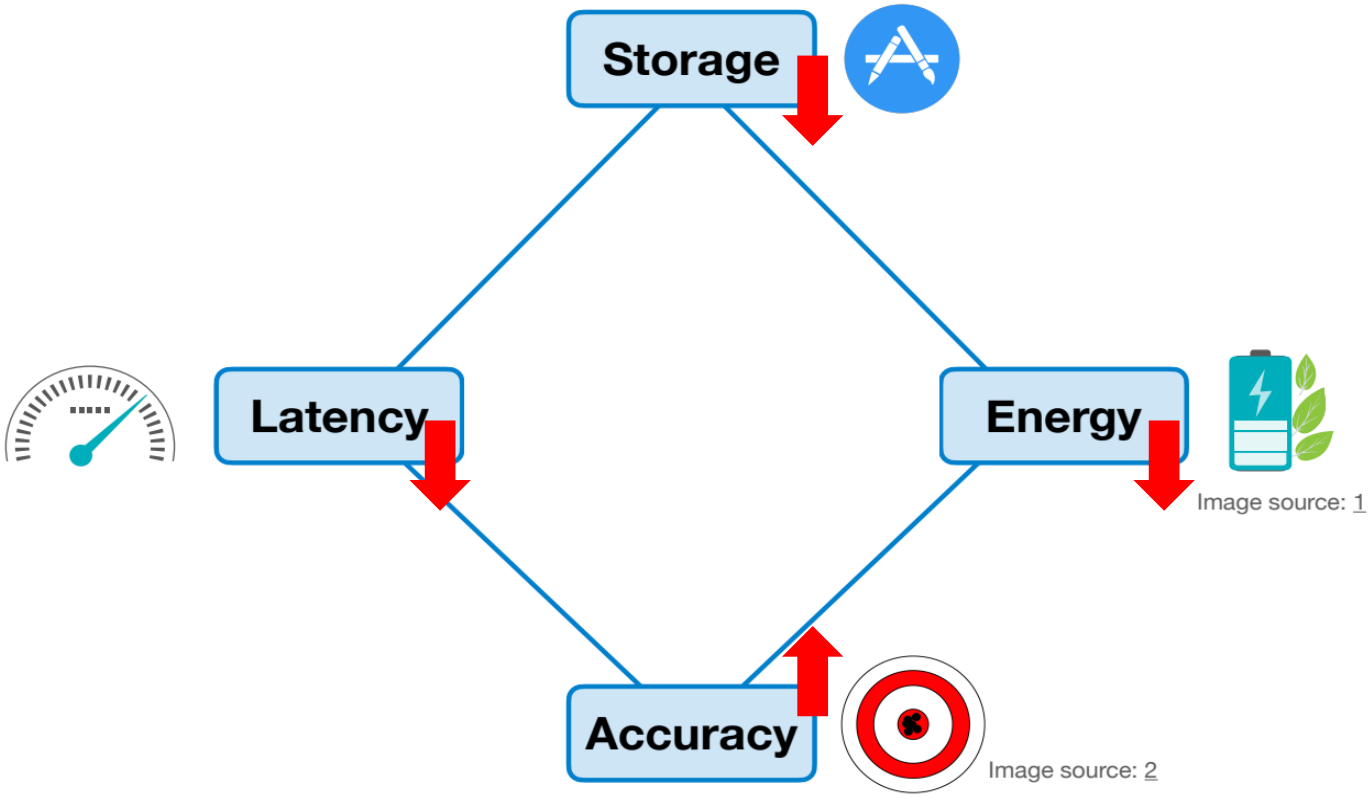
TinyML and Efficient Deep Learning Computing

Neural Architecture Seach (NAS)

발표자 신호주

Contents

Trade-off Between Efficiency & Accuracy

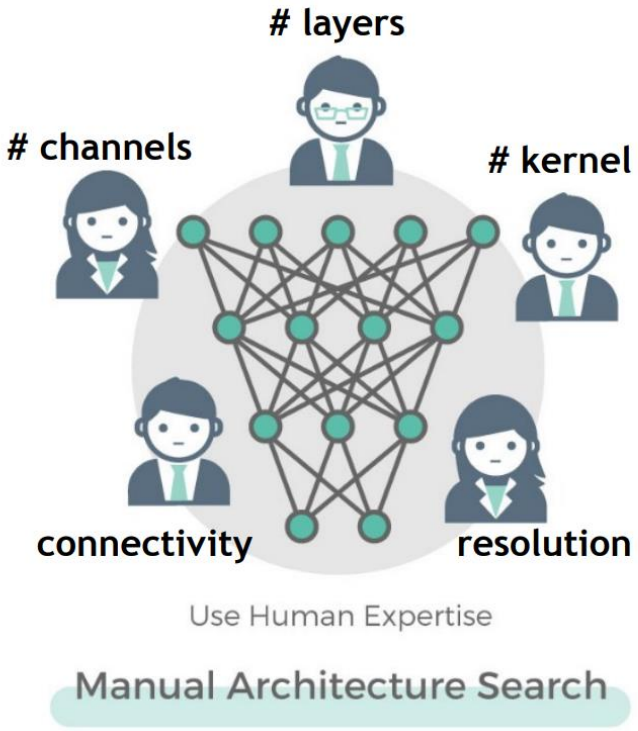


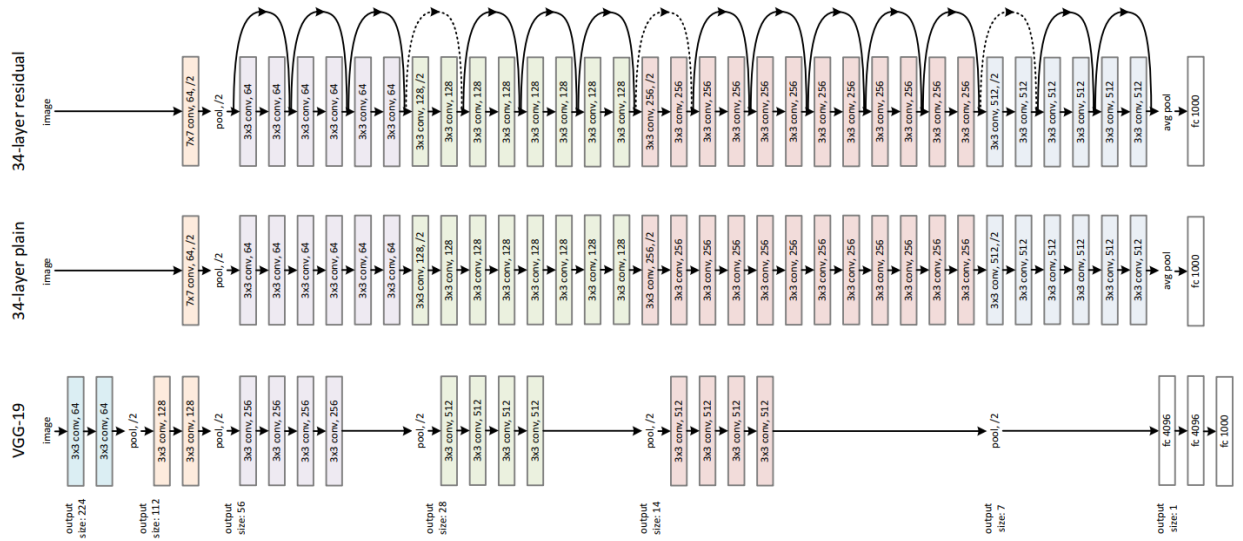
Design Philosophy of NAS

설계 단계에서부터
compact한 모델을 만들자!

From Manual Design to Automatic Design

- Architecture를 수작업으로 설계함
- Block 내부의 구조, 채널의 수, 레이어의 수 등 **넓은 Search Space** 존재
  ⇨ 모든 경우를 탐색할 수 없음 (manual design is unscalable!)





From Manual Design to Automatic Design

- Architecture를 수작업으로 설계함
- Block 내부의 구조, 채널의 수, 레이어의 수 등 **넓은 Search Space** 존재
  ⇨ 모든 경우를 탐색할 수 없음 (manual design is unscalable!)

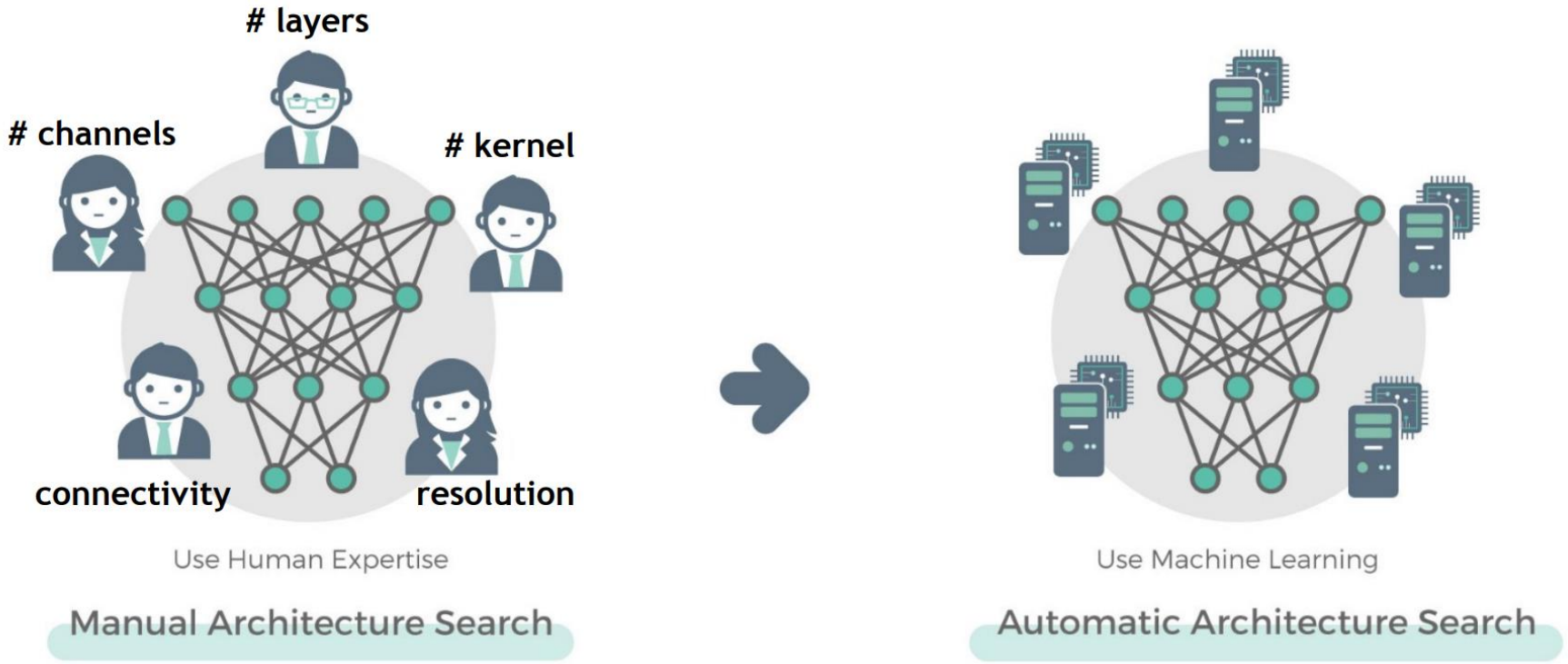
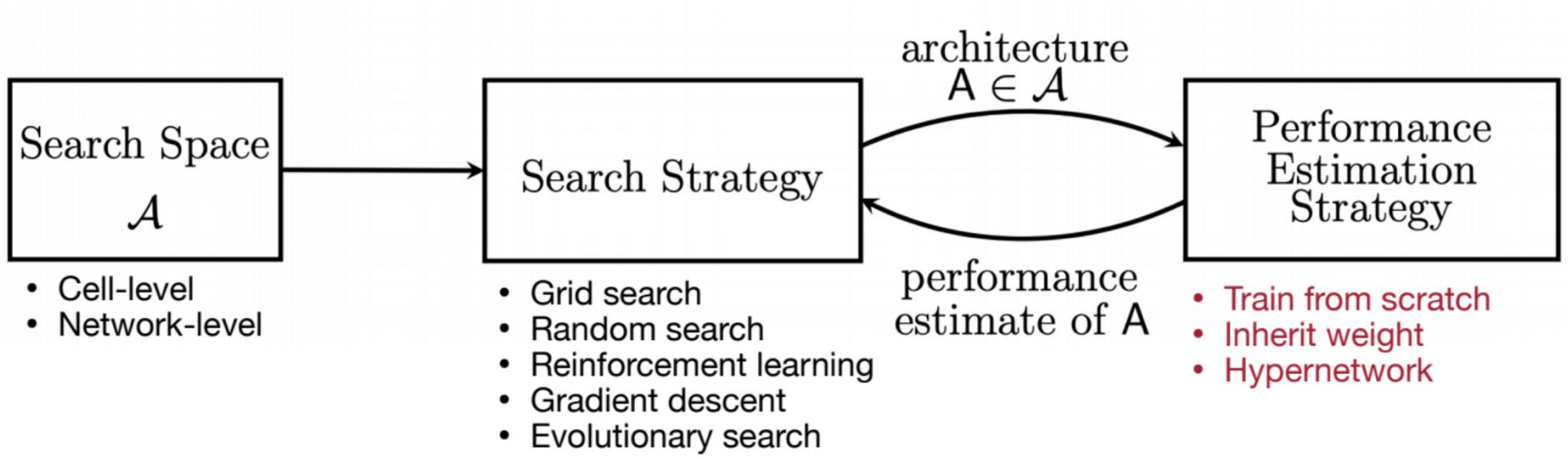


Illustration of NAS

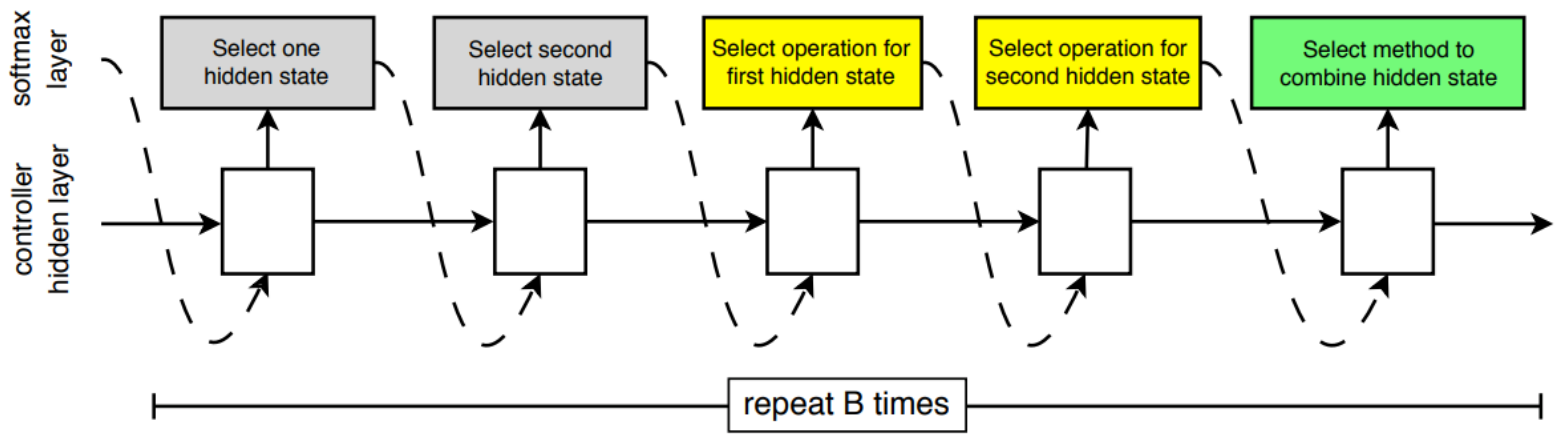
- **Goal**: Search space $\mathcal{A}$ 에서 최적화 대상 지표를 가장 잘 만족시키는 모델을 찾는 것
- **Component**:
  - Search Space: NN architecture 후보의 집합
  - Search Strategy: search space 탐색 전략
  - Performance Estimation Strategy: 후보의 성능 평가 방식

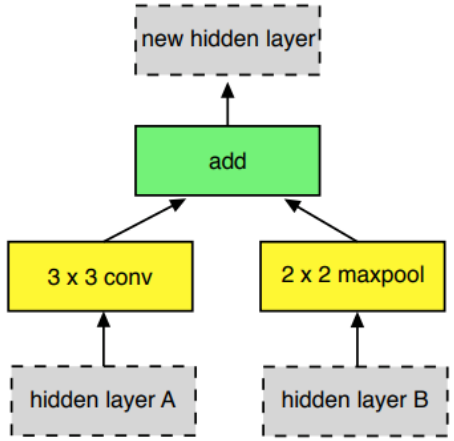


Search Space

- **Cell-level search space**
  - **RNN controller**: 후보 cell 만들어 냄
    1. Input 2개 선택
    2. 두 input에 각각 적용할 연산 선택 (e.g., conv / pooling / identity ...)
    3. 두 결과를 합칠 방법 선택 (e.g., add / element-wise multiplication ...)

경우의 수 $(2 \times 2 \times M \times M \times N)^B$

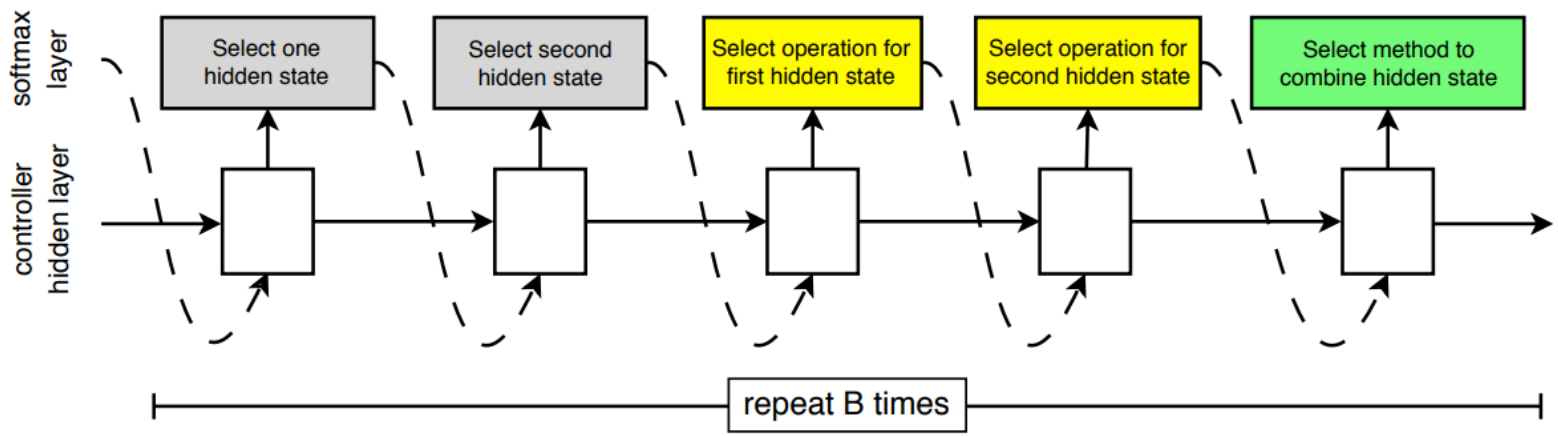


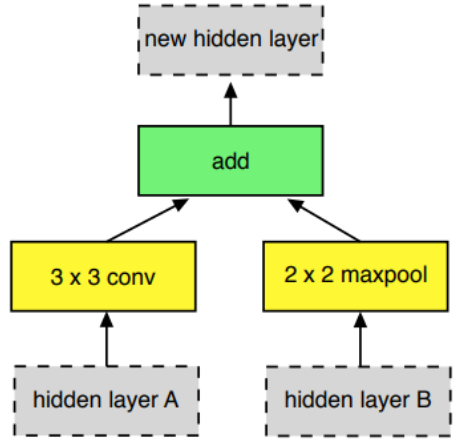


Search Space

- **Cell-level search space**
  - **RNN controller**: 후보 cell 만들어 냄
    1. Input 2개 선택
    2. 두 input에 각각 적용할 연산 선택 (e.g., conv / pooling / identity ...)
    3. 두 결과를 합칠 방법 선택 (e.g., add / element-wise multiplication ...)

경우의 수 $(2 \times 2 \times 5 \times 5 \times 2)^5 = 3.2 \times 10^{11}$



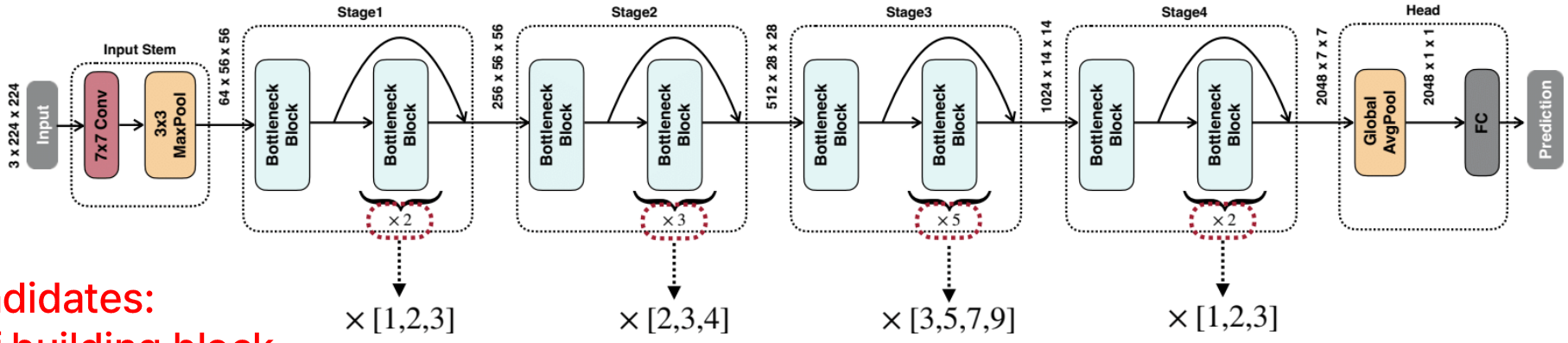


Search Space

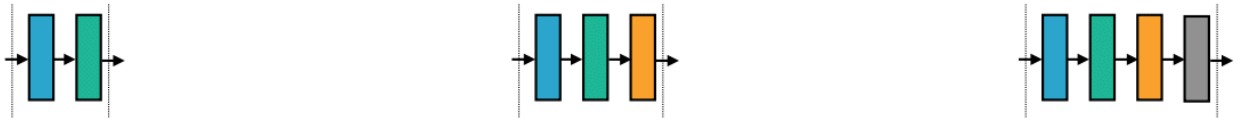
- **Network-level search space**
  - Deep vs. Shallow 단계에서 레이어의 특성이 매우 다름
  - 같은 block을 반복하는 것은 최적의 방식이 아님

Search Space

- **Network-level search space:** depth dimension
  - 각 stage 마다 사용할 block 수의 집합

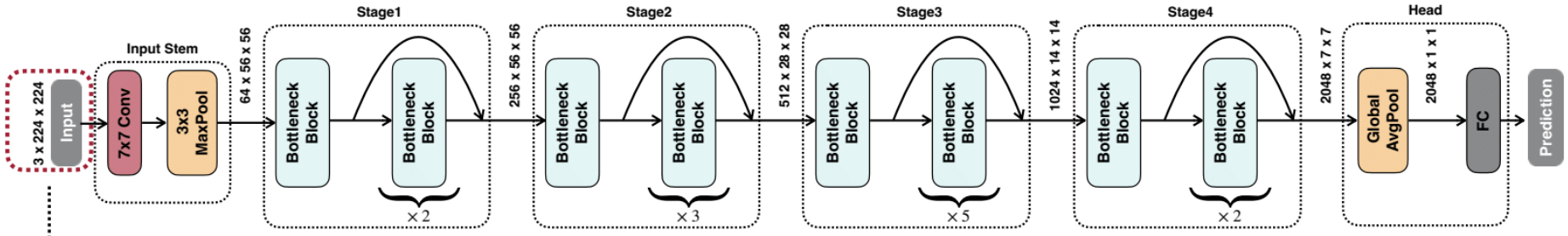


Candidates:
# of building block



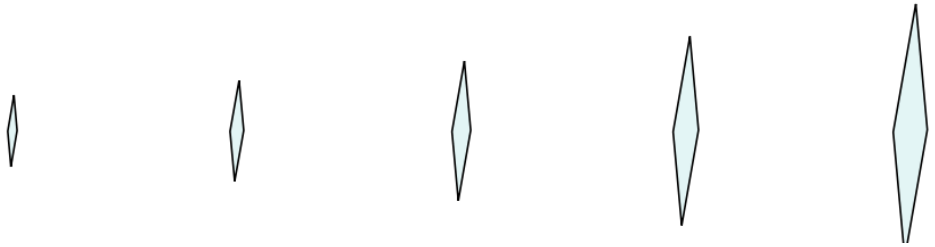
Search Space

- **Network-level search space:** input resolution dimension
  - 입력 이미지의 크기 집합



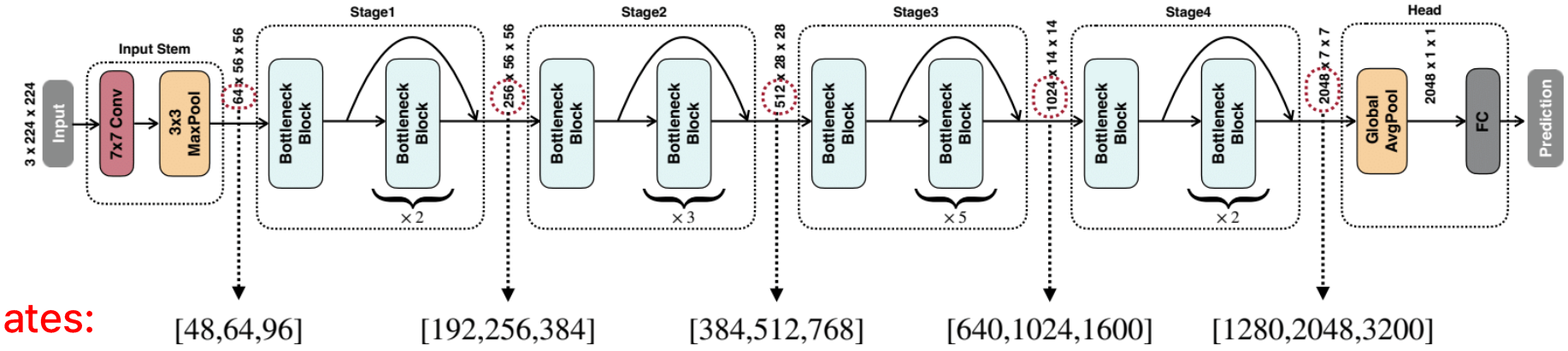
Candidates:
input resolution

[(3,128,128); (3,160,160); (3,192,192); (3,224,224); (3,256,256)]



Search Space

- **Network-level search space:** width dimension
  - 각 stage의 채널 수 집합

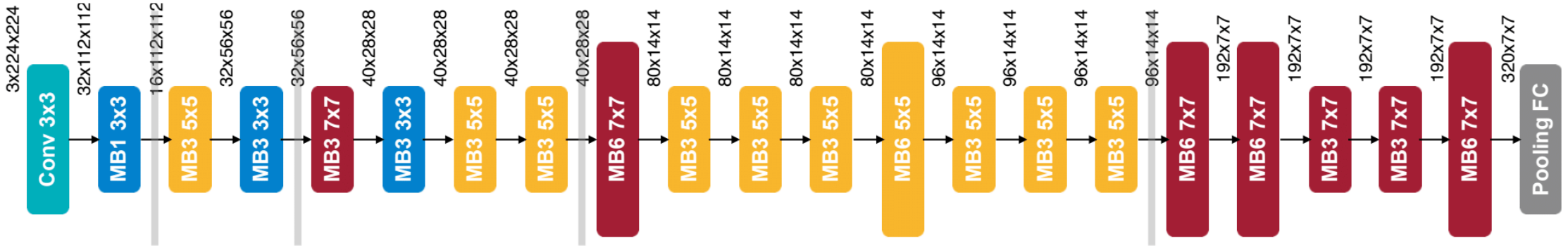


Candidates:
# of channels

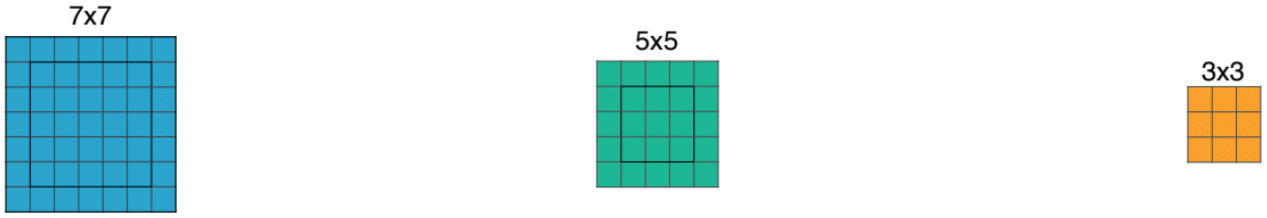
[48,64,96] [192,256,384] [384,512,768] [640,1024,1600] [1280,2048,3200]

Search Space

- **Network-level search space:** kernel size dimension

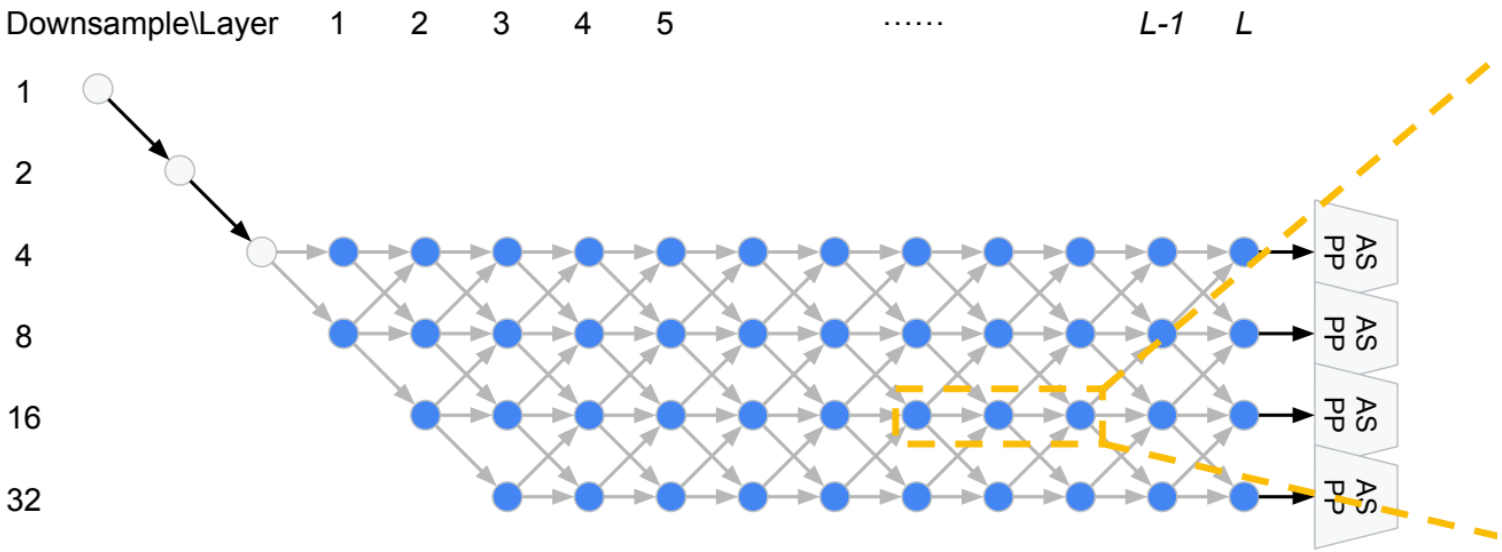


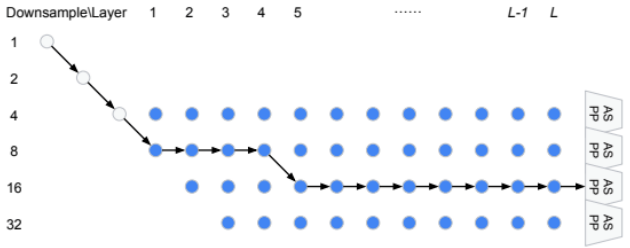
For models that use depthwise convolution, we can choose the kernel size for each depthwise convolution.



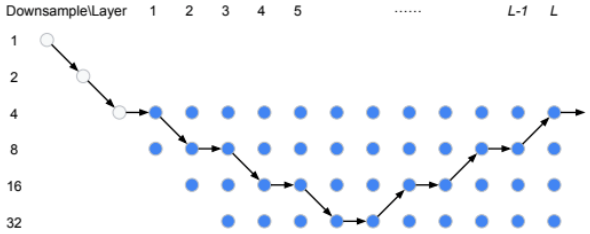
Search Space

- **Network-level search space:** topology connection

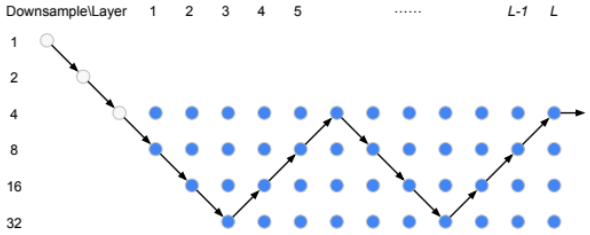




(a) Network level architecture used in DeepLabv3 [9].



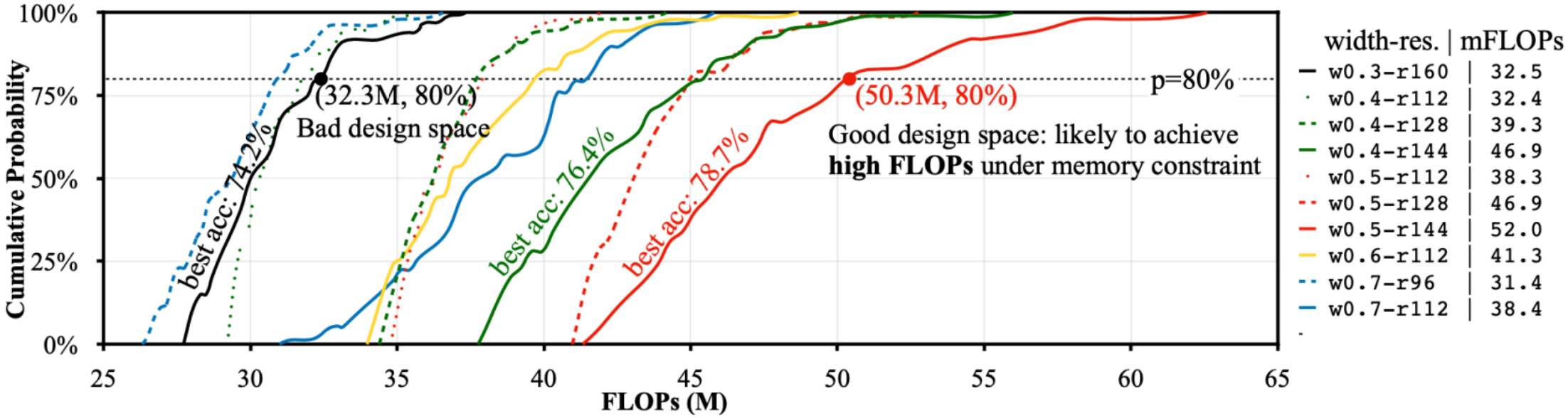
(b) Network level architecture used in Conv-Deconv [56].



(c) Network level architecture used in Stacked Hourglass [55].

Design the Search Space

- **Design the search space for TinyML**
  - 휴대폰, IoT 기기의 ML모델은 latency / battery / memory 제약 존재
  - 제약을 만족하는 모든 architecture를 학습한 후 성능을 확인하는 것은 비용 / 시간 측면에서 어려움
    ⇨ 효과적으로 search space를 설계해야 함



Search Strategy

- **Grid Search**
  - 전체 search space: 1차원 상의 데카르트 곱 (Cartesian product)
  - 후보 network의 정확도를 알기 위해서 모델 전체를 학습해야 함
  - 특정 제약 (e.g., latency) 을 만족하지 않는 후보를 사전에 제외할 수 있음

Resolution / Width	1.0x	1.1x	1.2x
1.0x	50.0%	53.0%	**54.9%**
1.1x	51.0%	53.5%	55.4%
1.2x	52.0%	54.1%	56.2%

Satisfies the latency constraint

Breaks the latency constraint

Search Strategy

- **Random search**
  - 전체 search space: grid search 와 동일
  - 랜덤하게 일부 architecture를 샘플링하는 방식으로 탐색

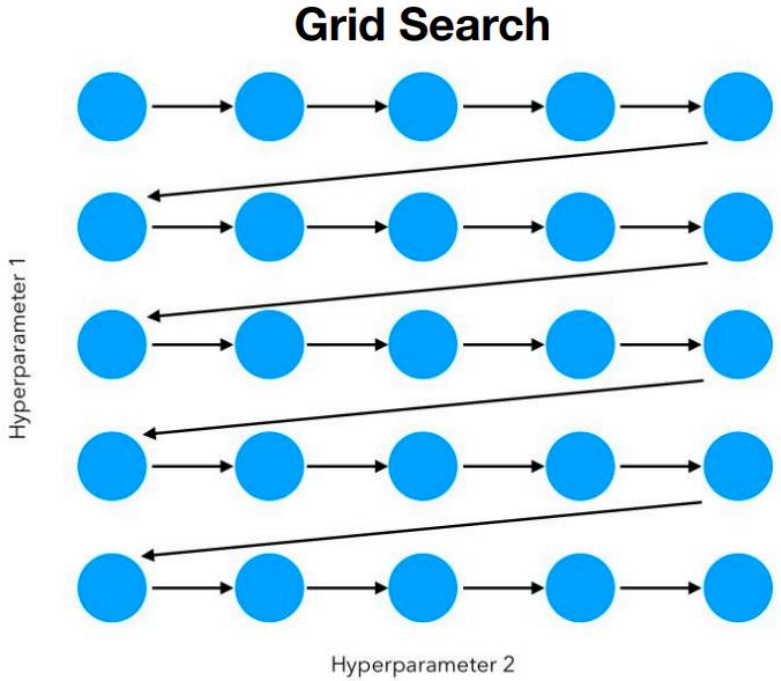


image source: 1

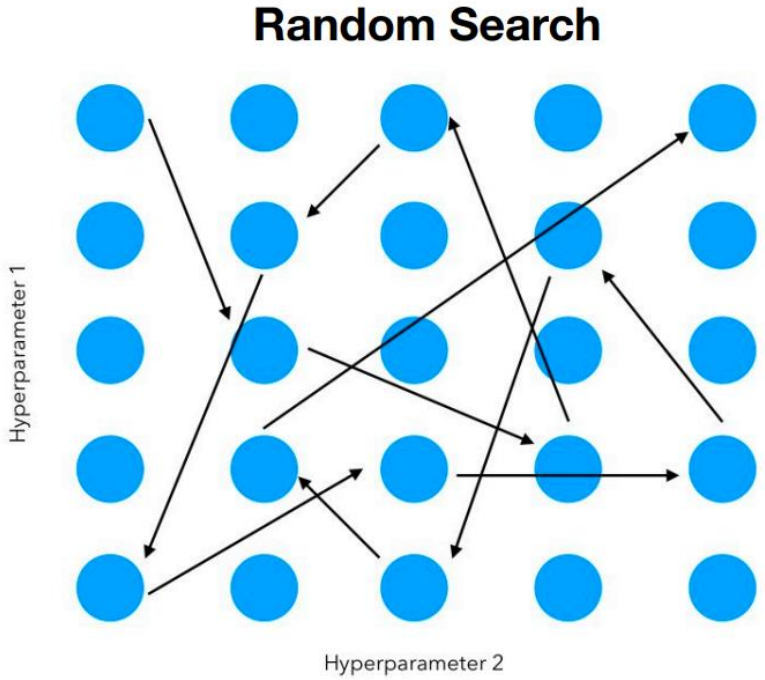
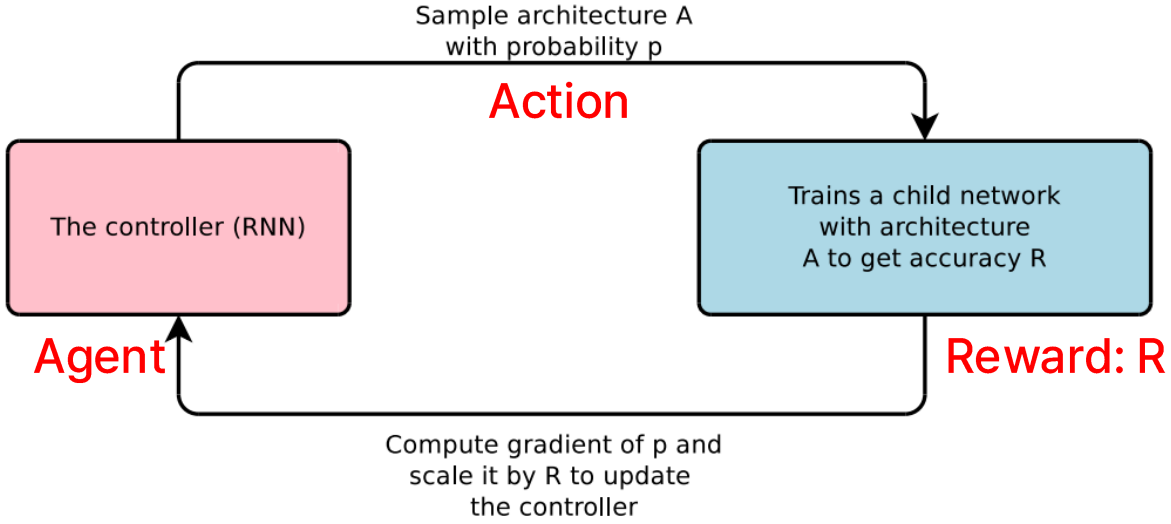


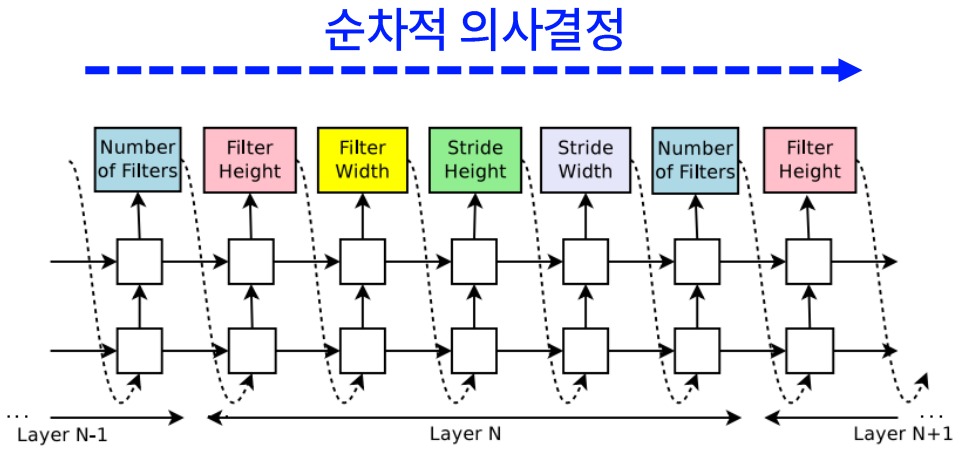
image source: 2

Search Strategy

- **Reinforcement learning**
  - Architecture 설계를 순차적 의사결정 문제로 정의
  - RNN controller를 학습하는데 강화학습 기법을 적용



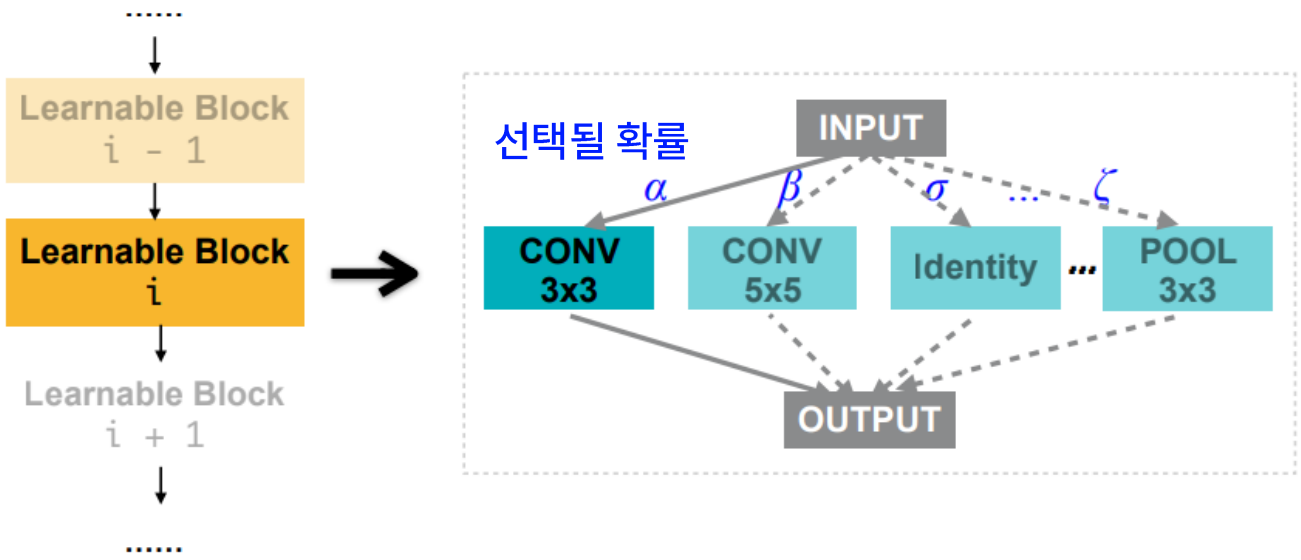
Overview of RL-based NAS



The RNN controller

Search Strategy

- **Gradient descent**
  - 한 node에 대해 가능한 연산을 모두 저장함
  - Latency를 미분 가능한 형태인 term ($\mathbb{E}[\text{latency}]$) 으로 만들어 최적화할 수 있음



Latency 예측 레이어 / Lookup table

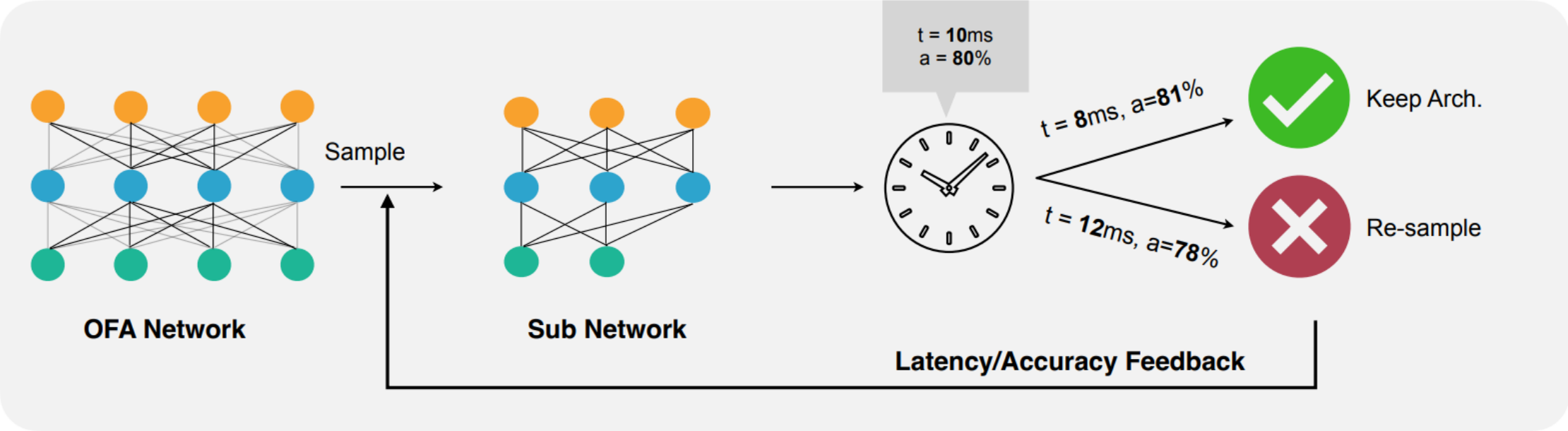
$$\mathbb{E}[\text{Latency}] = \alpha \times F(\text{conv\_3x3}) + \beta \times F(\text{conv\_5x5}) + \sigma \times F(\text{identity}) + \ldots\ldots + \zeta \times F(\text{pool\_3x3})$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i]$$

$$Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

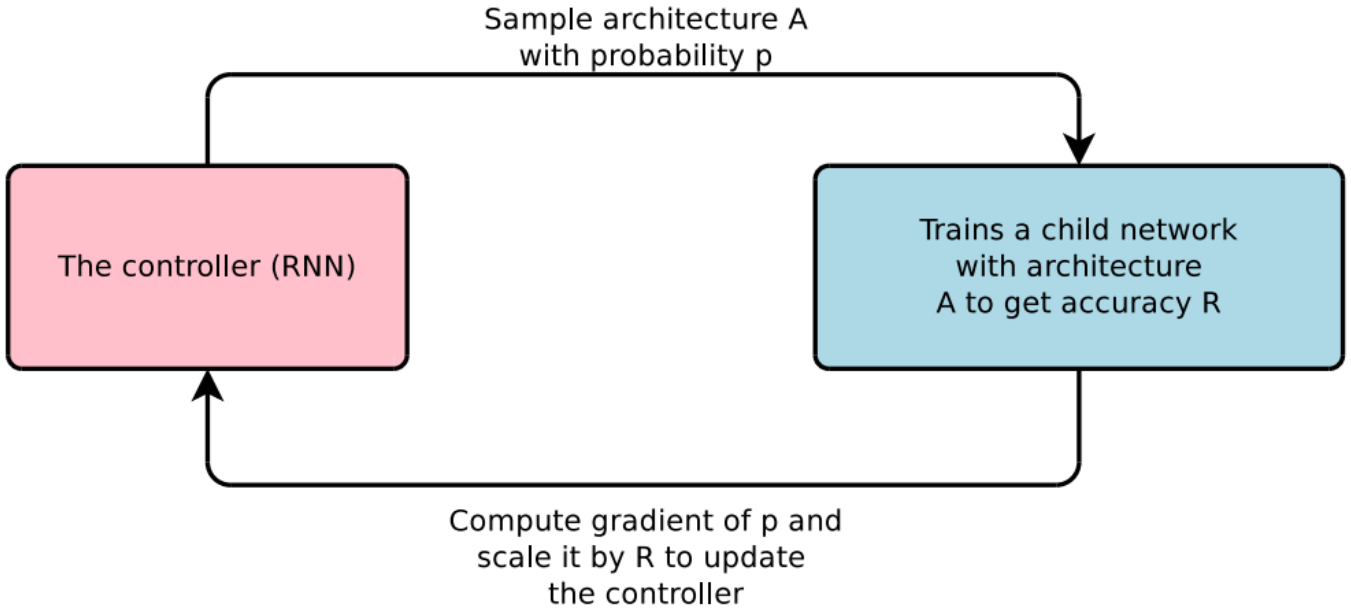
Search Strategy

- **Evolutionary search**
  - 생물학의 진화 이론에서 나온 방식
  - 모델의 적합도 F(정확도, 효율성) 측정 → Mutation → Crossover



Accuracy Estimation Strategy

- **Train from scratch**
  - 후보 architecture A 처음부터 학습한 뒤, 정확도 R 측정
  - 학습 시간 및 비용이 매우 많이 필요 ⇒ **NOT scalable!**



CIFAR10

Train 12,800 model architectures

22,400 GPU-hours

Large-Scale Datasets

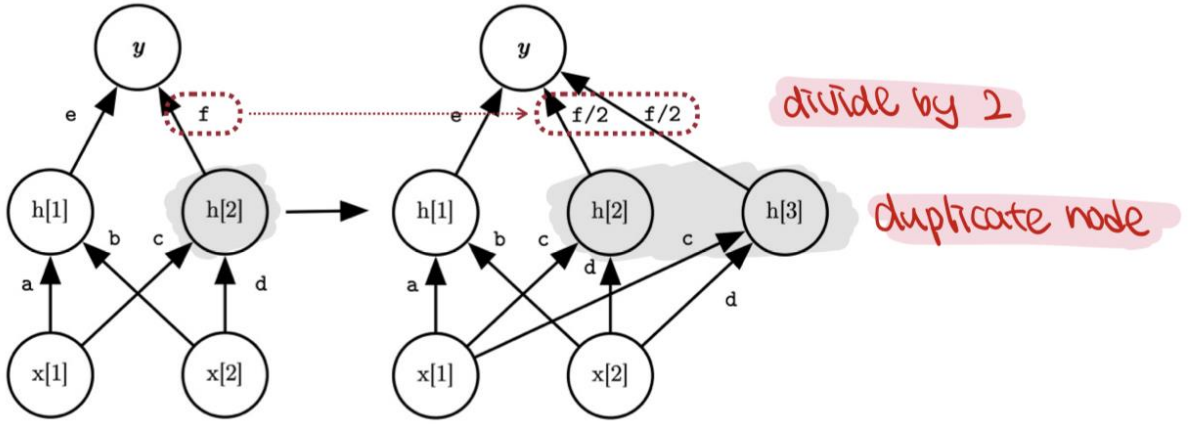
ImageNet
COCO
…

Accuracy Estimation Strategy

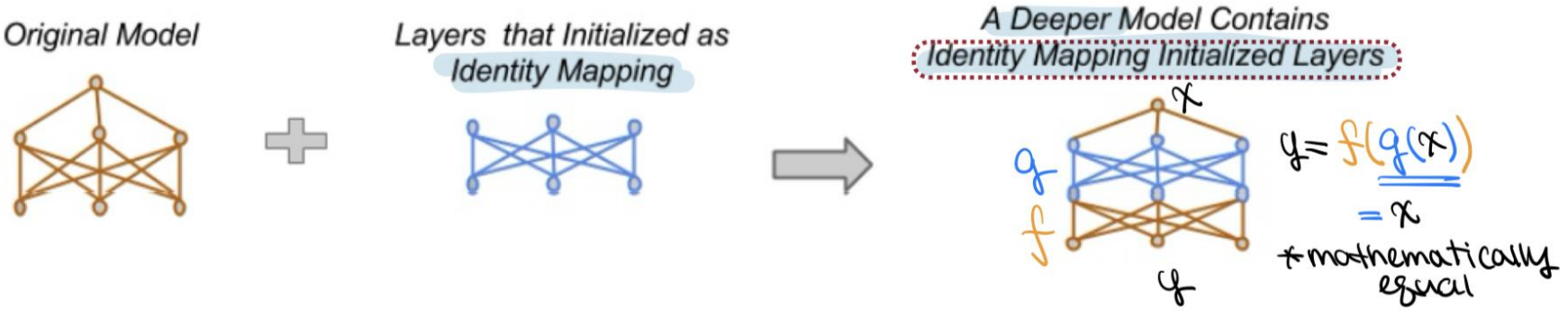
- **Inherit weight**
  - 부모 모델로부터 가중치를 상속받는 방식
  - 처음부터 학습하는 것보다 학습 비용을 줄일 수 있음

$$y = f/2 \times h[2] + f/2 \times h[3]$$
$$= f \times h[2]$$

Net2Wider

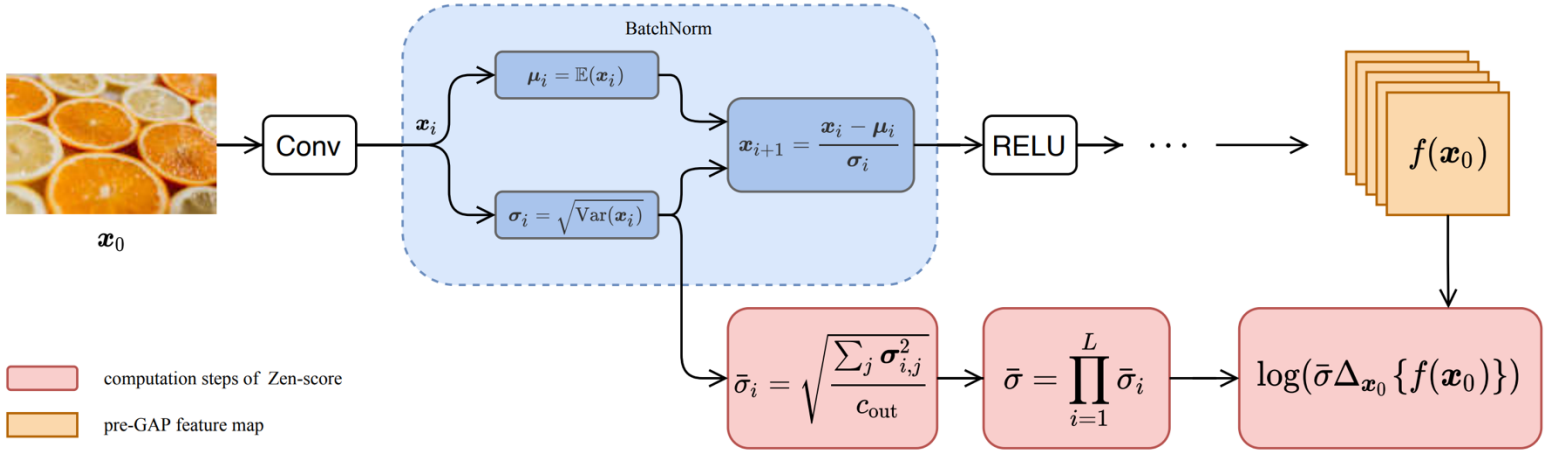


Net2Deeper



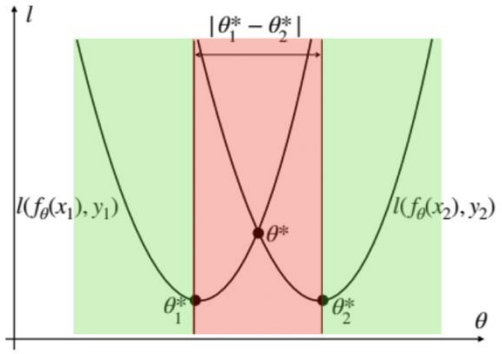
Zero-Shot NAS

- **ZenNAS**
  1. 입력 $x \sim \mathcal{N}(0,1)$ 분포 따르도록 초기화
  2. 입력에 매우 작은 값 (perturbation)을 더해 $x' = x + \epsilon$ 값 얻음
  3. 네트워크의 가중치를 정규 분포를 따르도록 초기화
  4. 가정 **좋은 모델은 입력의 변화에 민감**: $z_1 = \log(f(x') - f(x))$
     + 배치 정규화의 분산 작아야 함

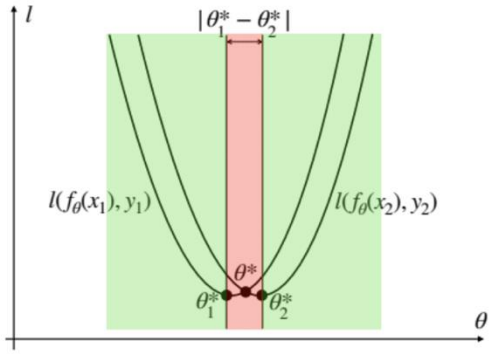


Zero-Shot NAS

- **GradSign**
  - 직관 좋은 모델은 샘플마다의 local minima 사이의 거리가 좁을 것이다!
  - 서로 다른 샘플에서의 gradient 부호가 같음 (초록색 영역)
  - 좋은 모델은 같은 부호를 가진 샘플의 수가 많음



(a) Optimization landscape with **sparser** sample-wise local optima corresponding to **worse** $J(\theta^*)$.



(b) Optimization landscape with **denser** sample-wise local optima corresponding to **better** $J(\theta^*)$.

Algorithm 1: GradSign

Result: GradSign score τ_f for a function class f_θ
Given $\mathcal{S} = \{(x_i, y_i)\}_{i\in[n]}$, randomly select initialization point θ_0;
Initialize $g[n, m]$;
for $i = 1, 2, \cdots, n$ **do** i, n corresponds to samples
  for $k = 1, 2, \cdots, m$ **do** k, m corresponds to layers
    $g[i, k] = \text{sign}([\nabla_\theta l(f_\theta(x_i), y_i)|_{\theta_0}]_k)$
  end
end
$\tau_f = \sum_k |\sum_i g[i, k]|$; $|\sum_{i=1}^{n} g[i, k]| \leq n$, "=" is achieved only when the gradients at all samples have the same sign.
return τ_f

Thank You