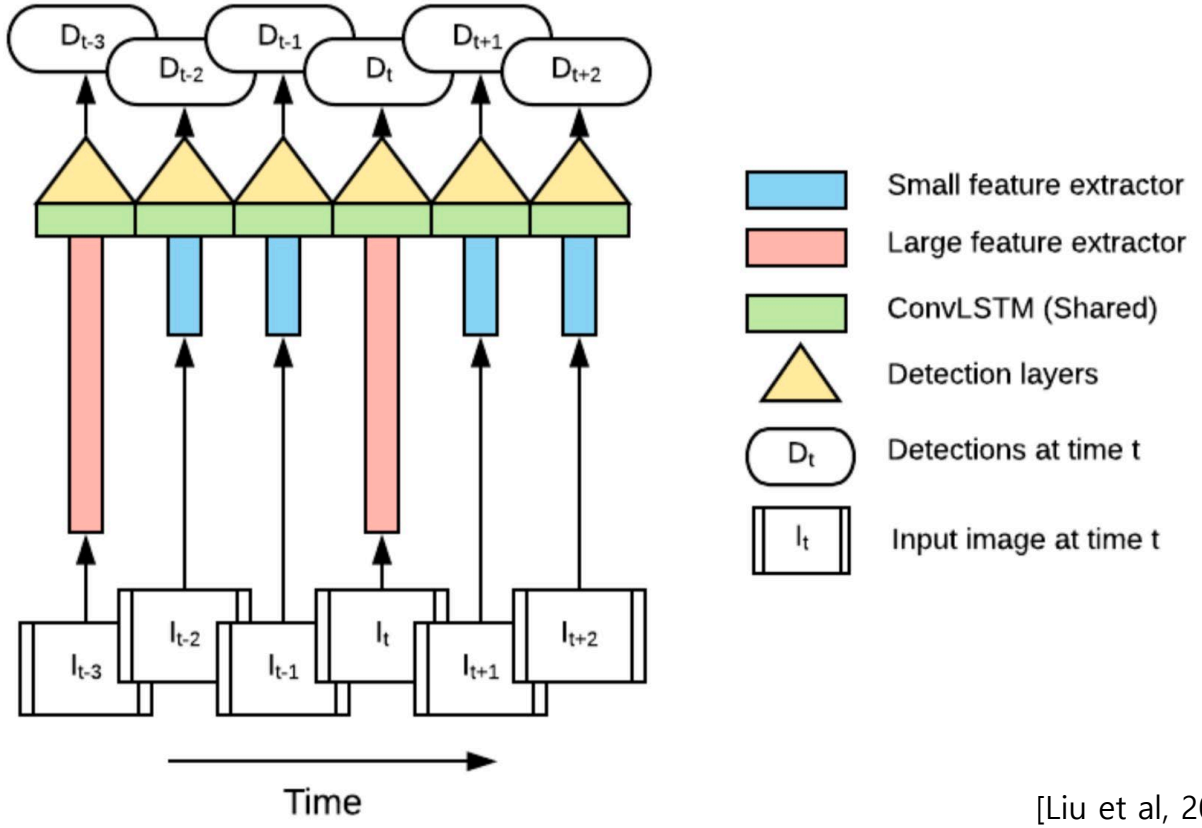


[Zhu et al, 2017]

Looking Fast and Slow

- Exploiting two fast/slow networks with temporal memory
  - MobileNetV2 x1.4 with 320 x 320 + MobileNetV2x0.35 with 160x160
  - Fusing prior knowledge with a convolutional LSTM

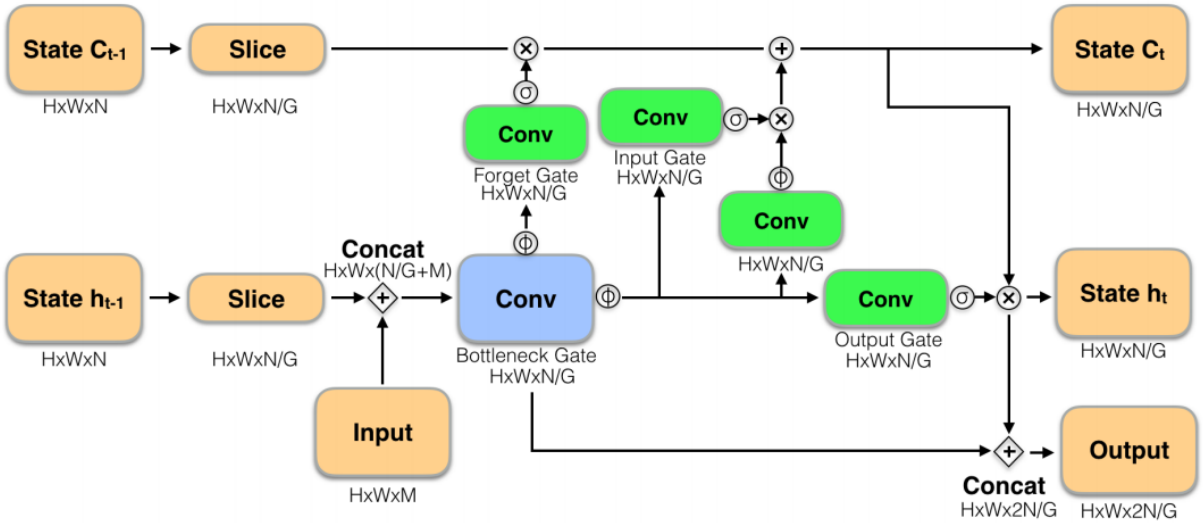




[Liu et al, 2019]

ConvLSTM for Memory Module

- Modify LSTM for 2-D data with light-weighted computation
  - LSTM cells are updated only with the output of the slow network
  - Adopt async feature fusion for consistent latency



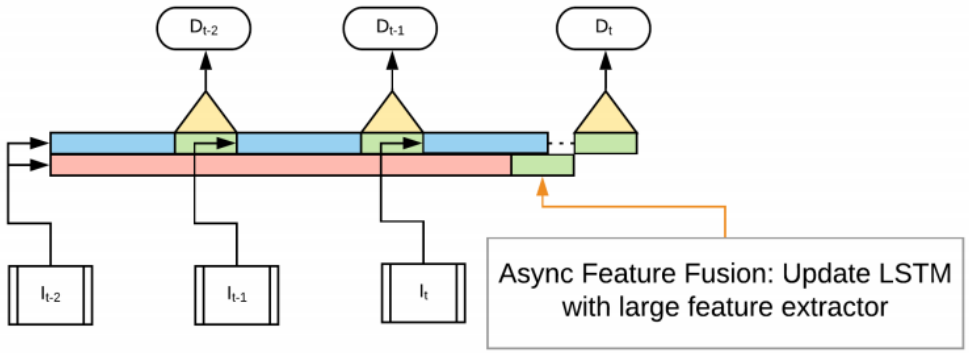


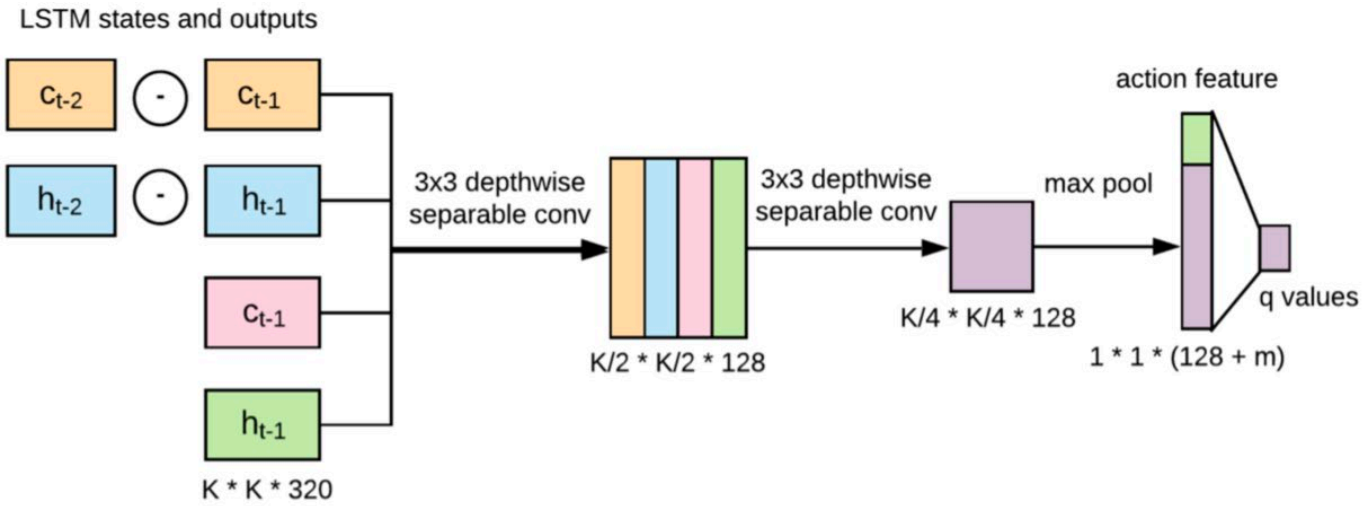
Figure 5: Illustration of asynchronous mode.

[Liu et al, 2019]

RL Agent Selects Key Frames

- Use loss difference as confidence measure
  - Use DDQN for training

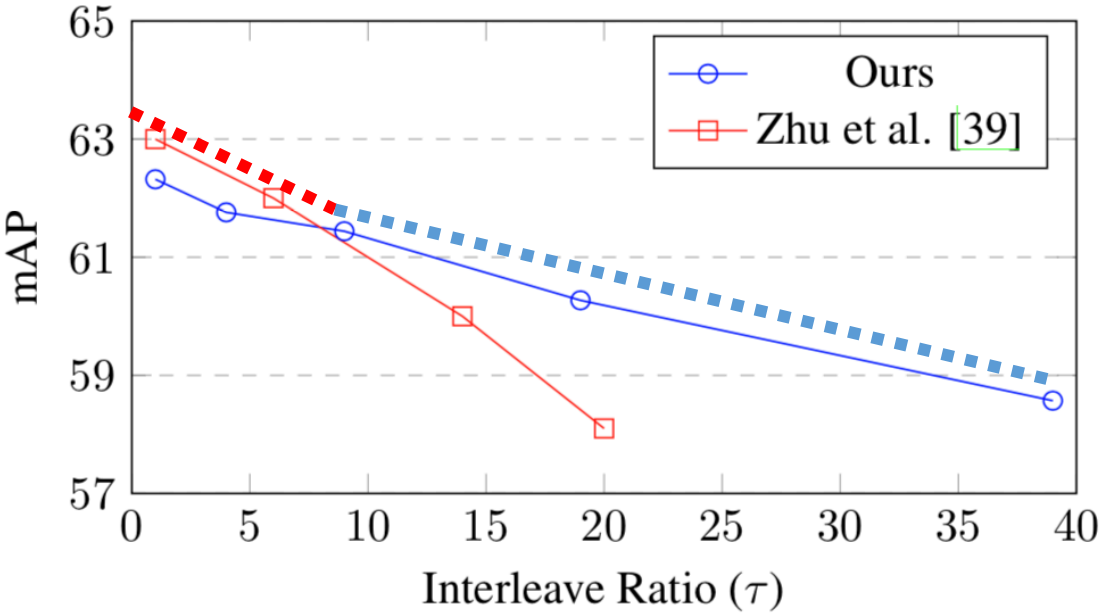
$$R(a) = \begin{cases} \min_i L(D^i) - L(D^0) & a = 0 \\ \gamma + \min_i L(D^i) - L(D^1) & a = 1\,, \end{cases}$$



[Liu et al, 2019]

Results and Discussion

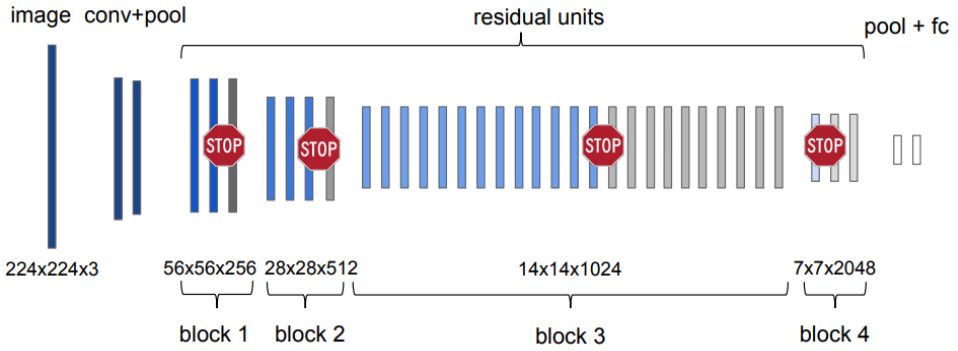
- Better accuracy than Deep Feature Flow [Zhu] in long-term inference
- Probably, an integration of big/little + feature propagation + *online training* can give better results
  - if fast pictures → DFF run (and little model *generation/training*) and
  - if slow pictures → Slow/fast model run (after a brief little model *training*)

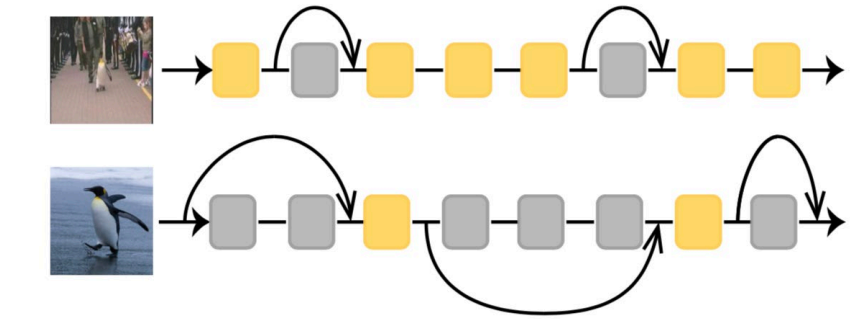


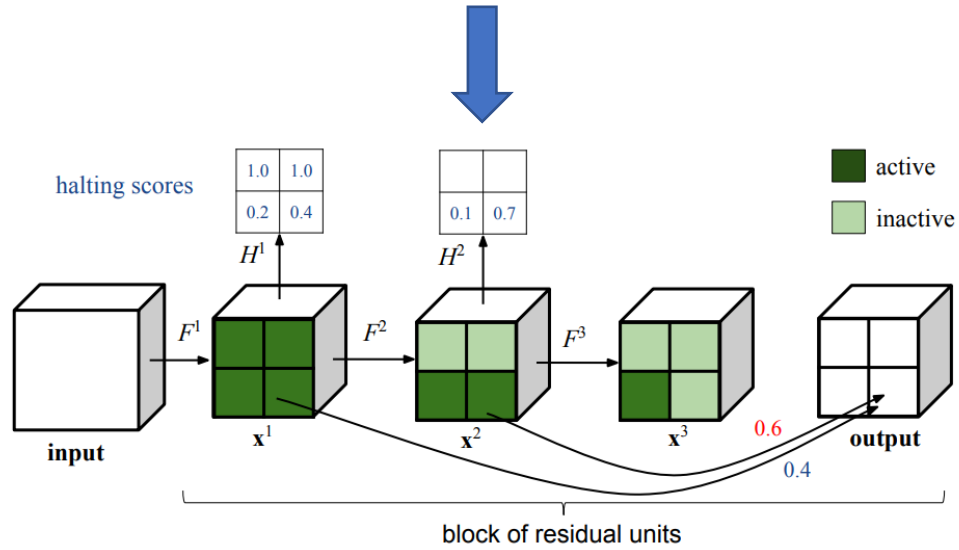
[Liu et al, 2019]
[S.Yoo]

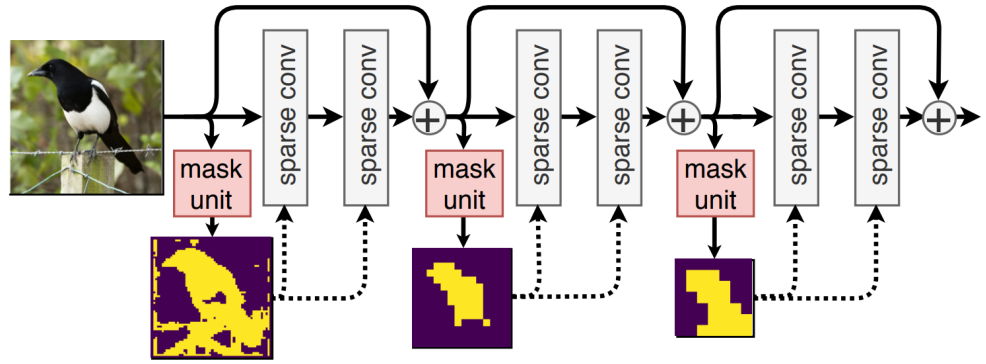
Dynamic Convolution

- How can we find the target region to consider in 2-D areas for each layer?



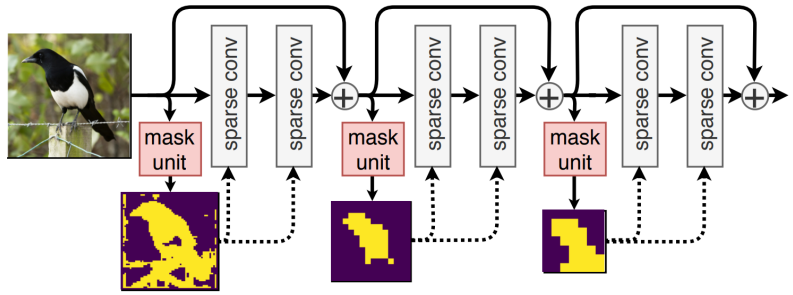






[Verelst et al, 2020]

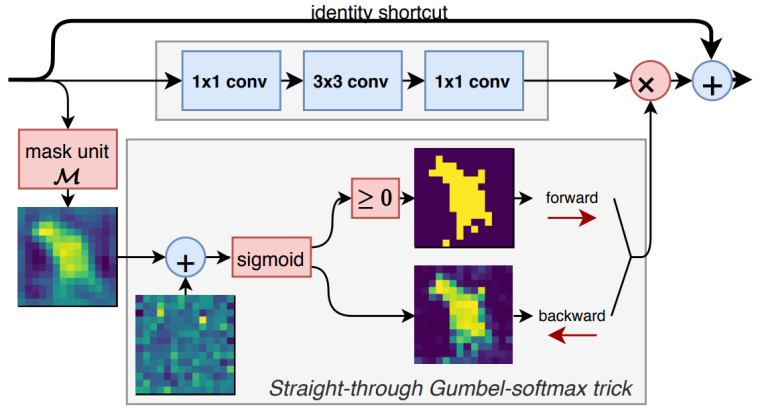
Dynamic Convolution

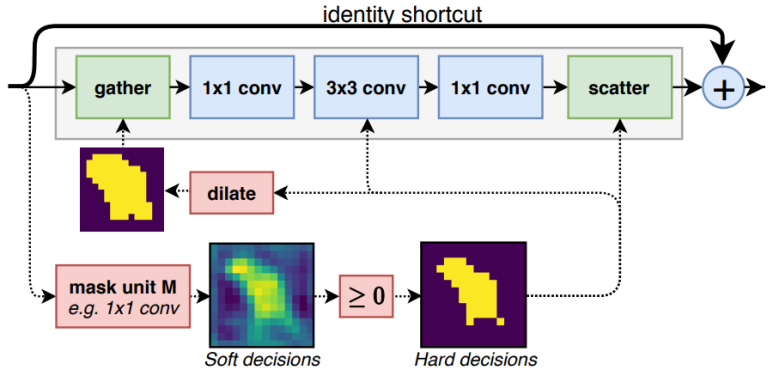


- Pixel-wise mask generation for interesting region

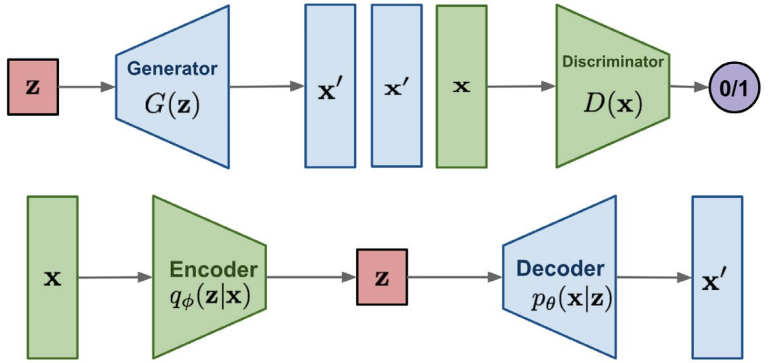
$$X_{b+1} = r(\mathcal{F}(X_b) + X_b) \longrightarrow X_{b+1} = r(\mathcal{F}(X_b) \circ G_b + X_b)$$

- End-to-end training using the Gumbel-Softmax trick
  - Turn the soft decision mask $\mathcal{M}$ into hard decision mask $G_b \in \{0, 1\}$
  - Gather-scatter-based fast computation
  - What is Gumbel-Softmax trick?



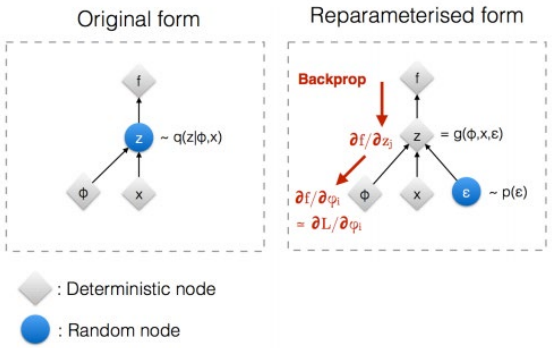


Reparameterization Trick?



- What if we want to use randomness inside a neural network?
  - Variational autoencoder(VAE)
  - Generative adversarial network (GAN)
  - And other networks with conditional probability
    - In dynamic convolution, the gating mask is modeled by discrete random variable notifying the probability of bypass/computation

- Reparameterization trick for contiguous random variable
  - Representative example: gaussian distribution modeled by mean and variance
    - $z \sim N(\mu, \sigma)$
      - Z is not differentiable…
    - $z' = \mu + \epsilon * \sigma, \quad \epsilon \sim N(0,1)$
      - Now we can differentiate μ & σ



Gumbel-Softmax Trick

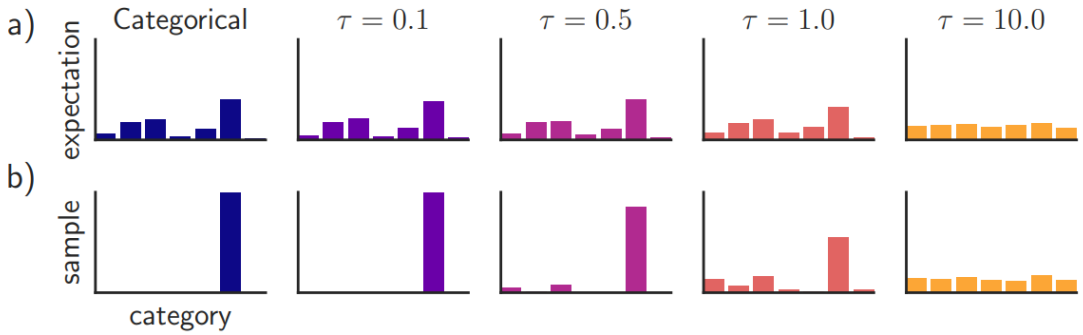
- Learning latent variable for discrete random variable with reparameterization trick
  - Random variable z is a categorical random variable modeled by probability distribution $\pi = (\pi_1, \dots, \pi_n)$
  - $z = \begin{cases} e_1 \; with \; probability \; \pi_1 \\ \cdots \\ e_k \; with \; probability \; \pi_k \end{cases}$, where $e_i = (0,0, \dots, 1, \dots, 0) = f_{onehot}(i)$

    $i - th \; index$
  - Gumbel-max trick(or sampling)
    - $\hat{z} = f_{onehot}\left(\arg\max_i (g_i + \log \pi_i)\right), g_i = -\log(-\log u_i), u_i \sim Uniform(0,1)$
    - However, argmax is not differentiable…
      - Softmax approximation or relaxation

Approximation of Gumbel-max Trick

- Use softmax instead of argmax
  - $\hat{z} = f_{onehot}\left(\arg\max_i(g_i + \log \pi_i)\right)$

    $y_i = softmax\left[\frac{g+log\,\pi}{\tau}\right]_i = \frac{exp\left(\frac{g_i+log(\pi_i)}{\tau}\right)}{\Sigma_j\, exp\left(\frac{g_j+log(\pi_j)}{\tau}\right)}$



  - Softmax approximates argmax function in low temperature, while approximates uniform distribution in high temperature
    - Exploration with high temperature in early stage
    - Progressively reduce temperature to converge to the argmax operation

Gumbel-Softmax Trick in Binary Decision

- Let assume the probability of the desired action $\pi_1 = \sigma(m)$,
  then that of the opposite action is $\pi_2 = 1 - \sigma(m)$ (in this case, skip computation)
- Then, $y_1 = \frac{\exp\left(\frac{\log \pi_1 + g_1}{\tau}\right)}{\exp\left(\frac{\log \pi_1 + g_1}{\tau}\right) + \exp\left(\frac{\log \pi_2 + g_2}{\tau}\right)} = \frac{(\exp(\log \sigma(m)) \exp(g_1))^{\frac{1}{\tau}}}{(\exp(\log \sigma(m)) \exp(g_1))^{\frac{1}{\tau}} + \left(\exp\left(\log\left(1-\sigma(m)\right)\right) \exp(g_2)\right)^{\frac{1}{\tau}}}$

$$= \frac{(\sigma(m)e^{g_1})^{\frac{1}{\tau}}}{(\sigma(m)e^{g_1})^{\frac{1}{\tau}} + \left(\left(1-\sigma(m)\right)e^{g_2}\right)^{\frac{1}{\tau}}} = \frac{1}{1 + \left(\frac{1-\sigma(m)}{\sigma(m)} e^{g_2 - g_1}\right)^{\frac{1}{\tau}}},$$

- $\sigma(m) = \frac{1}{1+e^{-m}}$, then $y_1 = \frac{1}{(1+e^{-m}e^{g_2-g_1})^{\frac{1}{\tau}}} = \frac{1}{1+e^{\frac{-m+g_2-g_1}{\tau}}} = \boldsymbol{\sigma\left(\frac{m+g_1-g_2}{\tau}\right)}$
- In binary decision, gumbel-softmax trick is converged to sampling of sigmoid variable

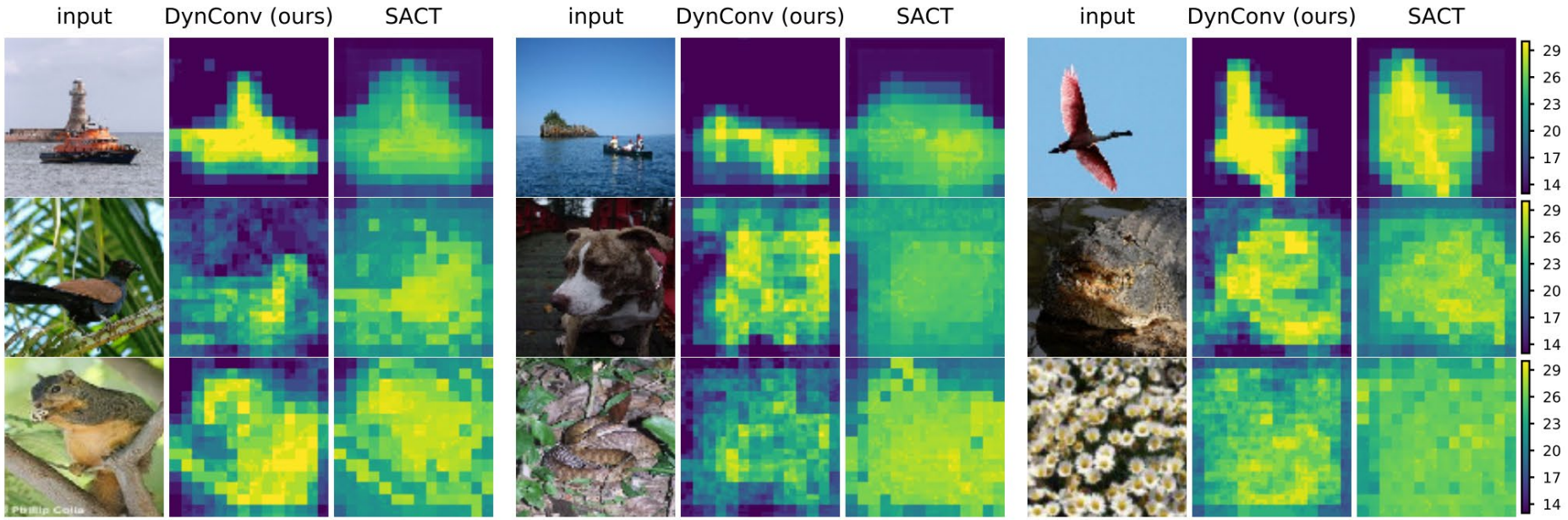
Gumbel-Softmax Trick in Binary Decision -2

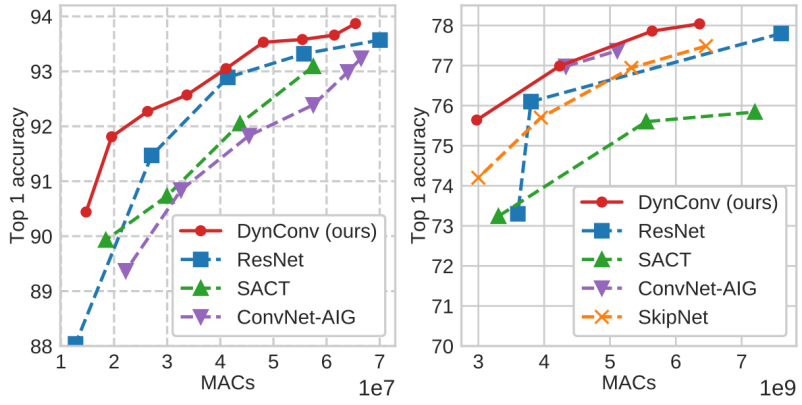
- Let assume the probability of the desired action $\pi_1 = \sigma(m)$,
  then that of the opposite action is $\pi_2 = 1 - \sigma(m)$ (in this case, skip computation)
- $y_1 = \frac{1}{(1+e^{-m}e^{g_2-g_1})^{\frac{1}{\tau}}} = \frac{1}{1+e^{\frac{-m+g_2-g_1}{\tau}}} = \boldsymbol{\sigma\left(\frac{m+g_1-g_2}{\tau}\right)}$
- In binary decision, gumbel-softmax trick is converged to sampling of sigmoid variable

- In this paper, they use a straight-through estimator instead of sampling

$$z = \begin{cases} y_1 > 0.5 \equiv \frac{m+g_1-g_2}{\tau} > 0 & \text{(forward)}, \\ y_1 & \text{(backward)}. \end{cases}$$

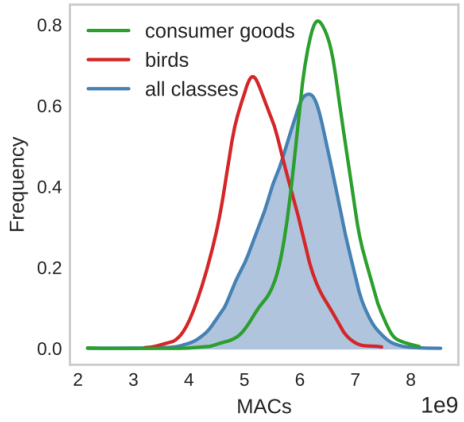
Results



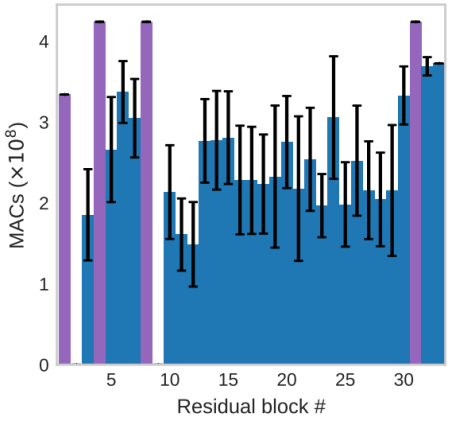


(a) CIFAR-10

(b) ImageNet-1K



(a) MACs per image



(b) MACs per layer