TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Neural Architecture Search

(NAS)

Table of Contents

Table of Contents

Neural Architecture Search (NAS)

Definition

NAS is a technique for automating the design of powerful deep neural networks on a specific task.

Neural Architecture Search (NAS)

Why do we need NAS?

Neural Architecture Search (NAS)

Requirement

- Manually design a normal network architecture (ignore the performance) → Easy
- Manually design a network architecture with high performance → Can perform (but requires a lot of experience and time-consuming)

Neural Architecture Search (NAS)

Drawbacks of <u>manual</u> architecture design

- Experience requirement from experts and time-consuming for designing an effective architecture
- Much of the search space is unexplored → potential architectures can be neglected

Neural Architecture Search (NAS)

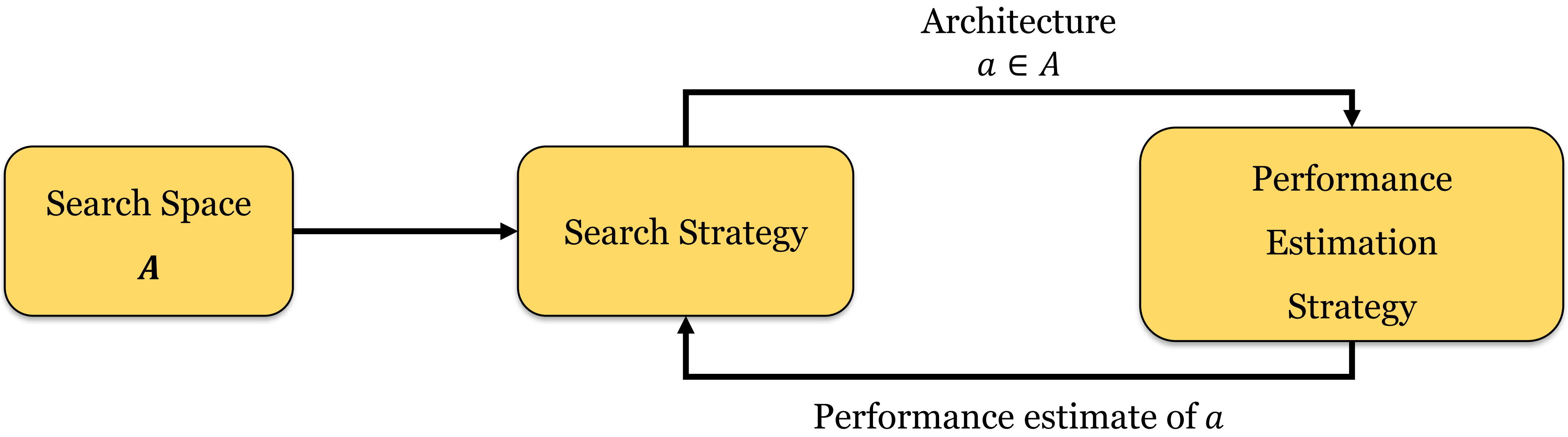
Definition

NAS is a technique for automating the design of powerful deep neural networks architectures on a specified task.

Aim

Automatically search for architecture networks with strong performance.

NAS components



Search Strategy

- Some common search strategies:
  - Evolutionary Algorithms (Xie and Yuille, 2017; Lu et al., 2018; Real et al., 2018)
  - Reinforcement Learning (Zoph and Le, 2017; Wang et al., 2019)
  - …
- We can combine each one together.

Performance Estimation Strategy

- One of the optimization objectives for optimizers when searching is the test performance.

  → However, the test performance is evaluated on the **unseen** data.
- To guide the optimizers in the search process, we need to have a strategy to calculate or estimate the test performance.

Performance Estimation Strategy

- Some common performance estimation strategies:
  - ❑ Full training and evaluating the performance (on the validation data).

  → <u>Advantages</u>:

  - High-correlation → The final performance is usually good.

  → <u>Drawbacks</u>:

  - Time-consuming.
  - Requires lot of computational resources.

Performance Estimation Strategy

- Some common performance estimation strategies:
  - ❑ Learning curve extrapolation (Early Stopping) (Ru et al., 2020; Zhou et al., 2020) (commonly-used approach)

  → Advantages:

  - Shorten the searching time and reduce the computational resources.

  → Drawbacks:

  - Need to determine the effective stopping epoch.
  - Requires time and computational resources to achieve the high-correlation.

Performance Estimation Strategy

- Some common performance estimation strategies:
  - ❑ Using training-free "proxy" metrics (Chen et al., 2021; Abdelfattah et al., 2021)

  → Advantages:

    - Extremely effective in shorting the searching time and reducing the computational resources.

  → Drawbacks:

    - Low-correlation.
    - Designing a high-correlation training-free "proxy" metric is a challenge.

Performance Estimation Strategy

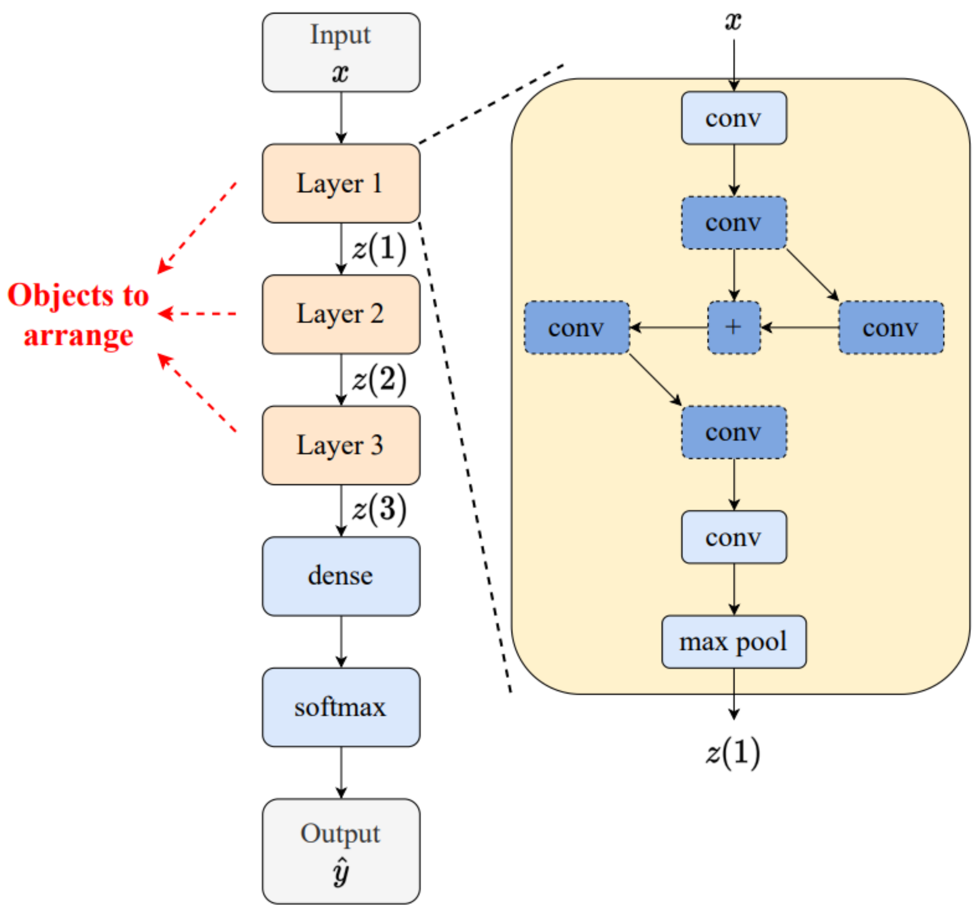
- Some common performance estimation strategies:
  - Full training and evaluating the performance (on the validation data).
  - Learning curve extrapolation (Ru et al., 2020; Zhou et al., 2020)
  - Using training-free "proxy" metrics (Chen et al., 2021; Abdelfattah et al., 2021)
  - …

Search Space

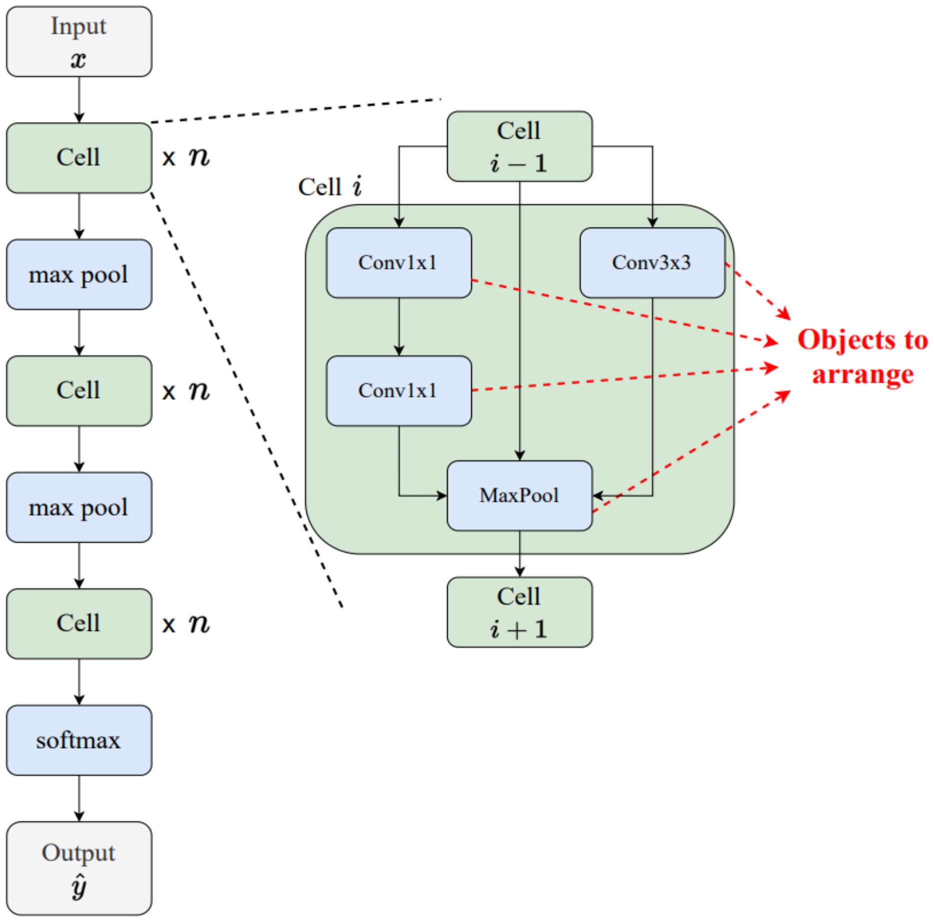
- In fact, each solution in the search space is not the entire architecture. It is just a part of the architecture.
- Specifically, each solution is the arrangement of some components in the architecture (e.g., layers; operations in cells).
- Based on the components which are used to arrange, there are 2 types of search space:
  - Macro-level
  - Micro-level

Search Space

Macro-level

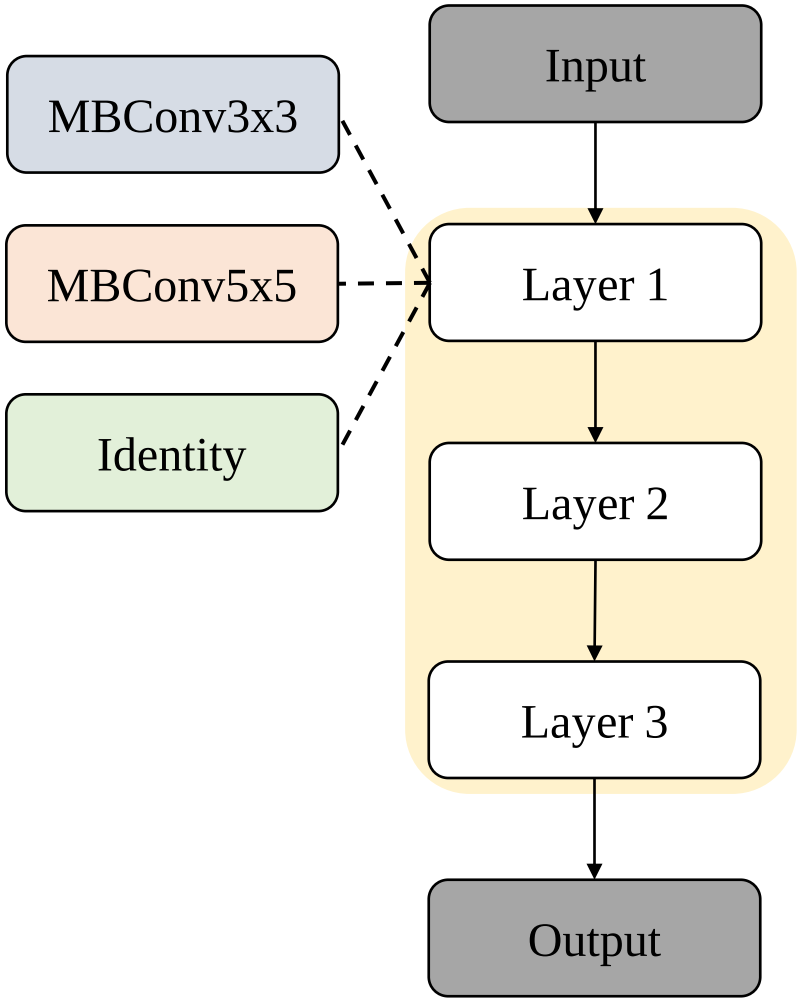


Micro-level

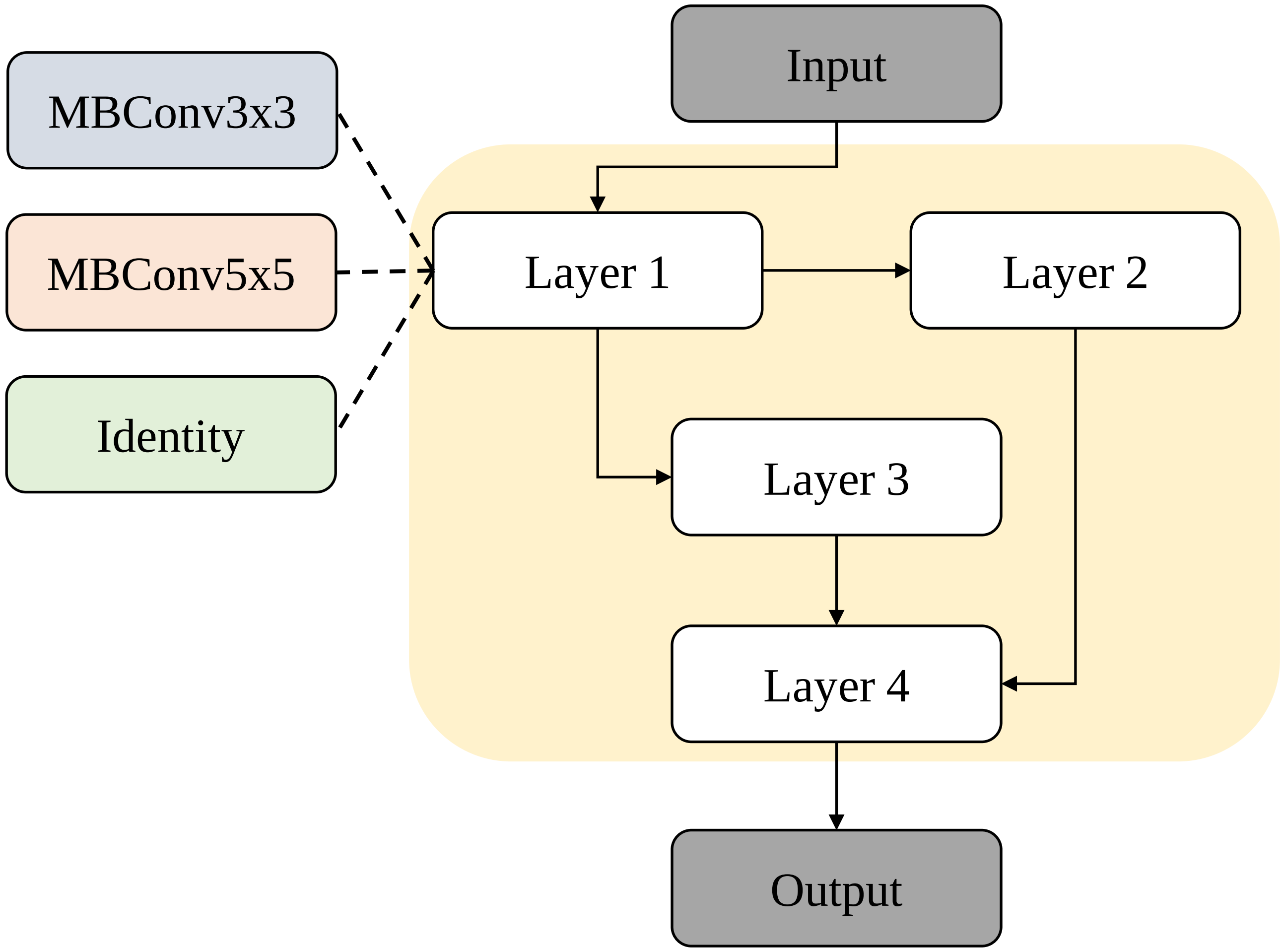


Macro-level

- Components for encoding → Layers
- A solution in search space → An arrangement of layers in the architecture.
- Layers can be in the same type or different type.
- The layers can be arranged as the chain structure or directed acyclic graph (DAG).



Example of an arrangement as a chain structure



Example of an arrangement as a DAG

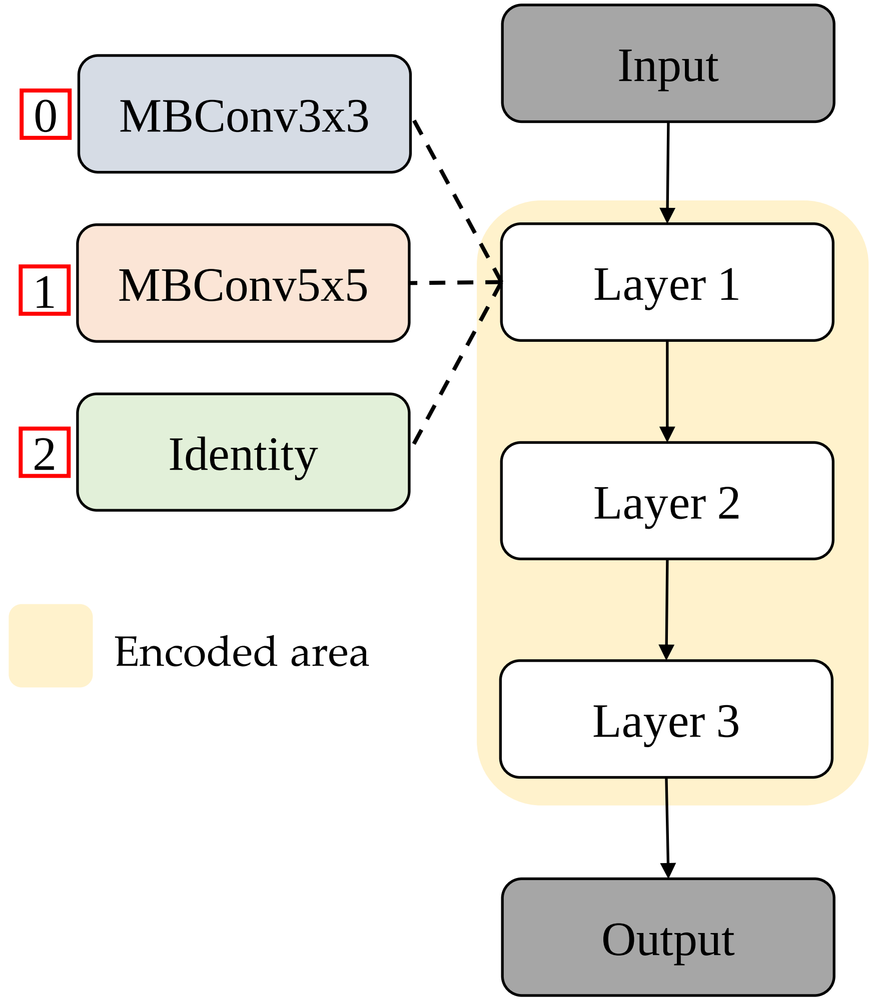
- Nodes represent layers; edges represent the flow of data.
- Edge e_{ij} ($i < j$) means that node ***j-th*** uses the output of node ***i-th*** as the input.
- A node can have more than 1 input node.

[Macro-level] Solution Encoding

Chain structure

- Use 1 vector (e.g., string, integer numbers)

Example



	Layer 1	Layer 2	Layer 3
Solution 1	MBConv3x3	MBConv5x5	MBConv5x5
Solution 2	Identity	MBConv3x3	Identity

Solution 1 (string) ['MBConv3x3', 'MBConv5x5', 'MBConv5x5']

Solution 2 (integer number) [2, 0, 2]

[Macro-level] Solution Encoding (cont.)

Chain structure

- Use 1 vector (e.g., string, integer numbers)

DAG

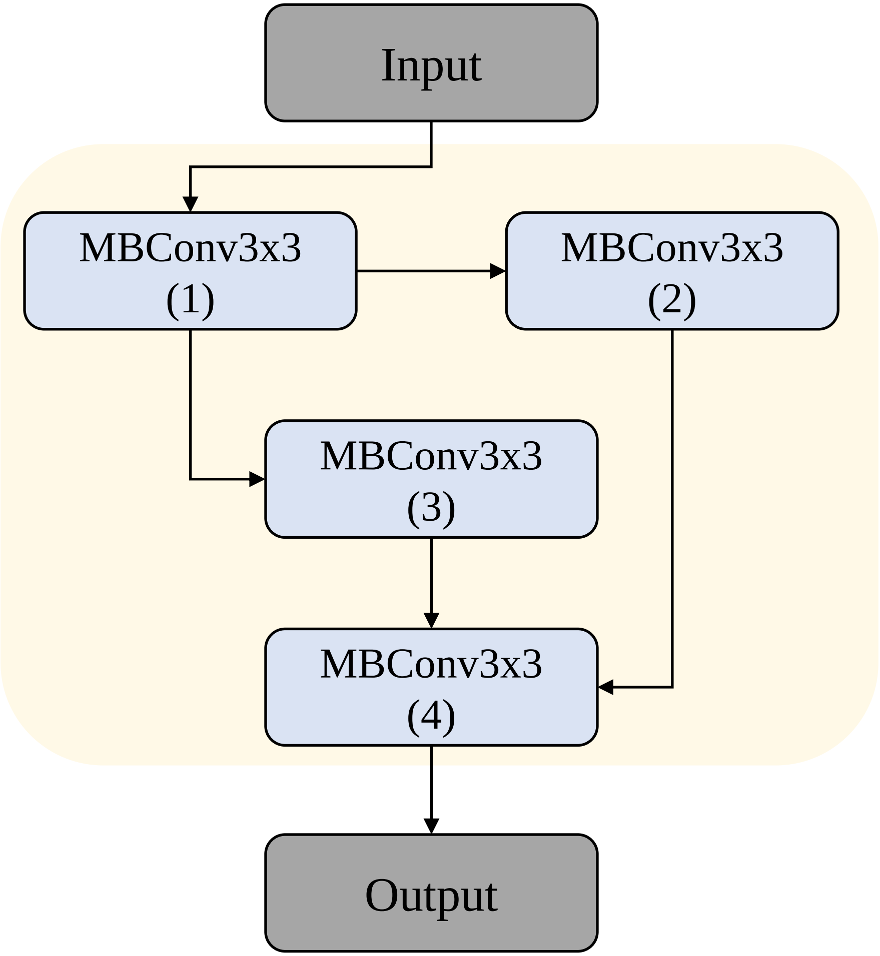
- **Case 1: Layers in the same type**

→ Represent the connections.

→ Use 1 binary vector (or upper-triangular binary matrix).

Case 1: Layers in the same type

Encoded area

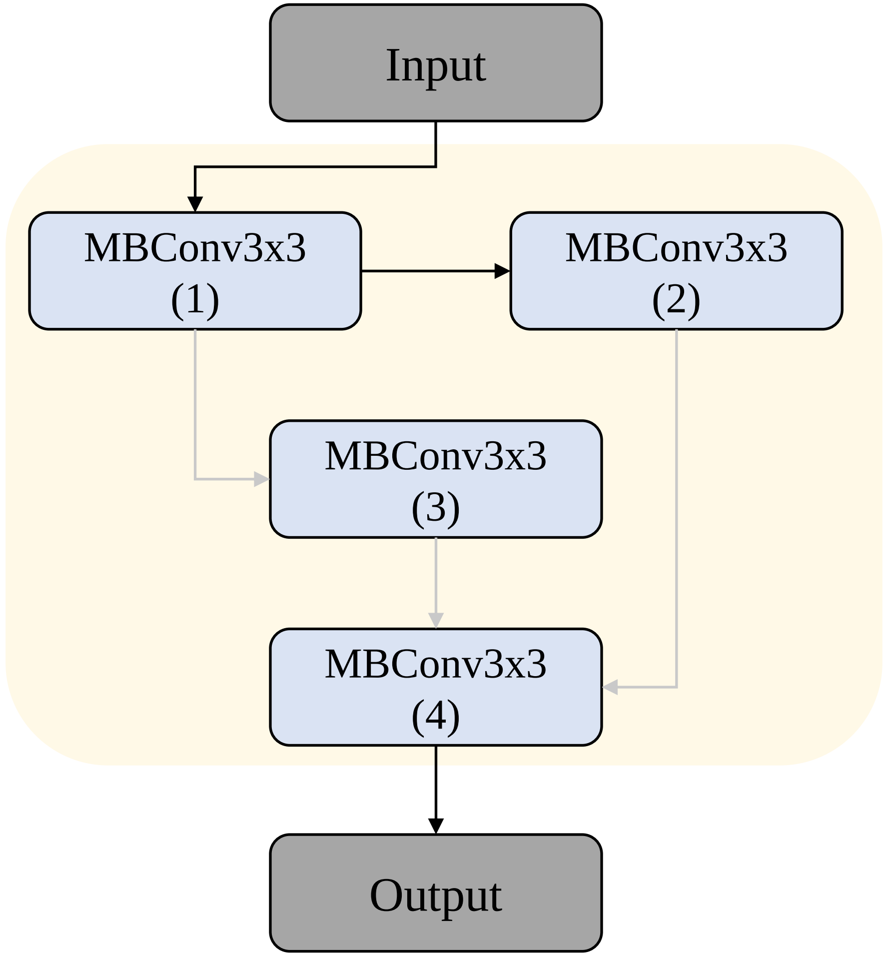


- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

Case 1: Layers in the same type

Encoded area



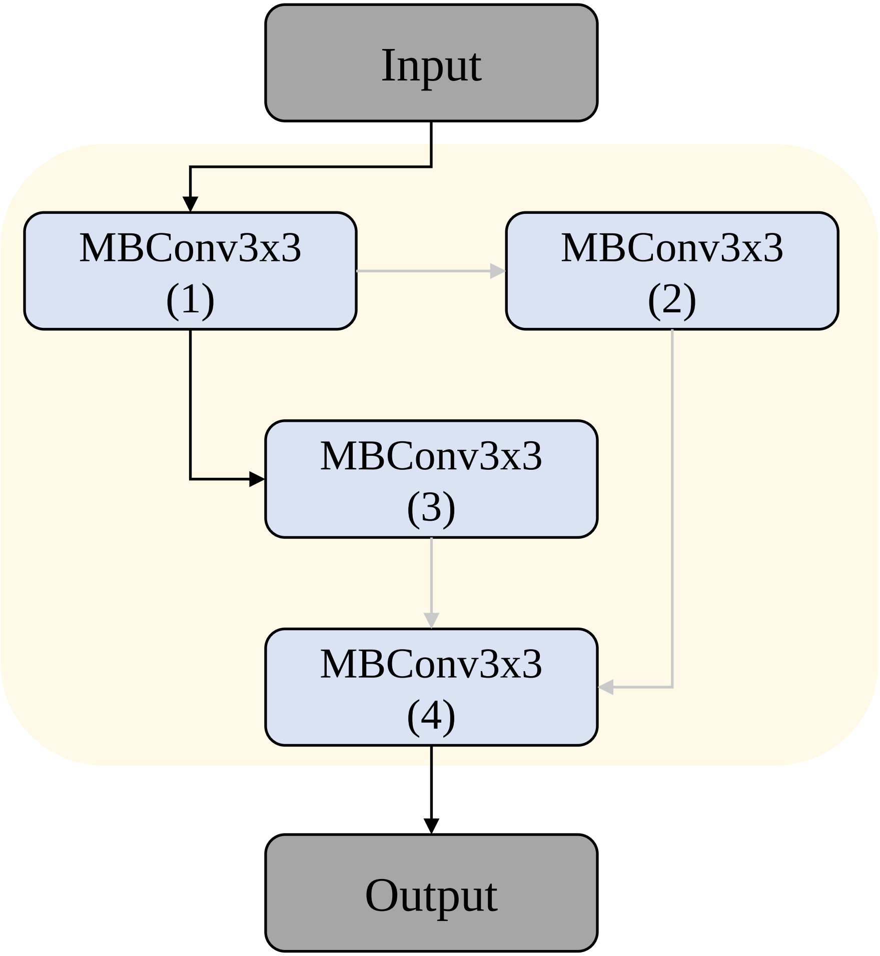
- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$\rightarrow [1, 1, 0, 0, 1, 1]$$

Case 1: Layers in the same type

Encoded area



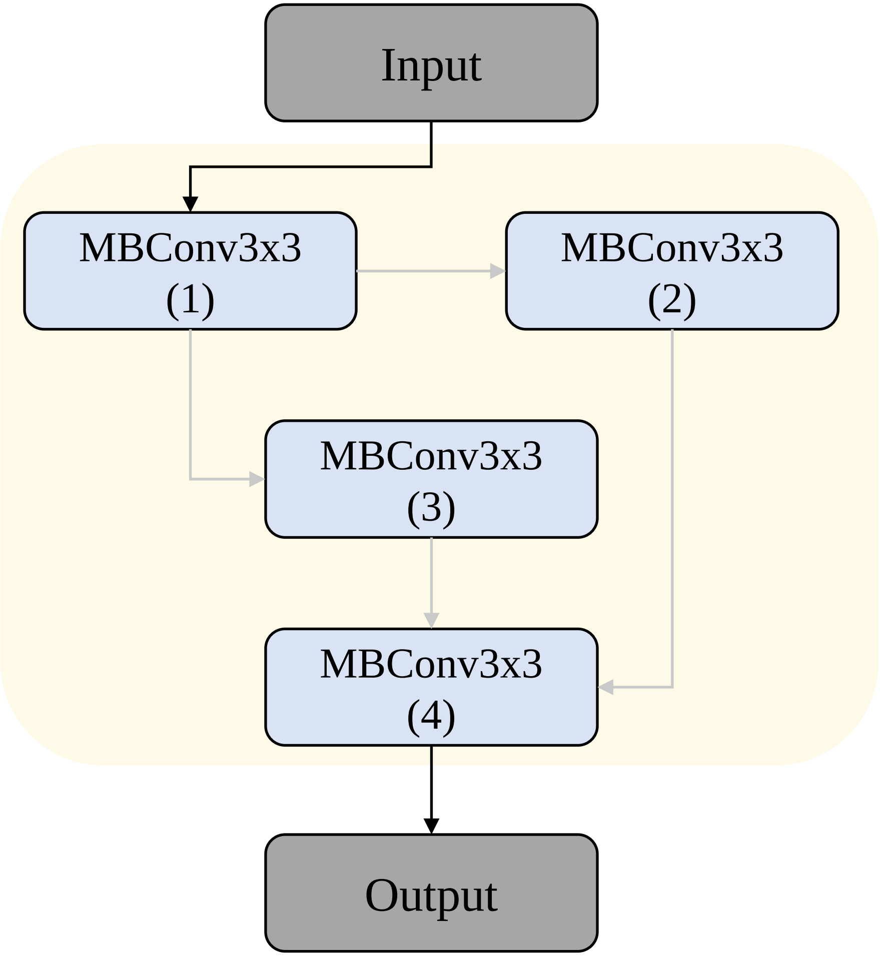
- Binary vector

$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$

$\rightarrow [1, 1, 0, 0, 1, 1]$

Case 1: Layers in the same type

Encoded area

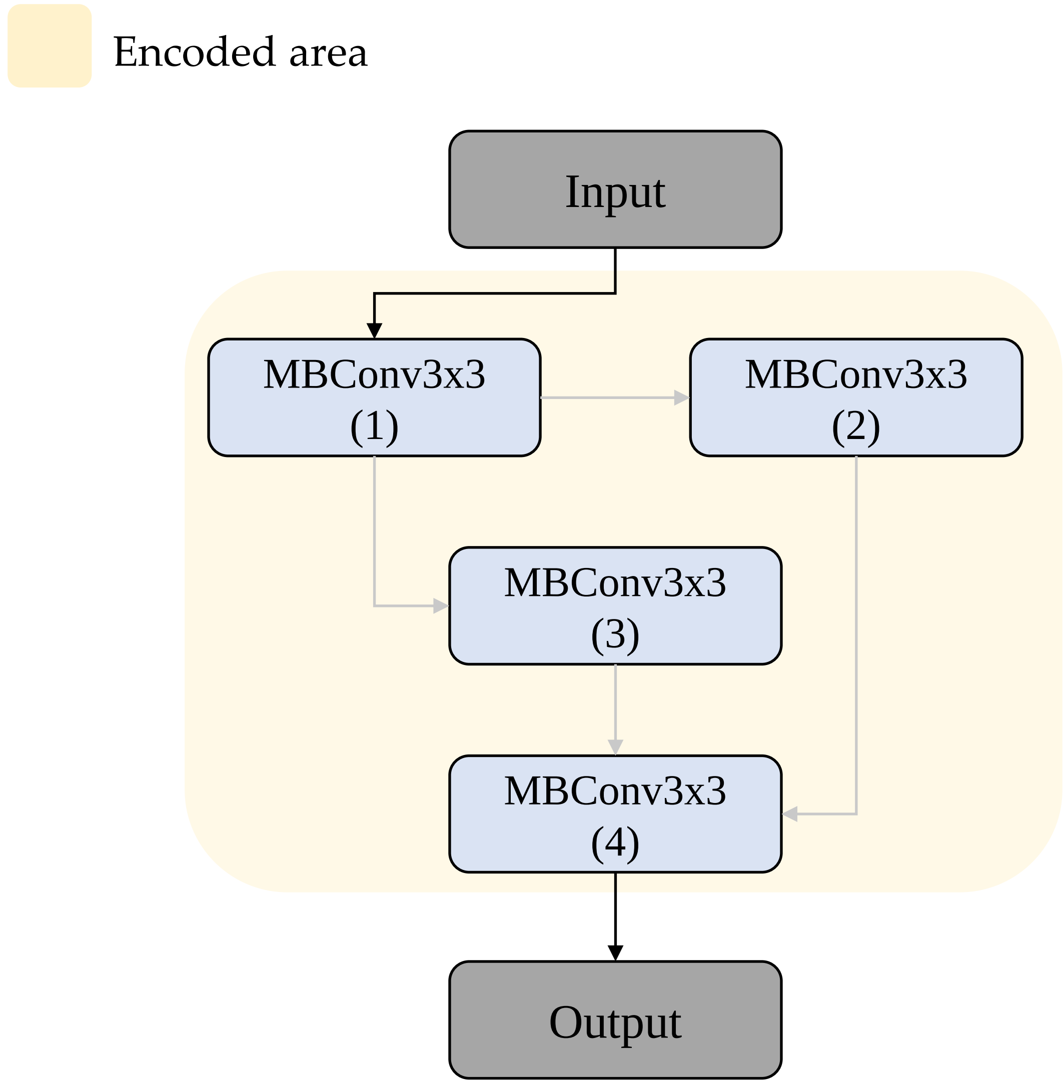


- Binary vector

$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$

$\rightarrow [1, 1, 0, 0, 1, 1]$

Case 1: Layers in the same type



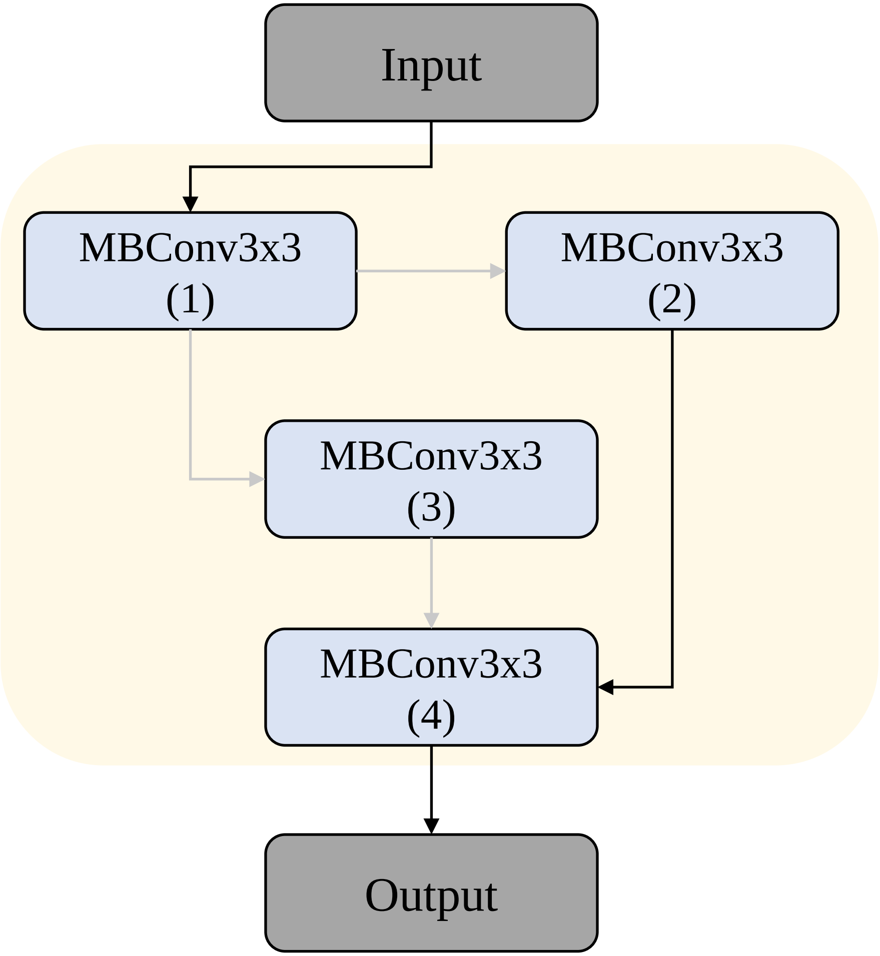
- Binary vector

$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$

$\rightarrow [1, 1, 0, 0, 1, 1]$

Case 1: Layers in the same type

Encoded area

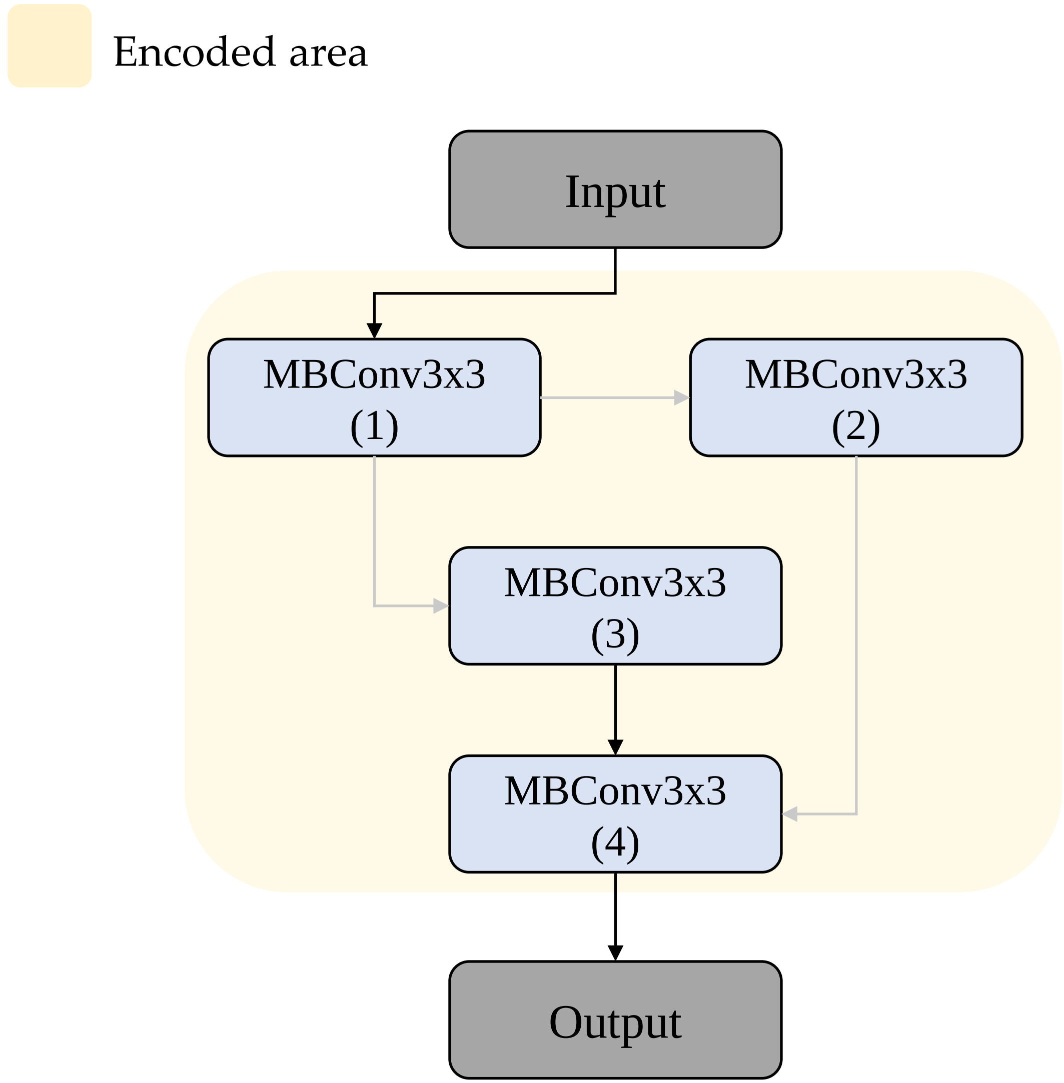


- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$\rightarrow [1, 1, 0, 0, 1, 1]$$

Case 1: Layers in the same type

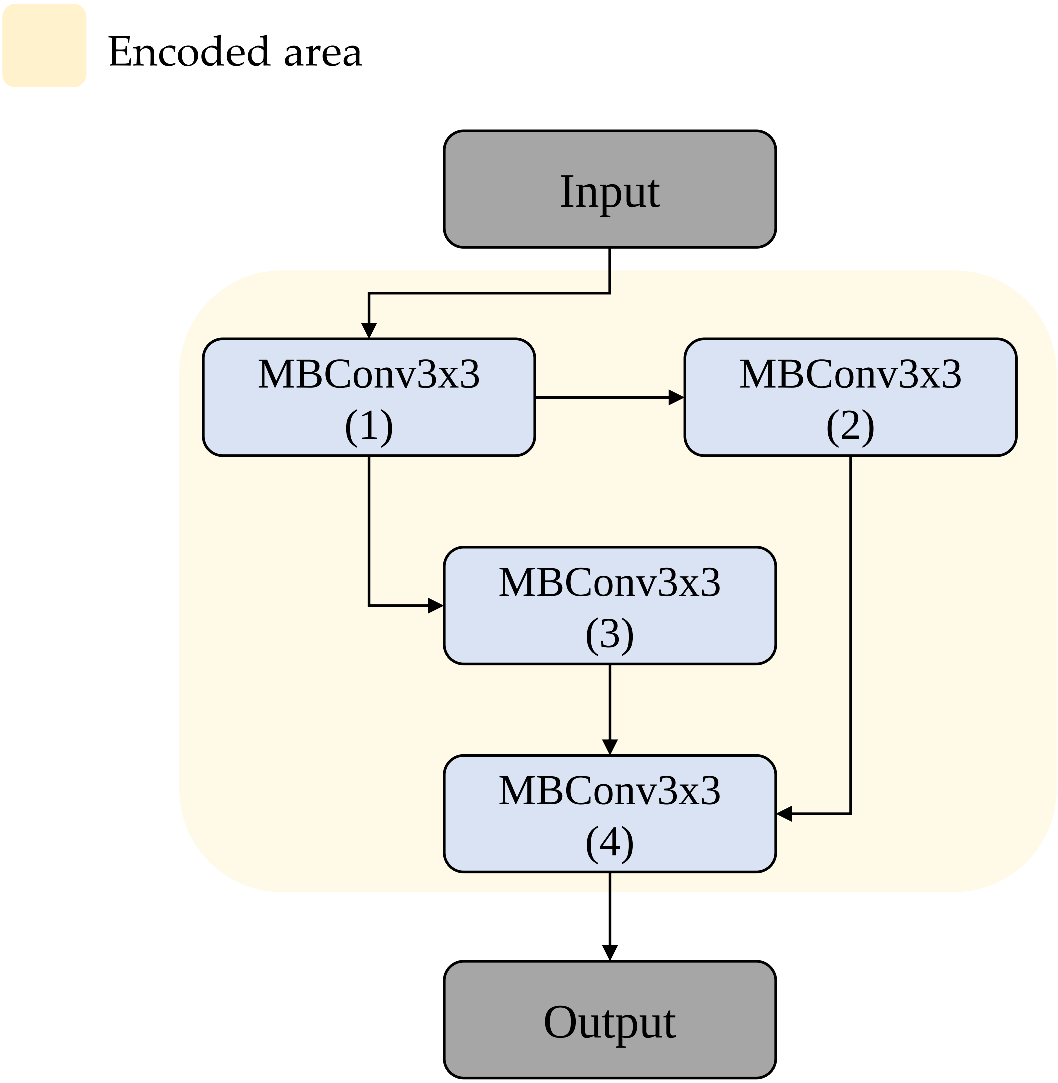


- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$\rightarrow [1, 1, 0, 0, 1, 1]$$

Case 1: Layers in the same type



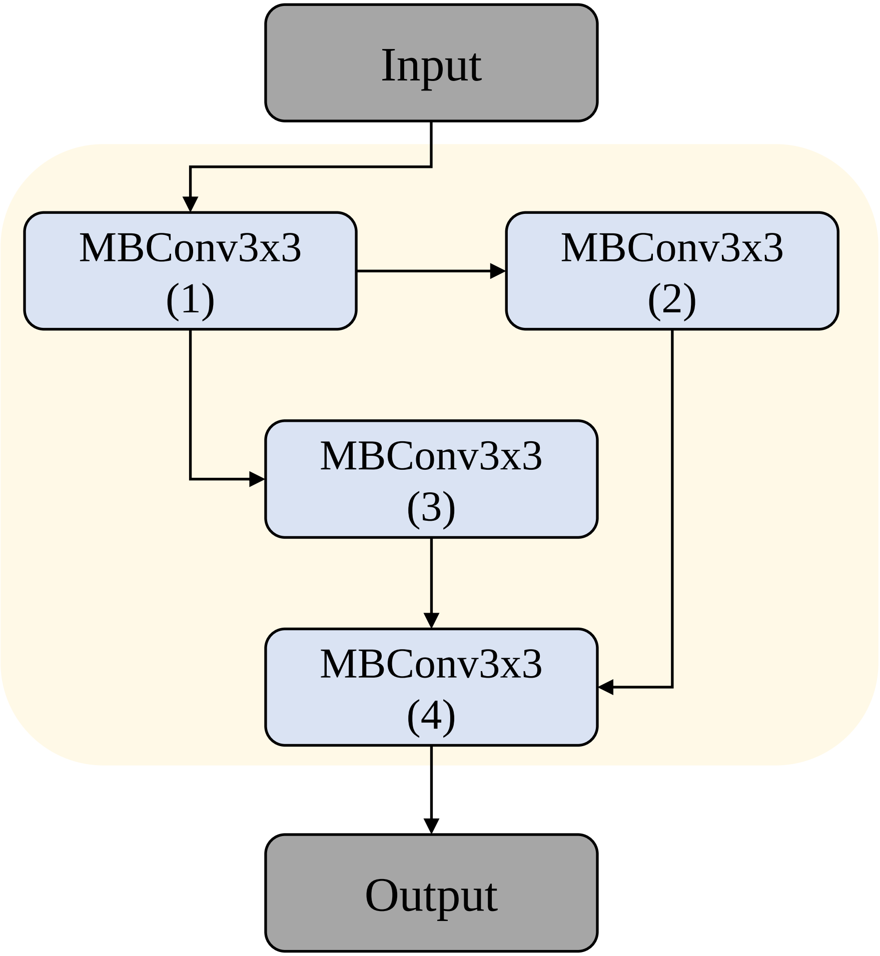
- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$\rightarrow [1, 1, 0, 0, 1, 1]$$

Case 1: Layers in the same type

Encoded area



- Binary vector

$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$\rightarrow [1, 1, 0, 0, 1, 1]$$

- Upper-triangular binary matrix

$$\begin{bmatrix} e_{11}, & e_{12}, & e_{13}, & e_{14} \\ 0, & e_{22}, & e_{23}, & e_{24} \\ 0, & 0, & e_{33}, & e_{34} \\ 0, & 0, & 0, & e_{44} \end{bmatrix} \Rightarrow \begin{bmatrix} 1, & 1, & 1, & 0 \\ 0, & 1, & 0, & 1 \\ 0, & 0, & 1, & 1 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

[Macro-level] Solution Encoding (cont.)

Chain structure

- Use 1 vector (e.g., string, integer numbers)

DAG

- **Case 1: Layers in the same type**

→ Represent the connections.

→ Use 1 binary vector (or upper-triangular binary matrix).

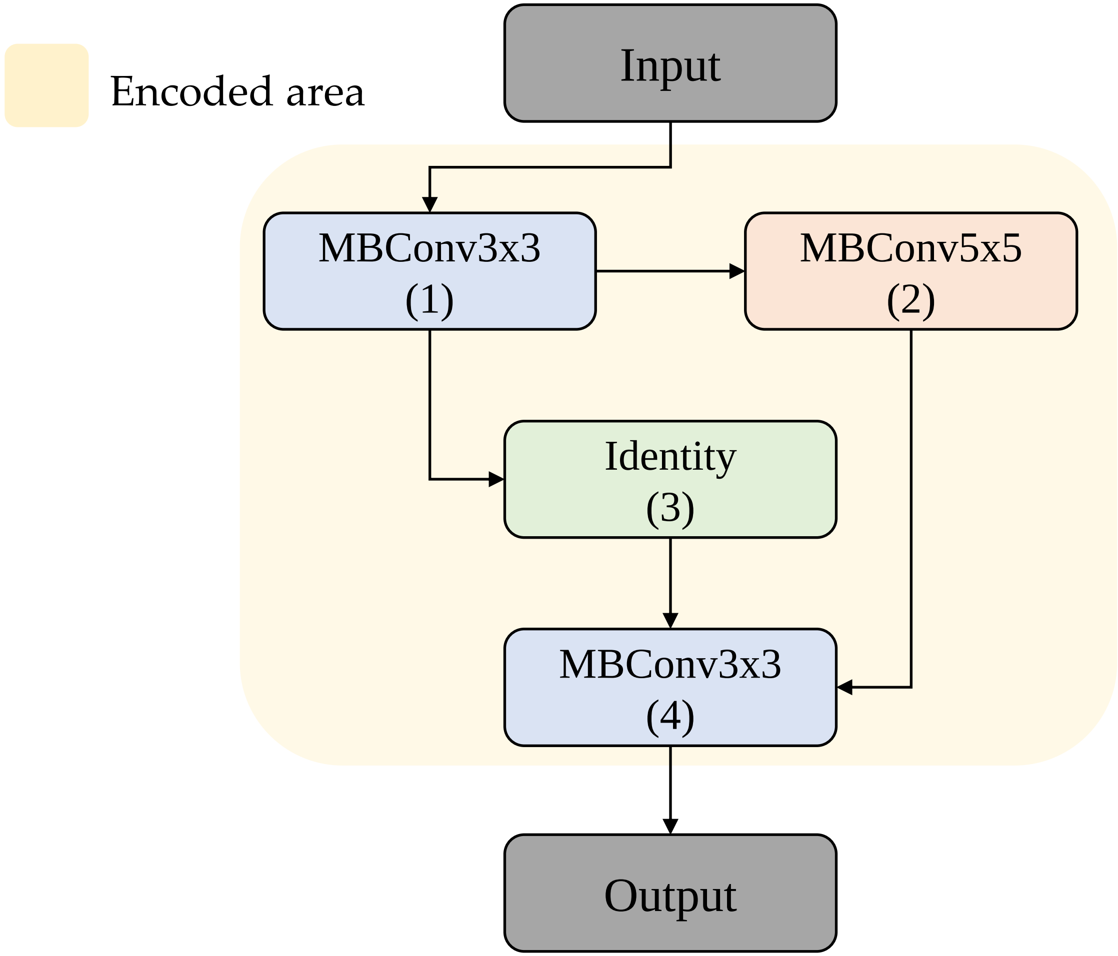
- **Case 2: Layers in the different type**

→ { Use 1 binary vector (or upper-triangular binary matrix) to represent the connections.
Use 1 vector (e.g., string, interger numbers) to represent the type of layers. }

Case 2: Layers in the different type

List of available layers

['MBConv3x3', 'MBConv5x5', 'Identity']



- Represent the connections (edges)

  $[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$

  → [1, 1, 0, 0, 1, 1]

- Represent the types of layers (nodes)

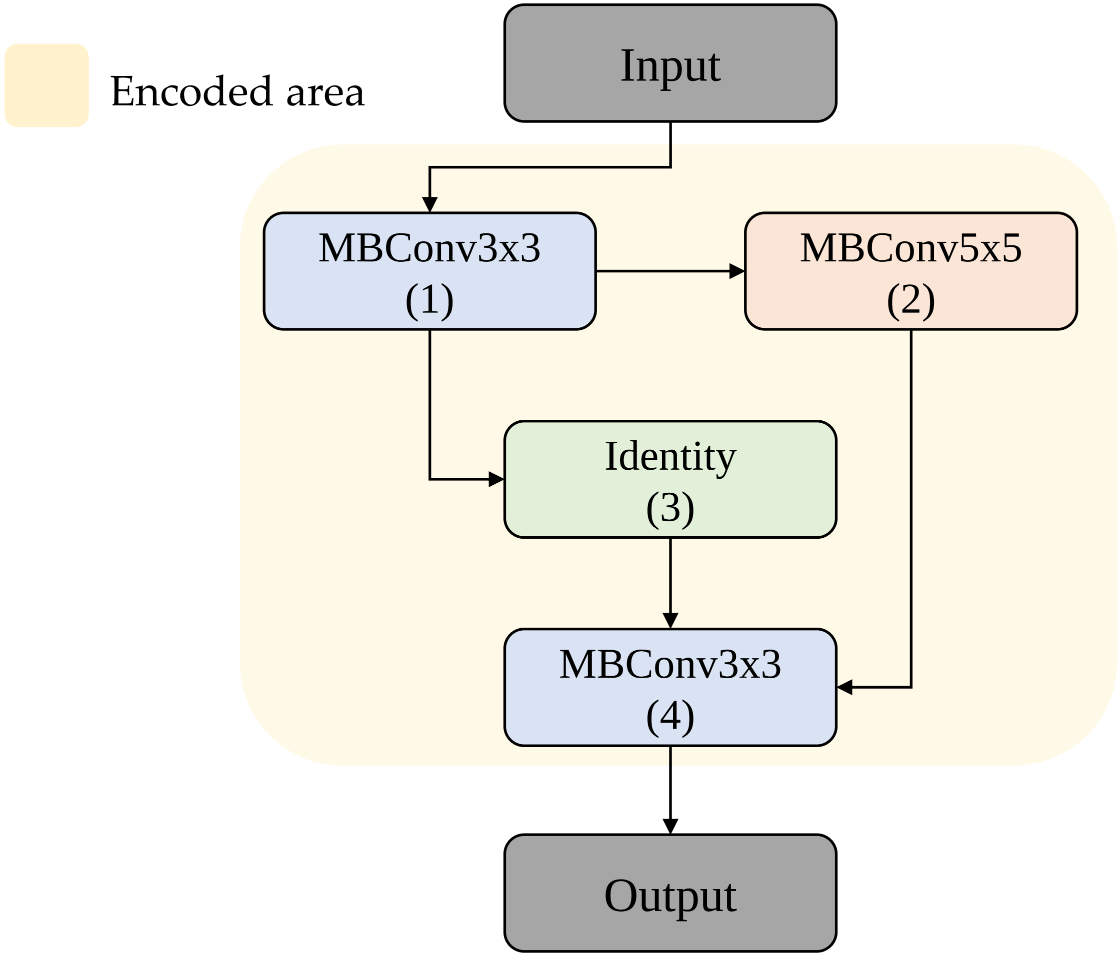
  $[n_1, n_2, n_3, n_4]$

  → [0, 1, 2, 0]

Case 2: Layers in the different type

List of available layers

['MBConv3x3', 'MBConv5x5', 'Identity']



- Represent the connections (edges)

  $[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$

  $\rightarrow$ [1, 1, 0, 0, 1, 1]

- Represent the types of layers (nodes)

  $[n_1, n_2, n_3, n_4]$

  $\rightarrow$ [0, 1, 2, 0]

  $\rightarrow$ [1, 1, 0, 0, 1, 1, 0, 1, 2, 0]

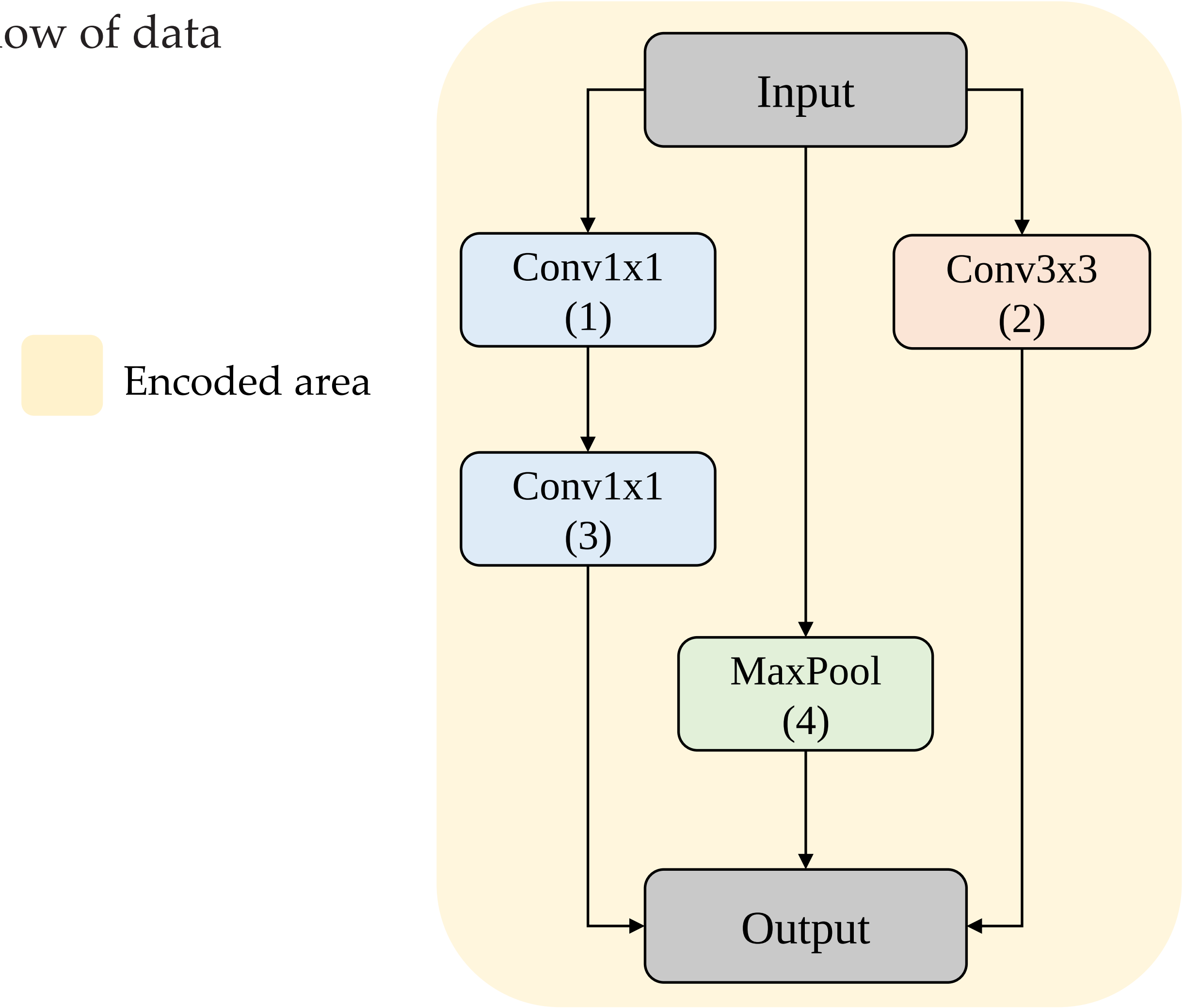
Micro-level

- Components for encoding → Operations
- A solution in search space → An arrangement of operations in the cell of an architecture.
- Operations are usually in <u>different</u> types.
- Operations are usually arranged as <u>a DAG</u>.
- There are two commonly-used types of DAGs:
  - Normal DAGs
  - Fully connected DAGs

Normal DAGs

Nodes → operations

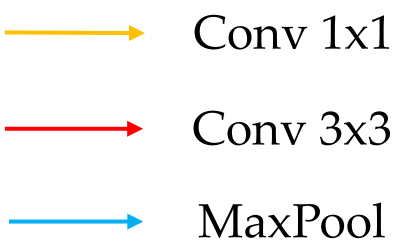
Edges → the flow of data

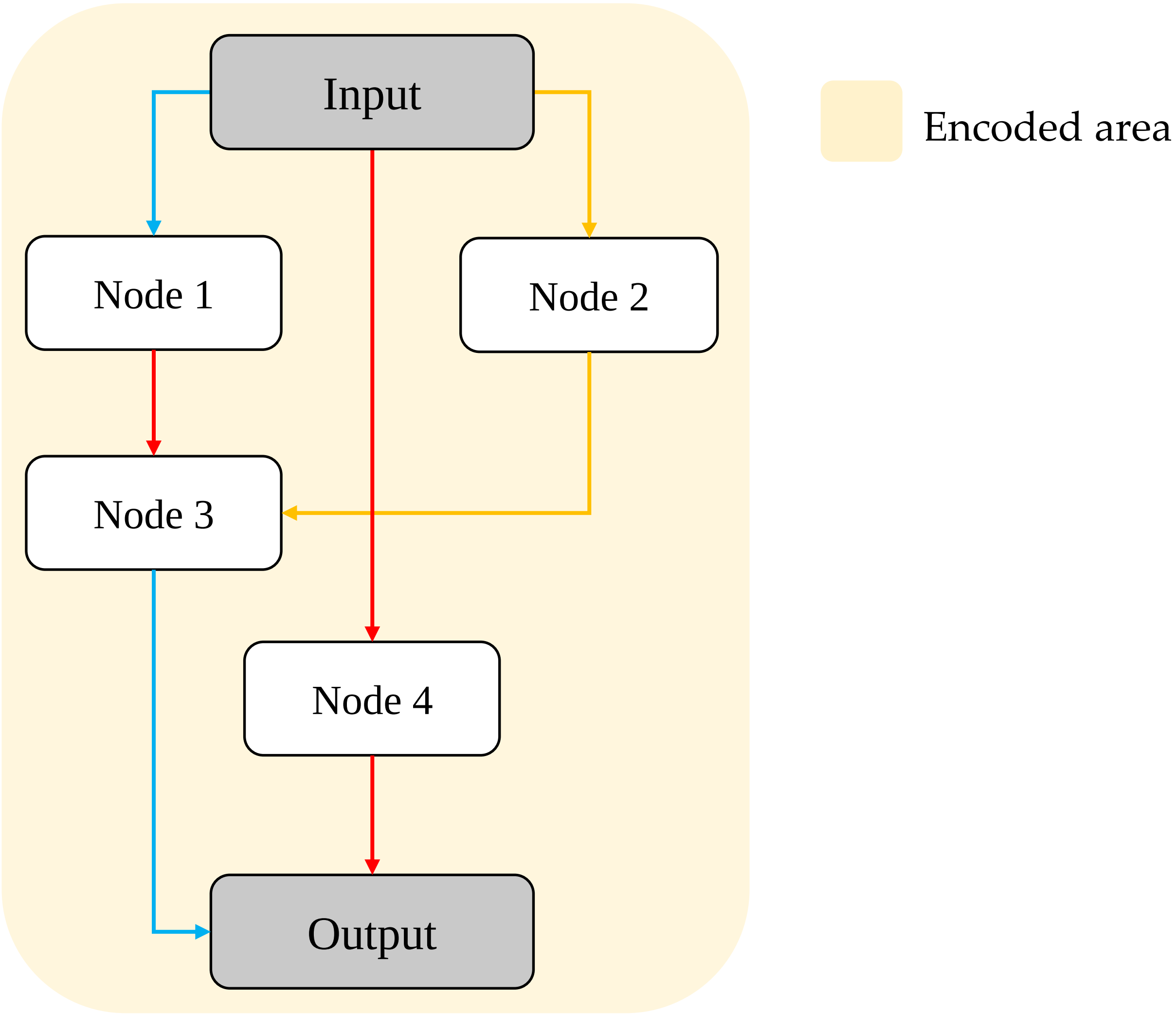


Normal DAGs

Nodes → the place to aggregate the output (data) from previous nodes

Edges → operations

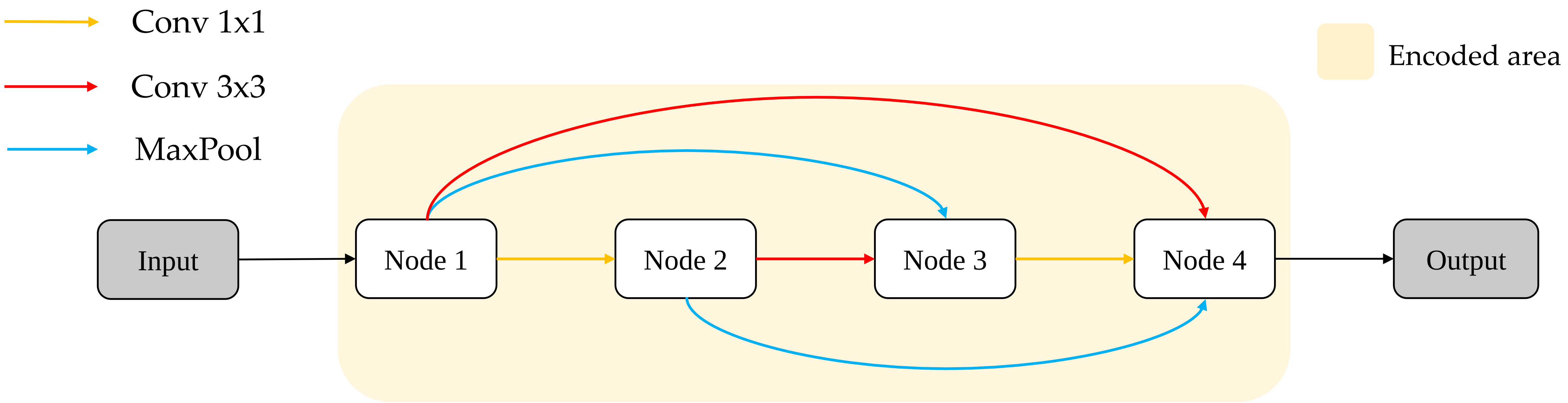




Fully connected DAGs

Nodes → the place to aggregate the output (data) from previous nodes

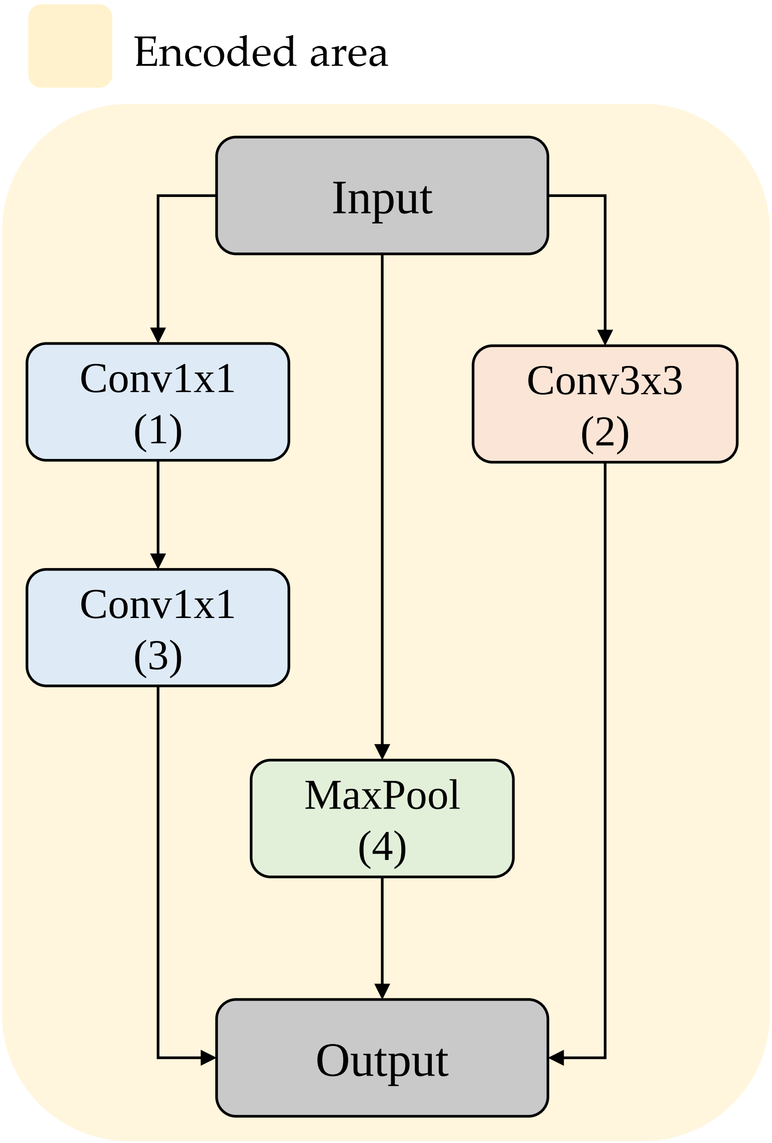
Edges → operations



[Micro-level] Solution Encoding

Normal DAG (Nodes → operations, edges → the flow of data)

- Use the same encoding mechanism in macro-level search space.



List of available operations = [‘Conv1x1’, ‘Conv3x3’, ‘MaxPool’]

- Represent the connections (edges)

$[e_{I1}, e_{I2}, e_{12}, e_{I3}, e_{13}, e_{23}, e_{I4}, e_{14}, e_{24}, e_{34}, e_{IO}, e_{1O}, e_{2O}, e_{3O}, e_{4O}]$

$\rightarrow [1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]$

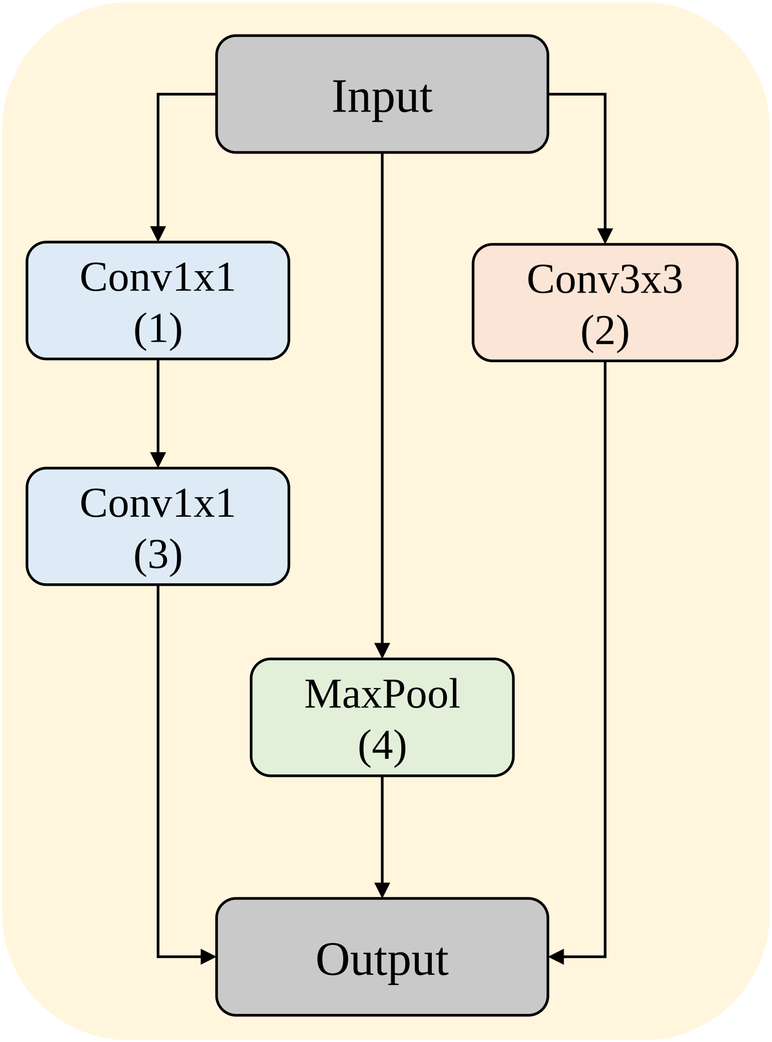
- Represent the type of operations (nodes)

[‘INPUT‘, Conv1x1’, ‘Conv3x3’, ‘Conv1x1’, ‘MaxPool’, ‘OUTPUT’]

$\rightarrow [-1, 0, 1, 0, 2, -1]$

$\rightarrow [0, 1, 0, 2]$

Encoded area



List of available operations = [‘Conv1x1’, ‘Conv3x3’, ‘MaxPool’]

- Represent the connections (edges)

[e_{I1}, e_{I2}, e_{12}, e_{I3}, e_{13}, e_{23}, e_{I4}, e_{14}, e_{24}, e_{34}, e_{IO}, e_{1O}, e_{2O}, e_{3O} , e_{4O}]

→ [1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1]

- Represent the type of operations (nodes)

[‘INPUT‘, Conv1x1’, ‘Conv3x3’, ‘Conv1x1’, ‘MaxPool’, ‘OUTPUT’]

→ [-1, 0, 1, 0, 2, -1]

→ [0, 1, 0, 2]

[1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 2]

[Micro-level] Solution Encoding (cont.)

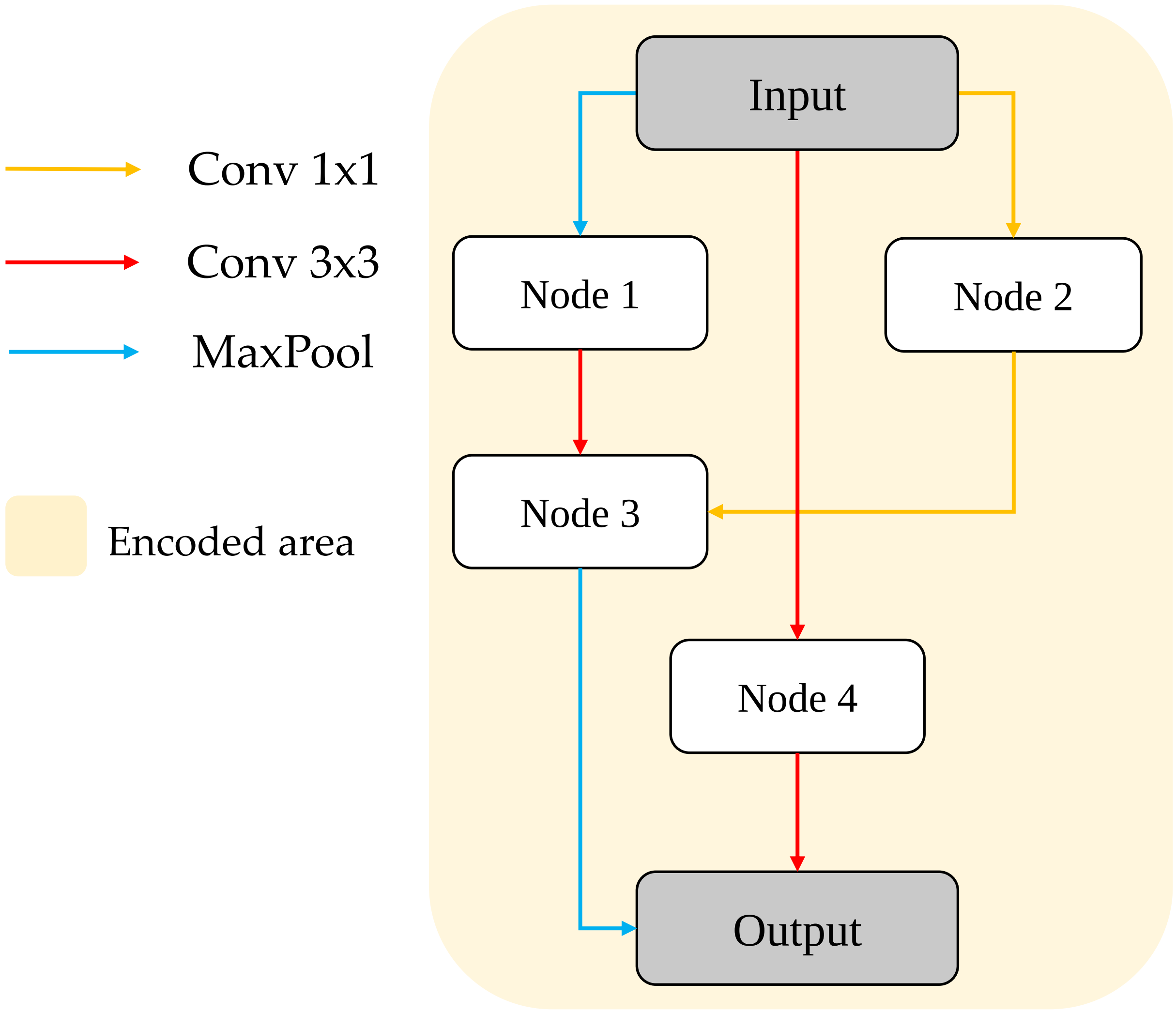
Normal DAG (Nodes → operations, edges → the flow of data)

- Use the same encoding mechanism in macro-level search space.

Normal DAG (Nodes → the place to aggregate the output (data) from previous nodes, edges → operations)

- Represent both the type of operators and connections

  → Use 1 vector of interger numbers



- 0: Conv1x1
- 1: Conv3x3
- 2: MaxPool
- -1: No connection

$$[e_{I1}, e_{I2}, e_{12}, e_{I3}, e_{13}, e_{23}, e_{I4}, e_{14}, e_{24}, e_{34}, e_{IO}, e_{1O}, e_{2O}, e_{3O}, e_{4O}]$$

$$\Downarrow$$

$$[2,\ 0,\ -1,\ -1,\ 1,\ 0,\ 1,\ -1,\ -1,\ -1,\ -1,\ -1,\ -1,\ 2,\ 1]$$

[Micro-level] Solution Encoding (cont.)

Normal DAG (Nodes → operations, edges → the flow of data)

- Use the same encoding mechanism in macro-level search space.

Normal DAG (Nodes → the place to aggregate the output (data) from previous nodes, edges → operations)

- Represent both the type of operators and connections

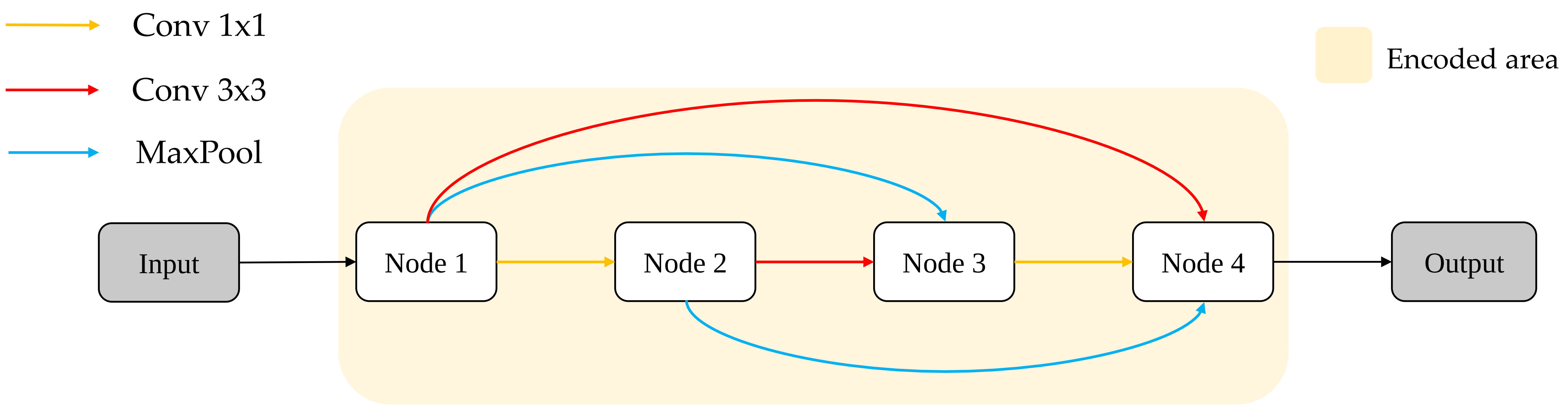
  → Use 1 vector of interger numbers

Fully connected DAG (Nodes → the place to aggregate the output (data) from previous nodes, edges → operations)

- Represent the type of operators.

  → Use 1 vector of interger numbers

- 0: Conv1x1
- 1: Conv3x3
- 2: MaxPool



$$[e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}]$$

$$[0, \quad 2, \quad 1, \quad 1, \quad 2, \quad 0]$$

Search Space

- In fact, each solution in the search space is not the entire architecture. It is just a part of the architecture.
- More specifically, each solution is the arrangement of some components in the architecture (e.g., layers; operations in cells).
- Based on the components which are used to arrange, there are 2 types of search space:
  - Macro-level
  - Micro-level
- Some common search space: NASNet (Zopl et al., 2018); DARTS (Liu et al., 2019), NAS-Bench-101, NAS-Bench-201.

Table of Contents

Approach NAS with Evolutionary Algorithms (EAs)

Fomulate NAS as an optimization problem:

- n-objectives problem:
  - Single-Objective Problem (SOP):

    Performance metrics (accuracy, error)

  - Multi-Objective Problem (MOP):

    Performance metrics + Computational metrics (#GPUs, #params).

- Ideal solution:

  A list of architectures (i.e., arrangement of architecture components) that maximize/ minimize the objective values.

Example of desirable results in solving MONAS

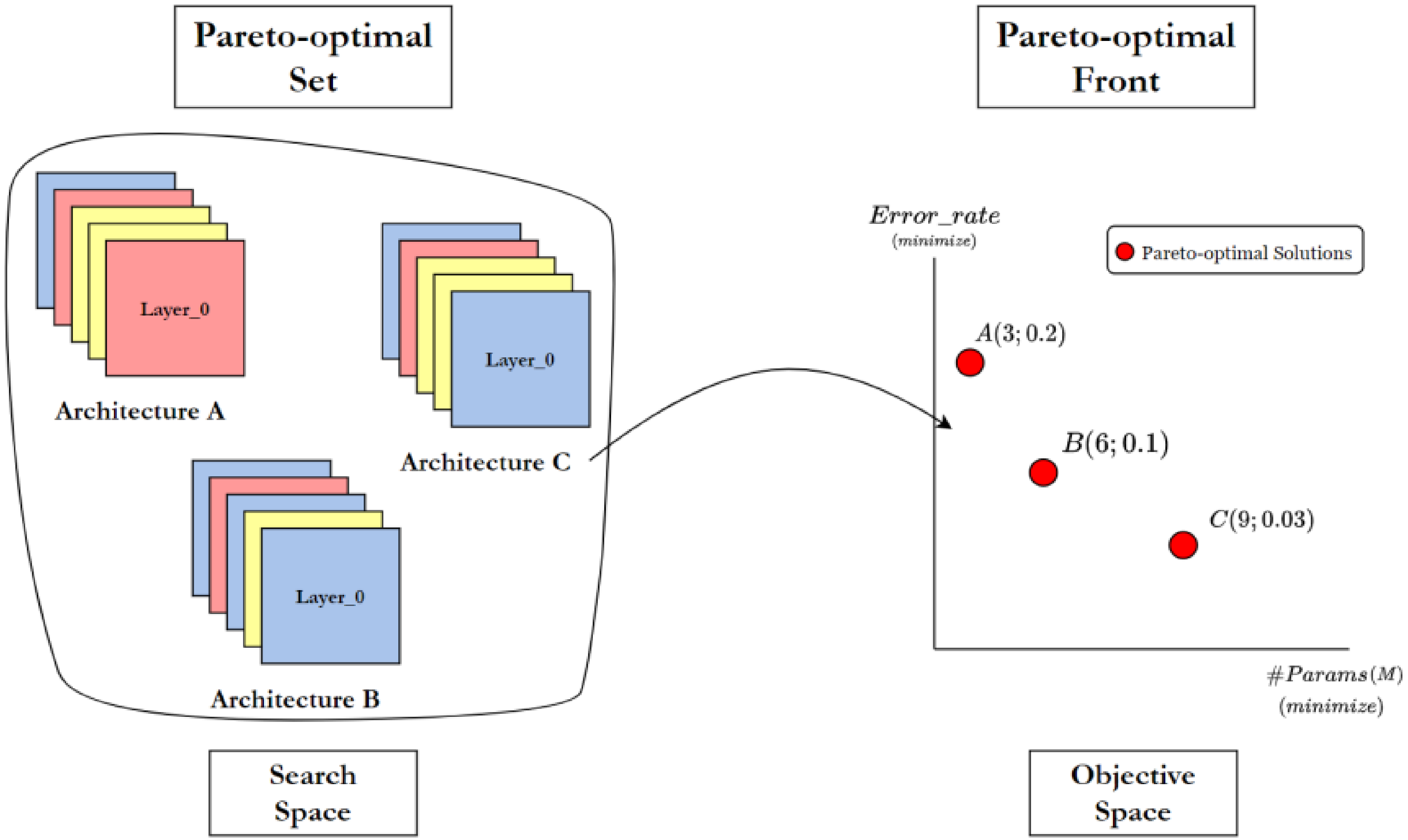


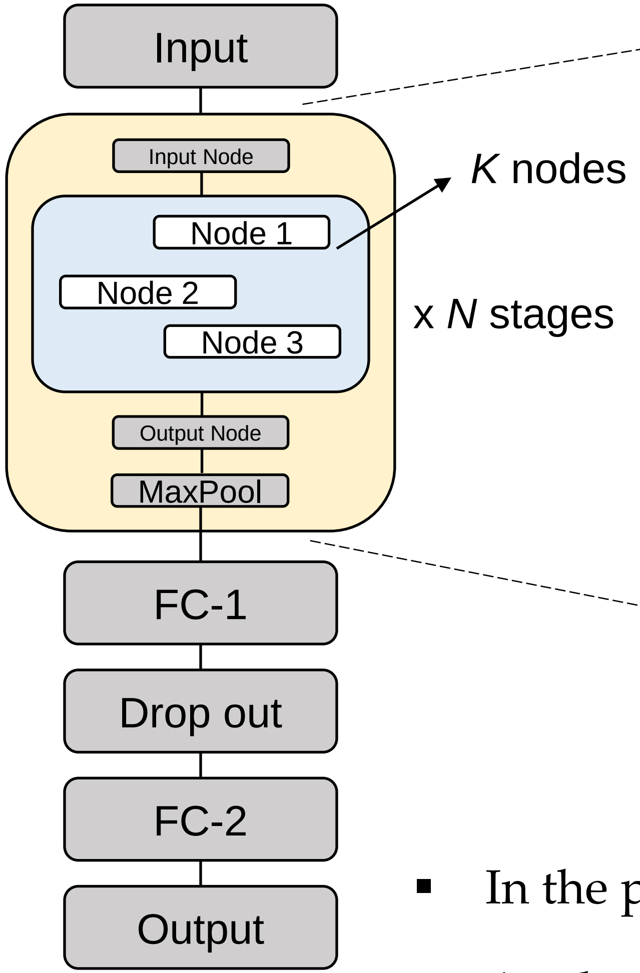
Table of Contents

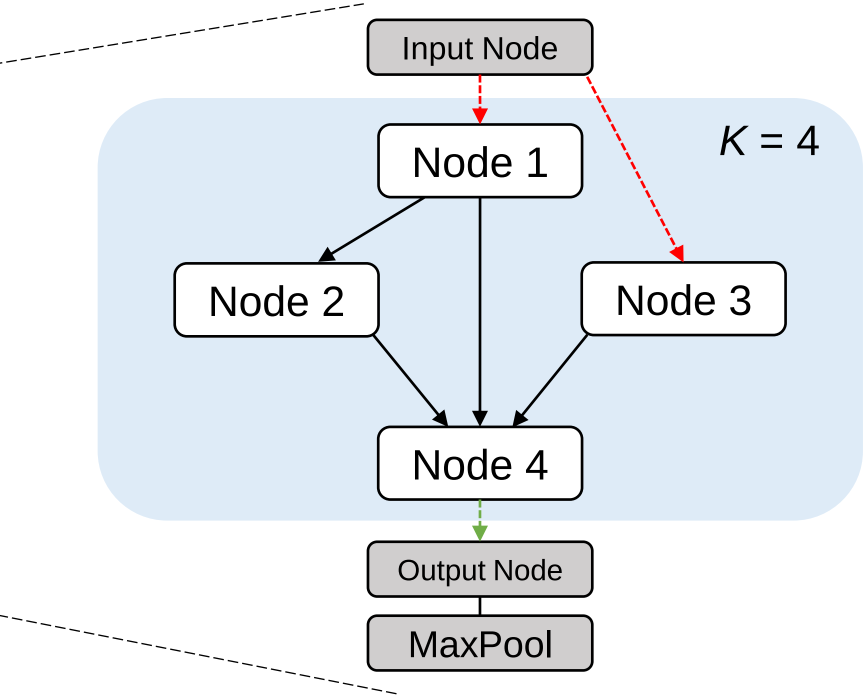
GeneticCNN

Some informations:

- Macro-level
- Single-objective
- Search strategy: Genetic Algorithm
- Fitness value: Validation accuracy
- Performance estimation strategy: Full training and evaluating on CIFAR-10 dataset.
- Transfer architectures found to another large-scale dataset (ILSVRC2012).

[GeneticCNN] Search Space





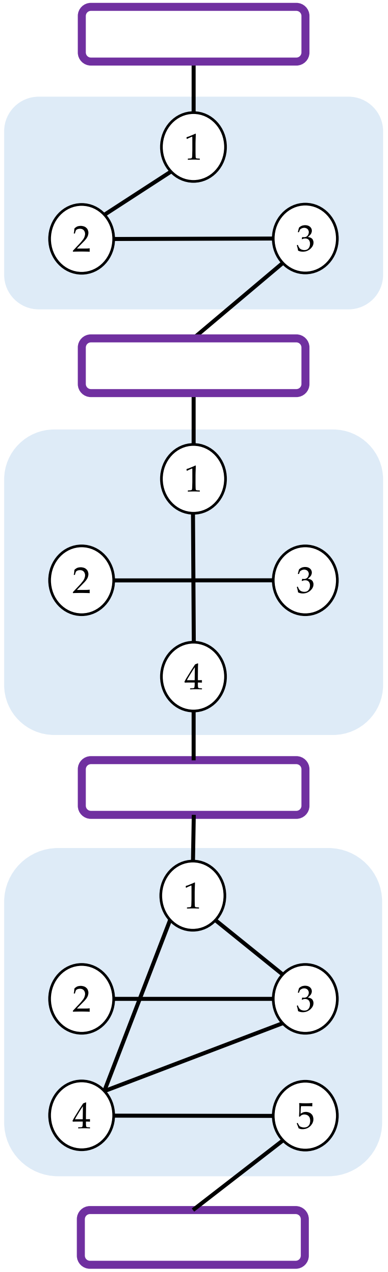
- A complete architecture includes a set of N stages S = {S_1; $S_{2;}$ …; S_N} with stage S_i has K_i nodes respectively.
- Each node (include ‘Input’ and ‘Output’ node) contains: Convolution operator, ReLU and BN. Each edge represents the flow of data.

- In the paper: $N = 3$ and $(K_1; K_2; K_3) = (3; 4; 5)$
- A solution is an arrangement of 12 nodes at 3 stages.

Represent solution in EA's search space

- Each solution is a binary vector has length of 19.
    - 0: no connection
    - 1: have connection
- The size of search space: 524,288 (= 2^{19})

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

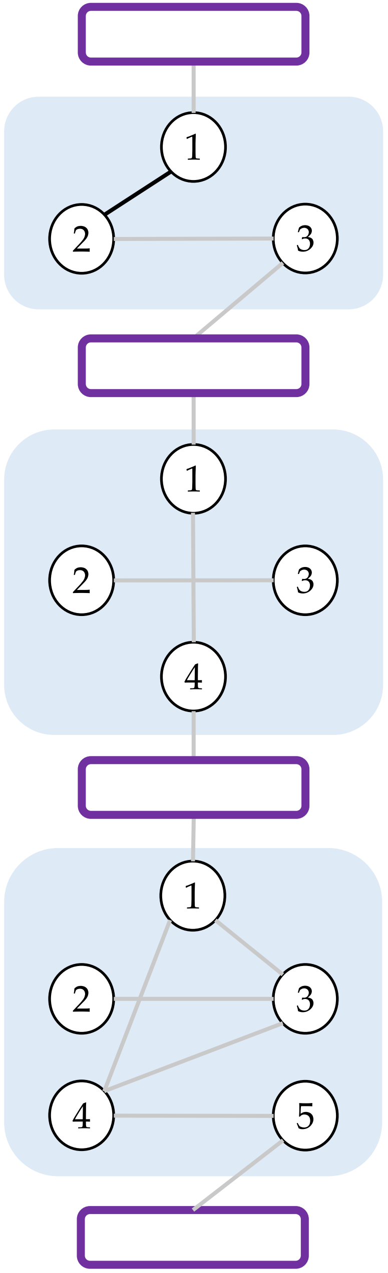


1, 0, 1

0, 0, 1, 1, 0, 0

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

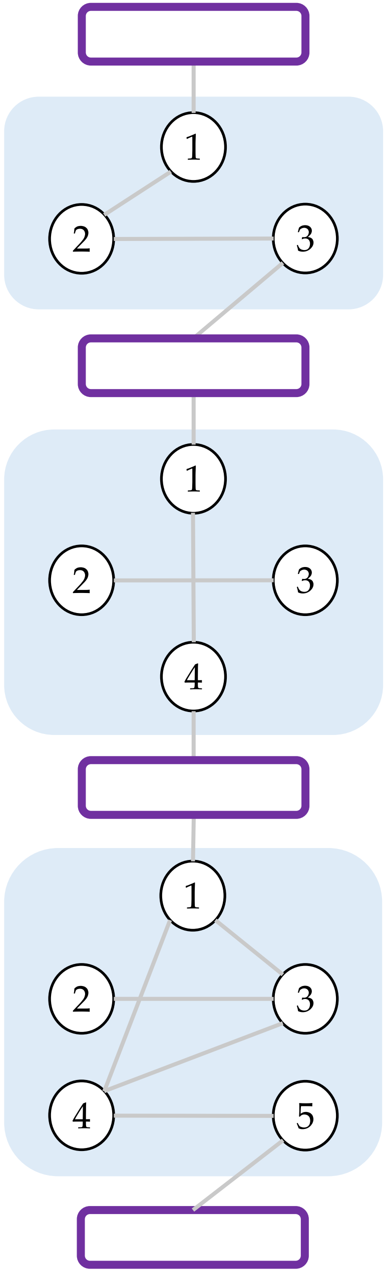


$\mathbf{1}, 0, 1$

$\mathbf{e_{12}}, e_{13}, e_{23}$

0, 0, 1, 1, 0, 0

0, 1, 1, 1, 0, 1, 0, 0, 0, 1



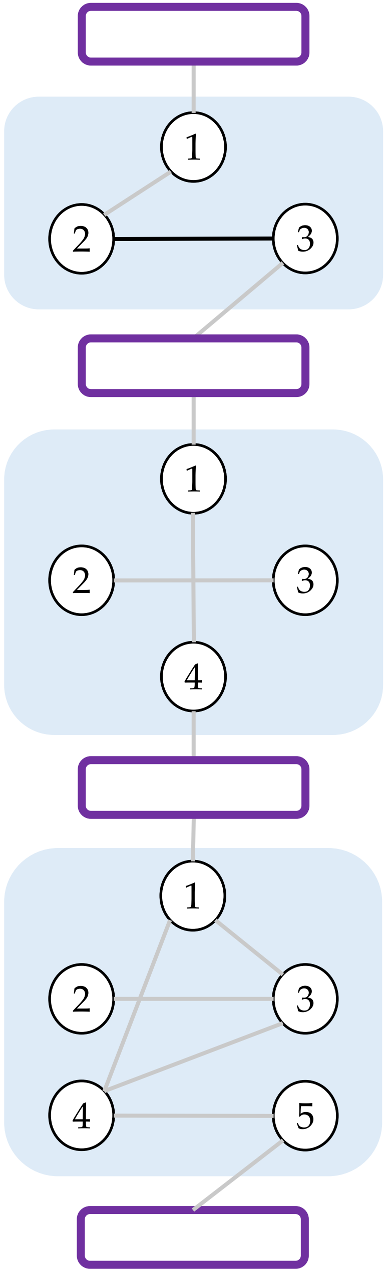
X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

1, **0**, 1

e_{12}, $\mathbf{e_{13}}$, e_{23}

0, 0, 1, 1, 0, 0

0, 1, 1, 1, 0, 1, 0, 0, 0, 1



$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

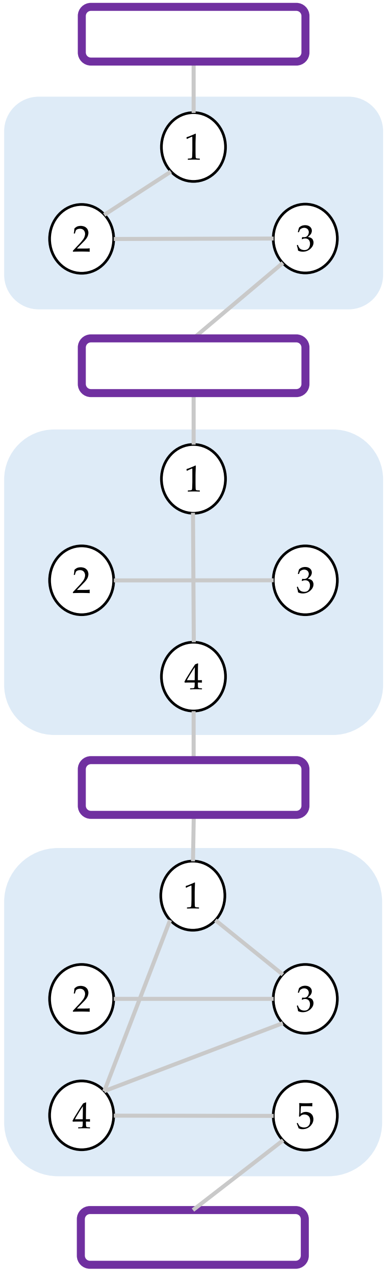
1, 0, **1**

$e_{12}, e_{13}, \mathbf{e_{23}}$

0, 0, 1, 1, 0, 0

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]



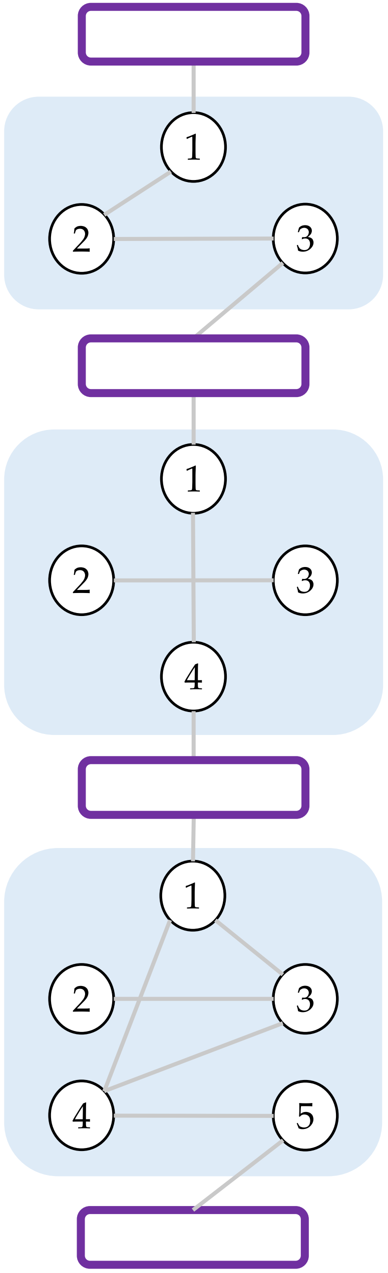
1, 0, 1

0, 0, 1, 1, 0, 0

$\mathbf{e_{12}}$, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$



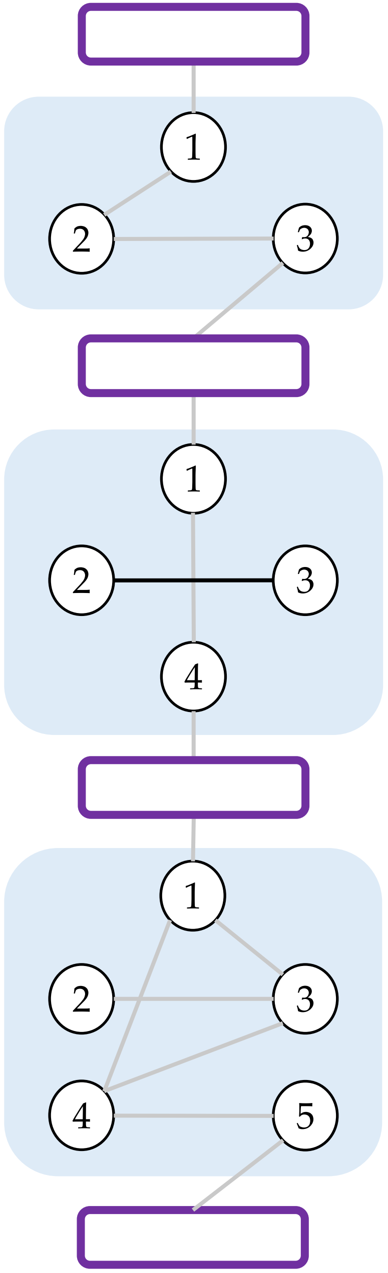
1, 0, 1

0, **0**, 1, 1, 0, 0

e_{12}, $\mathbf{e_{13}}$, e_{23}, e_{14}, e_{24}, e_{34}

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$



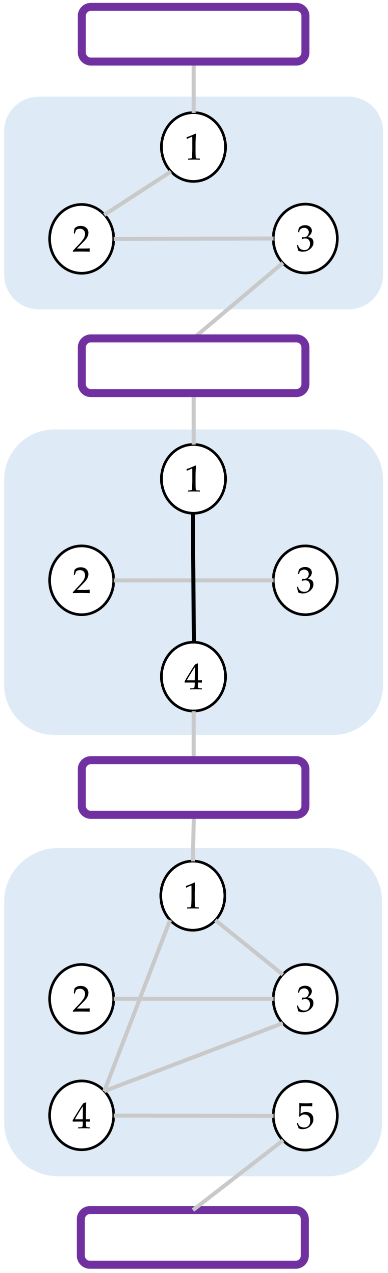
1, 0, 1

0, 0, **1**, 1, 0, 0

$e_{12}, e_{13}, \mathbf{e_{23}}, e_{14}, e_{24}, e_{34}$

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$



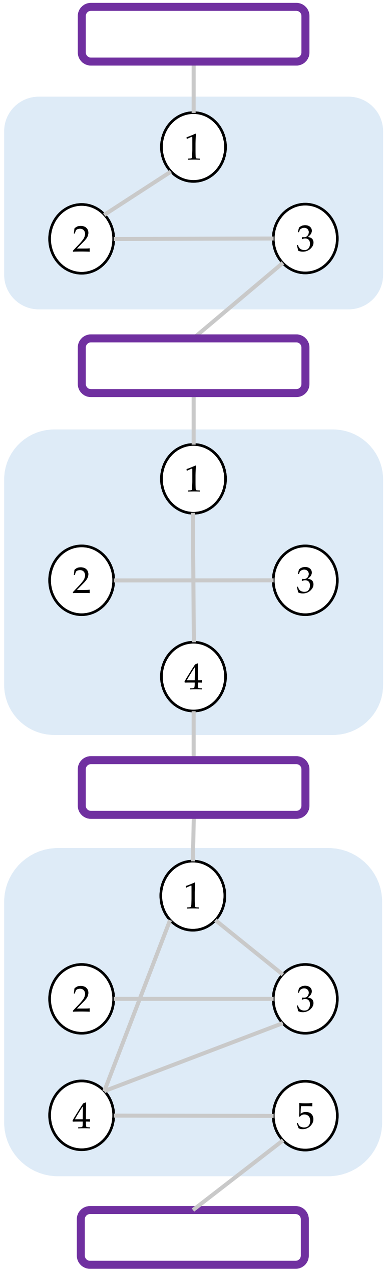
1, 0, 1

0, 0, 1, **1**, 0, 0

$e_{12}, e_{13}, e_{23}, \mathbf{e_{14}}, e_{24}, e_{34}$

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$



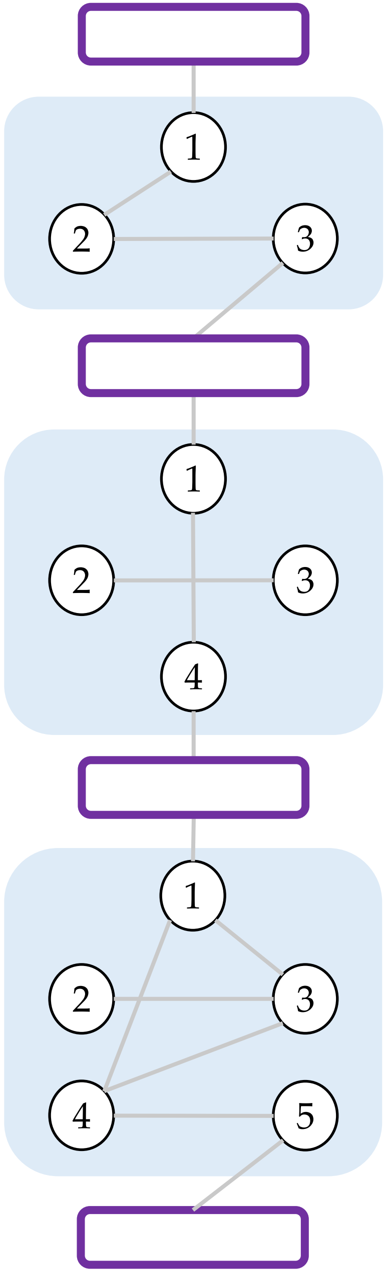
1, 0, 1

0, 0, 1, 1, **0**, 0

e_{12}, e_{13}, e_{23}, e_{14}, $\mathbf{e_{24}}$, e_{34}

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

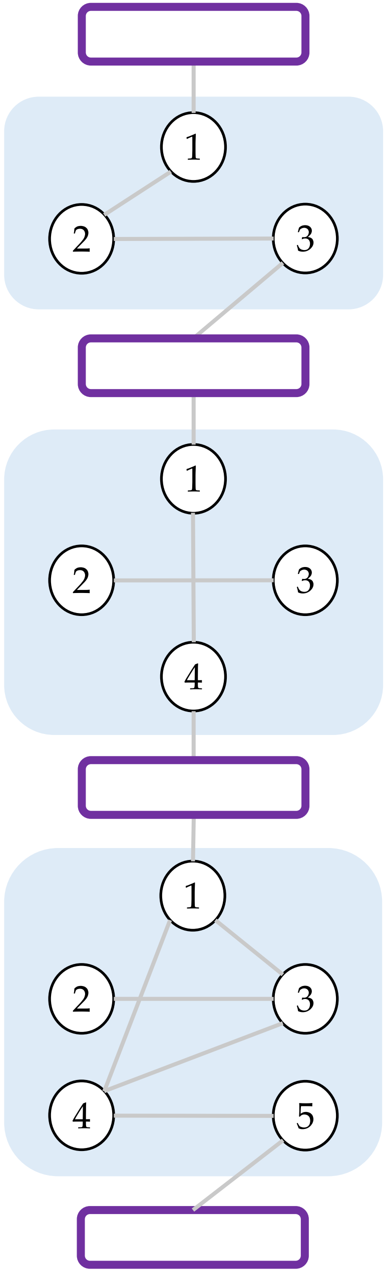


1, 0, 1

0, 0, 1, 1, 0, **0**

$e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, \mathbf{e_{34}}$

0, 1, 1, 1, 0, 1, 0, 0, 0, 1



$X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

1, 0, 1

0, 0, 1, 1, 0, 0

$e_{12}, e_{13}, e_{23}, e_{14}, e_{24}, e_{34}$

0, 1, 1, 1, 0, 1, 0, 0, 0, 1

$e_{12}, e_{13}, e_{23}, e_{14}, e_{24},$
$e_{34}, e_{15}, e_{25}, e_{35}, e_{45}$

Crossover (approach 1)

$X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

$X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]$

Crossover (approach 1)

X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

⬇

X_1 = [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1]

Crossover (approach 2)

$X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]$

$X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]$

Crossover (approach 2)

X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

↓

X_1 = [1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

X_2 = [1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

Crossover (approach 3)

X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

Crossover (approach 3)

X_1 = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

↓

X_1 = [1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

X_2 = [1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1]

Mutation

- Idea: Change the value of the current element to the new value.

X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

Mutation

- Idea: Change the value of the current element to the new value.

X = [1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1]

↓

X = [1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1]

Table of Contents

NSGA-Net

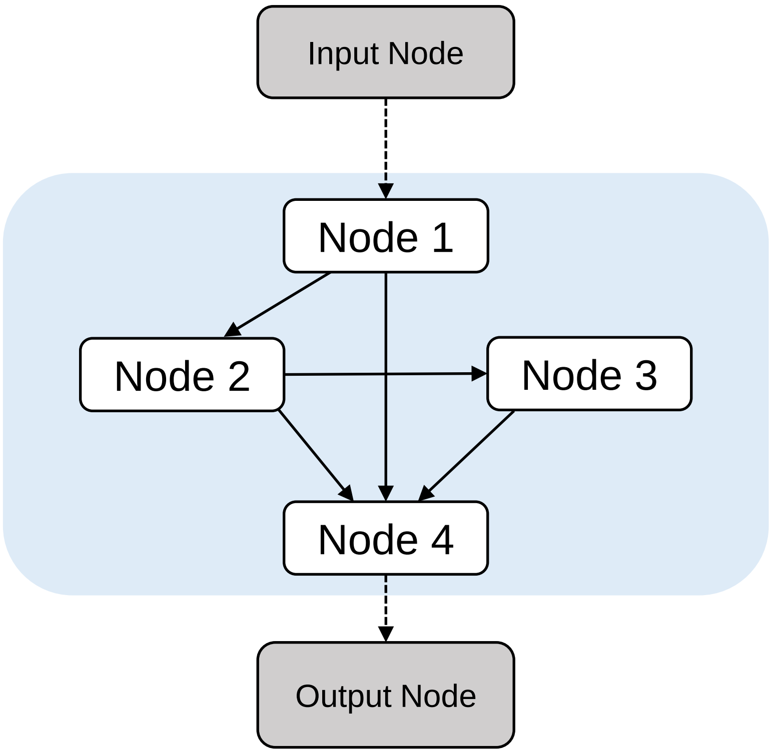
Some informations:

- Macro-level & Micro-level
- Multi-objective
- Search Strategy: NSGA-II
- Fitness values = {FLOPs; Classification error (validation)}
- Performance Estimation Strategy: Partial training (training in few epochs) and evaluating on validation set (CIFAR-10).
- Transfer architectures found to another large-scale dataset (ImageNet).

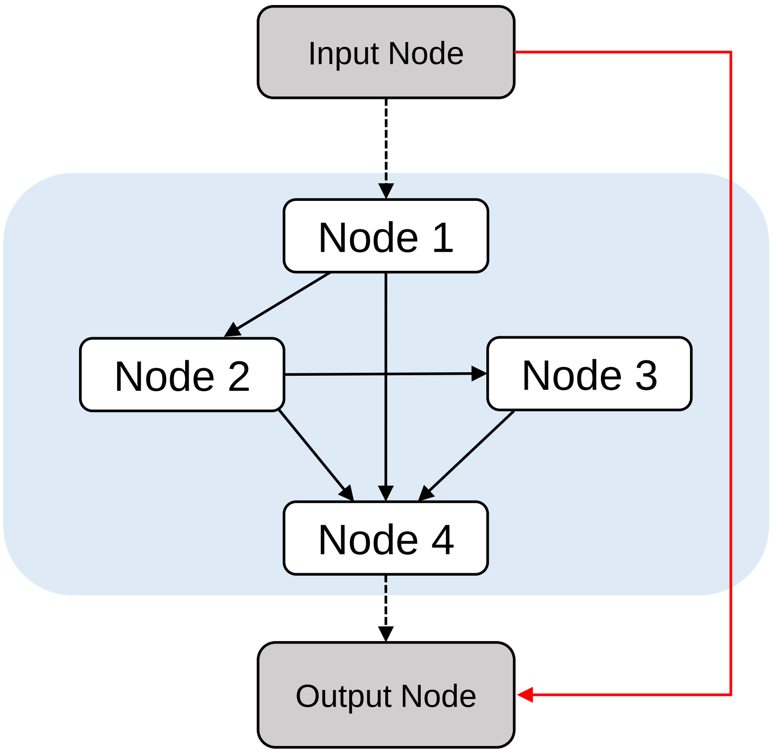
[NSGA-Net] Search Space (Macro-level)

- Use the same search space in GeneticCNN paper.
- Set $N = 3$ stages and $(K_1; K_2; K_3) = (6; 6; 6)$
- The difference between NSGA-Net search space and GeneticCNN search space is that at each stage of solution in NSGA-Net search space, there may be have a connection between Input Node and Output Node.

Genetic CNN



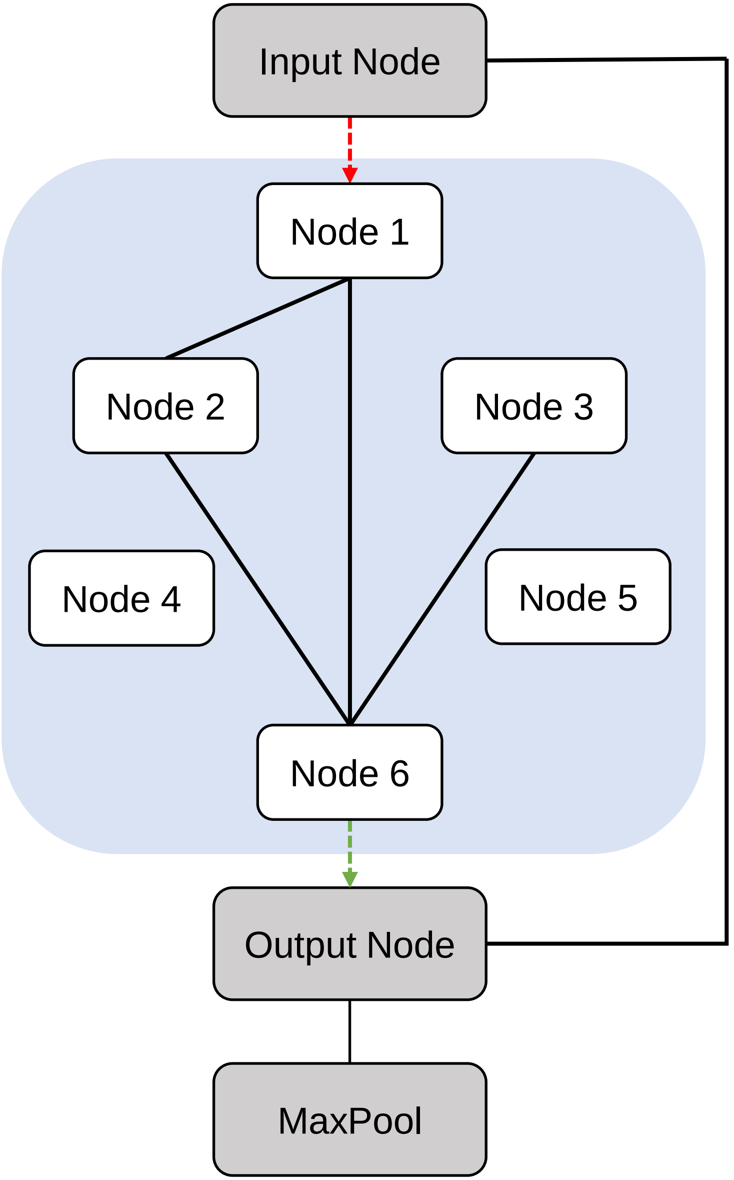
NSGA-Net



The difference between two search space

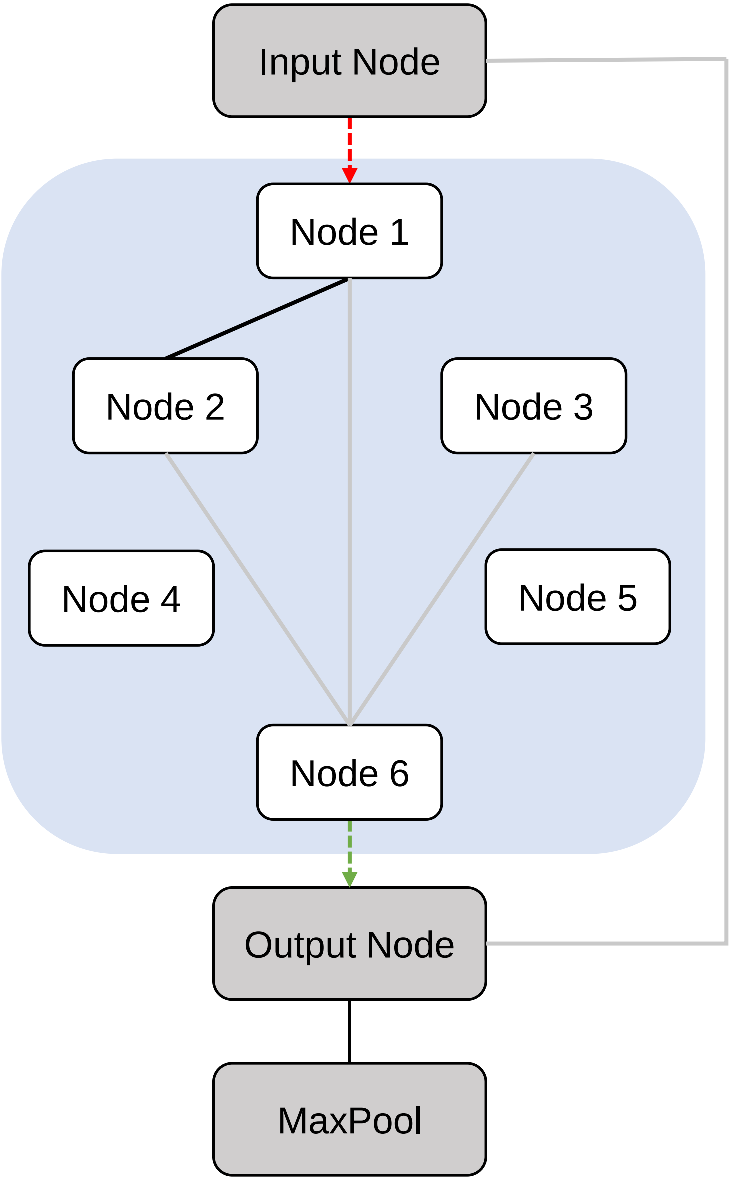
Represent solution in EA's search space (Macro-level)

- Use the same mechanism in GeneticCNN paper.
- In each stage, adding a bit to represent the connection between Input Node and Output Node.
- The length of each solution: 48
- The size of search space: 2^{48}



[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1]

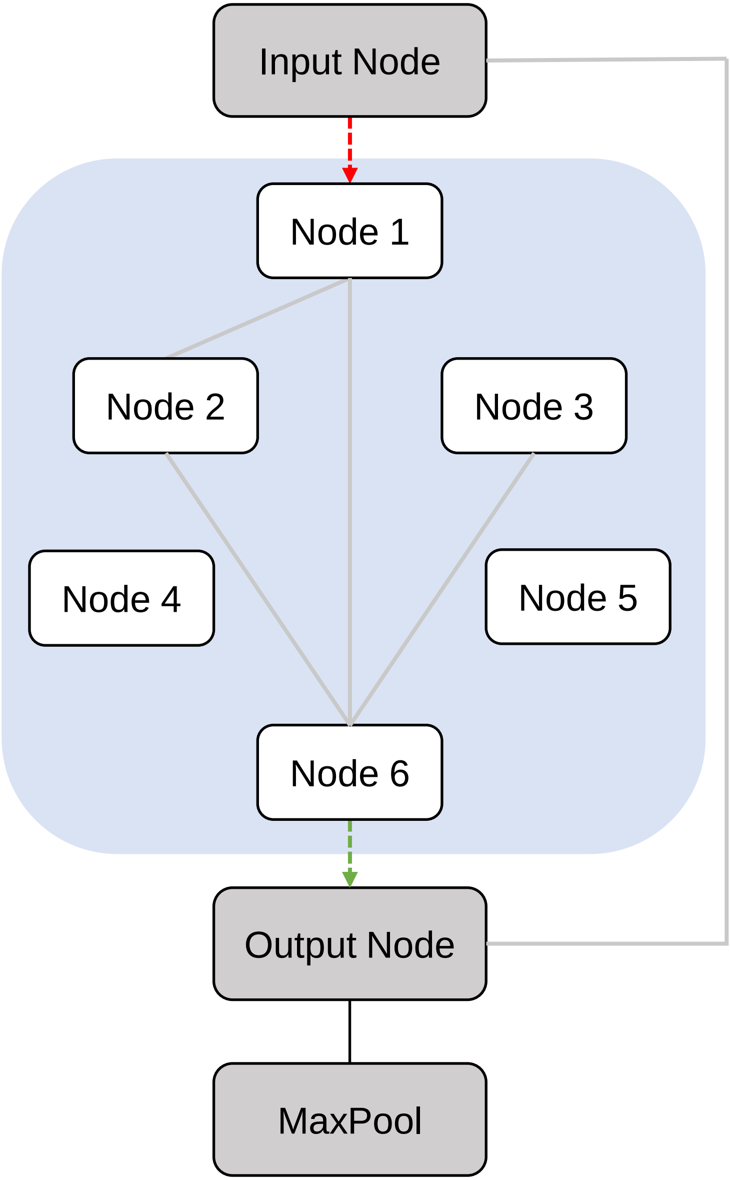
Example of encoding one stage in NSGA-Net



$$[\mathbf{1}, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1]$$

$$\mathbf{e_{12}},$$
$$e_{13}, e_{23},$$
$$e_{14}, e_{24}, e_{34},$$
$$e_{15}, e_{25}, e_{35}, e_{45},$$
$$e_{16}, e_{26}, e_{36}, e_{46}, e_{56},$$
$$e_{IO}$$

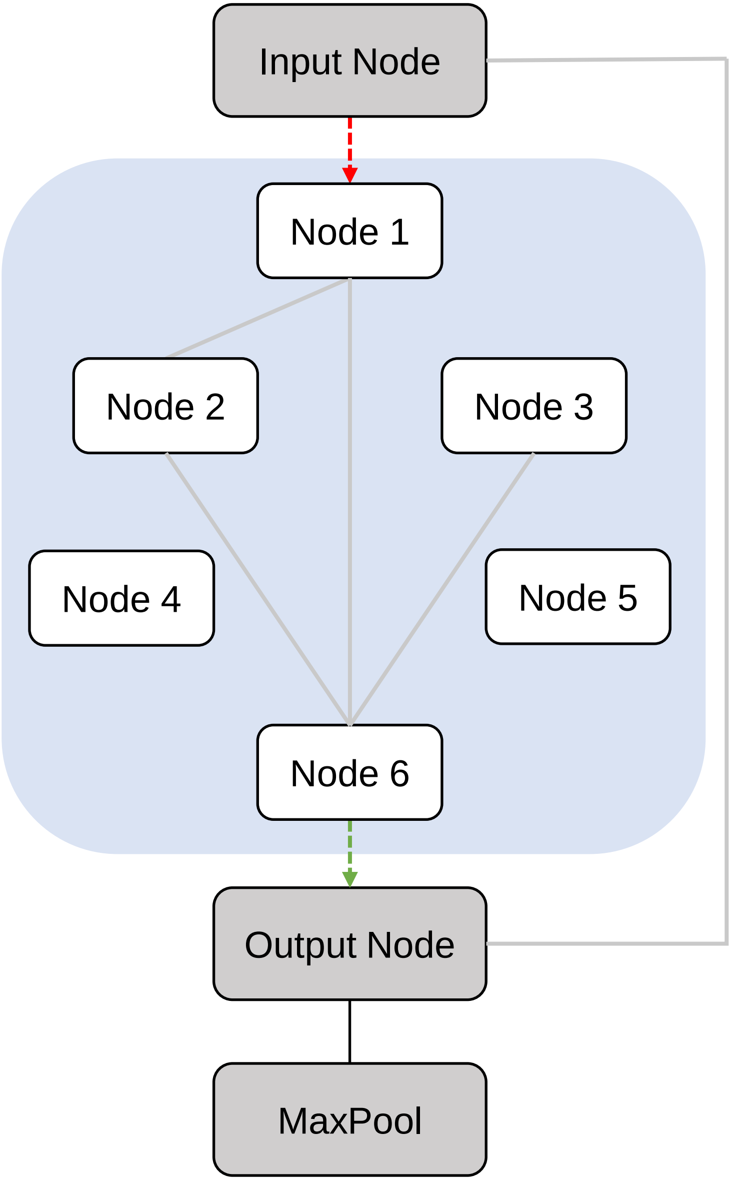
Example of encoding one stage in NSGA-Net



[1, **0, 0,** 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1]

$e_{12},$
$\mathbf{e_{13}, e_{23},}$
$e_{14}, e_{24}, e_{34},$
$e_{15}, e_{25}, e_{35}, e_{45},$
$e_{16}, e_{26}, e_{36}, e_{46}, e_{56},$
e_{IO}

Example of encoding one stage in NSGA-Net



[1, 0, 0, **0, 0, 0,** 0, 0, 0, 0, 1, 1, 1, 0, 0, 1]

$e_{12},$

$e_{13}, e_{23},$

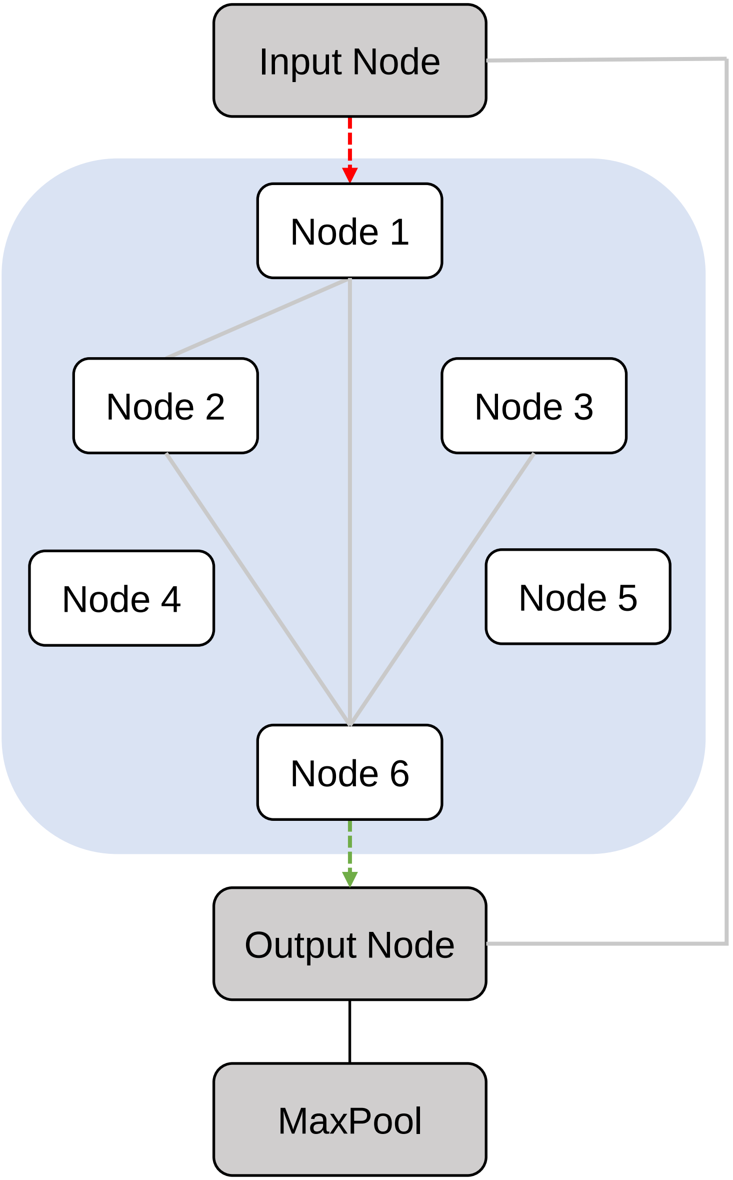
$\mathbf{e_{14}, e_{24}, e_{34},}$

$e_{15}, e_{25}, e_{35}, e_{45},$

$e_{16}, e_{26}, e_{36}, e_{46}, e_{56},$

e_{IO}

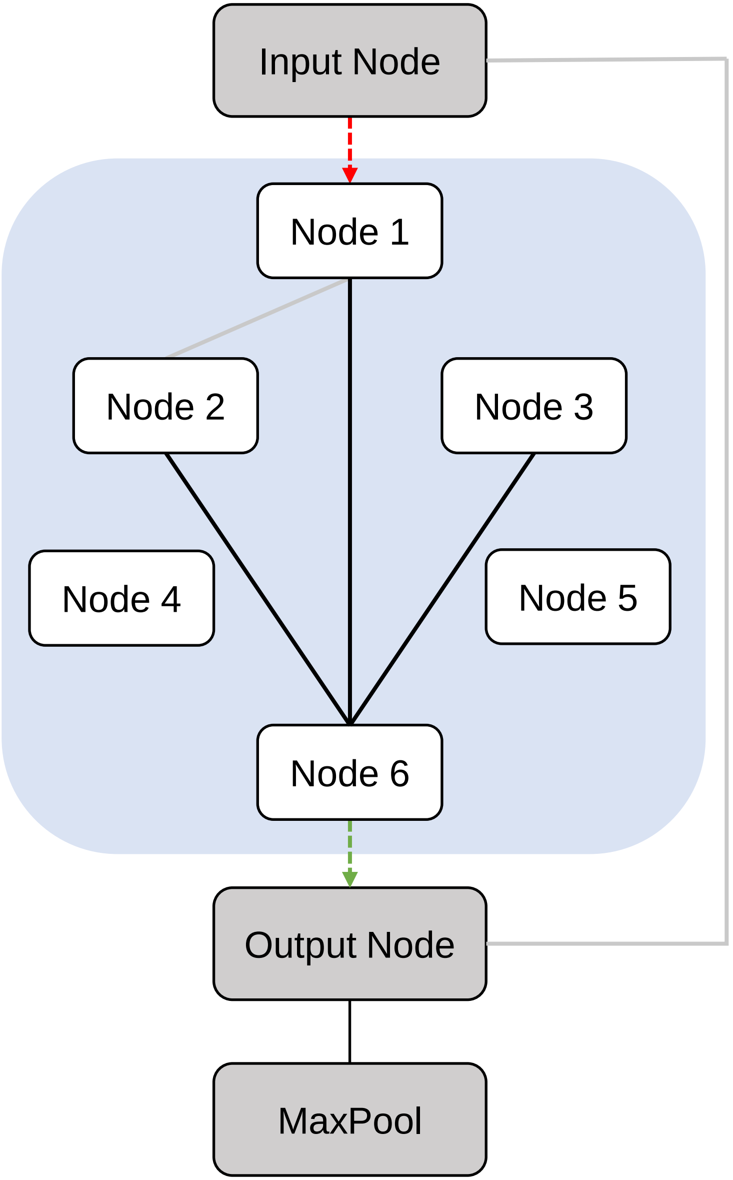
Example of encoding one stage in NSGA-Net



[1, 0, 0, 0, 0, 0, **0, 0, 0, 0,** 1, 1, 1, 0, 0, 1]

e_{12},
e_{13}, e_{23},
e_{14}, e_{24}, e_{34},
$\mathbf{e_{15}}$, $\mathbf{e_{25}}$, $\mathbf{e_{35}}$, $\mathbf{e_{45}}$,
e_{16}, e_{26}, e_{36}, e_{46}, e_{56},
e_{IO}

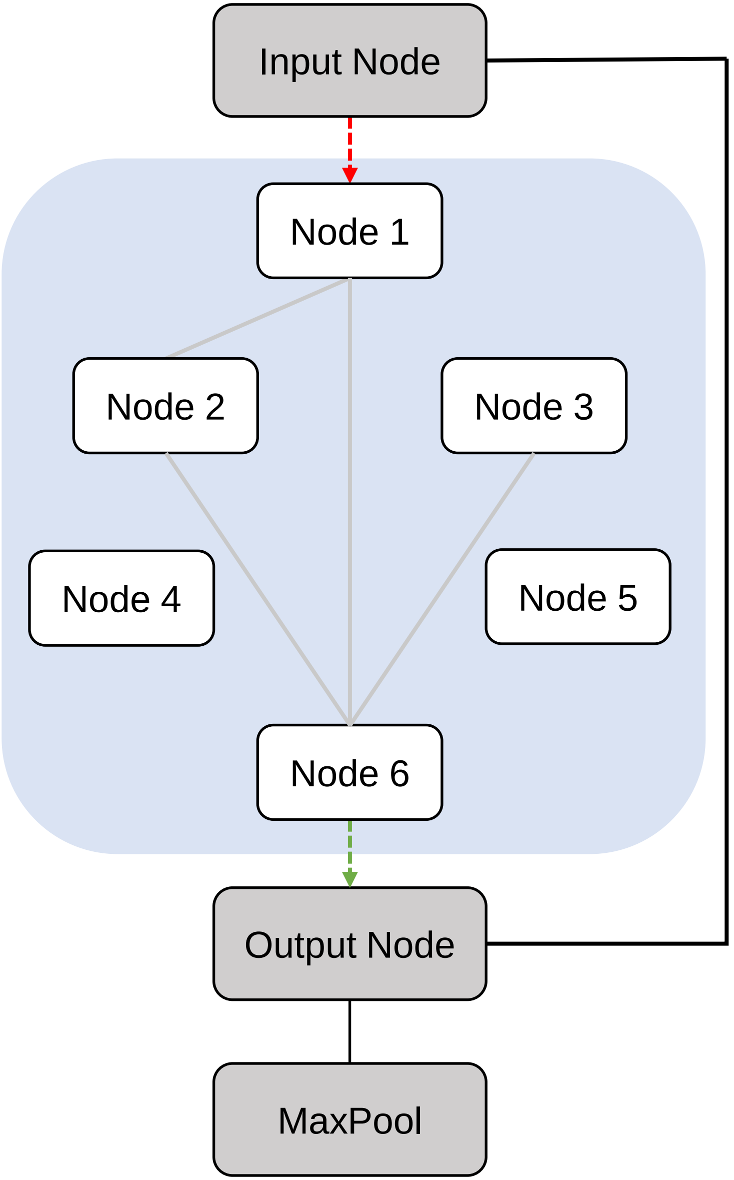
Example of encoding one stage in NSGA-Net



[1, 0, 0, 0, 0, 0, 0, 0, 0, **0, 1, 1, 1, 0, 0,** 1]

e_{12},
e_{13}, e_{23},
e_{14}, e_{24}, e_{34},
e_{15}, e_{25}, e_{35}, e_{45},
$\mathbf{e_{16}}$, $\mathbf{e_{26}}$, $\mathbf{e_{36}}$, $\mathbf{e_{46}}$, $\mathbf{e_{56}}$,
e_{IO}

Example of encoding one stage in NSGA-Net



$$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, \mathbf{1}]$$

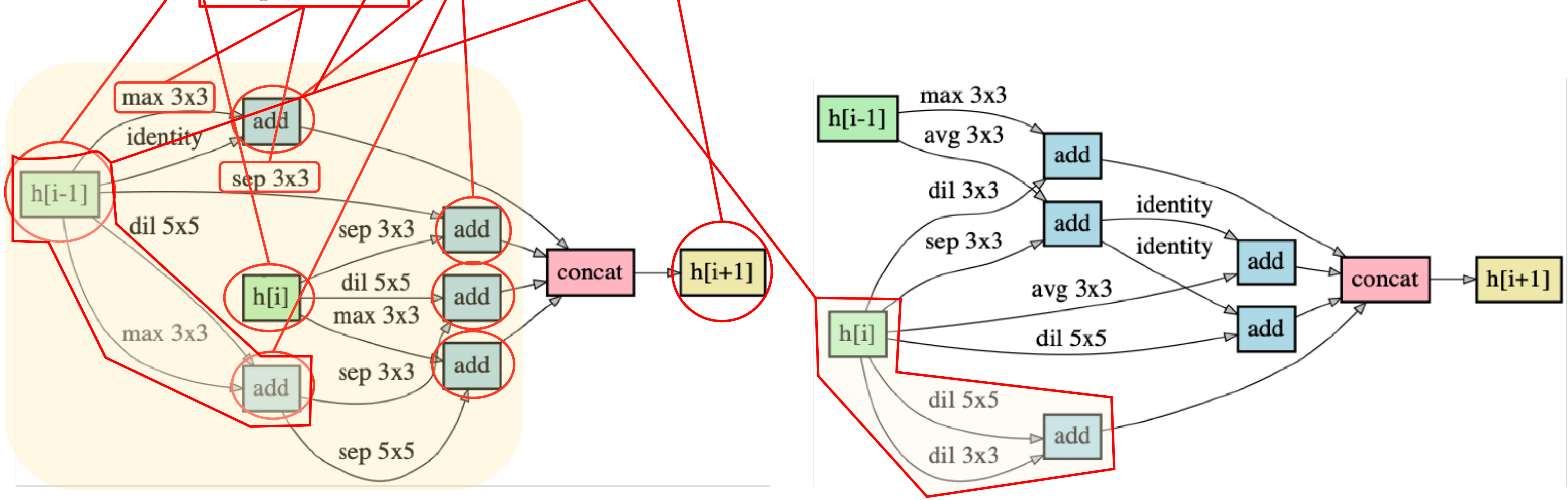
$$\begin{array}{c} e_{12}, \\ e_{13}, e_{23}, \\ e_{14}, e_{24}, e_{34}, \\ e_{15}, e_{25}, e_{35}, e_{45}, \\ e_{16}, e_{26}, e_{36}, e_{46}, e_{56}, \\ \mathbf{e_{IO}} \end{array}$$

Example of encoding one stage in NSGA-Net

[NSGA-Net] Search Space (Micro-level)

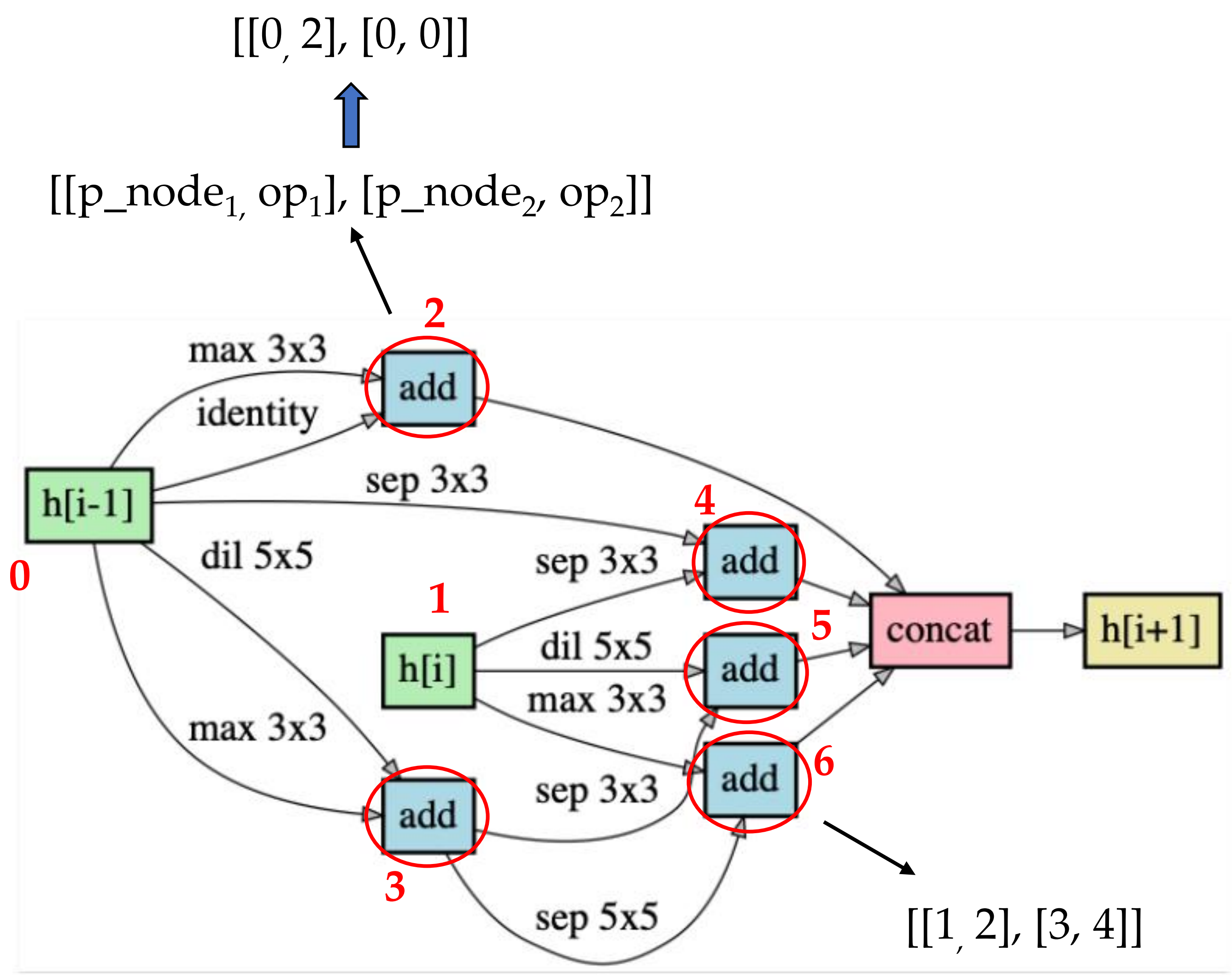
- Modified DARTS search space.
- Each solution is the arrangement of operations in two cells: normal cell; reduce cell
- The operations arrangement in each cell is presented as a normal DAG has 5 nodes. Each node is the place to aggregate the output (data) from 2 previous nodes (including 2 Input Nodes; can be duplicated). Each edge represents 1 out of 9 operations.

We can't use the vector $[e_1, e_2, \ldots]$ to encode the solution because a node can receive the output of 1 previous node 2 times



Normal Cell *Reduction Cell*

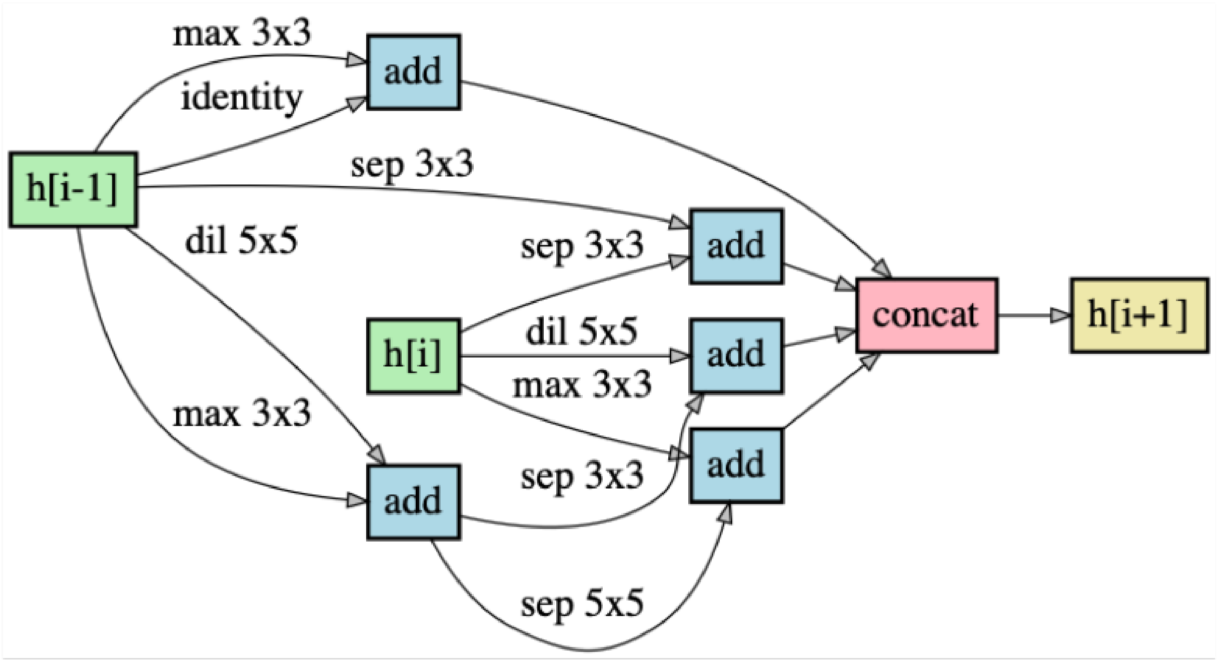
Example of a solution in NSGA-Net search space (micro-level)



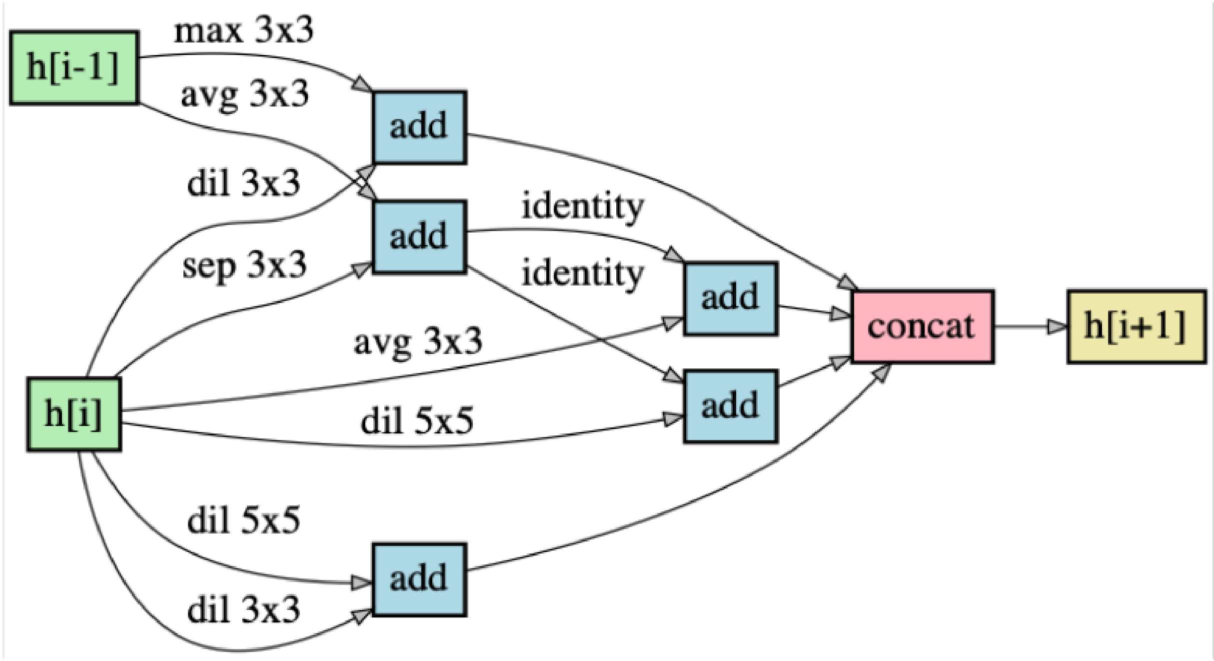
- 0: identity
- 1: avg 3x3
- 2: max 3x3
- 3: sep 3x3
- 4: sep 5x5
- 5: sep 7x7
- 6: dil 3x3
- 7: dil 5x5
- 8: conv 7x7

→ [[[0, 2], [0, 0]], [[0, 7], [0, 2]], [[0, 3], [1, 3]], [[1, 7], [3, 3]], [[1, 2], [3, 4]]] → Shape: [5, 2, 2]

Illustration of the encoding process in NSGA-Net search space (micro-level)



Normal Cell



Reduction Cell

→ [[[[0, 2], [0, 0]], [[0, 7], [0, 2]], [[0, 3], [1, 3]], [[1, 7], [3, 3]], [[1, 2], [3, 4]]],

[[[0, 2], [1, 6]], [[0, 1], [1, 3]], [[1, 7], [1, 6]], [[3, 0], [1, 1]], [[3, 0], [1, 7]]]]

→ Shape: [2, 5, 2, 2]

Illustration of the encoding process in NSGA-Net search space (micro-level)

Represent solution in EA's search space (Micro-level)

- Convert matrix 4D to vector 1D.
- Length of the solution: 40

[[[[0, 2], [0, 0]], [[0, 7], [0, 2]], [[0, 3], [1, 3]], [[1, 7], [3, 3]], [[1, 2], [3, 4]]],

[[[0, 2], [1, 6]], [[0, 1], [1, 3]], [[1, 7], [1, 6]], [[3, 0], [1, 1]], [[3, 0], [1, 7]]]]

⬇

[0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4,

0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

Crossover (approach 1)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

Crossover (approach 1)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

$\downarrow$

X_1 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

Crossover (approach 2)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

Crossover (approach 2)

$X_1 = [\boxed{0, 2}, 0, 0, \boxed{0, 7}, \boxed{0, 2}, 0, 3, 1, 3, \boxed{1, 7}, 3, 3, 1, 2, 3, 4, 0, 2, \boxed{1, 6}, 0, 1, \boxed{1, 3}, 1, 7, 1, 6, 3, 0, 1, 1, \boxed{3, 0}, 1, 7]$

$X_2 = [\boxed{0, 2}, 1, 6, \boxed{0, 1}, \boxed{1, 3}, 1, 7, 1, 6, \boxed{3, 0}, 1, 1, 3, 0, 1, 7, 0, 2, \boxed{0, 0}, 0, 7, \boxed{0, 2}, 0, 3, 1, 3, 1, 7, 3, 3, \boxed{1, 2}, 3, 4]$

$$\Downarrow$$

$X_1 = [\boxed{0, 2}, 0, 0, \boxed{0, 1}, \boxed{1, 3}, 0, 3, 1, 3, \boxed{3, 0}, 3, 3, 1, 2, 3, 4, 0, 2, \boxed{0, 0}, 0, 1, \boxed{0, 2}, 1, 7, 1, 6, 3, 0, 1, 1, \boxed{1, 2}, 1, 7]$

$X_2 = [\boxed{0, 2}, 1, 6, \boxed{0, 7}, \boxed{0, 2}, 1, 7, 1, 6, \boxed{1, 7}, 1, 1, 3, 0, 1, 7, 0, 2, \boxed{1, 6}, 0, 7, \boxed{1, 3}, 0, 3, 1, 3, 1, 7, 3, 3, \boxed{3, 0}, 3, 4]$

Crossover (approach 3)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

Crossover (approach 3)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

Crossover (approach 3)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, $\boxed{0, 2}$, $\boxed{1, 6}$, $\boxed{0, 1}$, $\boxed{1, 3}$, $\boxed{1, 7}$, $\boxed{1, 6}$, $\boxed{3, 0}$, $\boxed{1, 1}$, $\boxed{3, 0}$, $\boxed{1, 7}$]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, $\boxed{0, 2}$, $\boxed{0, 0}$, $\boxed{0, 7}$, $\boxed{0, 2}$, $\boxed{0, 3}$, $\boxed{1, 3}$, $\boxed{1, 7}$, $\boxed{3, 3}$, $\boxed{1, 2}$, $\boxed{3, 4}$]

Crossover (approach 3)

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4]

$\downarrow$

X_1 = [0, 2, 0, 0, 0, 7, 0, 2, 0, 3, 1, 3, 1, 7, 3, 3, 1, 2, 3, 4, 0, 2, 1, 6, 0, 7, 1, 3, 0, 3, 1, 6, 3, 0, 3, 3, 3, 0, 1, 7]

X_2 = [0, 2, 1, 6, 0, 1, 1, 3, 1, 7, 1, 6, 3, 0, 1, 1, 3, 0, 1, 7, 0, 2, 0, 0, 0, 1, 0, 2, 1, 7, 1, 3, 1, 7, 1, 1, 1, 2, 3, 4]

NAS Benchmarks

Issues in NAS Research and Evaluations

- The final results reported in different papers are typically incomparable
  - Different training code
  - Different search space
  - Different evaluation schemes
- We may wait for a long time to see the effectiveness of a new idea/method.

→ Using NAS Benchmarks can solve these problems.

NAS Benchmarks

- **Definition**

  A NAS Benchmark consists of architectures in the same search space; performance' architectures were obtained by training and evaluating on the same dataset, same setting configurations.

- **When should we use NAS Benchmarks?**

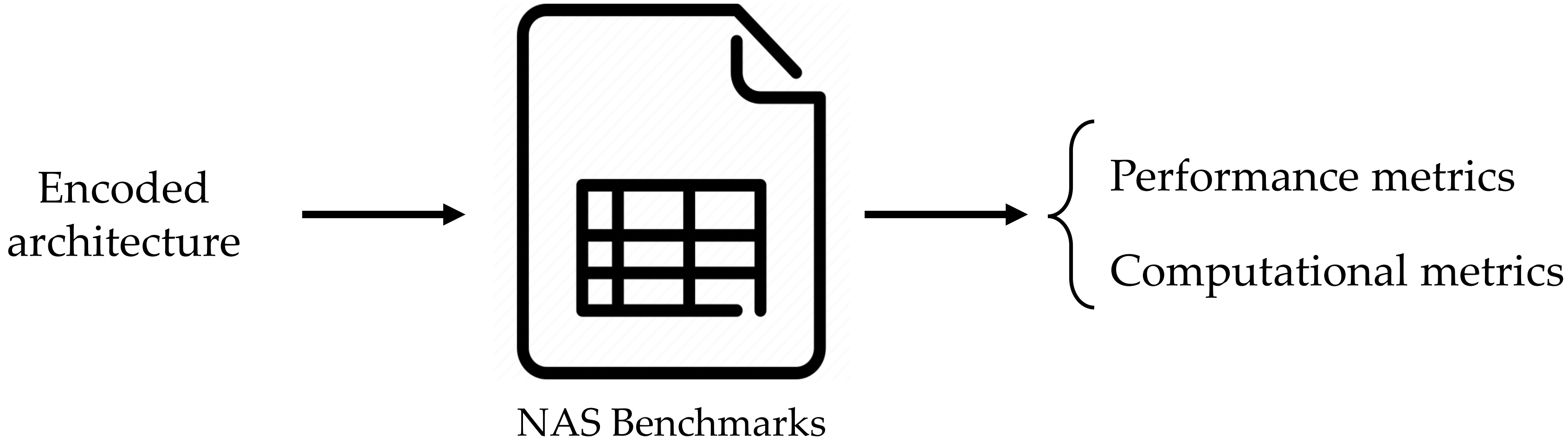
  Comparison between NAS methods

NAS Benchmarks

- **How to use NAS Benchmarks?**

  1) Encode the solution to the required format

  2)



NAS Benchmarks

- **Benefit of using NAS Benchmarks**
  - Fast
  - Can compare our NAS methods to other NAS methods
  - Only focus on "Search strategy" (NAS methods) and "Performance estimation strategy"
- **Drawbacks**
  - Not using for achieving the SoTA architecture
  - Constraints on the search space

NAS Benchmarks

- **Some NAS Benchmarks:**
  - MacroNAS
  - NAS-Bench-101
  - NAS-Bench-201
  - NAS-Bench-301
  - NAS-Bench-NLP

Table of Contents

MacroNAS

- **Some informations:**
  - Macro-level
  - Dataset: CIFAR-10; CIFAR-100
  - Performance metrics: {Training accuracy; Validation Accuracy; Testing Accuracy}
  - Computational metric: MMACs
  - Seach space size: 4,784,969
  - More detail: [2004.08996] Local Search is a Remarkably Strong Baseline for Neural Architecture Search (arxiv.org)

MacroNAS

- **Search space informations:**
  - Chain structure
  - 14 layers. Each layer is 1 out of 3 types.

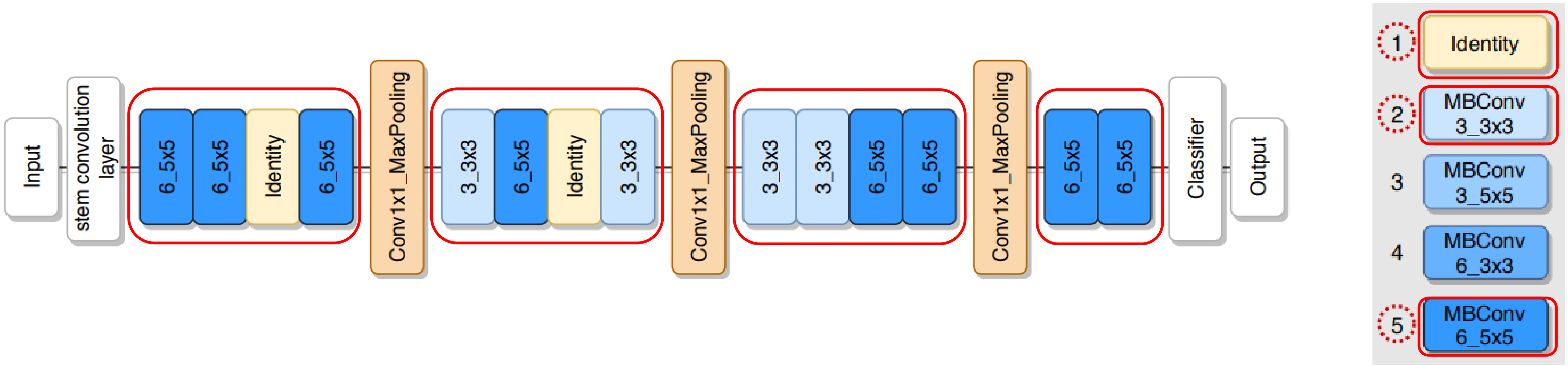


Table of Contents

NAS-Bench-101

- **Some informations:**
  - Micro-level
  - Dataset: CIFAR-10
  - Performance metrics: {Training accuracy; Validation Accuracy; Testing Accuracy}
  - Computational metric: #Params
  - Seach space size: ≈ 423,000
  - More details: [1902.09635] NAS-Bench-101: Towards Reproducible Neural Architecture Search (arxiv.org)

NAS-Bench-101

- **Search space informations:**
  - Normal DAG
  - Nodes represent operations. Edges represent the flow of data.
  - Each DAG has 7 nodes. There are 2 fixed nodes: Input Node; Output Node. Each remaining node is 1 of 3 operations.
  - Maximum number of edges in DAG is 9.

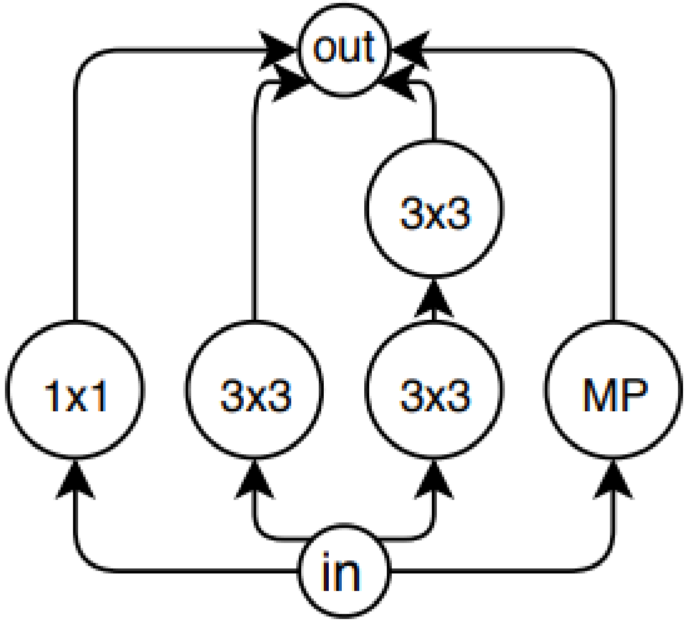


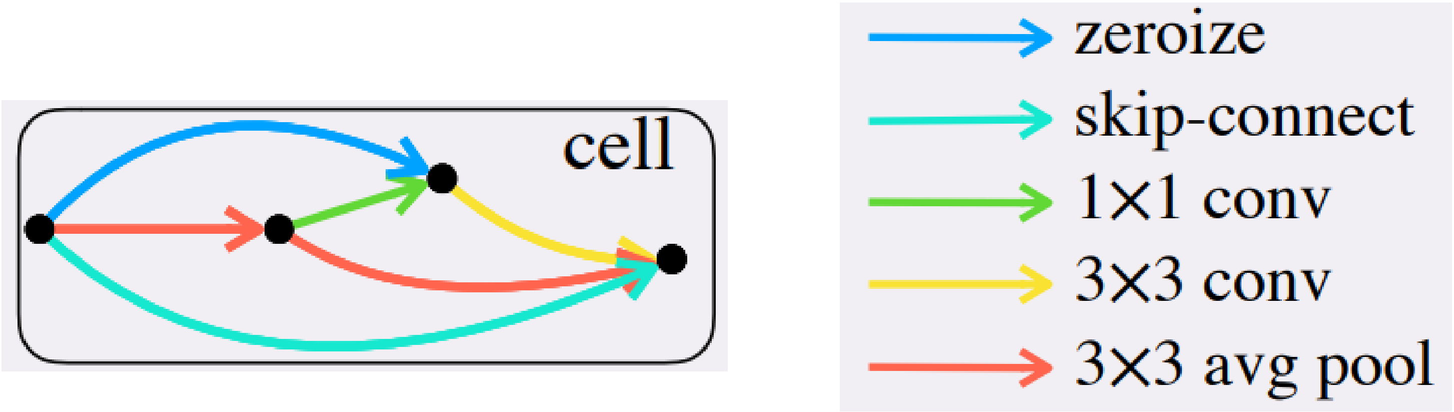
Table of Contents

NAS-Bench-201

- **Some informations:**
  - Micro-level
  - Dataset: CIFAR-10; CIFAR-100; ImageNet16-120
  - Performance metrics: {Training accuracy; Validation Accuracy; Testing Accuracy}
  - Computational metric: FLOPs, #Params
  - Seach space size: 15,625
  - More details: [2001.00326] NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search (arxiv.org)

NAS-Bench-201

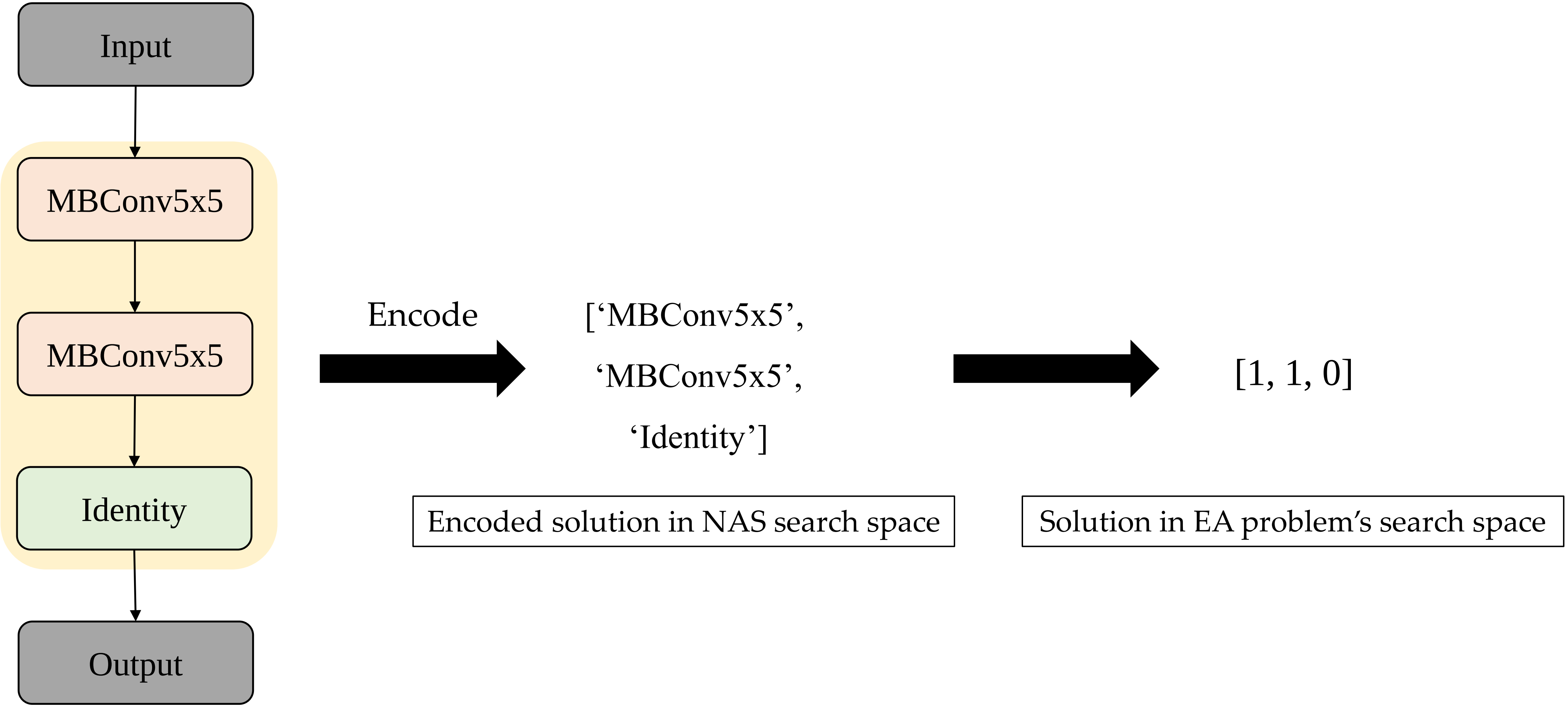
- **Search space informations:**
  - Fully connected DAG
  - Nodes is the place to aggregate the output (data) from previous nodes, edges represent operations.
  - Each DAG has 4 nodes. Each edge is 1 of 5 operators.



I recommend you should consider these questions before experimenting:

- Which kind of problem would you want to solve (SOP; MOP)?
- Which EA would you choose (GA, NSGA-II, MO-GOMEA, MOEA/D, …)? Is it suitable for the problem that you are solving? Search strategy
- What are solutions in the NAS search space? How to represent the solution of NAS search space in the EA problem's search space? ⟹ Search space

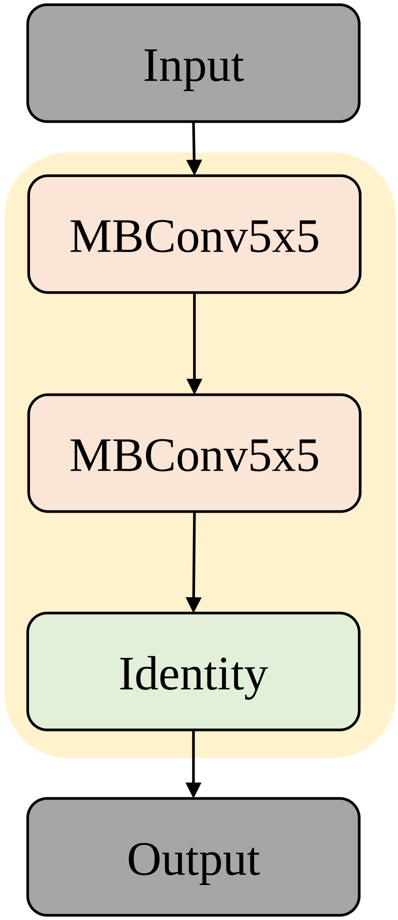
- What are solutions in the NAS search space? How to represent the solution of NAS search space in the EA problem's search space?



- How do you build a complete architecture? (PyTorch, Tensorflow, …)

[1, 1, 0]

Solution in EA problem's search space



A complete architecture

I recommend you should consider these questions before experimenting:

- Which kind of problem would you want to solve (SOP; MOP)?
- Which EA would you choose (GA, NSGA-II, MO-GOMEA, MOEA/D, …)?
  Is it suitable for the problem that you are solving?

  Search strategy
- What are solutions in the NAS search space? How to represent the solution of NAS search space in the EA problem's search space? ⟹ Search space
- What are fitness values? How to evalutate? ⟹ Performance Estimation Strategy
  - NAS-Benchmarks
  - Truly train ⟹ How do you build a complete architecture? (PyTorch, Tensorflow, …)

I recommend you should consider these questions before experimenting:

- Which kind of problem would you want to solve (SOP; MOP)?
- Which EA would you choose (GA, NSGA-II, MO-GOMEA, MOEA/D, …)? Is it suitable for the problem that you are solving? $\Big\}$ Search strategy
- What are solutions in the NAS search space? How to represent the solution of NAS search space in the EA problem's search space? $\Longrightarrow$ Search space
- What are fitness values? How to evalutate? $\Longrightarrow$ Performance Estimation Strategy
- How to perform crossover, mutation?

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

- During the search process, you should log the results (e.g., the population; the best solution; the approximation front; the number of evaluations) at the end of each generation.