Differentiable NAS

Shusen Wang

http://wangshusen.github.io/

Reference

1. Liu, Simonyan, & Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019.

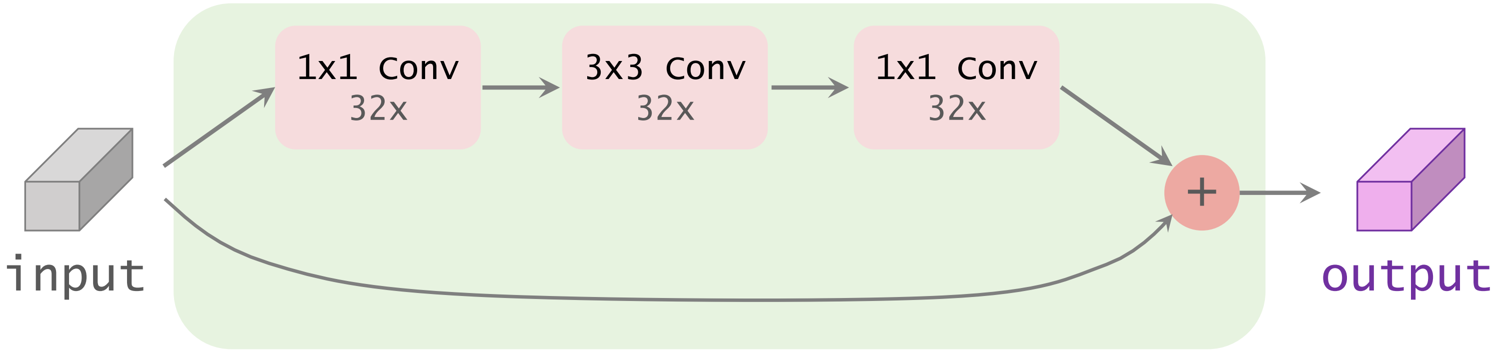
2. Wu et al. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*, 2019.

Basic Idea

- User manually defines some (e.g., 9) candidate blocks.

Basic Idea

- User manually defines some (e.g., 9) candidate blocks.



Example: a candidate block

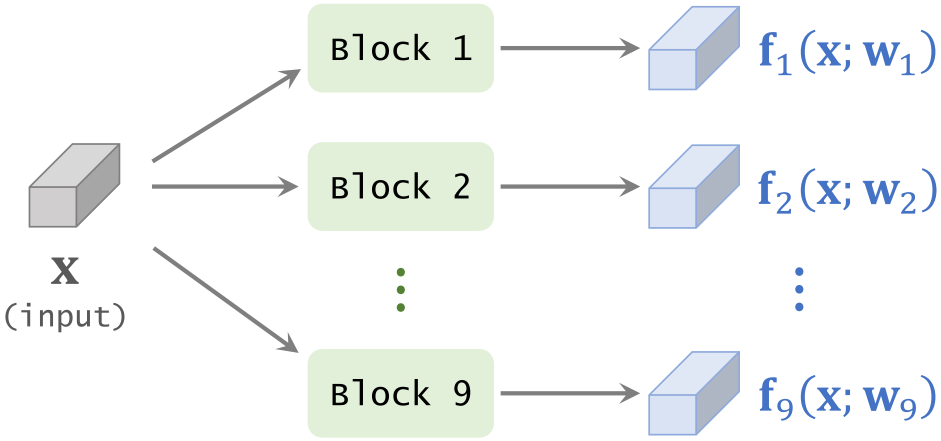
Basic Idea

- User manually defines some (e.g., 9) candidate blocks.
- User specifies the number of layers, e.g., 20 layers. 把一个模块看作一层，
- Each layer can be one of the 9 candidate blocks.
- Size of search space (i.e., # of possible architectures) is 9^{20}.

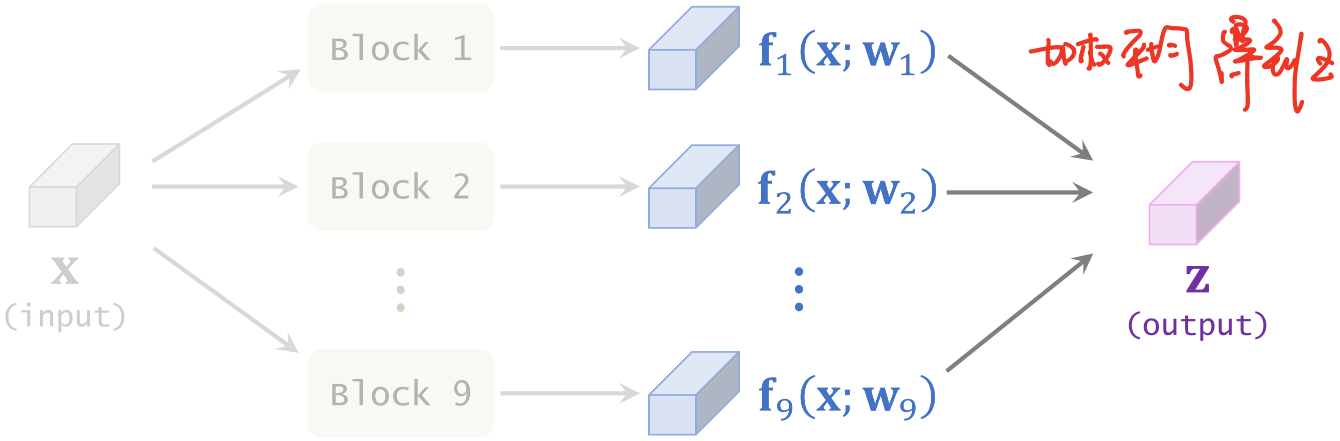
Super-net

One Layer of Super-net

每一层由9个模块 => 得到输出结果

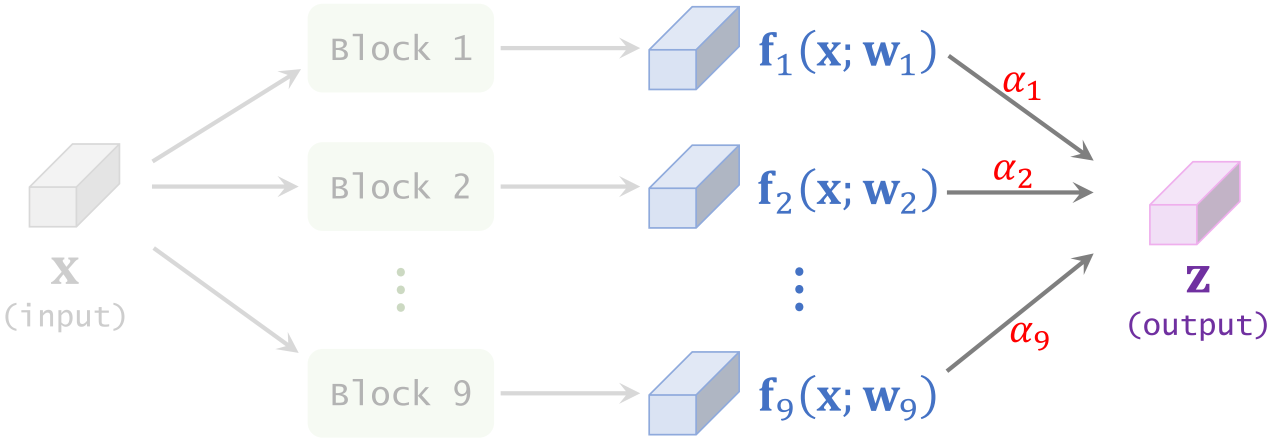


One Layer of Super-net

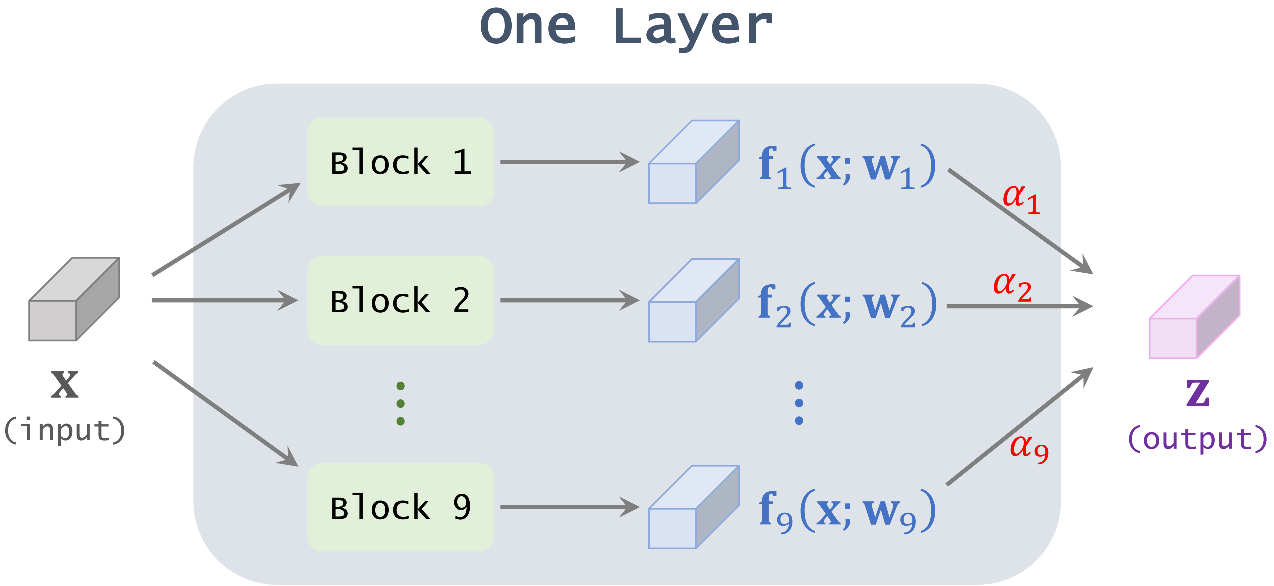


One Layer of Super-net

- $[\alpha_1, \cdots, \alpha_9] = \text{Softmax}(\theta_1, \cdots, \theta_9)$.
- Output: $\mathbf{z} = \sum_{j=1}^{9} \alpha_j \cdot \mathbf{f}_j(\mathbf{x}; \mathbf{w}_j)$.

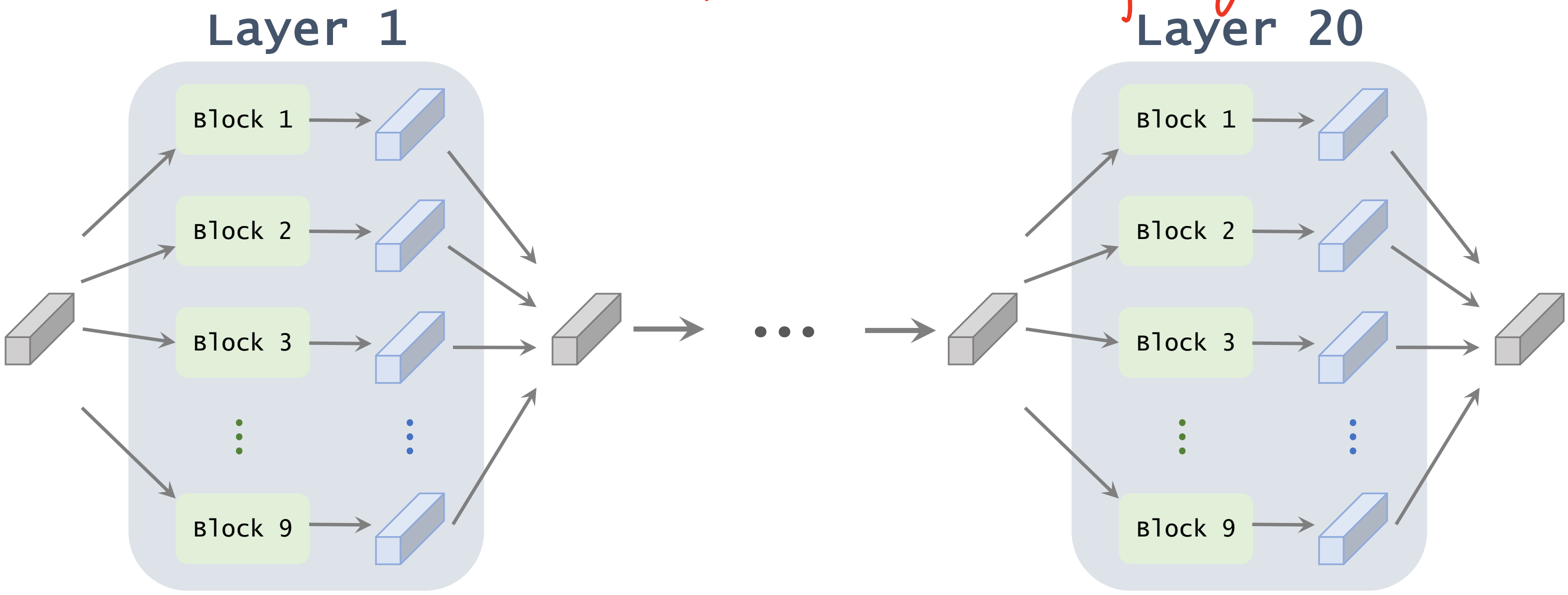


One Layer of Super-net



The super-net has 20 layers; each layer contains 9 parallel blocks.

每一层结构一样，不同层之参数不同　　部分层有pooling，手工调



Trainable Parameters of Super-net

- Each layer has the following trainable parameters:
  - $\mathbf{w}_1, \cdots, \mathbf{w}_9$ (tensors): parameters of the 9 blocks.
  - $\theta_1, \cdots, \theta_9$ (scalars): parameters that determine the weights, $\alpha_1, \cdots, \alpha_9$.

- Layers do not share parameters.
  - Each layer has its own parameters, $\mathbf{w}_1, \cdots, \mathbf{w}_9$ and $\theta_1, \cdots, \theta_9$.
  - Parameters are not shared across layers.

Trainable Parameters of Super-net

- Blocks: $j = 1, \cdots, 9$.
- Layers: $l = 1, \cdots, 20$.
- Trainable parameters of the l-th layer and j-th block:
  - $\mathbf{w}_j^{(l)}$ (tensors) and $\theta_j^{(l)}$ (a scalar).
- All the trainable parameters of the super-net:
  - $\mathcal{W} = \left\{ \mathbf{w}_j^{(l)} \right\}_{j,l}$ and $\Theta = \left\{ \theta_j^{(l)} \right\}_{j,l}$.

Train the Super-net

- $\mathbf{x}_1, \cdots, \mathbf{x}_n$: training images.
- $\mathbf{y}_1, \cdots, \mathbf{y}_n$: targets (aka labels).
- $\mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)$: a prediction made by the 20-layer super-net.

Train the Super-net

- $\mathbf{x}_1, \cdots, \mathbf{x}_n$: training images.
- $\mathbf{y}_1, \cdots, \mathbf{y}_n$: targets (aka labels).
- $\mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)$: a prediction made by the 20-layer super-net.
- Learn $\mathcal{W}$ and Θ from the training set by minimizing the cross-entropy loss:

$$\min_{\mathcal{W}, \Theta} \ \frac{1}{n} \sum_{i=1}^{n} \text{Loss}\Big(\mathbf{y}_i, \ \mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)\Big).$$

Train the Super-net

- $\mathbf{x}_1, \cdots, \mathbf{x}_n$: training images.
- $\mathbf{y}_1, \cdots, \mathbf{y}_n$: targets (aka labels).
- $\mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)$: a prediction made by the 20-layer super-net.
- Learn $\mathcal{W}$ and Θ from the training set by minimizing the cross-entropy loss:

$$\min_{\mathcal{W}, \Theta} \ \frac{1}{n} \sum_{i=1}^{n} \text{Loss}\big(\mathbf{y}_i, \ \mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)\big).$$

用反向传播求损失函数关于θ的梯度 再用梯度估计θ

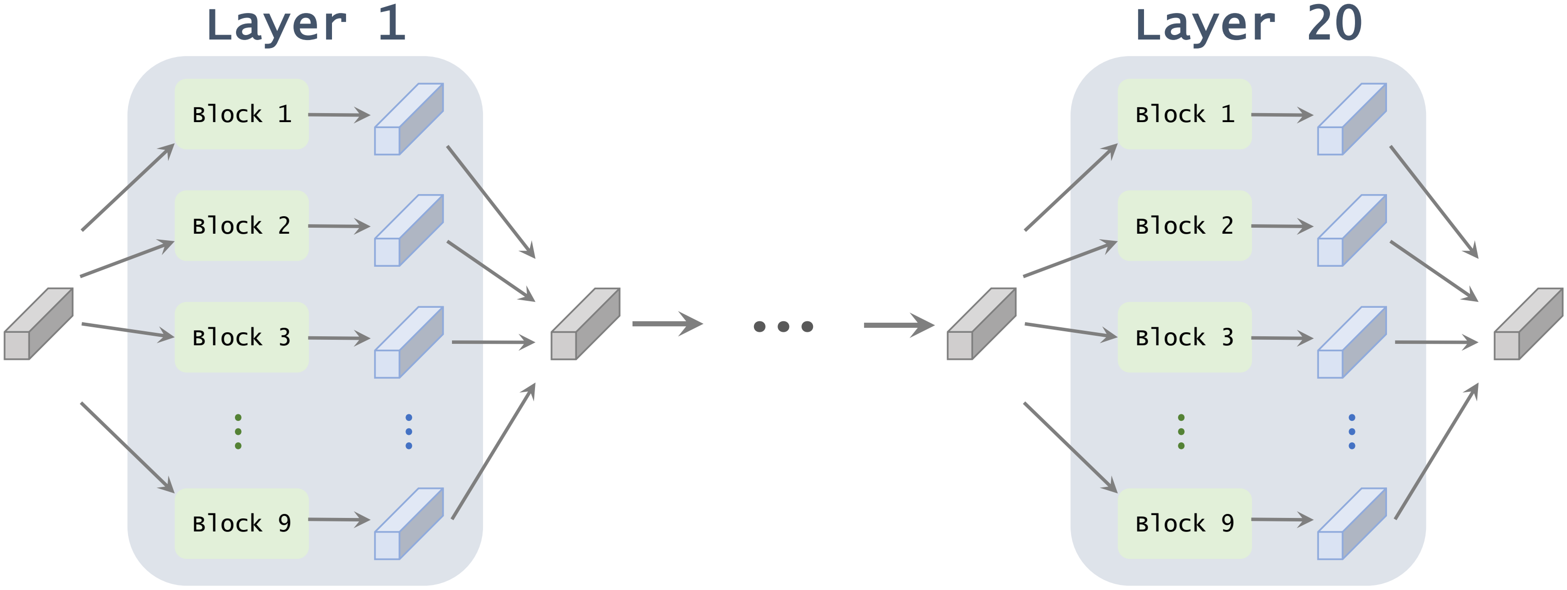
$$\Theta \longrightarrow \mathbf{p} \longrightarrow \text{Loss}$$

θ

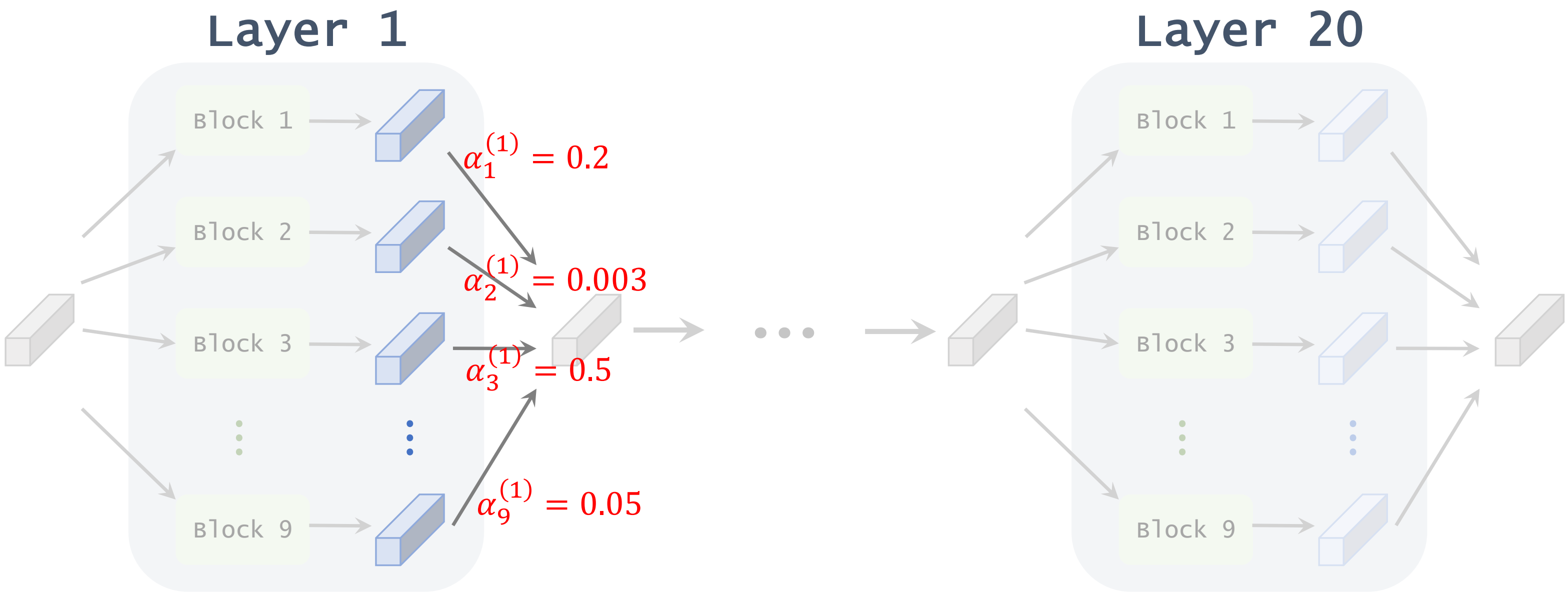
p是θ的函数 可微

softmax变换

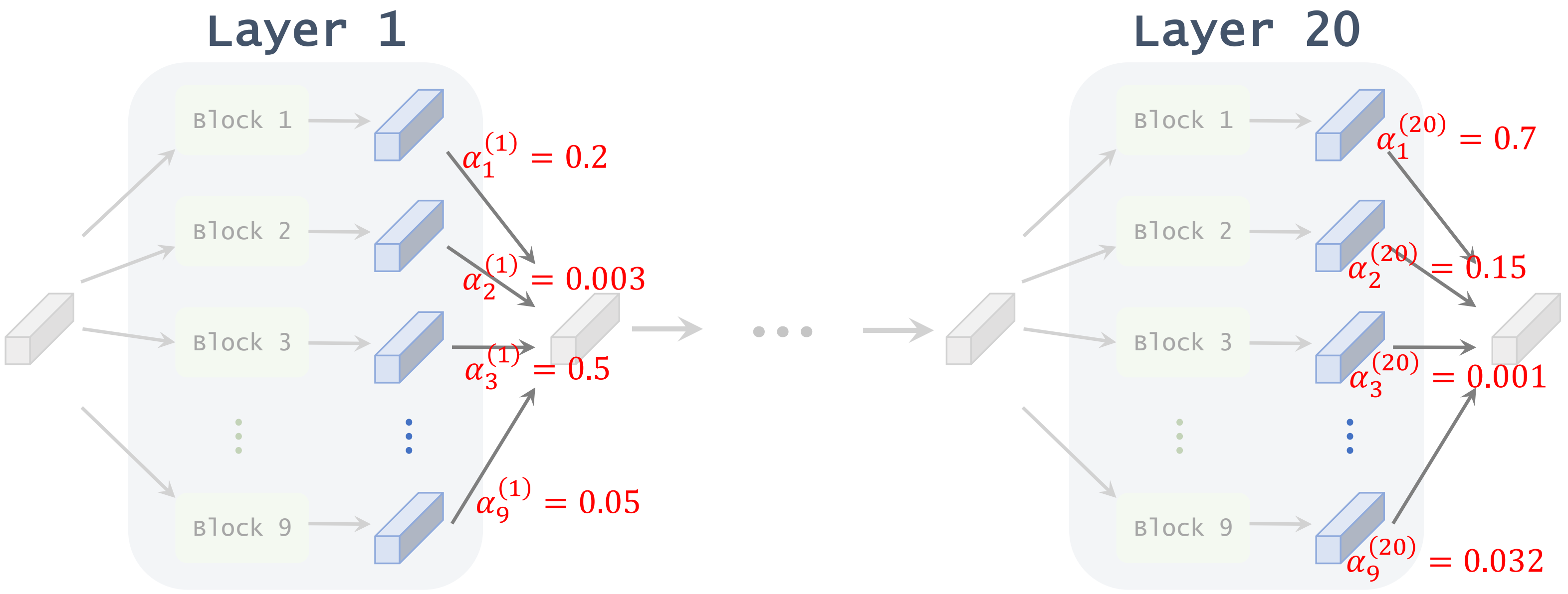
Knowing the optimal Θ, we have the weights $\alpha_j^{(l)} = \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}$.



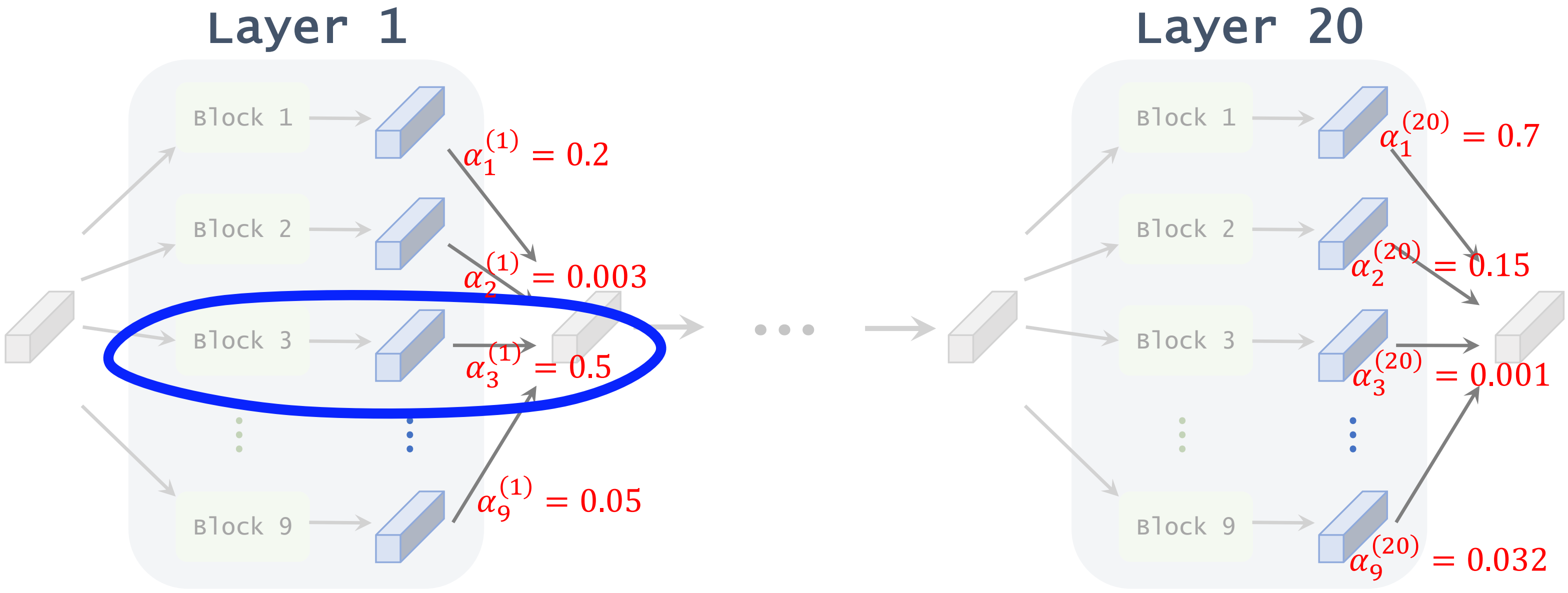
Knowing the optimal Θ, we have the weights $\alpha_j^{(l)} = \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9}\exp\left(\theta_k^{(l)}\right)}$.



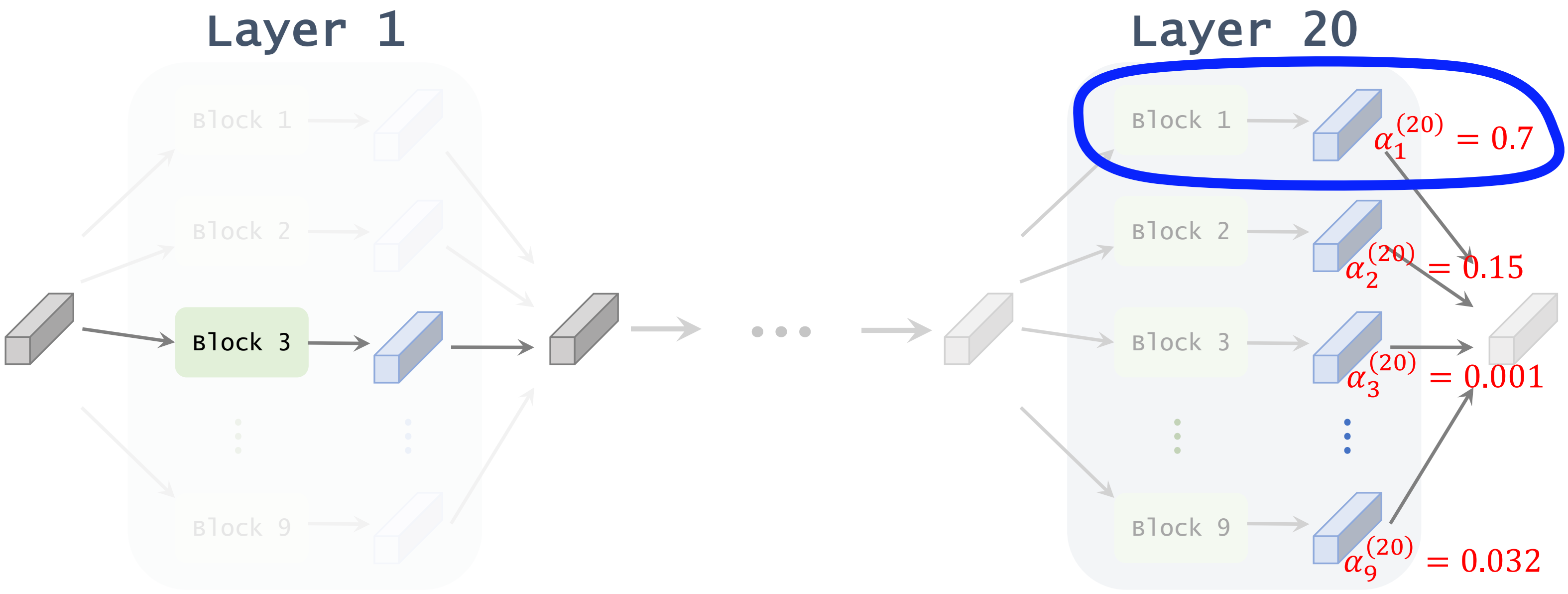
Knowing the optimal Θ, we have the weights $\alpha_j^{(l)} = \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}$.



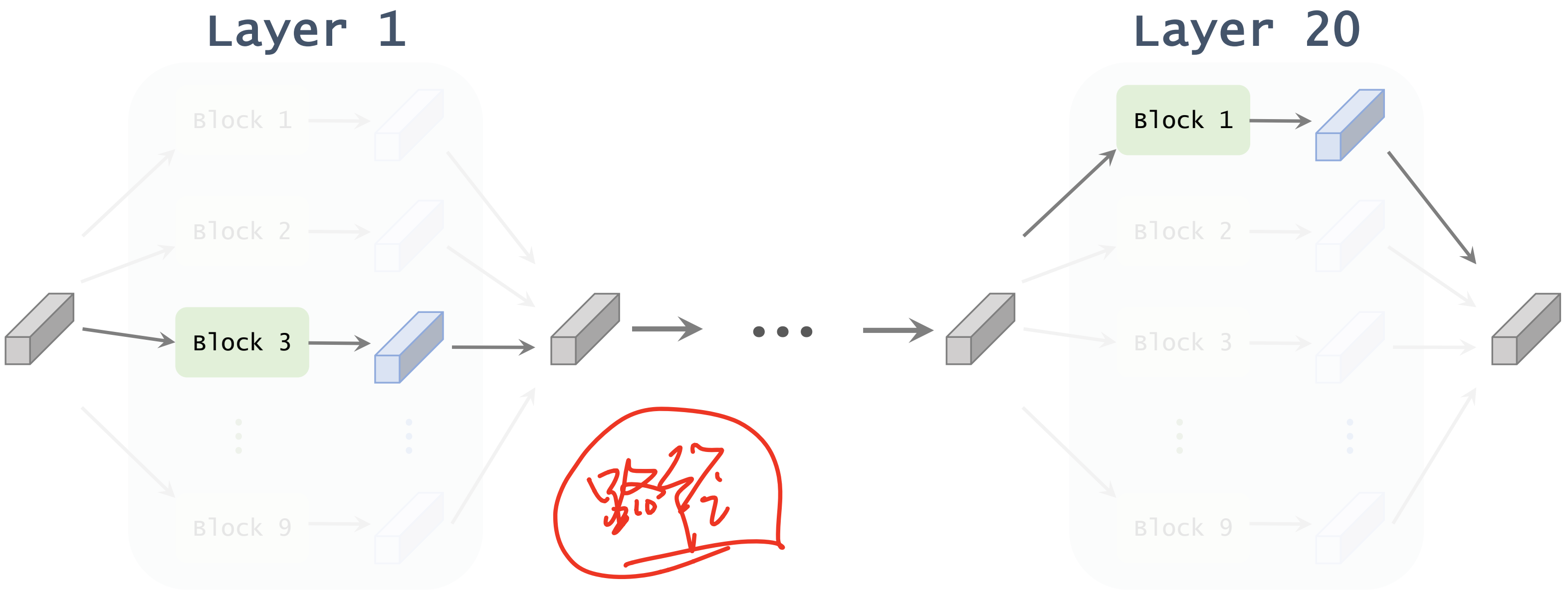
For each layer, select the block that has the biggest weight, α.



For each layer, select the block that has the biggest weight, α.



For each layer, select the block that has the biggest weight, α.



Computational Efficient Design

权衡 预测量与计算准确率

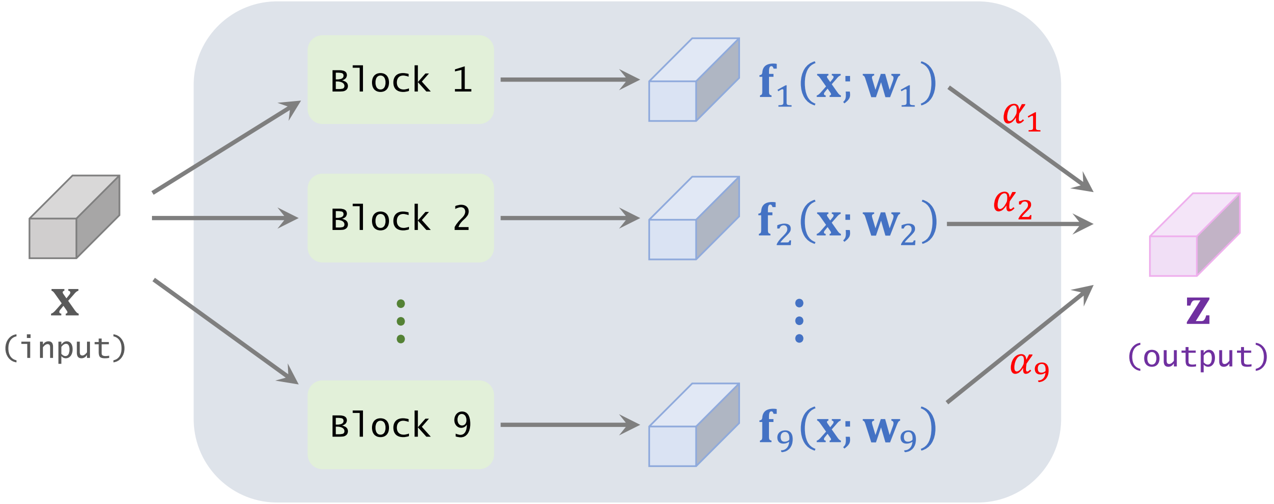
Reference:

1. Wu et al. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Basic Idea

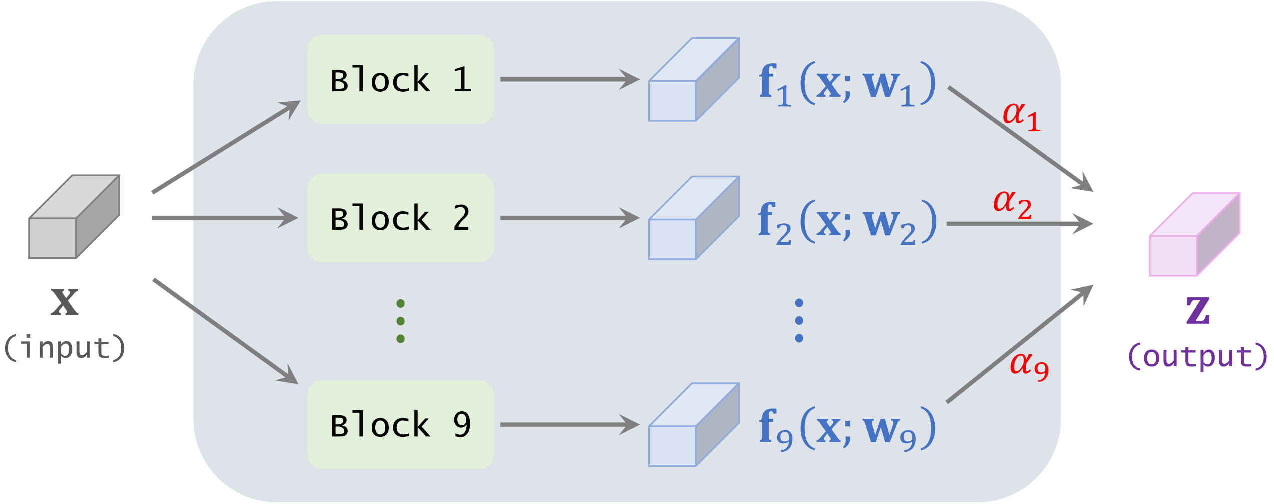
- A trained CNN takes as input an image and makes a prediction.
- Small latency (i.e., time cost of prediction making) is preferable.
- Latency can be considered during architecture search.
    - Different candidate blocks cause different accuracies and different latencies.
    - Trade off accuracy and latency.

One Layer of the Super-net



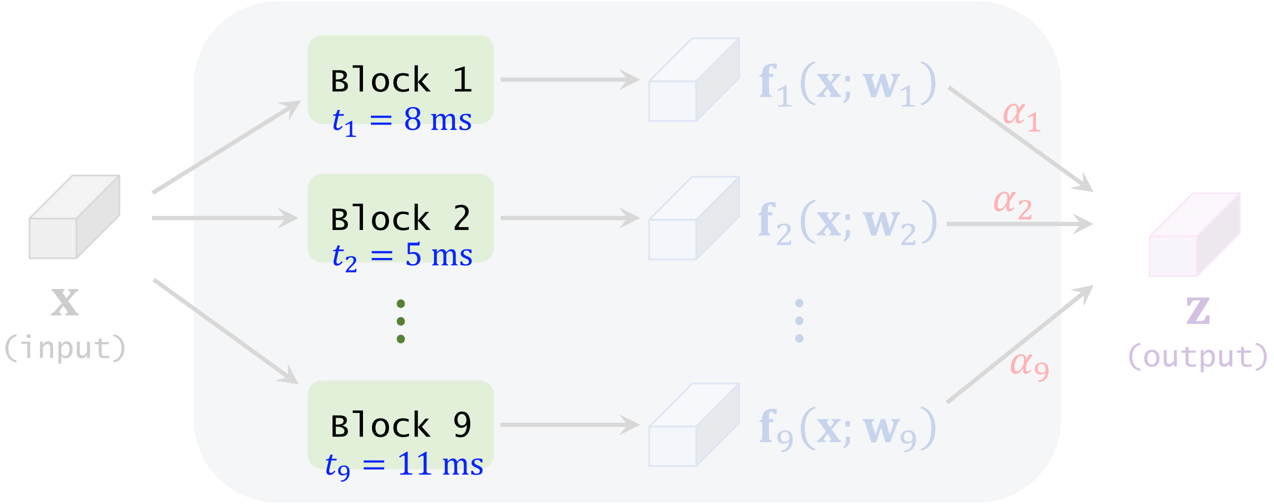
Latency

- Suppose the selected CNN will be deployed to iPhone 12.
- On iPhone 12, measure the latency caused by each block.



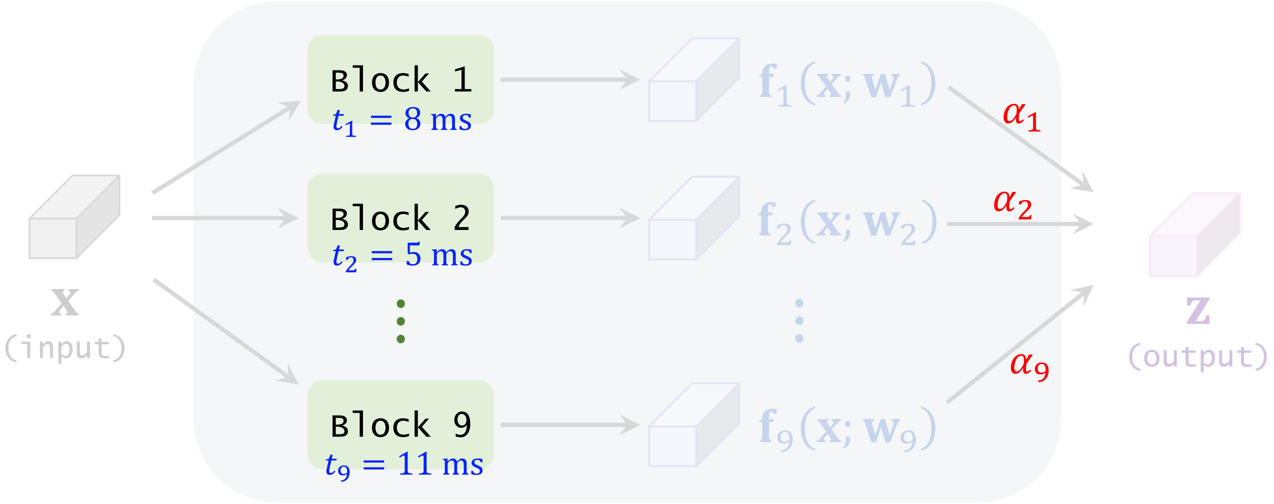
Latency

- Suppose the selected CNN will be deployed to iPhone 12.
- On iPhone 12, measure the latency caused by each block.



Latency

Weighted average of latencies: $\sum_{j=1}^{9} t_j \cdot \alpha_j$.



Latency

- For layers $l = 1, \cdots, 20$ and blocks $j = 1, \cdots, 9$:
  - Denote the measured latency (ms) by $t_j^{(l)}$.
  - Denote the weights by $\alpha_j^{(l)} = \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}$.

$\rightarrow \exp\left(\theta_j^{(l)}\right)$

Latency

- For layers $l = 1, \cdots, 20$ and blocks $j = 1, \cdots, 9$:
  - Denote the measured latency (ms) by $t_j^{(l)}$.
  - Denote the weights by $\alpha_j^{(l)} = \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}$.
- Define: $\underline{\mathrm{Lat}(\Theta)} = \underline{\sum_{l=1}^{20}} \underline{\sum_{j=1}^{9}} \underline{t_j^{(l)}} \cdot \boxed{\alpha_j^{(l)}}$.

$$= \sum_{l=1}^{20} \sum_{j=1}^{9} \underline{t_j^{(l)}} \cdot \boxed{\frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}}.$$

Latency

Latency caused by the 20 layers:

$$\underline{\mathrm{Lat}(\Theta)} = \sum_{l=1}^{20} \sum_{j=1}^{9} t_j^{(l)} \cdot \frac{\exp\left(\theta_j^{(l)}\right)}{\sum_{k=1}^{9} \exp\left(\theta_k^{(l)}\right)}.$$

- Encourage $\mathrm{Lat}(\Theta)$ to be small.
- Apply (add or multiply) $\mathrm{Lat}(\Theta)$ to the loss function.
- $\mathrm{Lat}(\Theta)$ is a differential function of the parameters, Θ.

Trade off accuracy and latency

- Additive:

$$\min_{\mathcal{W}, \Theta} \; \frac{1}{n} \sum_{i=1}^{n} \underline{\mathrm{Loss}\big(\mathbf{y}_i, \; \mathbf{p}(\mathbf{x}_i; \mathcal{W}, \Theta)\big)} \; + \; \lambda \cdot \underline{\mathrm{Lat}(\Theta)}.$$

Trade off accuracy and latency

- Additive:

$$\min_{\mathcal{W},\Theta} \ \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}\big(\mathbf{y}_i,\ \mathbf{p}(\mathbf{x}_i;\mathcal{W},\Theta)\big) \ + \ \lambda \cdot \mathrm{Lat}(\Theta).$$

- Multiplicative [1]:

$$\min_{\mathcal{W},\Theta} \ \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}\big(\mathbf{y}_i,\ \mathbf{p}(\mathbf{x}_i;\mathcal{W},\Theta)\big) \ \cdot \log^{\lambda}[\mathrm{Lat}(\Theta)].$$

Reference:

1. Wu et al. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Hardware Awareness

- Some candidate blocks are suitable for GPU, while some are suitable for mobile devices.
  - For example, a candidate block is too small to fit in GPU.
  - But it can make full use of the A14 processor on iPhone 12.
- The optimal architectures for GPU and iPhone 12 are different.
  - For a GPU, the optimal architecture contains big Conv layers (good for accuracy, bad for latency.)
  - For iPhone 12, the optimal architecture contains DepthWise Conv layers (bad for accuracy, good for latency.)

Summary

Differentiable Architecture Search

- DARTS [1] automatically search neural architectures.
- This lecture explains DARTS using the example of [2].
- The objective function is a differentiable function of the parameters, $\Theta = \left\{\theta_j^{(l)}\right\}$, that determine network architecture.

Reference:

1. Liu, Simonyan, & Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019.
2. Wu et al. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Candidate Blocks & Super-net

- User manually prepare some (e.g., 9) candidate blocks.
- User manually specify the number of layers (e.g., 20.)
- Build a **super-net**: 20 layers; each layer contains the 9 parallel blocks.
- The output of each layer is the weighted sum of the 9 blocks; the weights are $\left\{\alpha_j^{(l)}\right\}$.

Candidate Blocks & Super-net

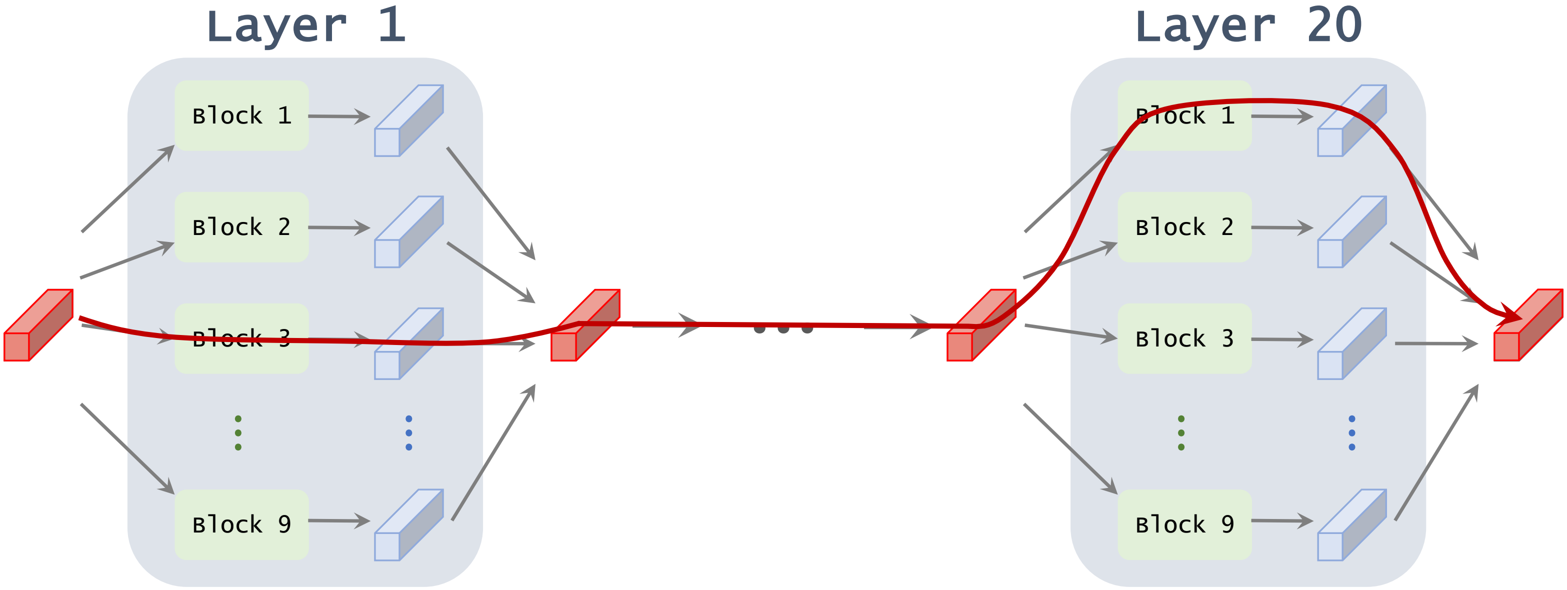
- Train the super-net (on the training set) to find the weights, $\alpha_j^{(l)}$ (for blocks $j = 1, \ldots, 9$ and layers $l = 1, \ldots, 20$).

- For the l-th layer, select the one among the 9 candidate blocks that has the biggest weight:

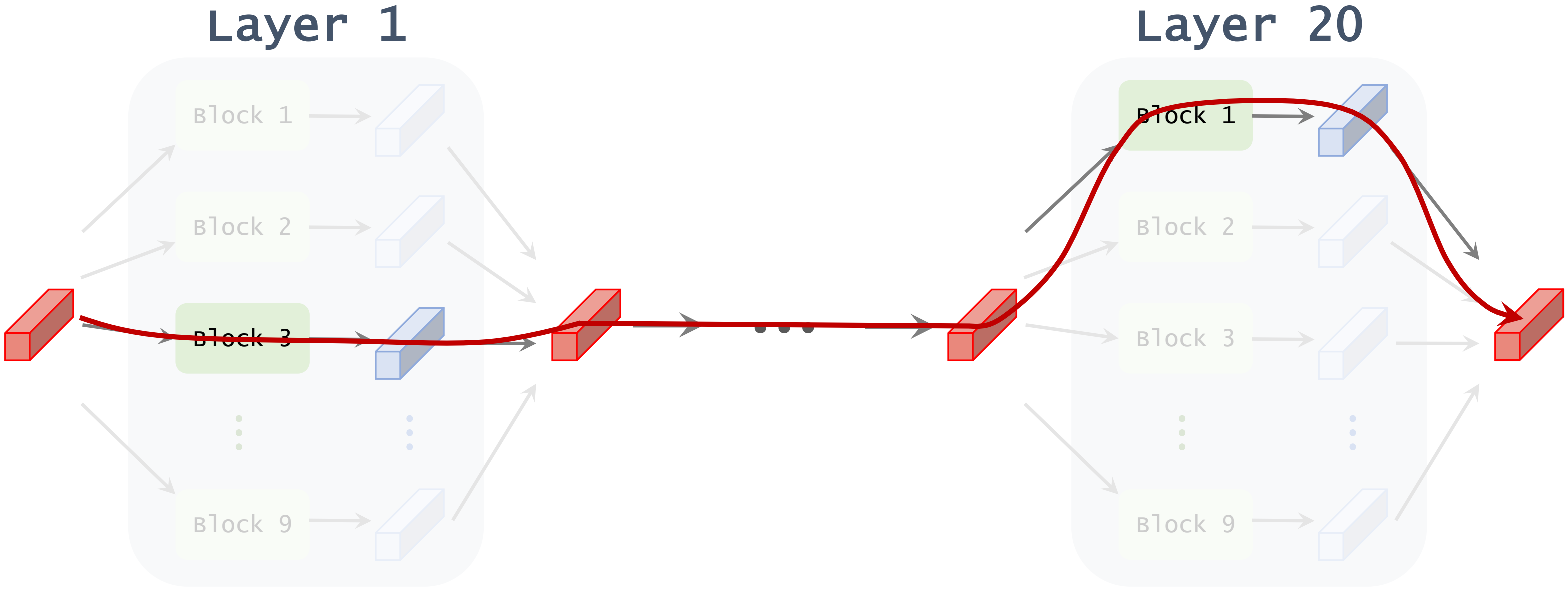
$$\operatorname*{argmax}_{j \in \{1, \cdots, 9\}} \alpha_j^{(l)} .$$

- The selected architecture has 20 layers, and each layer is one of the 9 candidate blocks.

Graph Perspective



Graph Perspective



Take efficiency into account

- Measure the latency (i.e., runtime of prediction making) caused by each of the $9 \times 20 = 180$ blocks.
- Take the weighted average (weights: $\alpha_j^{(l)}$) of the measured latencies for the 9 blocks in the l-th layer.
- $\text{Lat}(\Theta)$: sum of the latencies across the 20 layers.
- Apply (add or multiply) $\text{Lat}(\Theta)$ to the loss function.

Thank You!