Neural Architecture Search without Training

Joseph Mellor, Jack Turner, Amos Storkey, Elliot J. Crowley

Section: A2

Group: 3

2105036 - Anisa Binte Asad

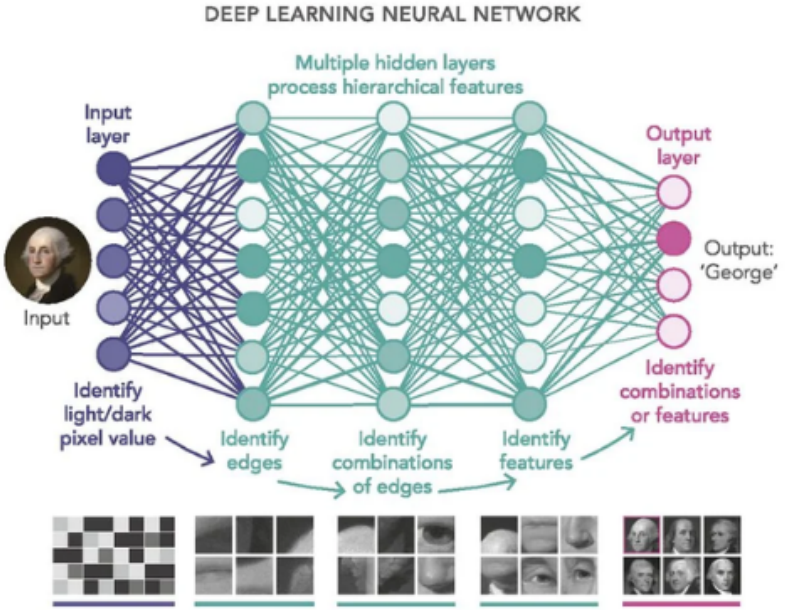
2105050 - Dipit Saha

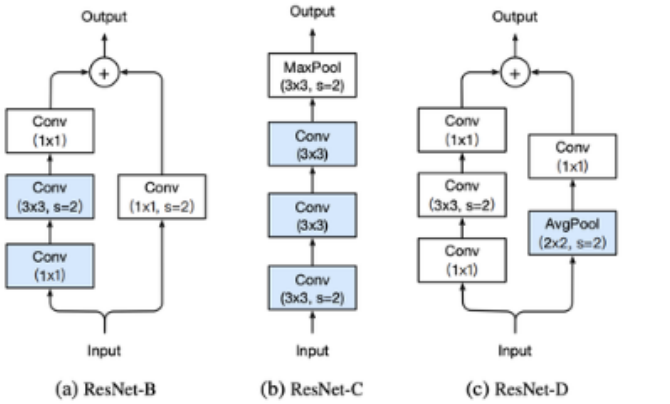
2105053 - Atika Tabassum Suchi

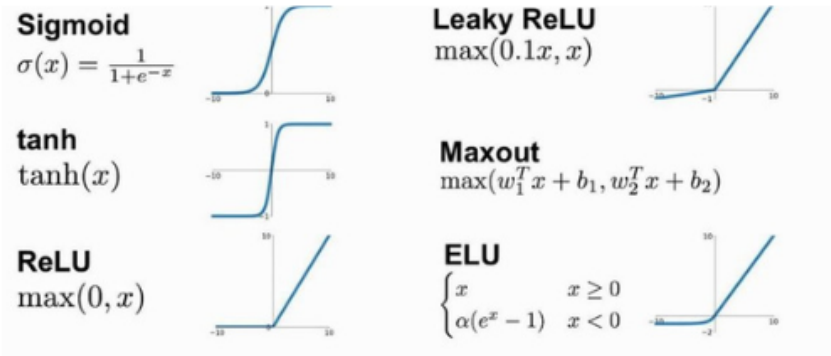
Department of Computer Science and Technology,

Bangladesh University of Engineering and Technology

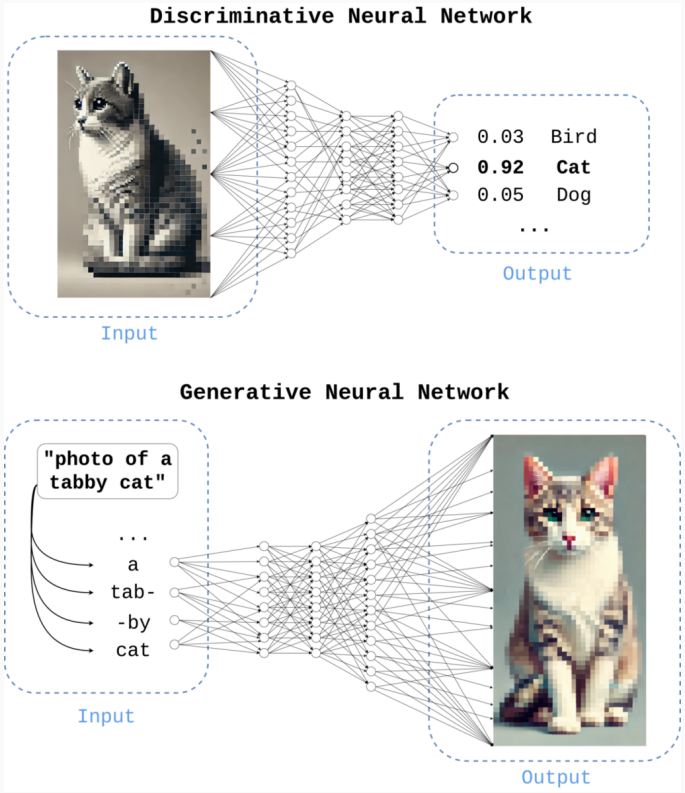
Why Designing Neural Networks is so Complex?



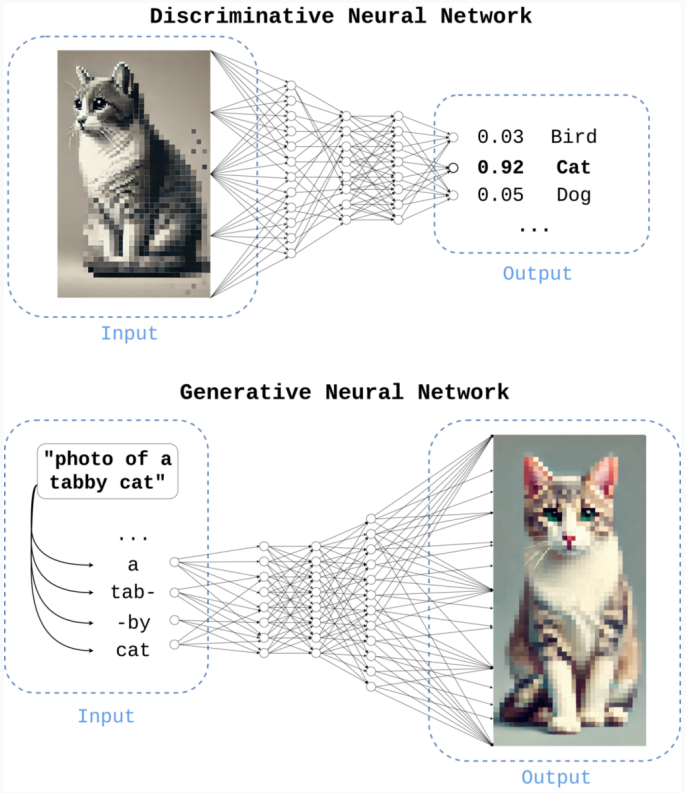




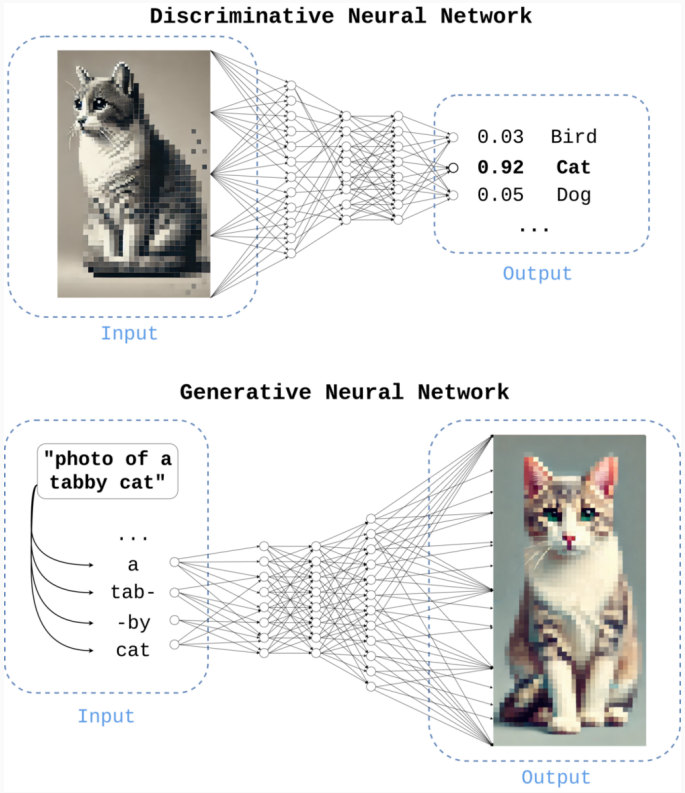
Designing Neural Networks vs Brewing Coffee

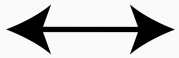


Designing Neural Networks vs Brewing Coffee



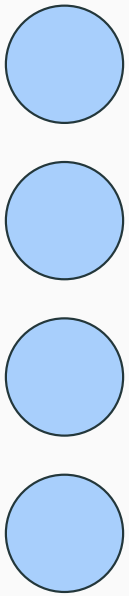
Designing Neural Networks vs Brewing Coffee





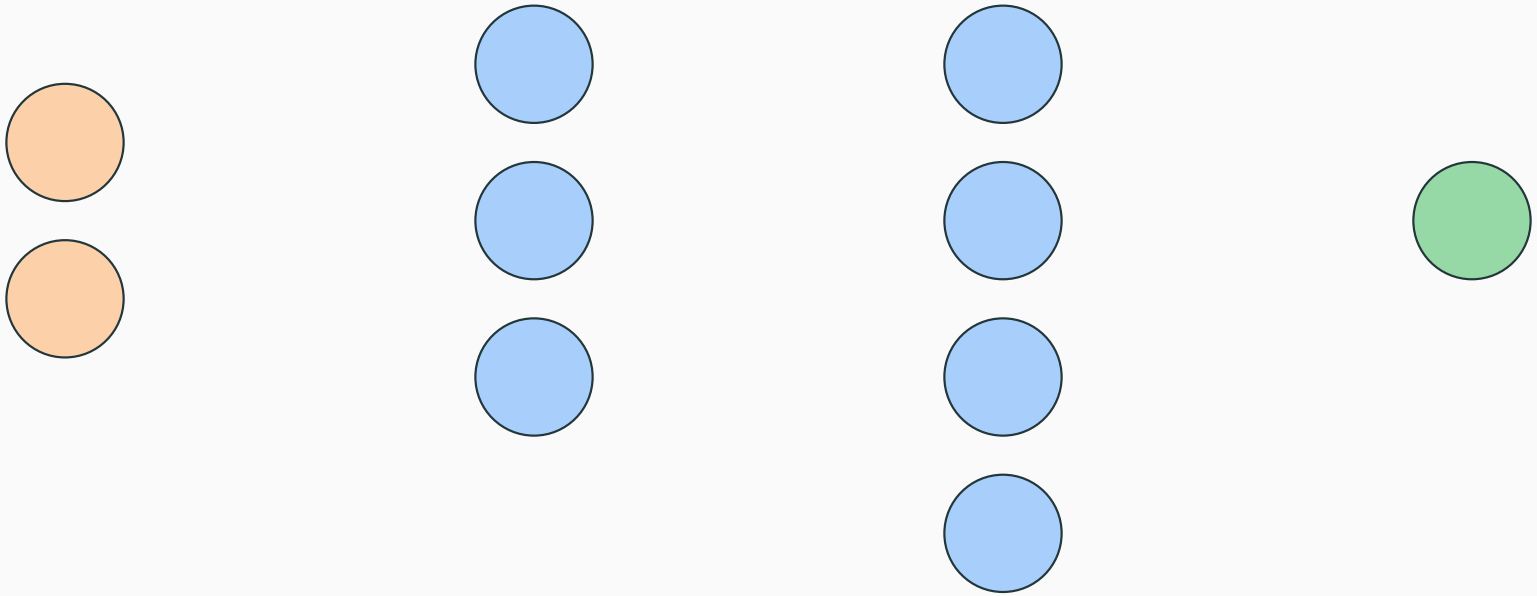
Key Considerations for Designing Neural Architecture

- Number of **neurons** in each layer



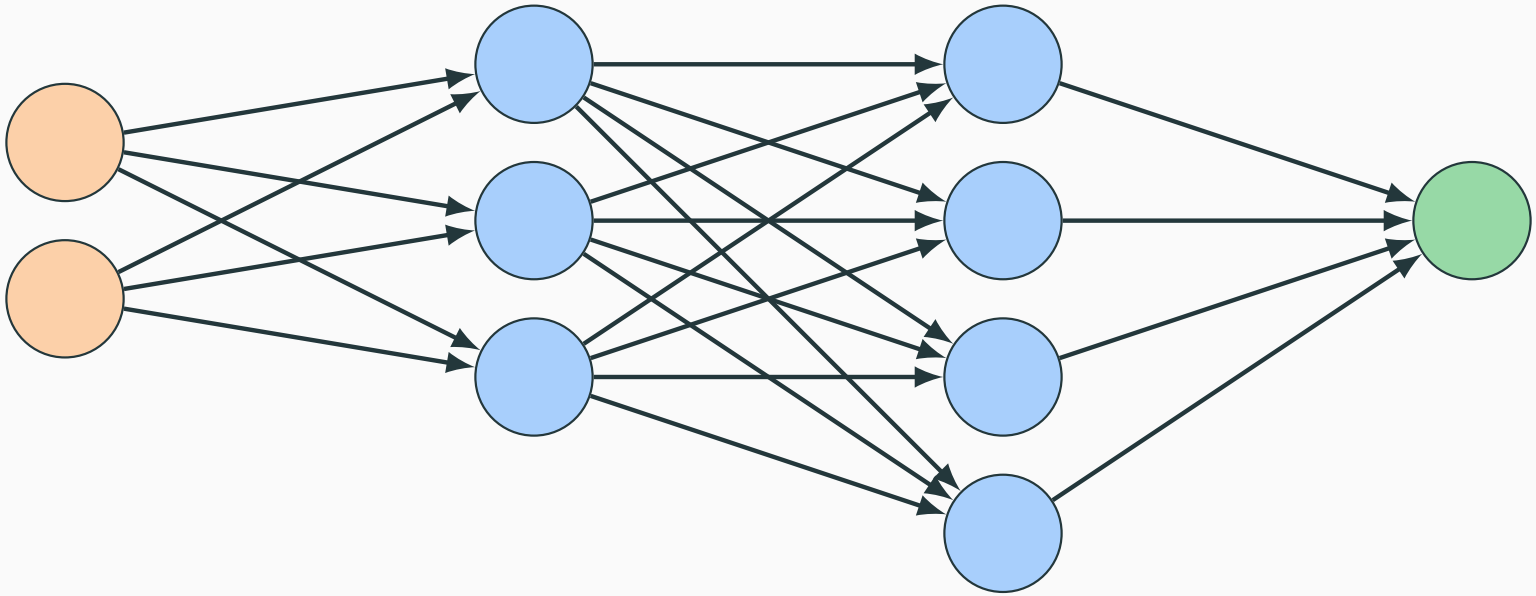
Key Considerations for Designing Neural Architecture

- Number of **layers** in the network



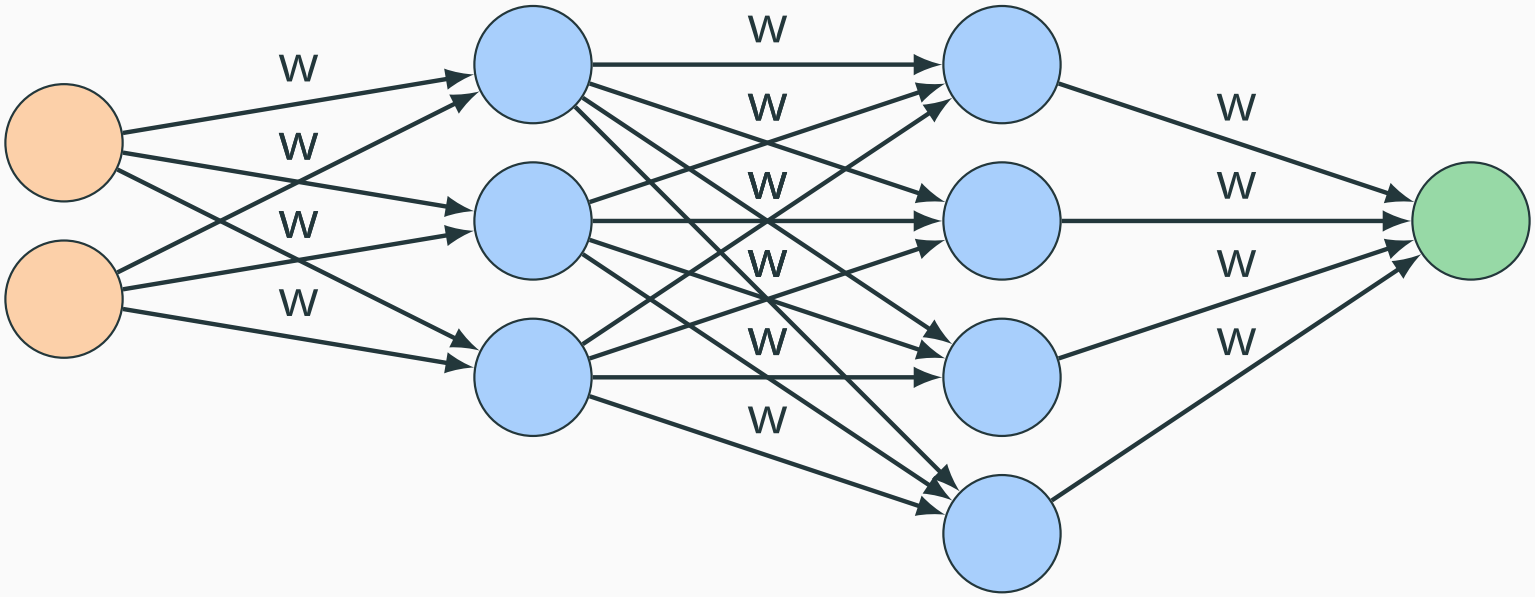
Key Considerations for Designing Neural Architecture

- Layer connectivity



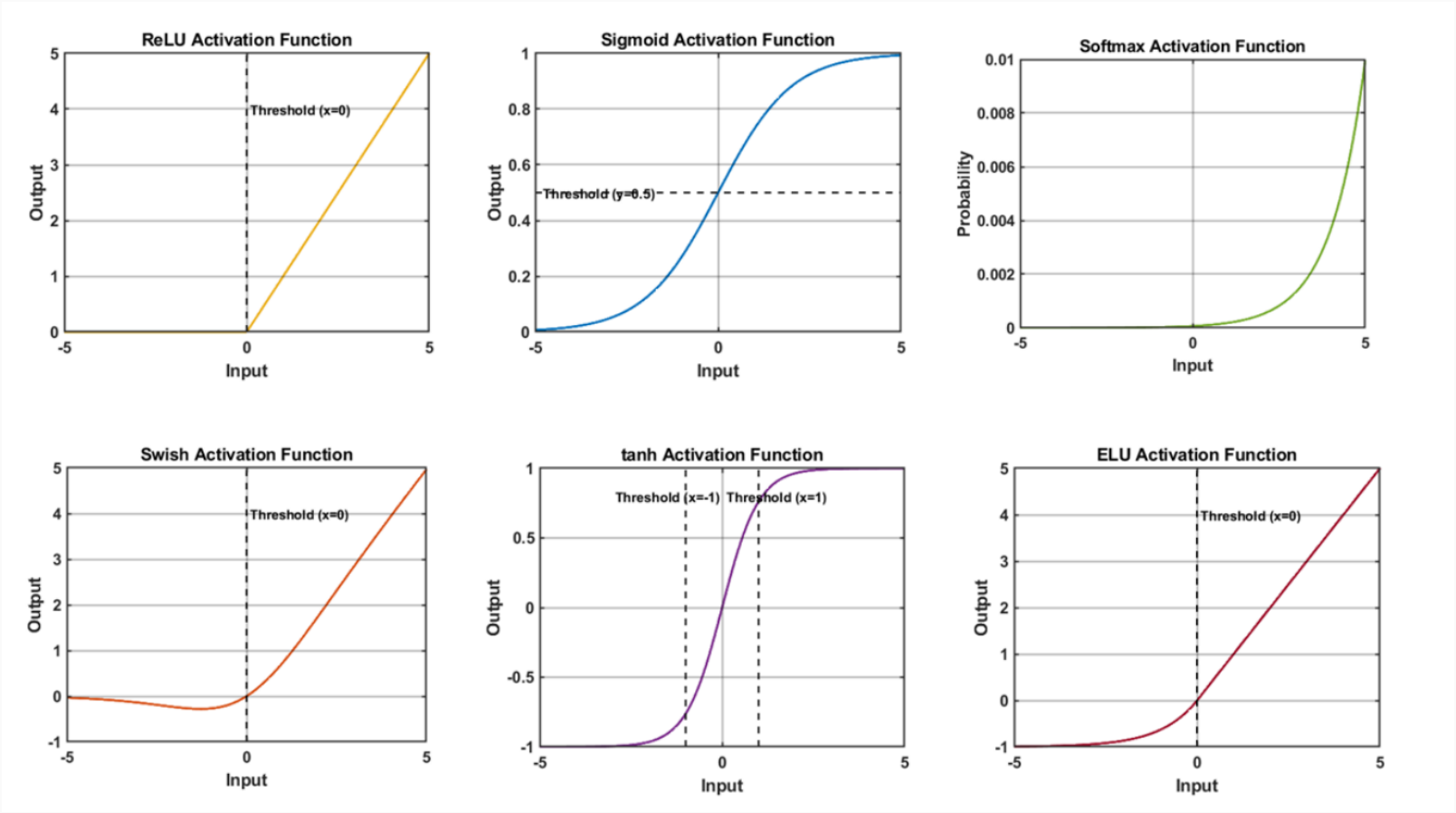
Key Considerations for Designing Neural Architecture

- **Parameters** (weights, biases)

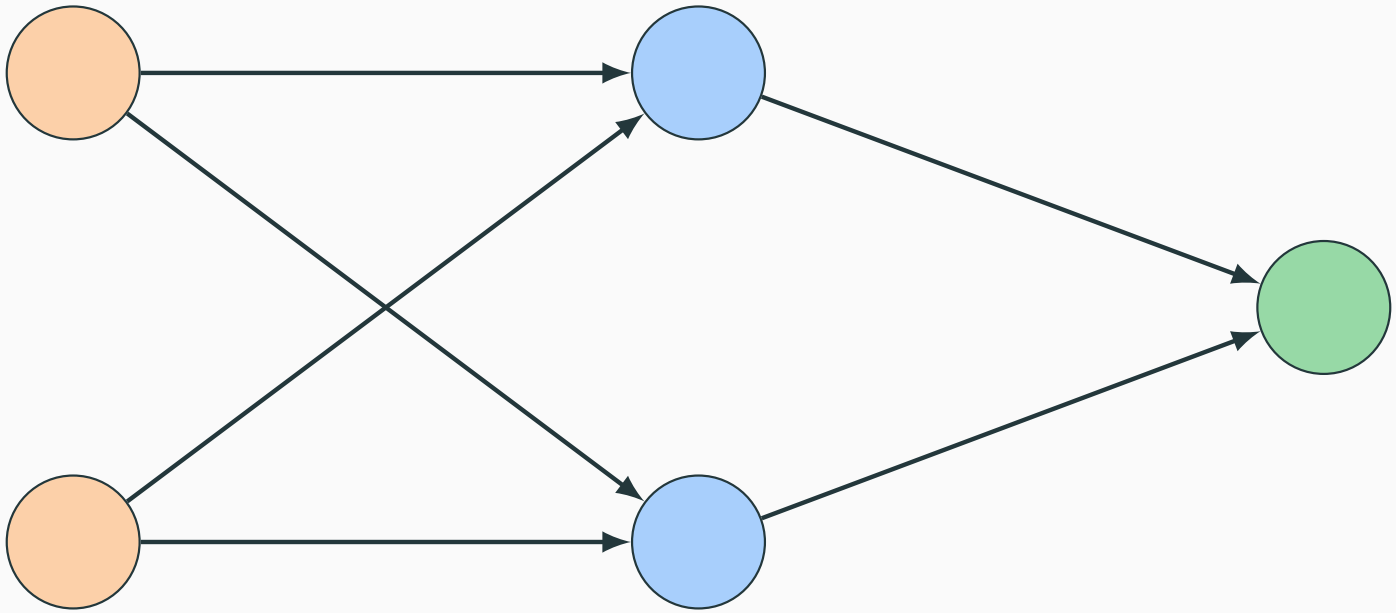


Key Considerations for Designing Neural Architecture

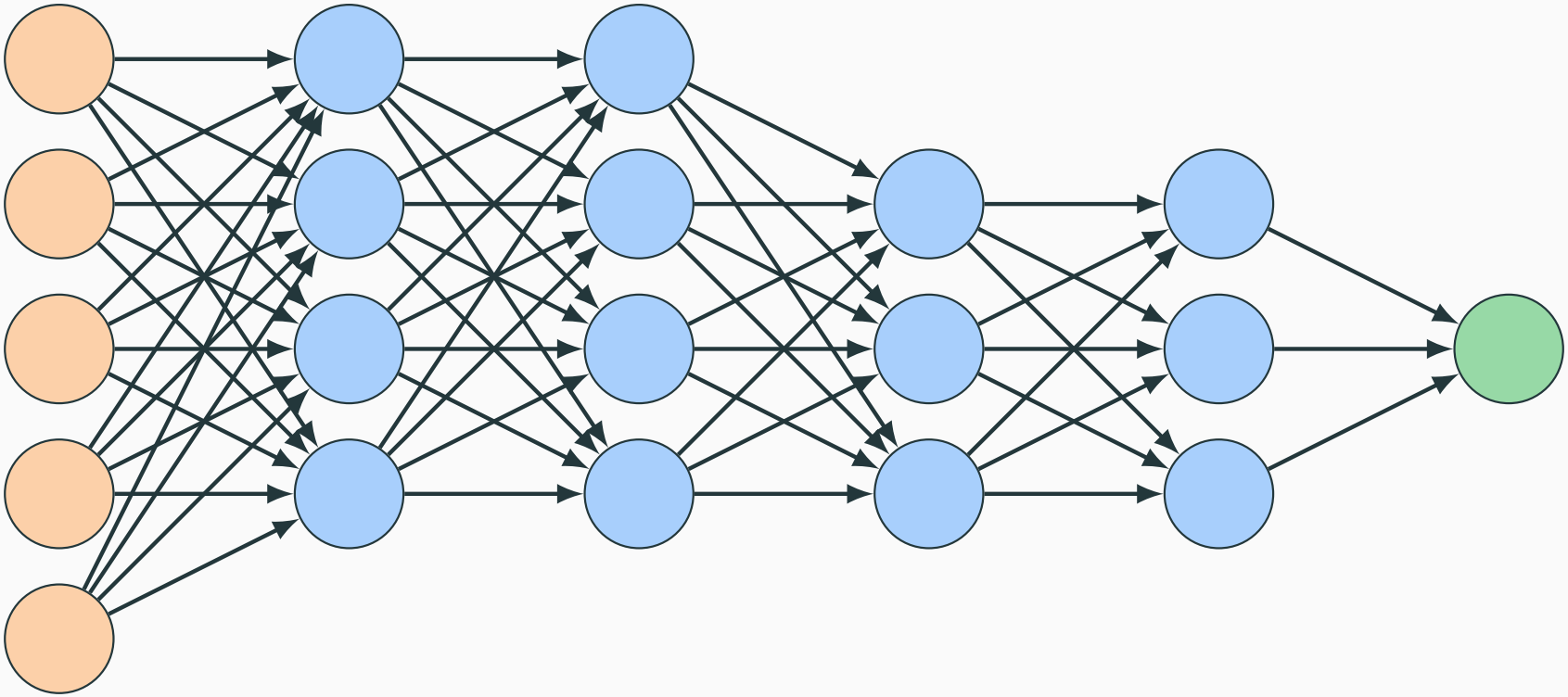
- Activation Function



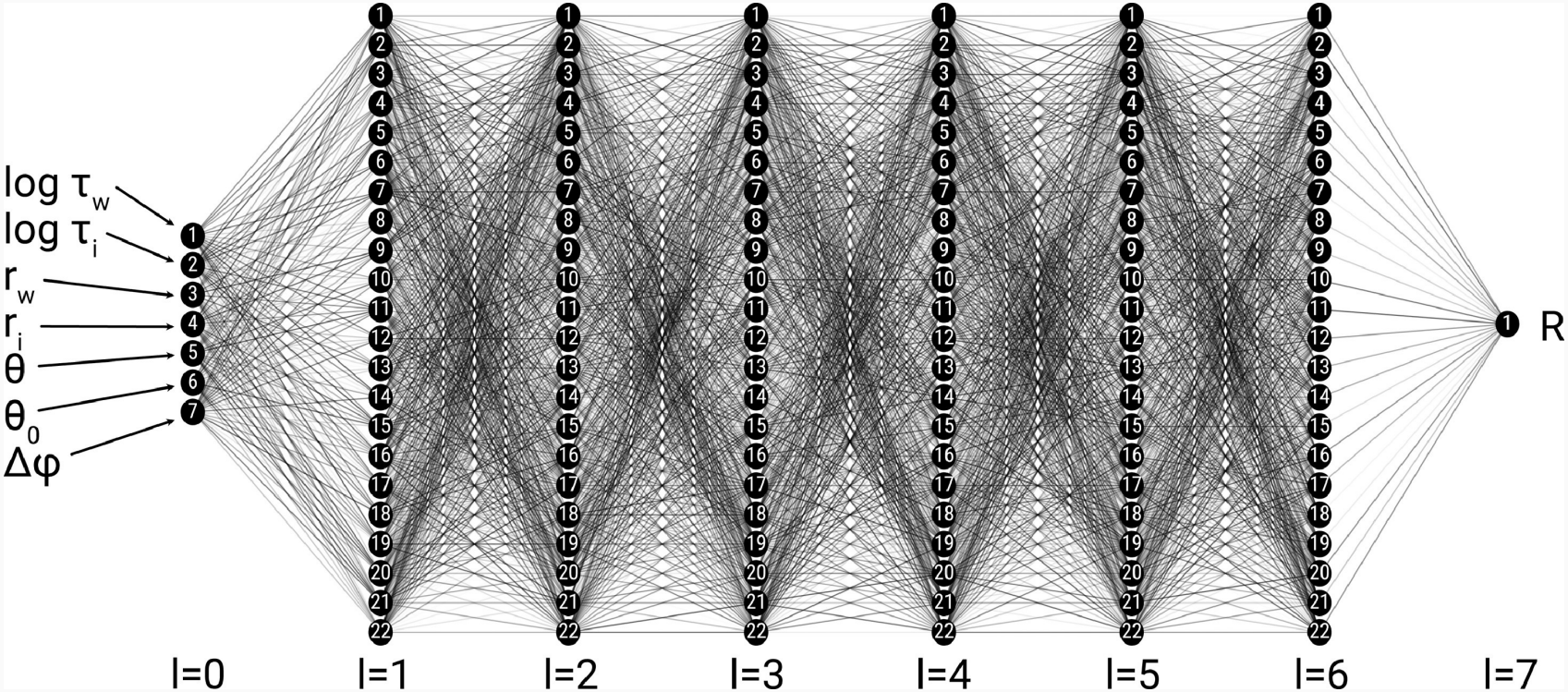
Simple Neural Architecture



Multilayered Neural Architecture



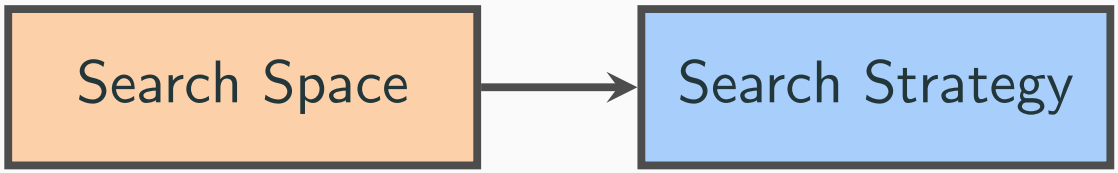
Complex Neural Architecture



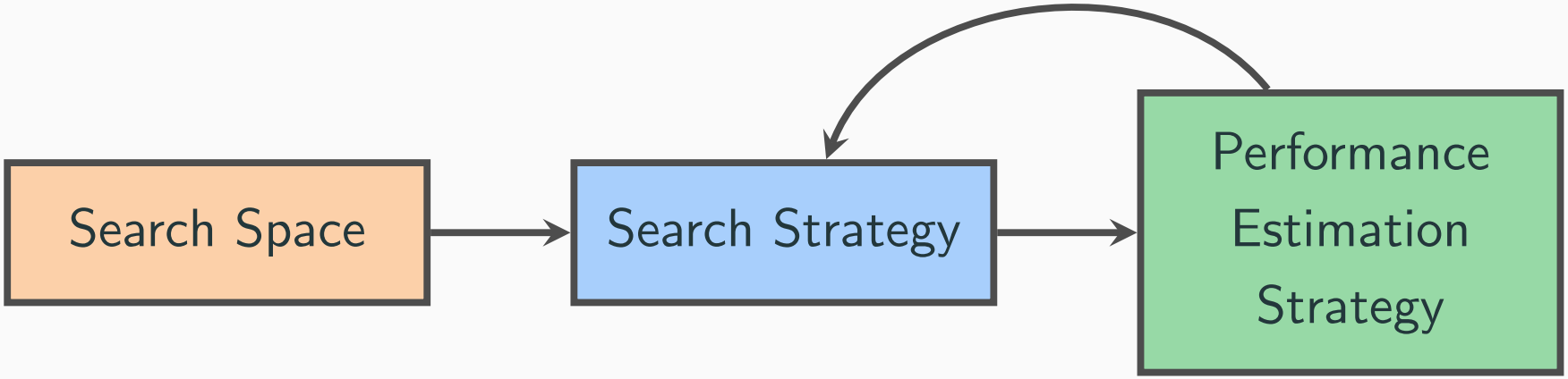
Neural Architecture Search

Core Components of NAS

Core Components of NAS

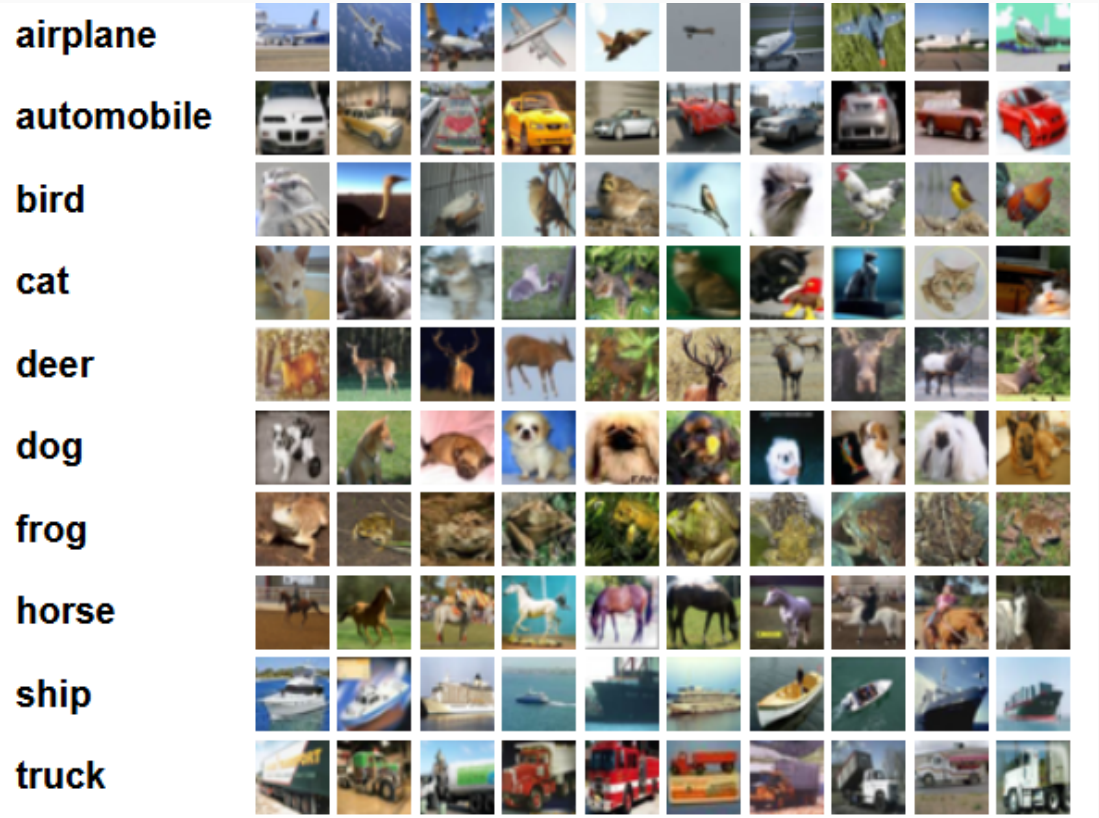


Core Components of NAS

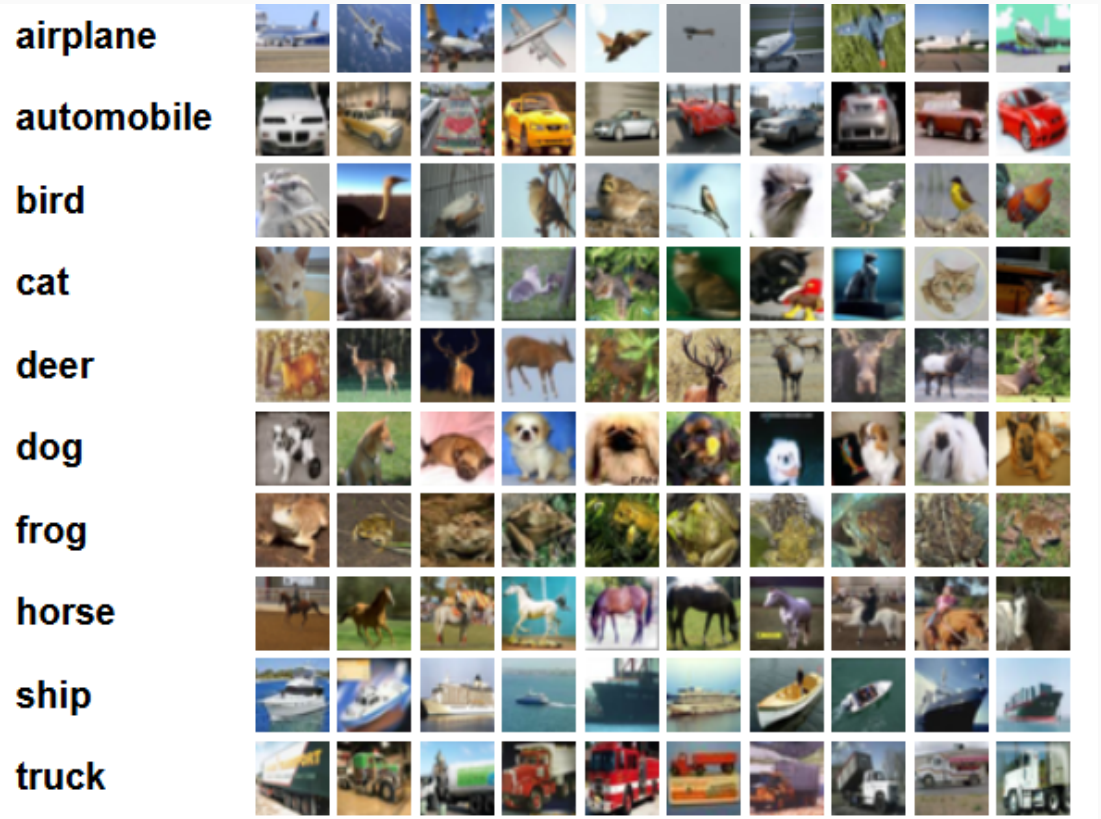


Application of NAS : CIFAR-10

Application of NAS : CIFAR-10



Application of NAS : CIFAR-10



- **60,000** 32x32 color images in **10** classes.
- **50,000** training images.
- **10,000** testing images.

Application of NAS : CIFAR-10

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet $(L = 40, k = 12)$ Huang et al. (2016a)	40	1.0M	5.24
DenseNet$(L = 100, k = 12)$ Huang et al. (2016a)	100	7.0M	4.10
DenseNet $(L = 100, k = 24)$ Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC $(L = 100, k = 40)$ Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

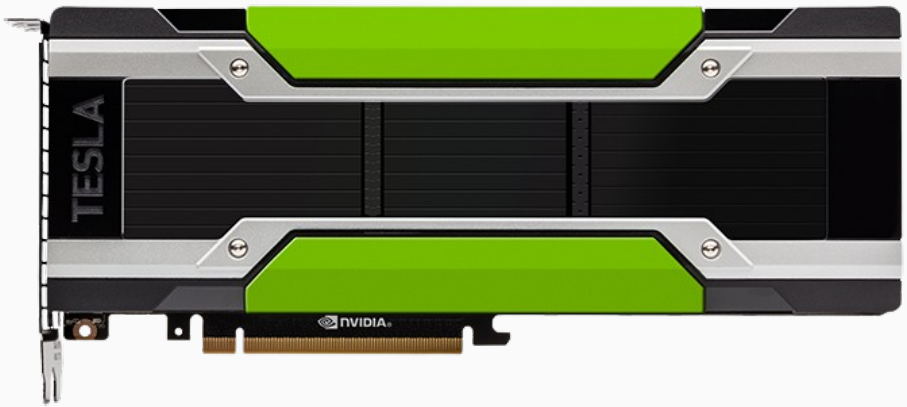
Application of NAS : CIFAR-10

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

Requirements for CIFAR-10 Classification with NAS

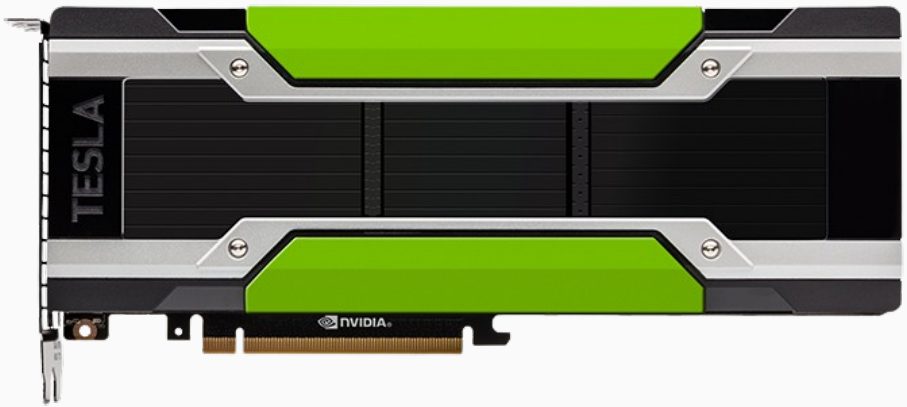
GPUs required:



x 100

Requirements for CIFAR-10 Classification with NAS

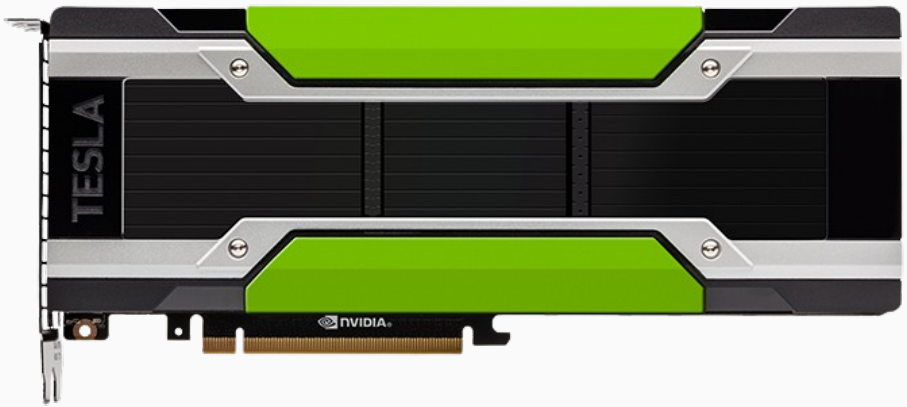
GPUs required:



x 200

Requirements for CIFAR-10 Classification with NAS

GPUs required:



x 800

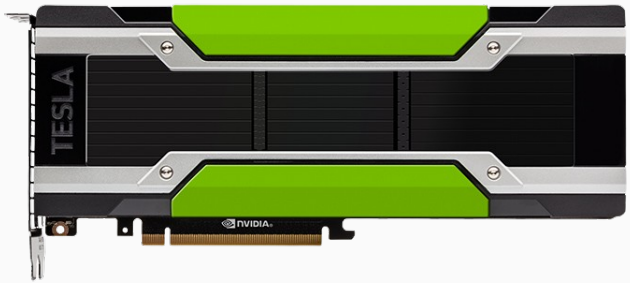
Requirements for CIFAR-10 Classification with NAS

Time required: 28 days

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Limitations of NAS

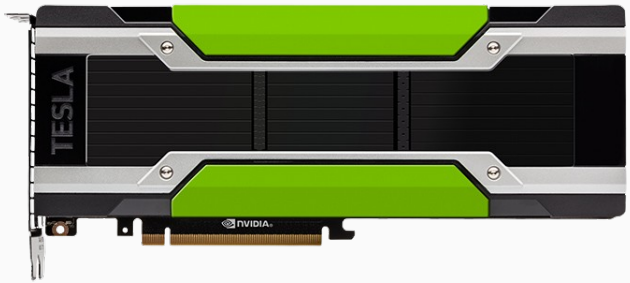
- Computationally expensive



x 800

Limitations of NAS

- Computationally expensive



x 800

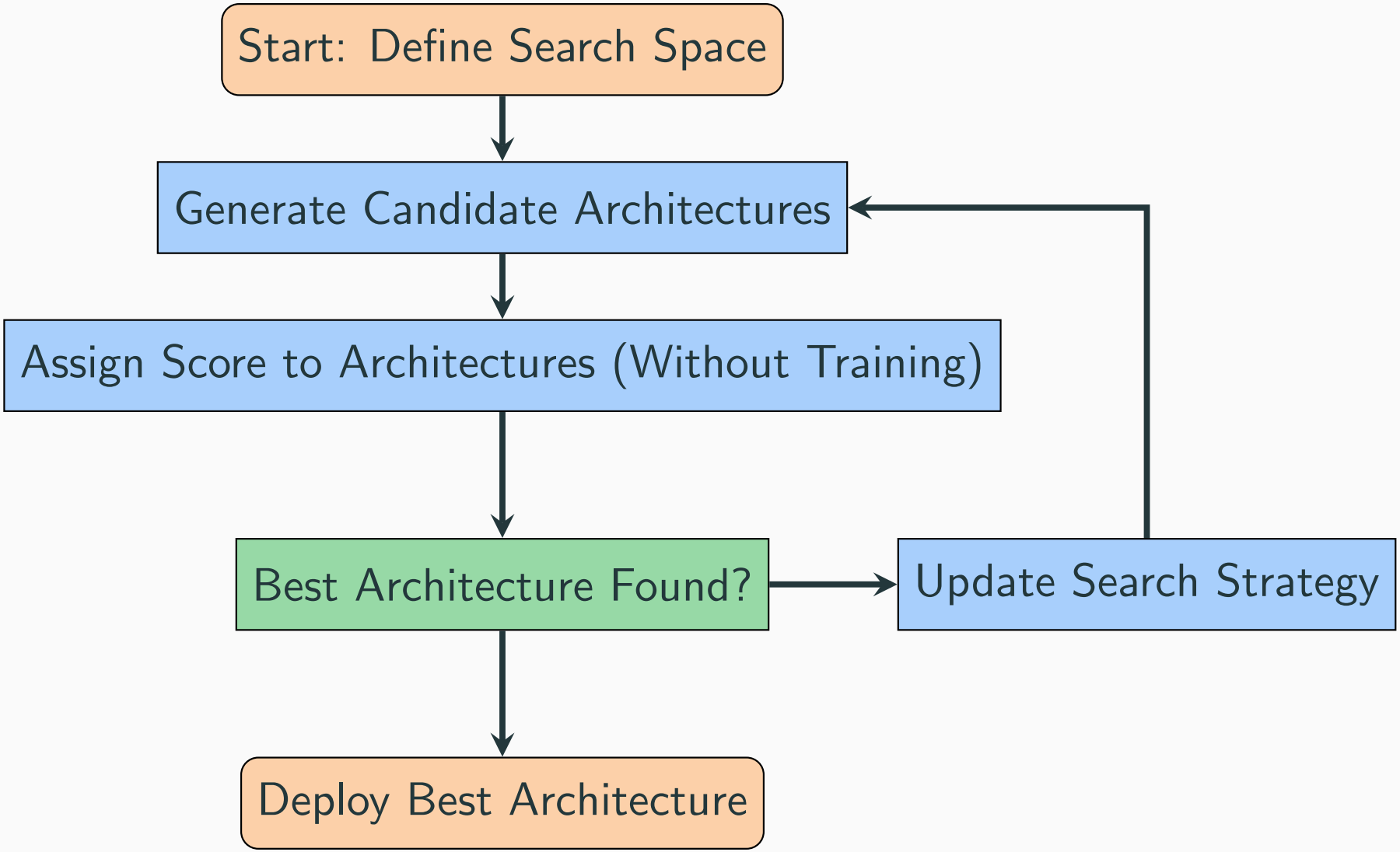
- Time consuimg

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

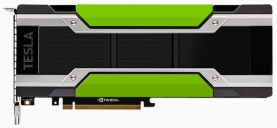
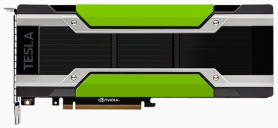
28 days

Neural Architecture Search without Training

Overview of NAS without Training



NAS vs NASWOT

Aspect	NAS	NASWOT
Computational Cost	 x 800	 x 1
Time Consumption	 28 days	 Seconds

How to score an architecture?

Let's break down the Scoring procedure

Steps for the scoring process

Step 1: Mini-batch selection

Step 2: Binary code generation

Step 3: Hamming distance calculation

Step 4: Kernel calculation.

Step 5: Assigning performance score

Steps for the scoring process

Step 1: Mini-batch selection

Step 2: Binary code generation

Step 3: Hamming distance calculation

Step 4: Kernel calculation.

Step 5: Assigning performance score

Steps for the scoring process

Step 1: Mini-batch selection

Step 2: Binary code generation

Step 3: Hamming distance calculation

Step 4: Kernel calculation.

Step 5: Assigning performance score

Steps for the scoring process

Step 1: Mini-batch selection

Step 2: Binary code generation

Step 3: Hamming distance calculation

Step 4: Kernel calculation.

Step 5: Assigning performance score

Steps for the scoring process

Step 1: Mini-batch selection

Step 2: Binary code generation

Step 3: Hamming distance calculation

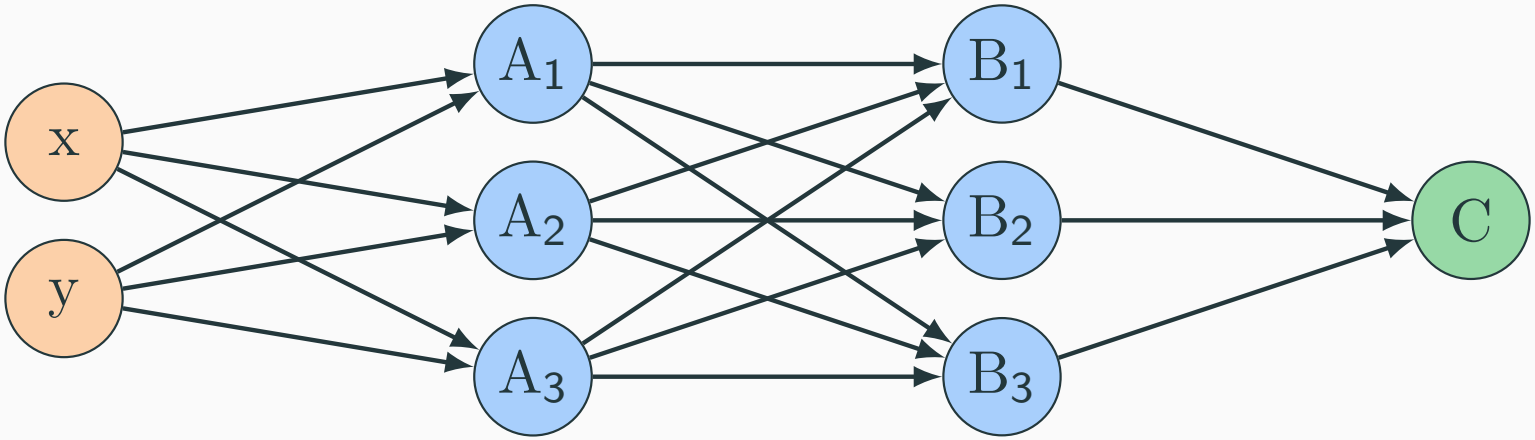
Step 4: Kernel calculation.

Step 5: Assigning performance score

Binary Indicator and Linear Region

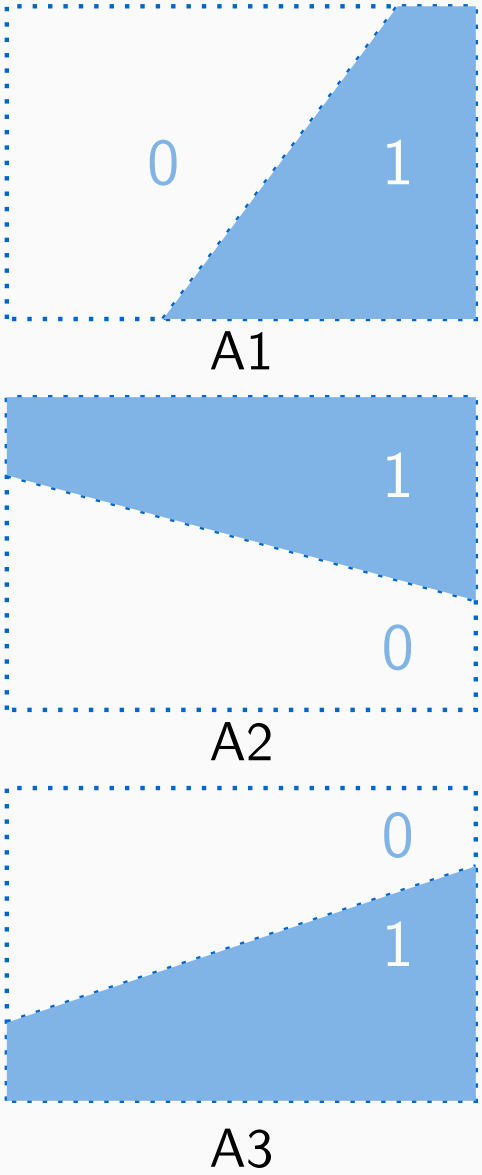
- For a mini-batch of data $X = \{x_i\}_{i=1}^{N}$, the output $f(x_i)$ forms a **binary code** c_i, which defines the **linear region**.
- The binary indicator is derived from the **ReLU activations** of the network units.

A Simple Neural Network



This figure shows a simple feedforward neural network with input nodes (x, y, hidden nodes (A_1, A_2, A_3, B_1, B_2, B_3) in blue, and an output node (C), fully connected with weights.

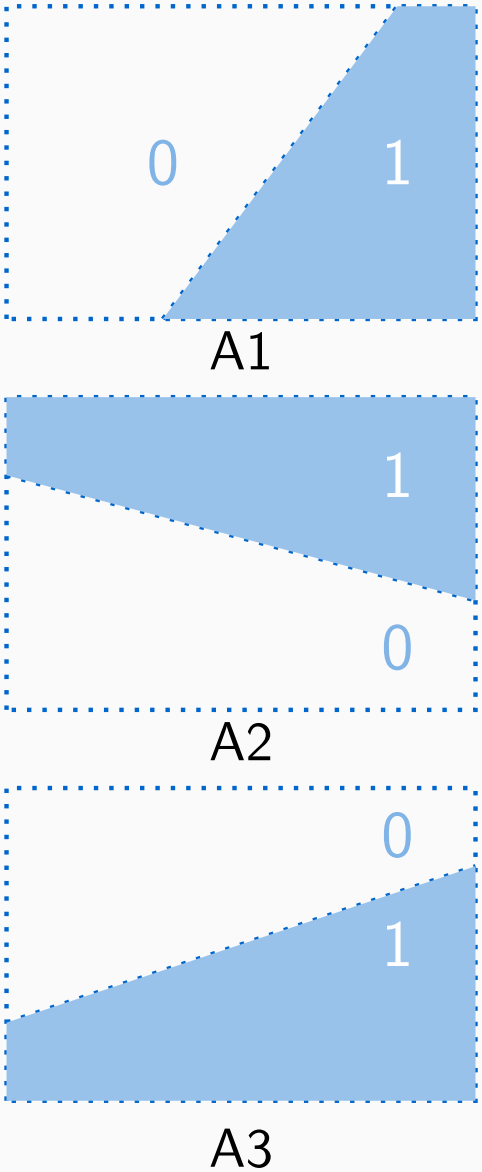
Binary Activation Codes



Each ReLU node splits the input space into two regions:

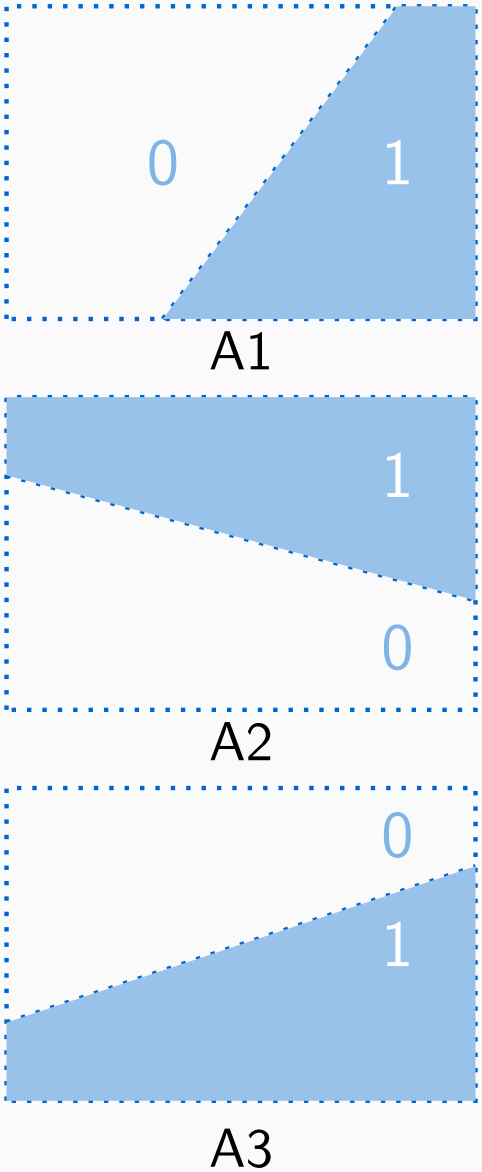
- Active (1)
- Inactive (0)

Binary Activation Codes



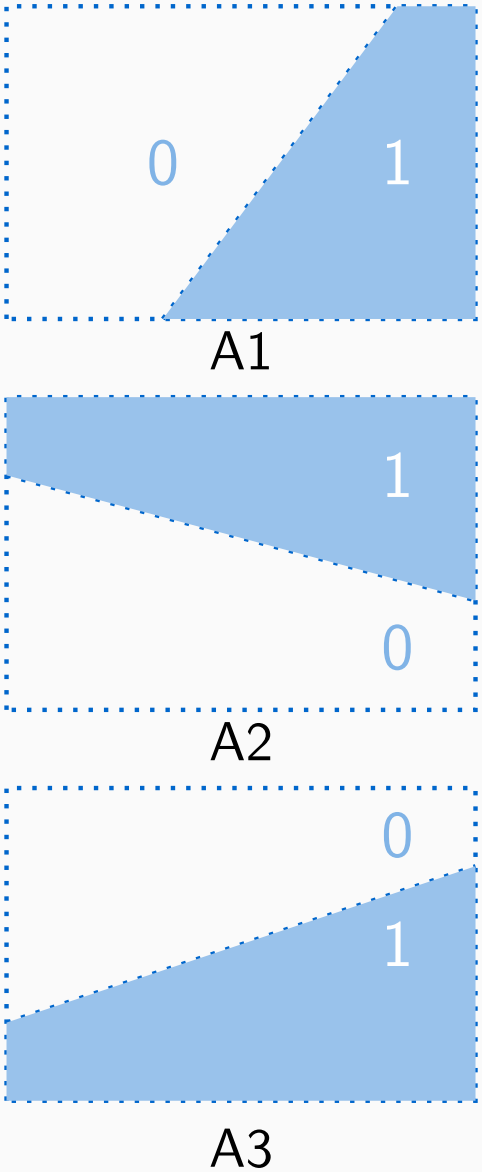
Here,intersection of regions defines linear regions

Binary Activation Codes



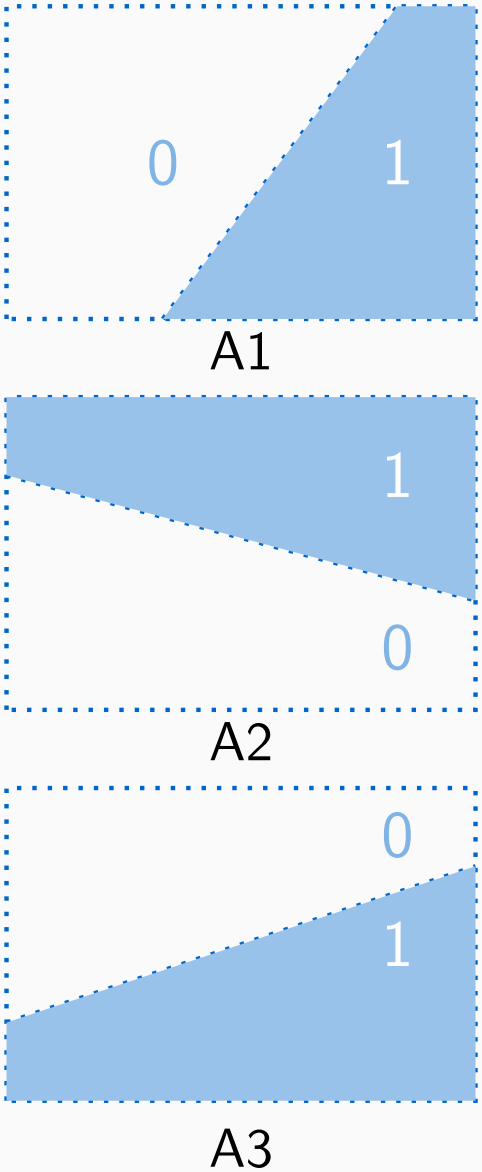
Here,intersection of regions defines linear regions

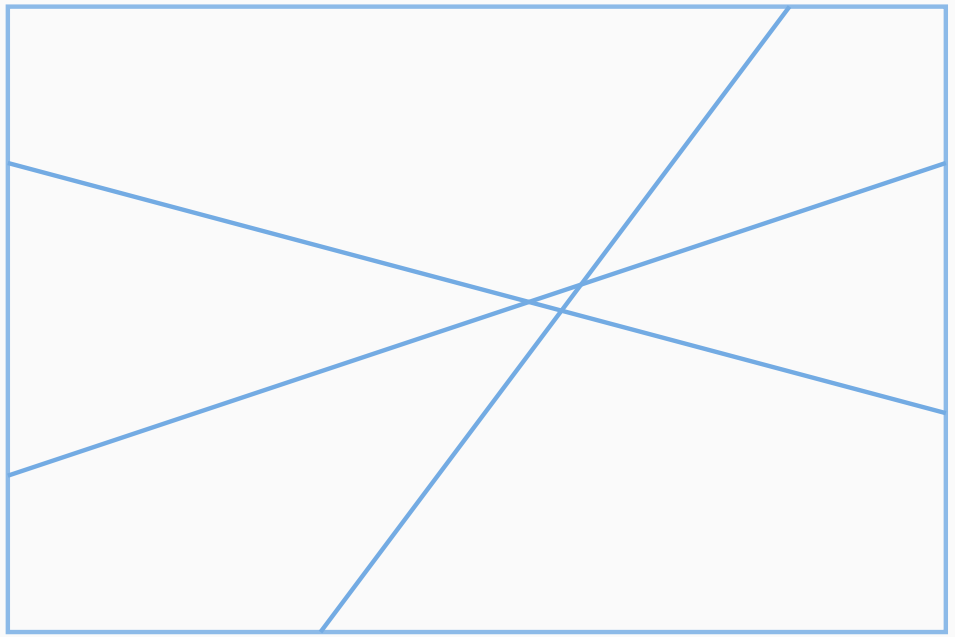
Binary Activation Codes



Here,intersection of regions defines linear regions

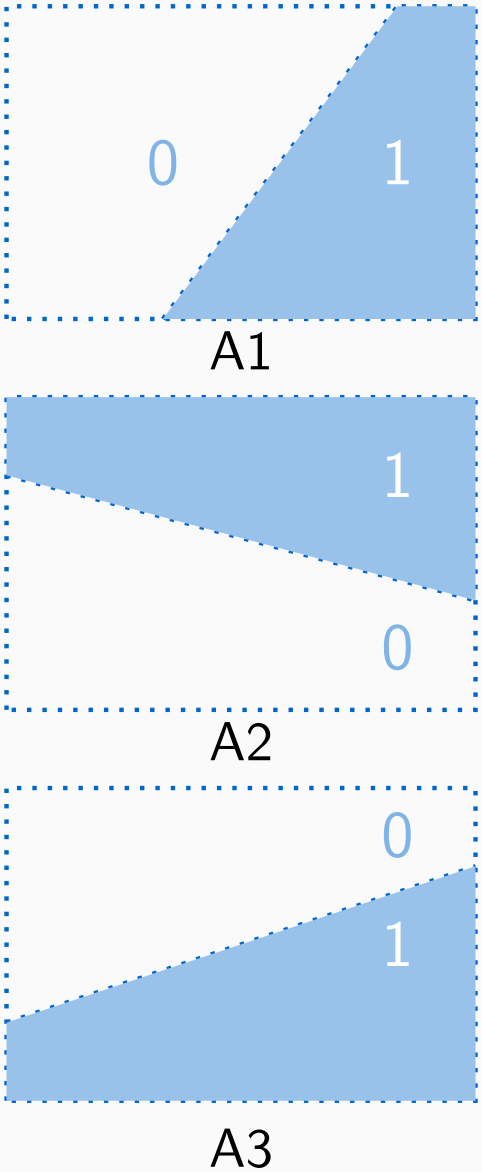
Binary Activation Codes

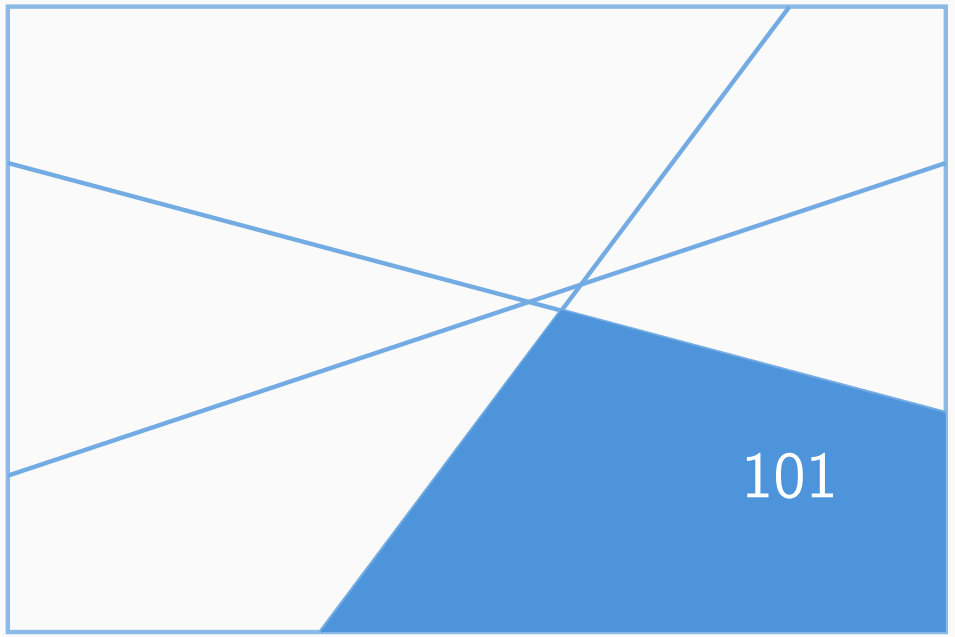




Here,intersection of regions defines linear regions

Binary Activation Codes





Here,intersection of regions defines linear regions

Neural Network and Linear regions

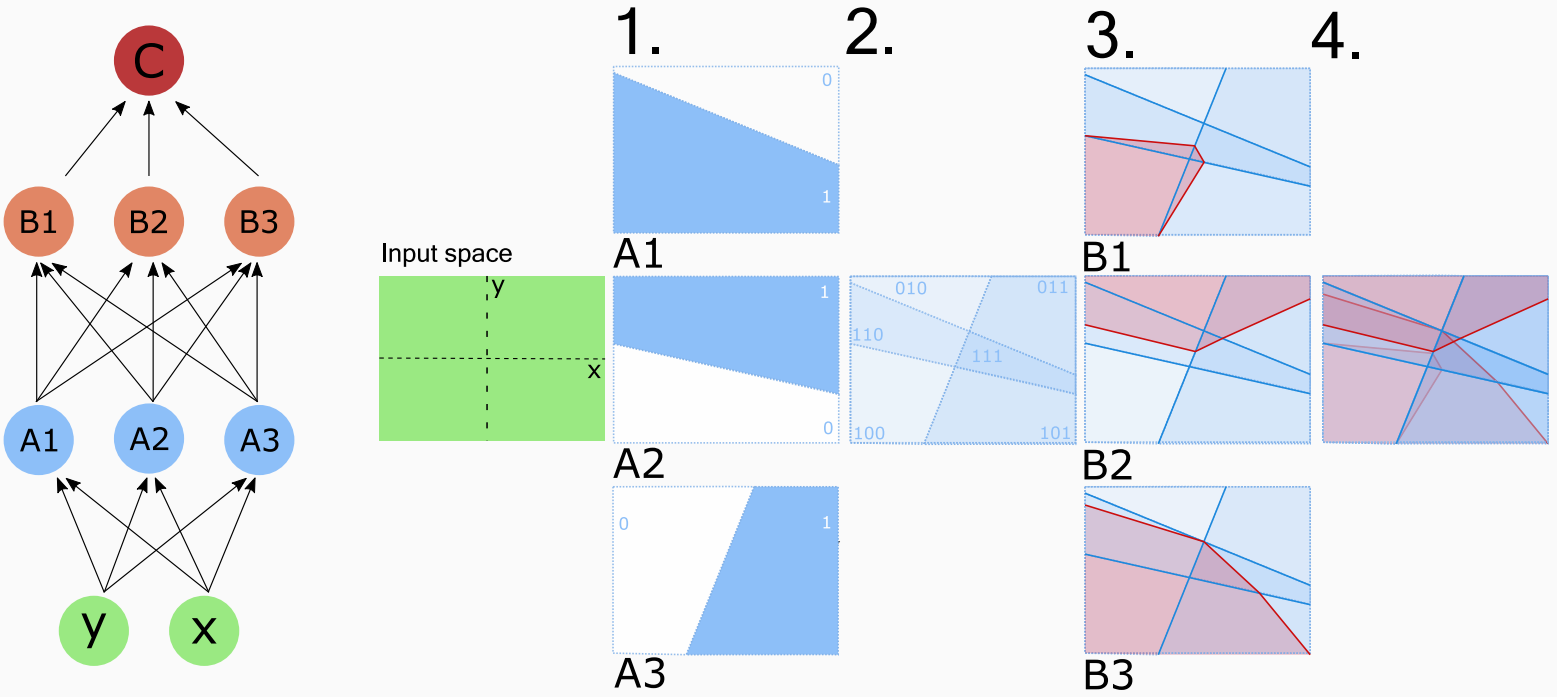
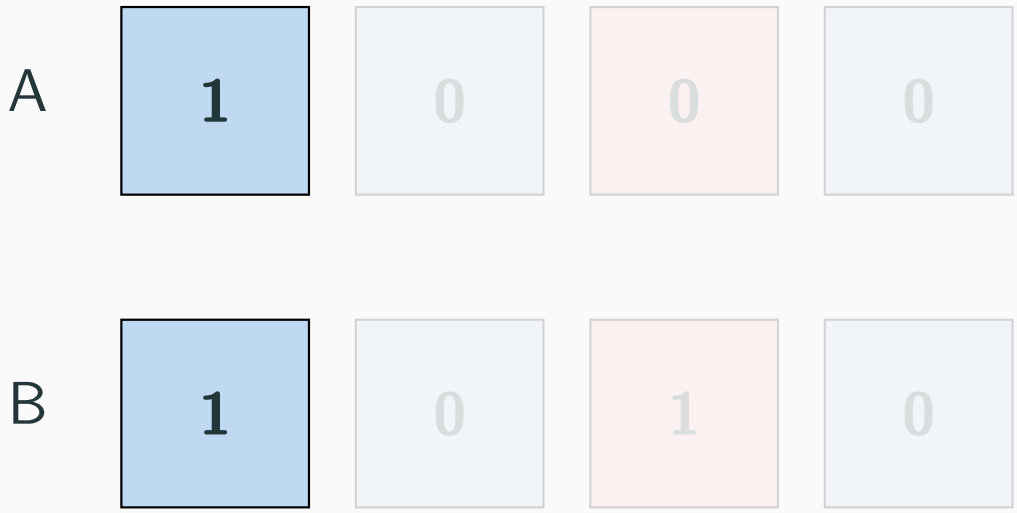


Figure 1: Visualizing binary activation codes of ReLU units and their corresponding linear regions.

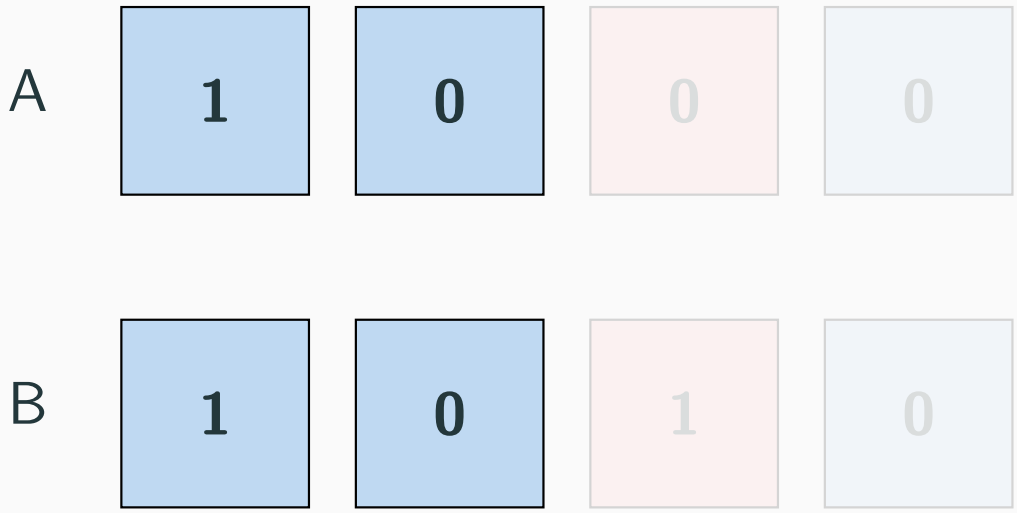
Hamming Distance

Hamming distance between two binary codes



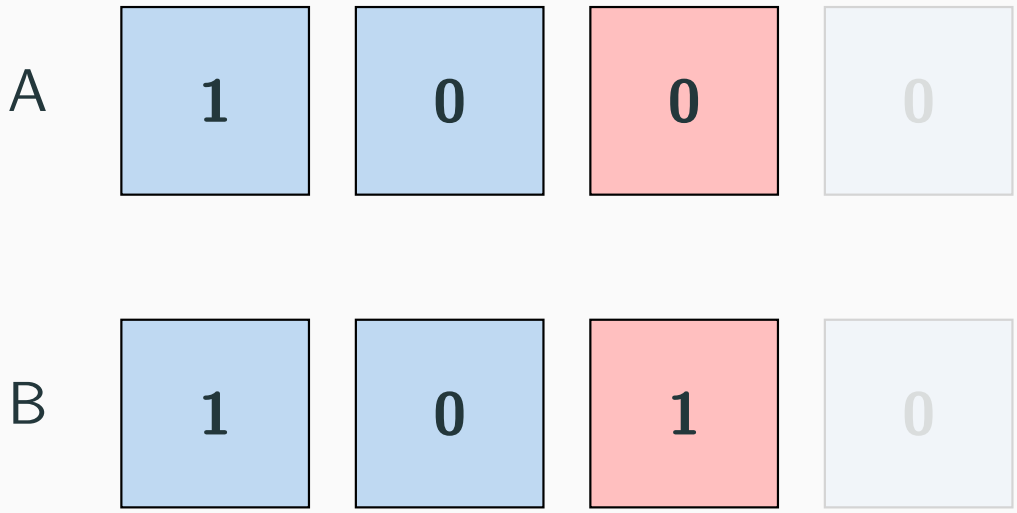
Hamming distance = 0

Hamming distance between two binary codes



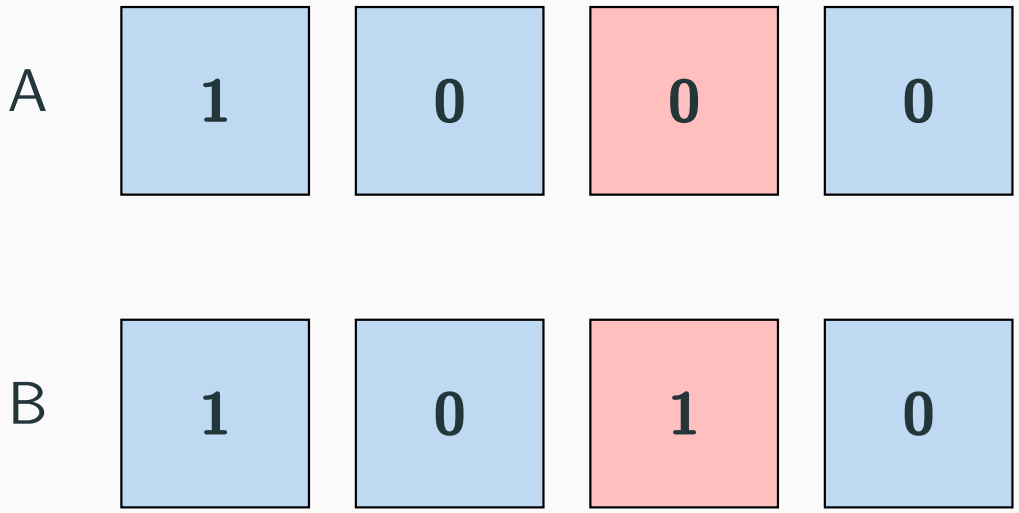
Hamming distance $= 0$

Hamming distance between two binary codes



Hamming distance $= 1$

Hamming distance between two binary codes



Hamming distance $= 1$

Kernel Matrix

Kernel Matrix Representation and Similarity

$$K_H = \begin{bmatrix} N_A - d_H(c_1, c_2) & \cdots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \cdots & N_A - d_H(c_N, c_N) \end{bmatrix}$$

- Each element represents $N_A - d_H(c_i, c_j)$, where d_H is the Hamming distance between **architecture codes**.
- The kernel matrix K_H encodes **pairwise similarities between architectures** based on Hamming distances.

Scoring Function

The performance score is calculated using the formula:

$$s = \log |K_H|$$

where K_H is the **kernel matrix** derived from activation overlaps.

- A **higher determinant** indicates more **similar architectures** and **better performance**.

Scoring Function

The performance score is calculated using the formula:

$$s = \log |K_H|$$

where K_H is the **kernel matrix** derived from activation overlaps.

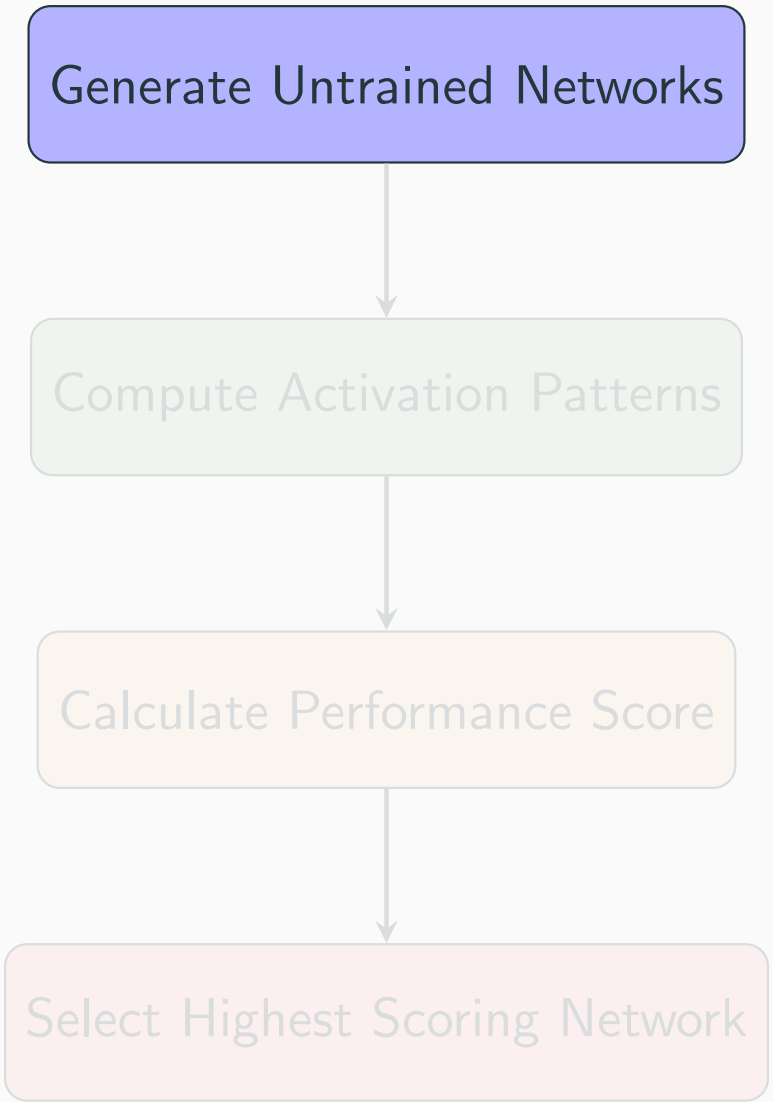
- A **higher determinant** indicates more **similar architectures** and **better performance**.

Proposed Method

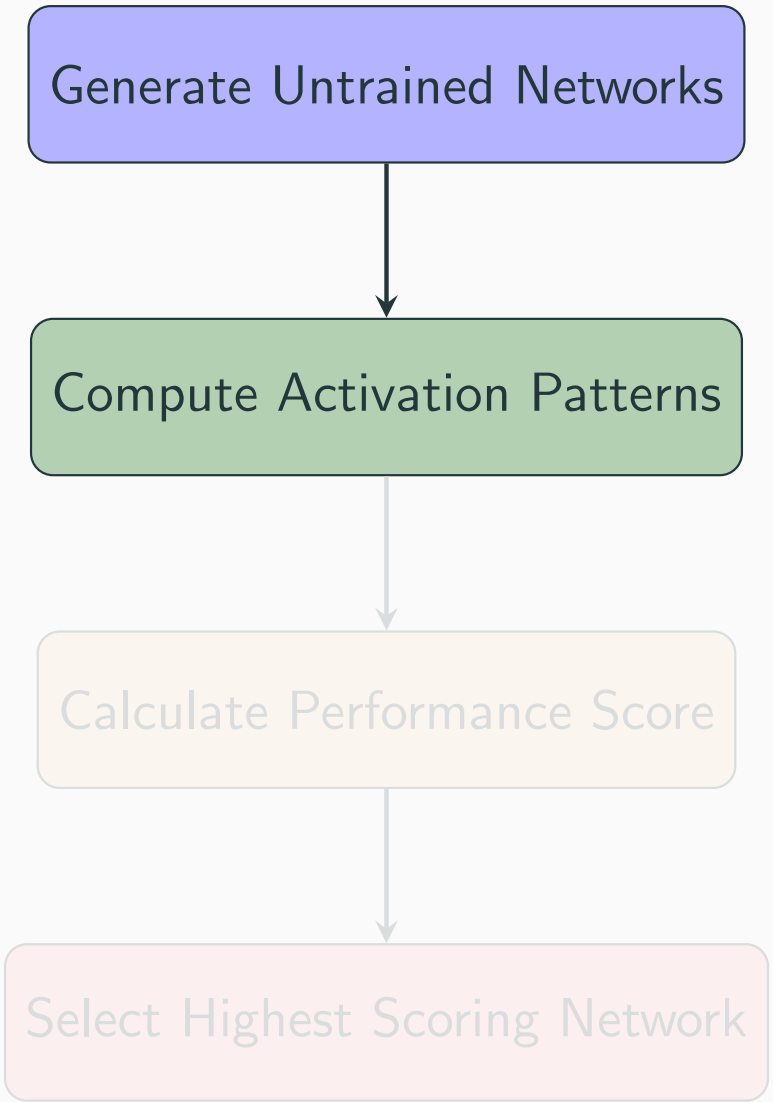
Proposed Method: NAS Without Training (NASWOT)

- Evaluates **untrained networks** using the activation patterns of ReLU layers.
- Uses **Hamming distance** of binary activation patterns to compute a performance score.
- $s = \log |K_H|$, where K_H is a **kernel matrix** derived from activation overlaps.

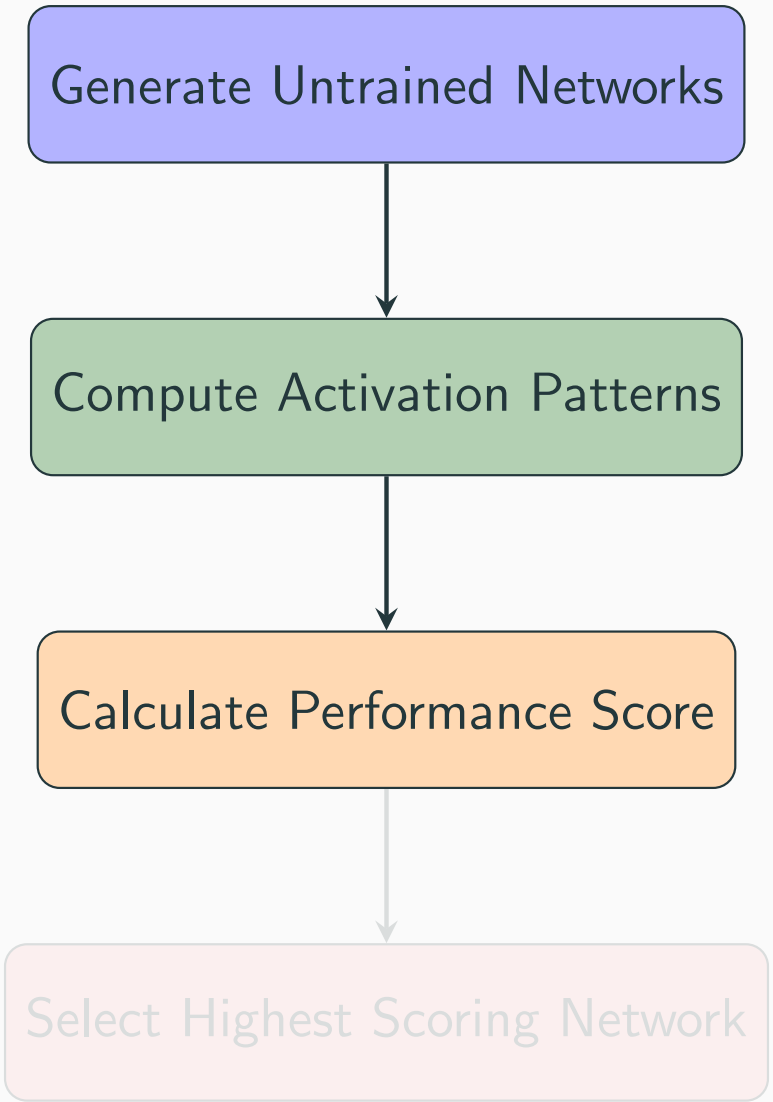
Flowchart: NASWOT Process



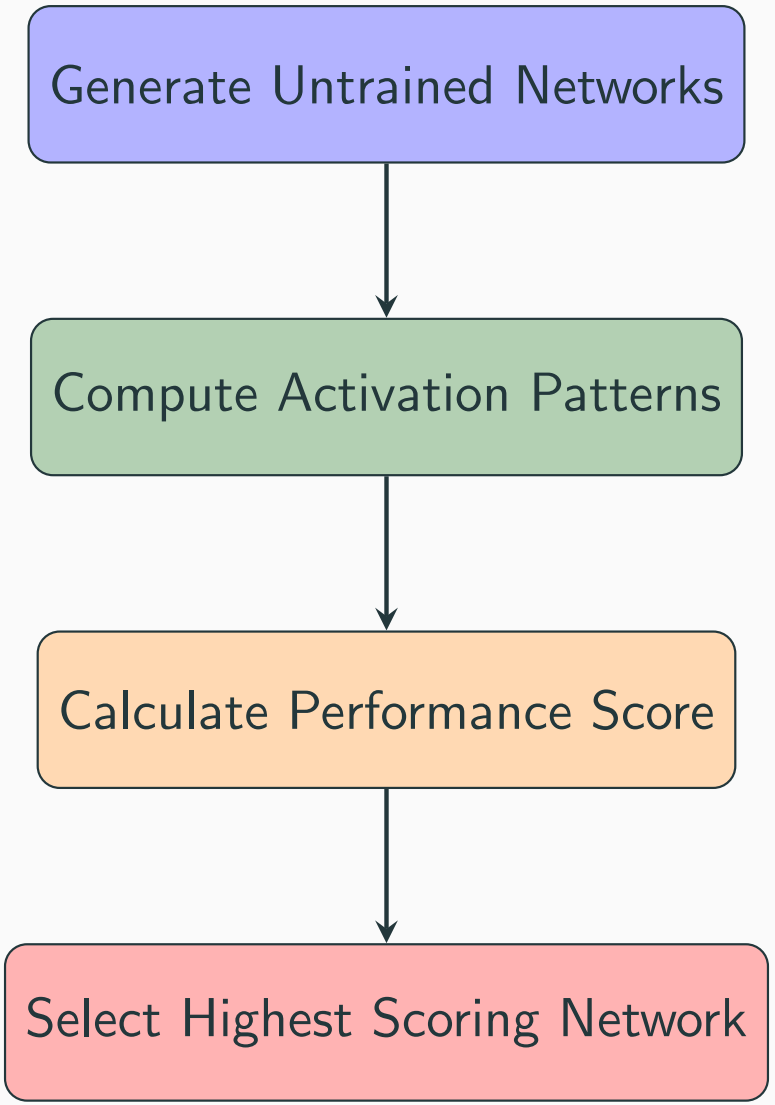
Flowchart: NASWOT Process



Flowchart: NASWOT Process



Flowchart: NASWOT Process



Algorithm

NASWOT Algorithm

Steps

1. Randomly generate architectures from the search space.
2. Compute scores for untrained architectures using Equation $s = \log |K_H|$.
3. Select the best architecture based on the computed score.
4. **Fast Execution:** Can be completed in seconds using a single GPU.

Algorithm 1: NASWOT

NASWOT Algorithm

Input: Random network generator *generator*, number of iterations N

Initialize: best_net $\leftarrow$ None, best_score $\leftarrow 0$

For $i = 1$ to N:

- net $\leftarrow$ generator.generate()
- score $\leftarrow$ net.score()
- **If** score $>$ best_score:
  - best_net, best_score $\leftarrow$ net, score

Output: chosen_net $\leftarrow$ best_net

Algorithm 1: NASWOT

NASWOT Algorithm

Input: Random network generator *generator*, number of iterations N

Initialize: best_net $\leftarrow$ None, best_score $\leftarrow 0$

For $i = 1$ to N:

- net $\leftarrow$ generator.generate()
- score $\leftarrow$ net.score()
- **If** score $>$ best_score:
  - best_net, best_score $\leftarrow$ net, score

Output: chosen_net $\leftarrow$ best_net

Algorithm 1: NASWOT

NASWOT Algorithm

Input: Random network generator *generator*, number of iterations N

Initialize: best_net $\leftarrow$ None, best_score $\leftarrow 0$

For $i = 1$ to N:

- net $\leftarrow$ generator.generate()
- score $\leftarrow$ net.score()
- **If** score $>$ best_score:
  - best_net, best_score $\leftarrow$ net, score

Output: chosen_net $\leftarrow$ best_net

Algorithm 1: NASWOT

NASWOT Algorithm

Input: Random network generator *generator*, number of iterations N

Initialize: best_net $\leftarrow$ None, best_score $\leftarrow 0$

For $i = 1$ to N:

- net $\leftarrow$ generator.generate()
- score $\leftarrow$ net.score()
- **If** score $>$ best_score:
  - best_net, best_score $\leftarrow$ net, score

Output: chosen_net $\leftarrow$ best_net

Algorithm 1: NASWOT

NASWOT Algorithm

Input: Random network generator *generator*, number of iterations N

Initialize: best_net $\leftarrow$ None, best_score $\leftarrow 0$

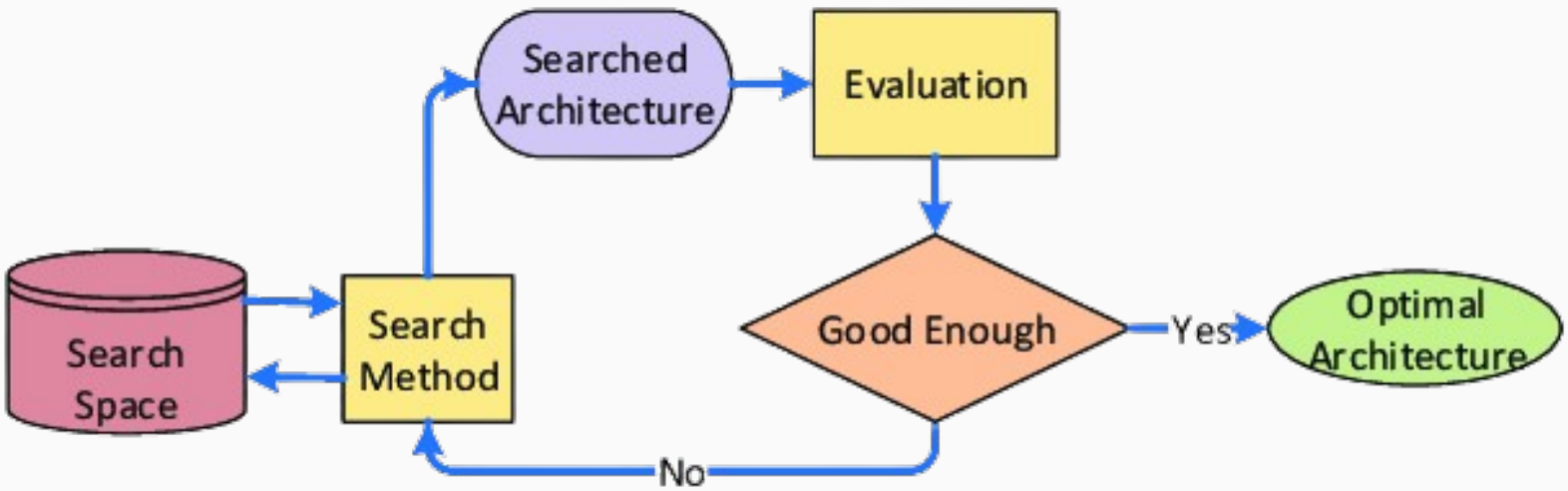
For $i = 1$ to N:

- net $\leftarrow$ generator.generate()
- score $\leftarrow$ net.score()
- **If** score $>$ best_score:
  - best_net, best_score $\leftarrow$ net, score

Output: chosen_net $\leftarrow$ best_net

NASWOT (Continued)

Algorithm 1: NASWOT



Traditional NAS vs NASWOT

Traditional NAS vs NASWOT

Features	Traditional NAS	NASWOT
Training Approach	**Generator network** for proposing and training.	**No training** is required.

Traditional NAS vs NASWOT

Features	Traditional NAS	NASWOT
Training Approach	**Generator network** for proposing and training.	**No training** is required.
Computational Cost	High **cost** due to training.	**Low cost** with no training.

Traditional NAS vs NASWOT

Features	Traditional NAS	NASWOT
Training Approach	**Generator network** for proposing and training.	**No training** is required.
Computational Cost	High **cost** due to training.	**Low cost** with no training.
Training Time	**Longer time** for each architecture.	**Faster** iteration without training.

Traditional NAS vs NASWOT

Features	Traditional NAS	NASWOT
Training Approach	**Generator network** for proposing and training.	**No training** is required.
Computational Cost	High **cost** due to training.	**Low cost** with no training.
Training Time	**Longer time** for each architecture.	**Faster** iteration without training.
Performance	High accuracy, but requires **more resources**.	High performance with fewer resources.

Let's See Some Accuracy Scores!

Performance Comparsions

Performance on NAS-Bench-101

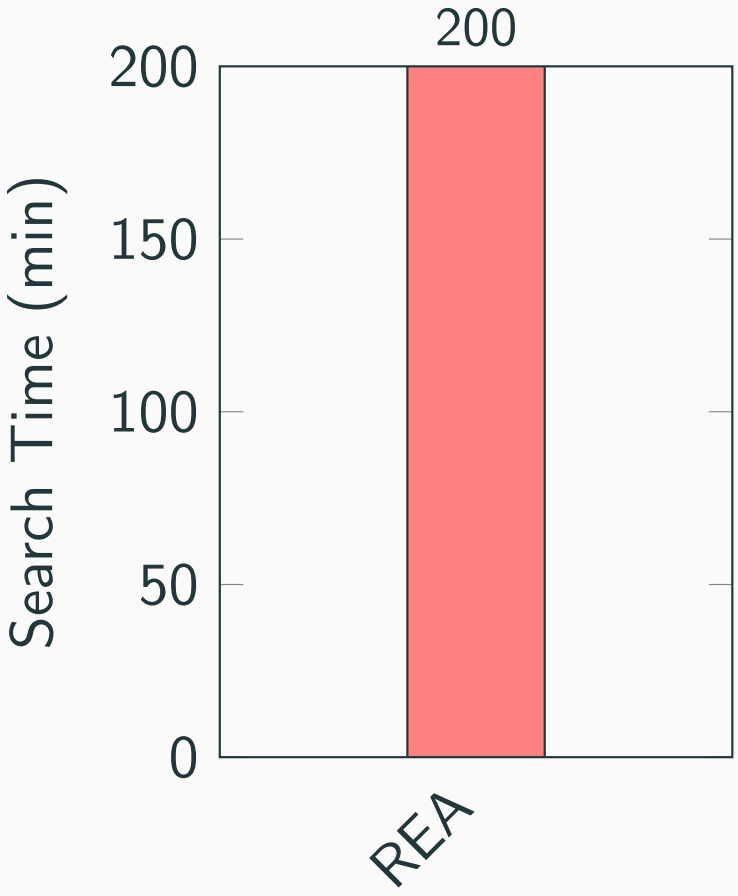


Figure 1: Search Time

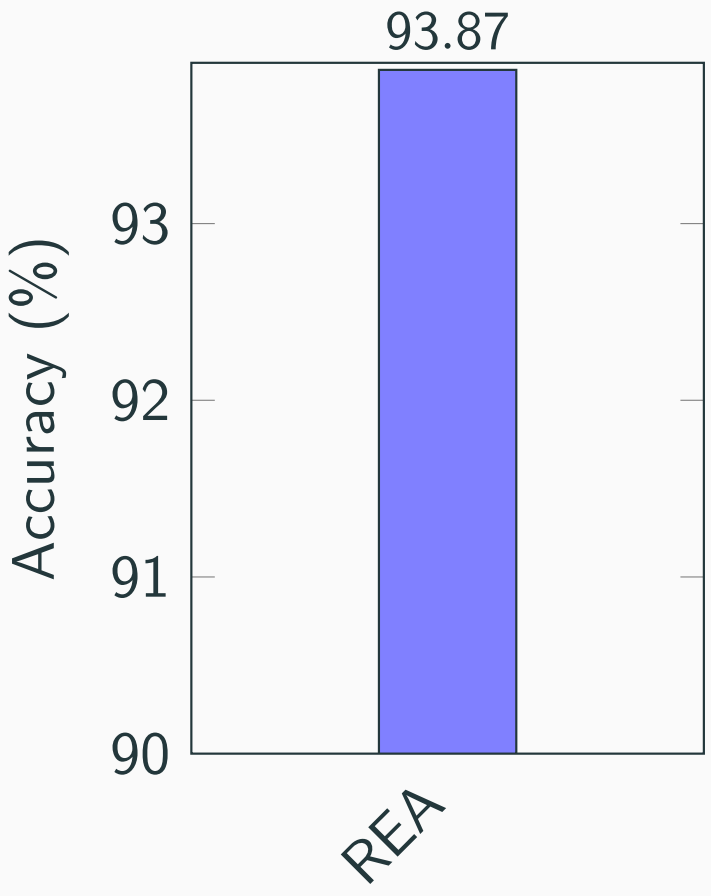


Figure 2: CIFAR-10 Accuracy

Performance on NAS-Bench-101

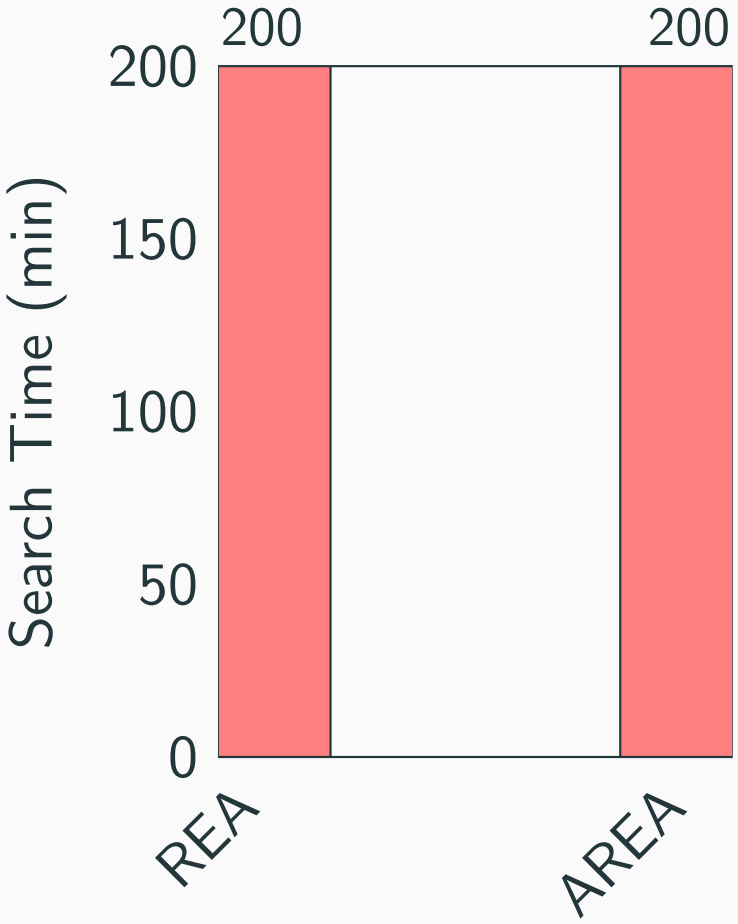


Figure 1: Search Time

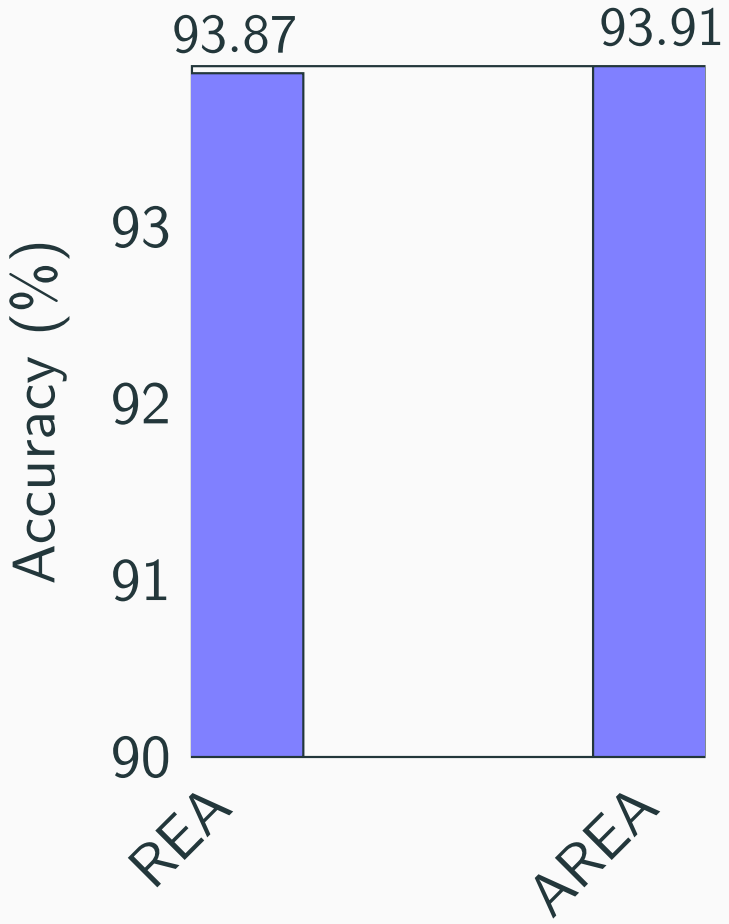


Figure 2: CIFAR-10 Accuracy

Performance on NAS-Bench-101

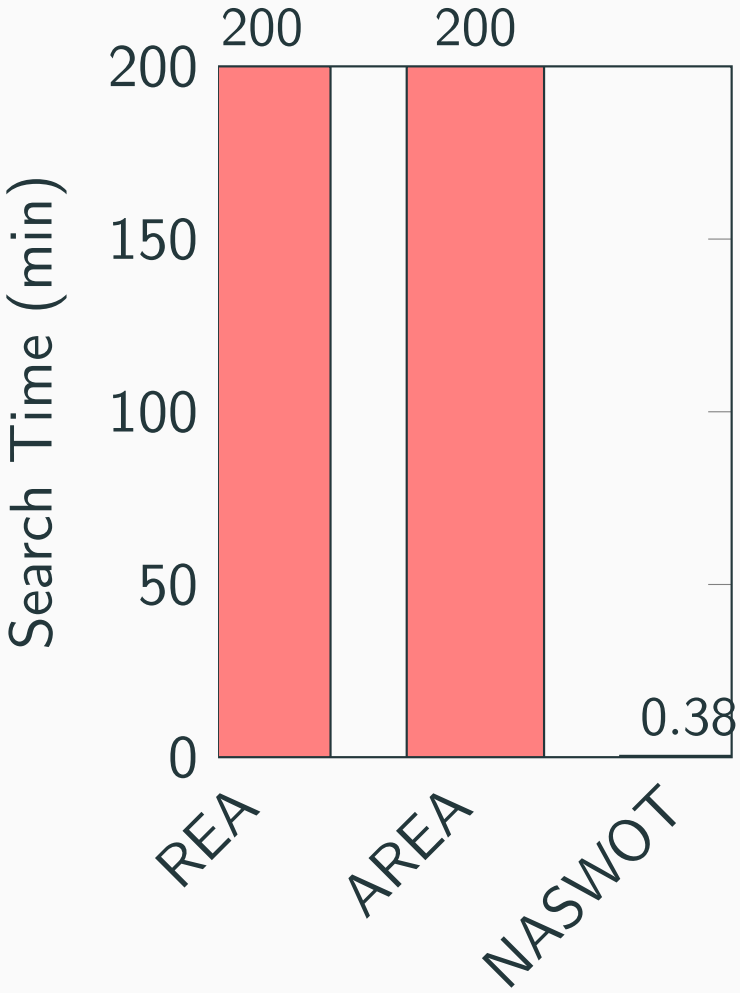


Figure 1: Search Time

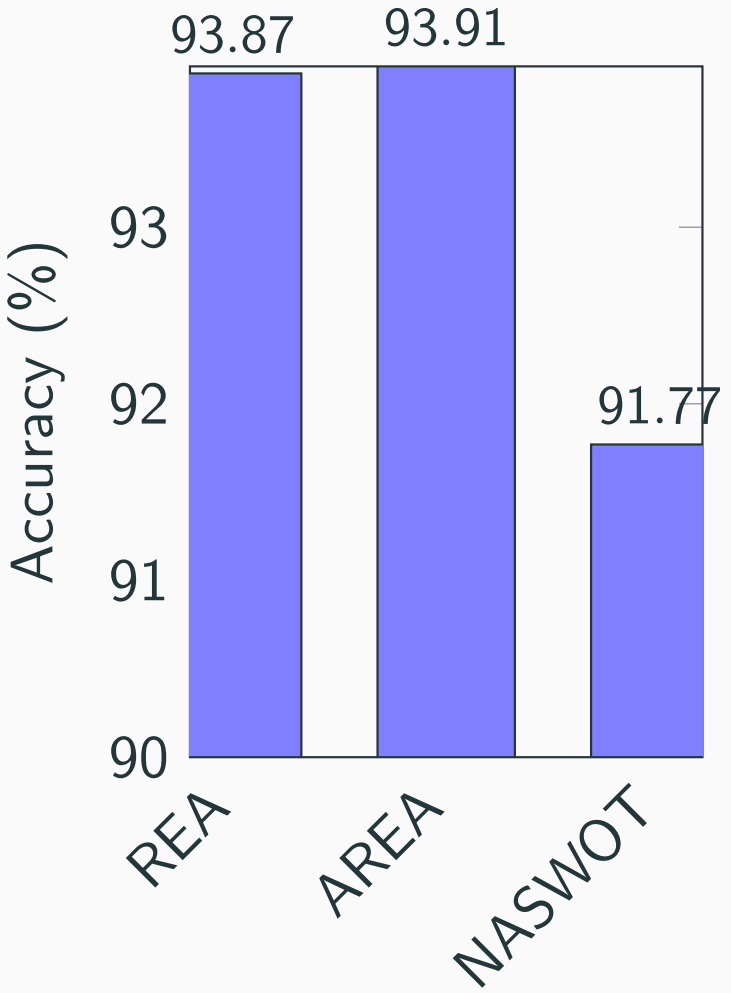


Figure 2: CIFAR-10 Accuracy

Performance on NAS-Bench-101

Method	Search (s)	CIFAR-10 Accuracy
REA	12000	**93.87±0.22**
AREA	12000	**93.91±0.29**
NASWOT (N=100)	**23**	**91.77±0.05**

Table 1: Comparison on NAS-Bench-101

- NASWOT is much faster and performs comparably to **REA** and **AREA**.

Sample Size Effects

Performance on NAS-Bench-201

Method	Search Time (s)	CIFAR-10 Test Accuracy
NASWOT (N=10)	3.05	92.44±1.13

Performance on NAS-Bench-201

Method	Search Time (s)	CIFAR-10 Test Accuracy
NASWOT (N=10)	3.05	92.44±1.13
NASWOT (N=100)	30.01	92.81±0.99

Performance on NAS-Bench-201

Method	Search Time (s)	CIFAR-10 Test Accuracy
NASWOT (N=10)	3.05	92.44±1.13
NASWOT (N=100)	30.01	92.81±0.99
NASWOT (N=1000)	306.19	**92.96±0.81**

- Accuracy improves with **sample size N.**
- NASWOT outperforms weight-sharing methods with much **less search time.**

Performance on NAS-Bench-201

Method	Search Time (s)	CIFAR-10 Test Accuracy
NASWOT (N=10)	3.05	92.44±1.13
NASWOT (N=100)	30.01	92.81±0.99
NASWOT (N=1000)	306.19	**92.96±0.81**

- Accuracy improves with **sample size N.**
- NASWOT outperforms weight-sharing methods with much **less search time.**

Search Time vs. Final Accuracy

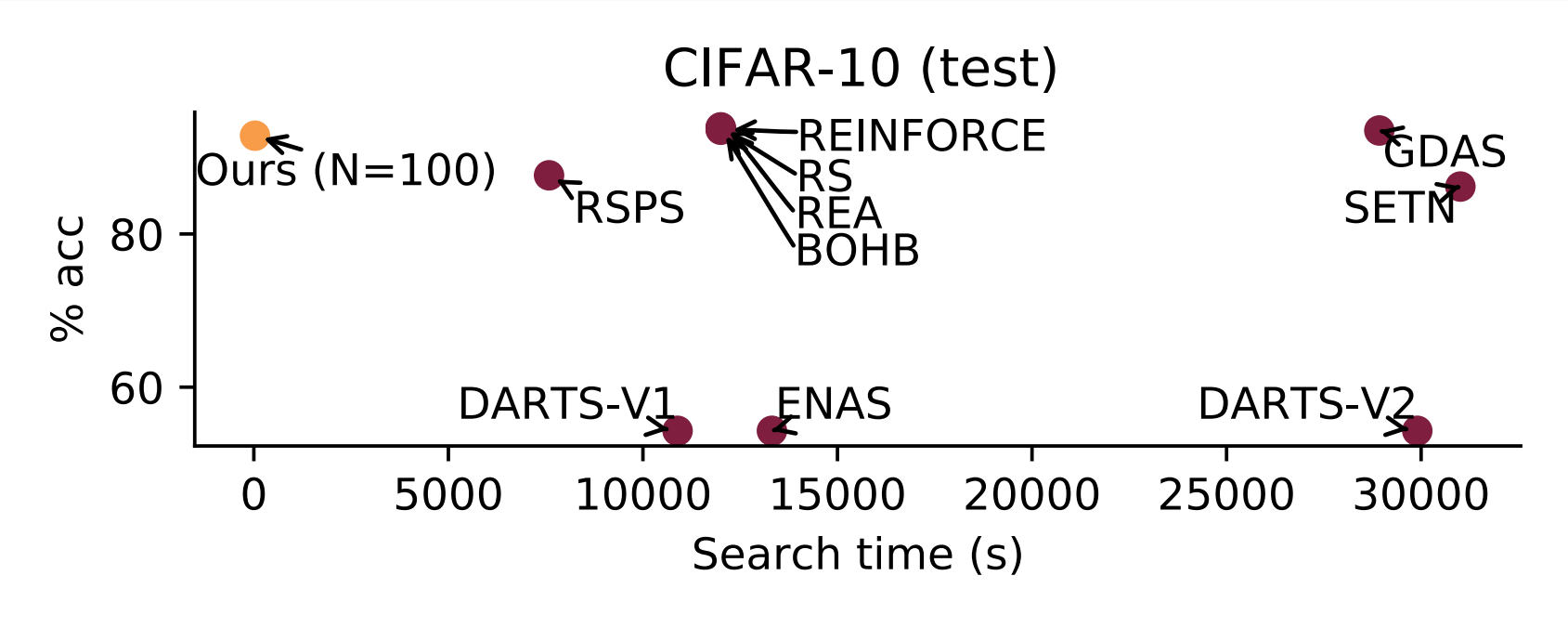
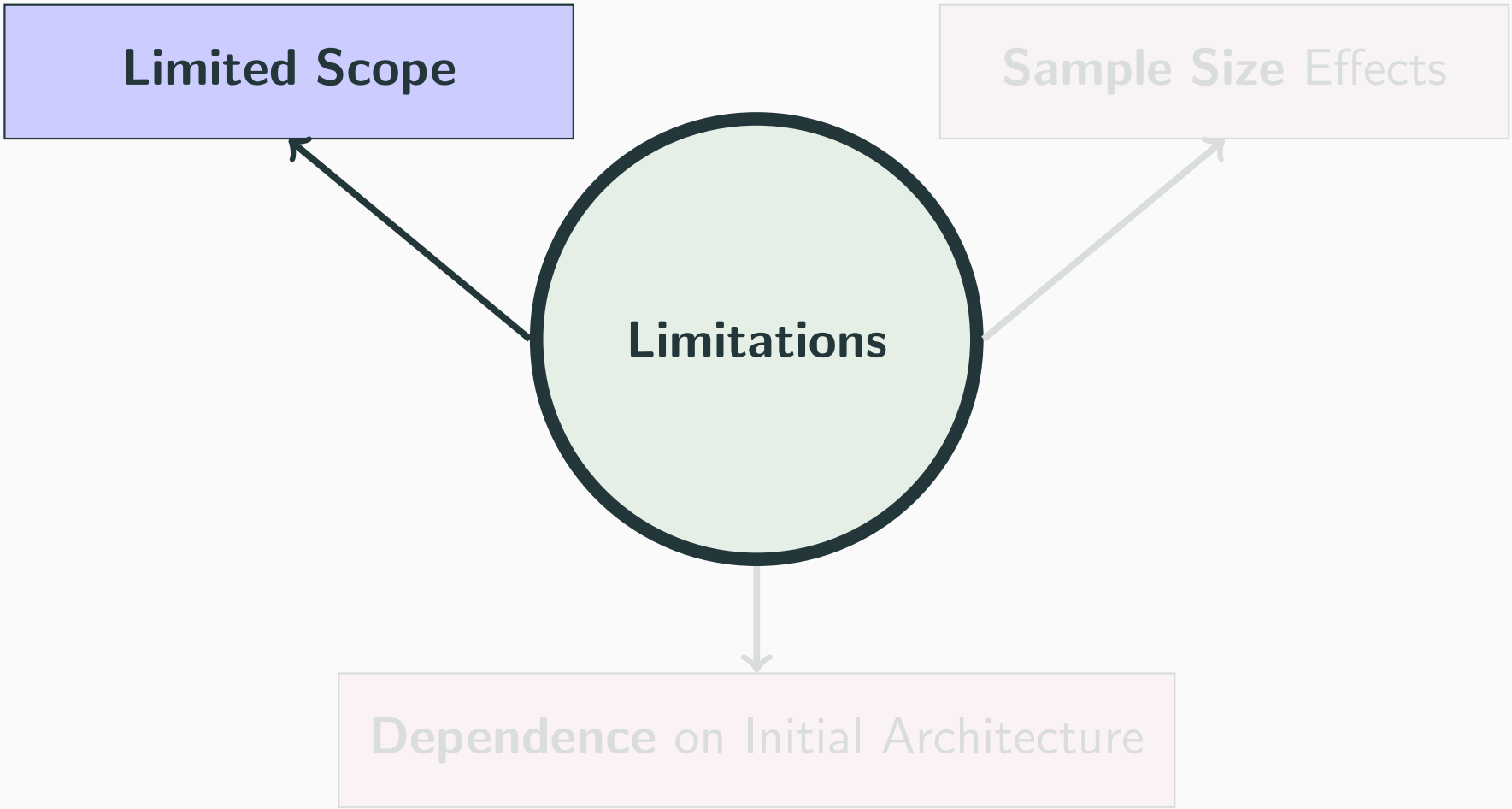


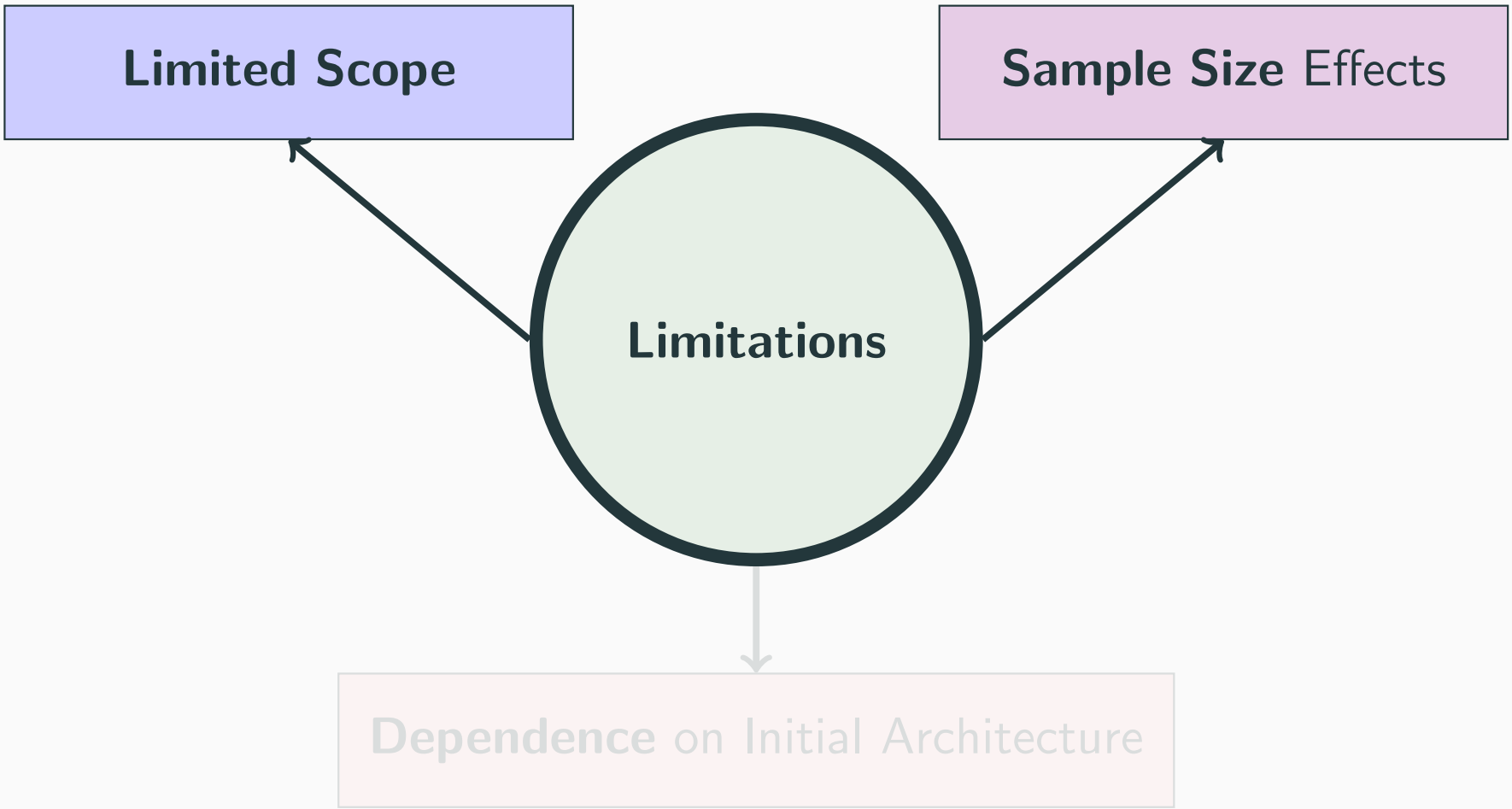
Figure 2: Plot showing the search time

Limitations of NASWOT

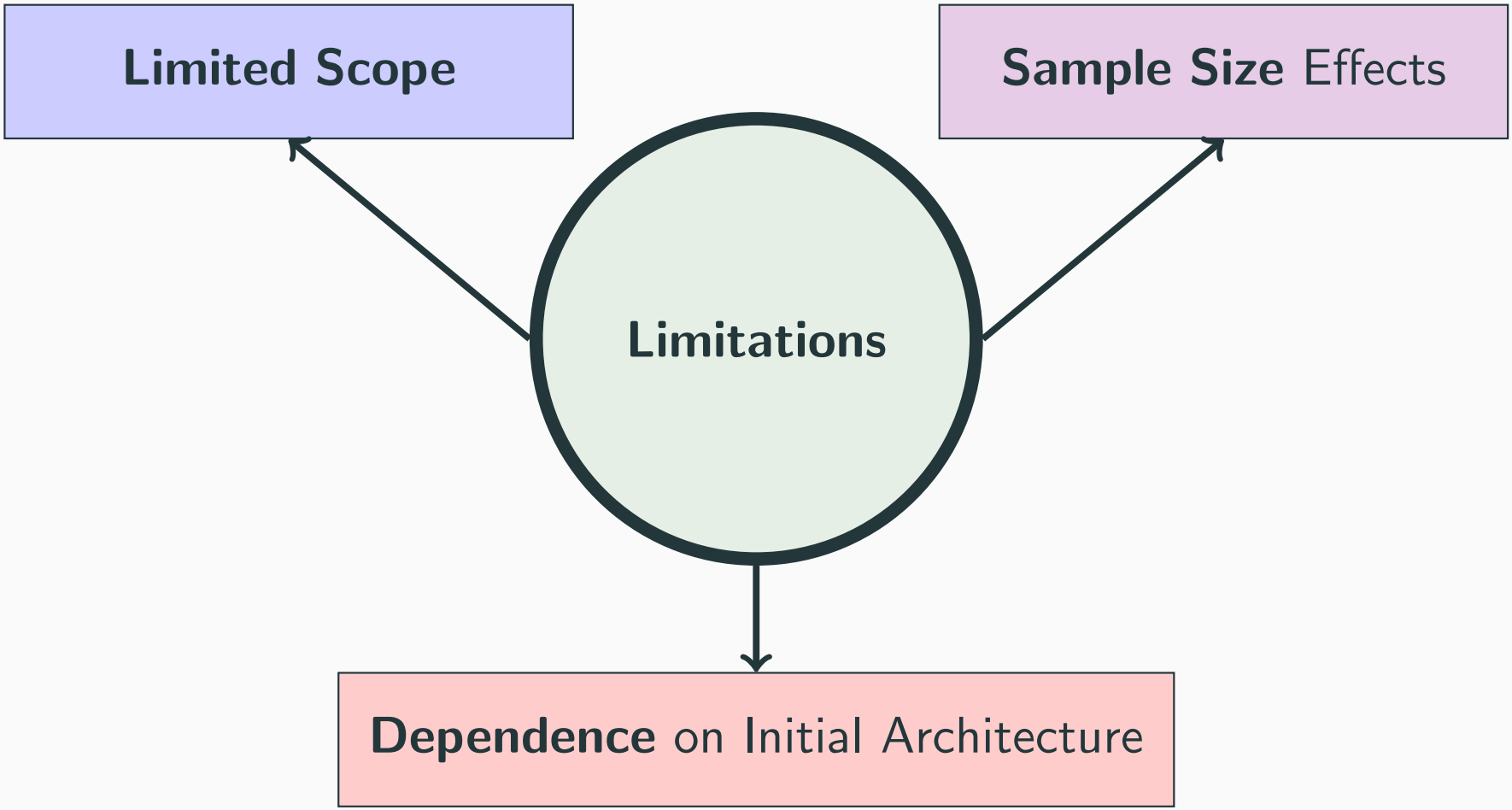
Limitations of NASWOT



Limitations of NASWOT



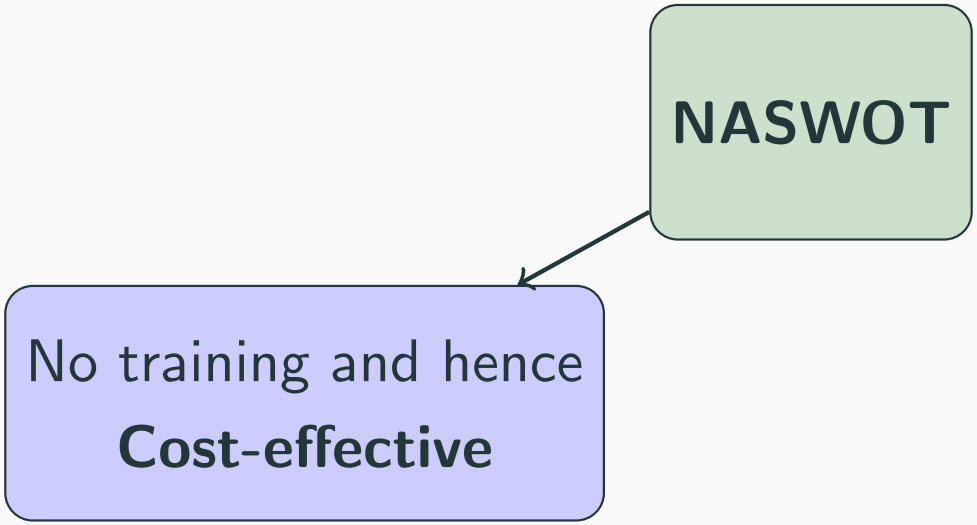
Limitations of NASWOT



With every challenge, a **new opportunity** emerges!

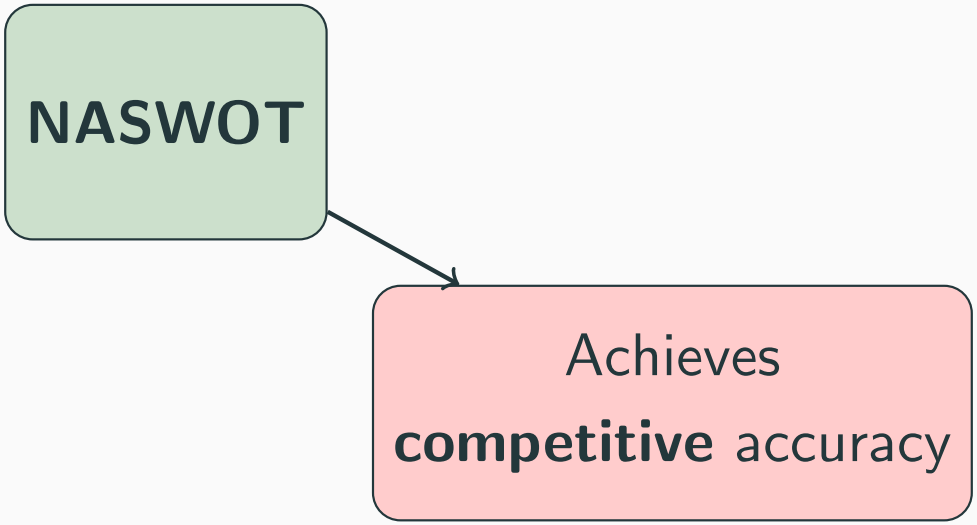
Let's Revisit the Highlights of Our Journey

Future is Here



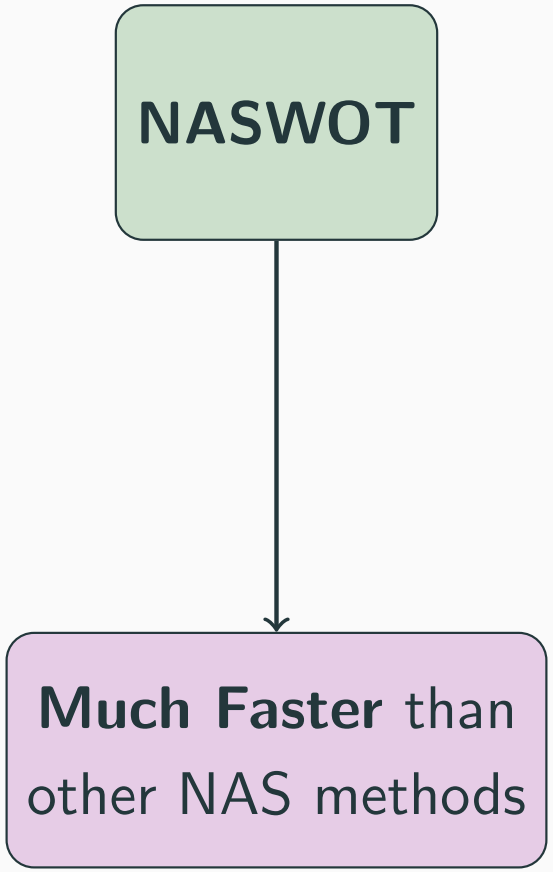
- NASWOT - No need for training architectures and hence **cost effective alternative** to traditional NAS methods.

Future is Here



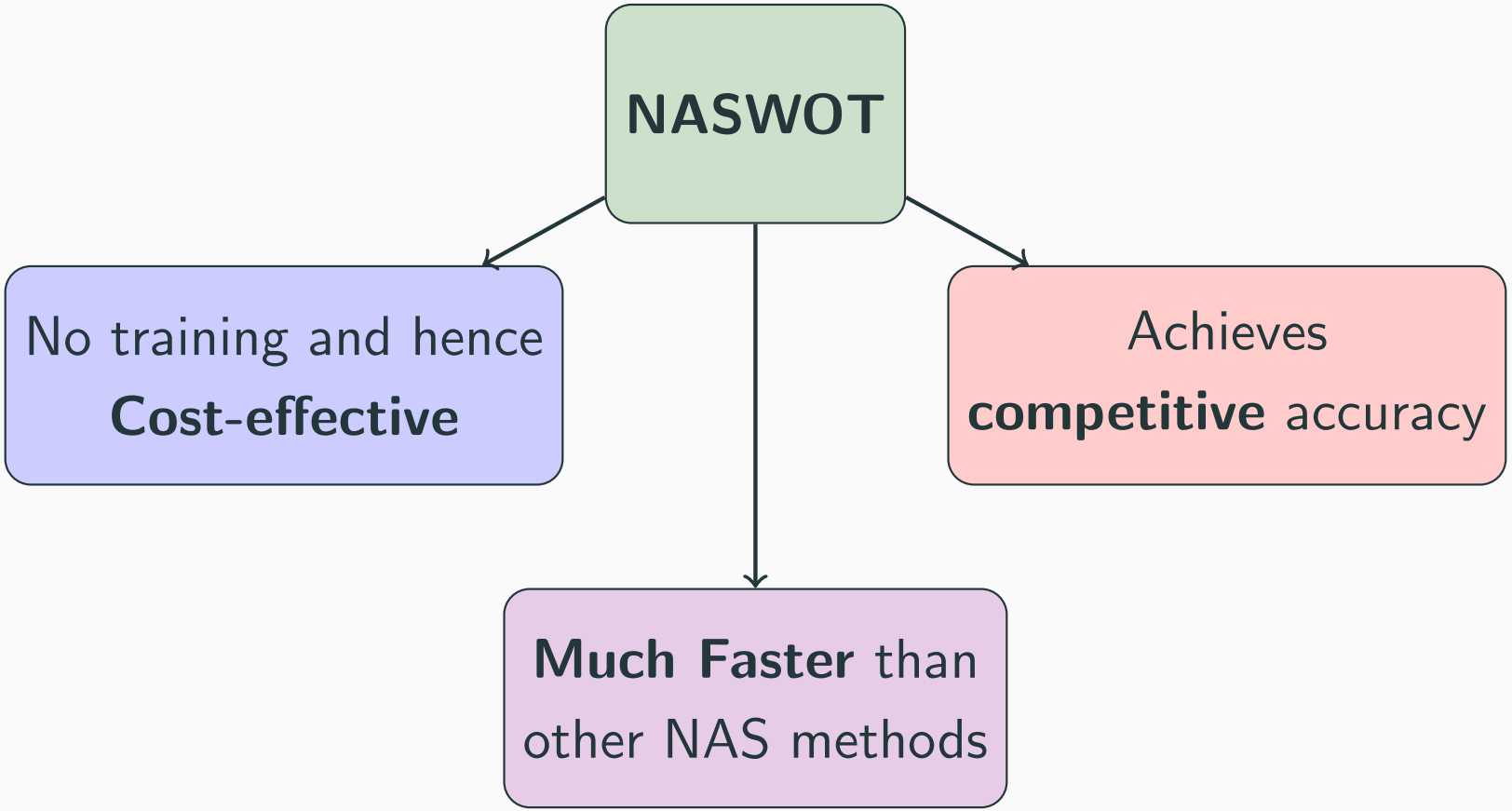
- **No need** for training architectures, achieves **competitive accuracy**

Future is Here



- **NASWOT's performance** is comparable to other NAS methods like **REA and AREA**, but much faster.

Future is Here



- **In the future,** NASWOT could be **extended to more tasks** and further optimized for different datasets and hardware.

References

References

1. Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *International Conference on Learning Representations*.
2. Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2018). SMASH: One-Shot Model Architecture Search through HyperNetworks. *International Conference on Learning Representations*.
3. Cai, H., Zhu, L., & Han, S. (2019). ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *International Conference on Learning Representations*.
4. Dong, X., & Yang, Y. (2019). One-shot neural architecture search via self-evaluated template network. *Proceedings of the International Conference on Computer Vision*.

References (Continued)

5. Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four GPU hours. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.
6. Chen, W., Gong, X., & Wang, Z. (2021). Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective. *International Conference on Learning Representations*.
7. Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., & Lane, N. D. (2021). Zero-Cost Proxies for Lightweight NAS. *International Conference on Learning Representations*.
8. Dong, X., & Yang, Y. (2020). NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. *International Conference on Learning Representations*.

Thank You For Your Attention!