Neural Architecture Search without Training

Dmitry Protasov

MIPT, 2023

February 13, 2024

Motivation

Why NAS without Training?

Traditional Neural Architecture Search (NAS) methods require extensive computational resources and time due to the need for training each candidate architecture. This process can be prohibitively expensive and slow. Our goal is to explore a methodology that allows for the quick evaluation of neural architectures without the need for full training, significantly reducing computational costs and time.

Advancements in NAS

- **Early NAS Efforts:** NAS using RNN controllers, requiring significant computational resources (800 GPUs for 28 days)
- **Modular Search:** Zoph et al. (2018) improved efficiency by searching over neural building blocks (cells), reducing resources (500 GPUs over 4 days).
- **ENAS and Weight Sharing:** Pham et al. (2018): Efficient NAS (ENAS) with weight sharing, drastically lowering search costs to half a day on a single GPU
- **Limitations and Baselines:** Evidence suggests weight sharing might hinder finding optimal architectures, with random search emerging as a strong baseline.
- **Exploring Training-Free Heuristics:** Recent work incorporates training-free heuristics into NAS to enhance performance, exploring novel directions like neural network Gaussian processes and the neural tangent kernel.

Benchmarks

NAS Benchmarks

- NAS-Bench-101 – 423,624 networks, trained on CIFAR-10 dataset
- NAS-Bench-201 – 15,625 networks trained multiple times on CIFAR-10, CIFAR-100, and ImageNet-16-120
- NATS-Bench
  - a topology search space (NATS-Bench TSS)
  - size search space (NATS-Bench SSS)
- Network Design Spaces (NDS)
  - Where the NAS benchmarks aim to compare search algorithms, NDS aims to compare the search spaces themselves
  - NDS-AmoebaNet, NDSDARTS, NDS-ENAS, NDS-NASNet, NDS-PNAS

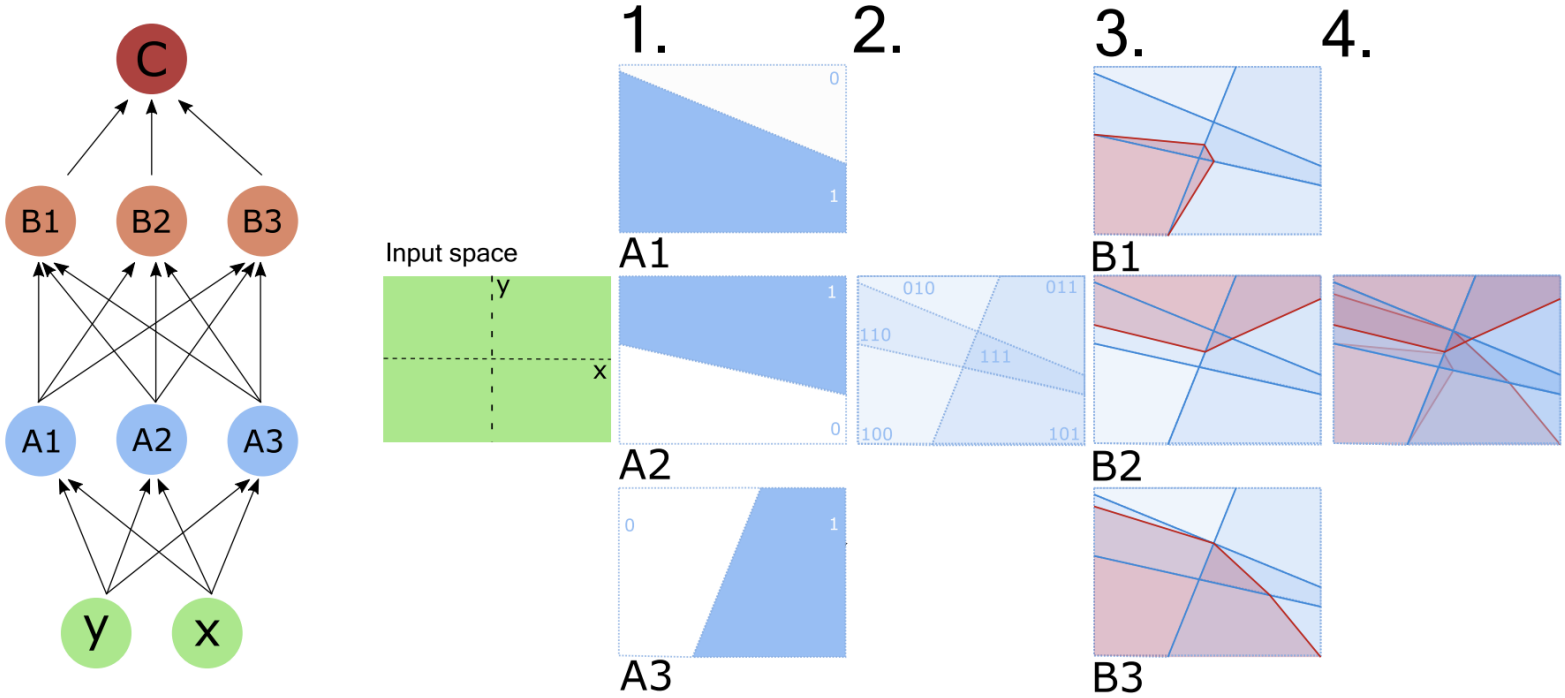

Figure: Visualising how binary activation codes of ReLU units correspond to linear regions. 1: Each ReLU node Ai splits the input into an active and inactive region We label the active region 1 and inactive 0. 4: Each linear region at a given node can be uniquely defined by the activation pattern of all the ReLU nodes that preceded it

Scoring Networks at Initialisation

The objective is to score network architectures at initialisation to predict their final trained accuracy without the costly training process.

- A neural network with ReLU units assigns binary codes to inputs, indicating active and inactive states.
- A mini-batch $X = \{x_i\}_{i=1}^{N}$ generates binary codes $\{c_i\}$ per input x_i, defining the linear regions.
- The Hamming distance between these codes measures input dissimilarity, used to construct a kernel matrix K_H.

$$K_H = \begin{pmatrix} N_A - d_H(c_1, c_1) & \cdots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \cdots & N_A - d_H(c_N, c_N) \end{pmatrix} \tag{1}$$

- Networks are scored by $s = \log |K_H|$, relating to the likelihood of higher final accuracy. This score has shown positive correlation with validation accuracy across multiple datasets and search spaces.

Scoring Networks at Initialisation

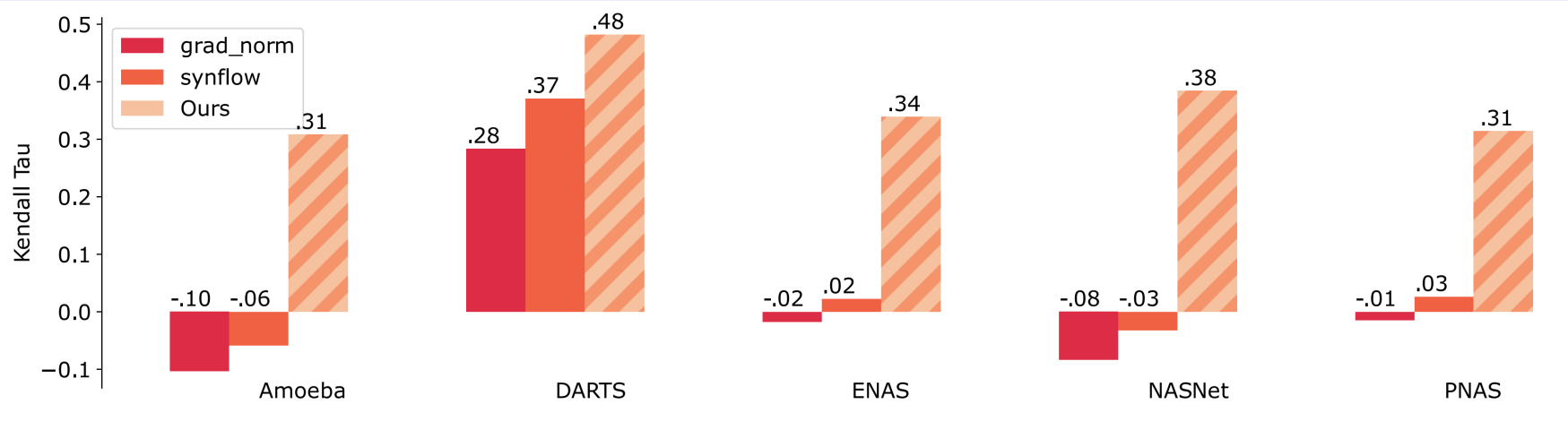

Figure: Kendall's Tau correlation across each of the NDS CIFAR-10 search spaces vs alternative measures: grad norm (Euclidean-norm of the gradients over random mini-batch data) and synflow (Tanaka et al. (2020))

Scoring Networks at Initialisation

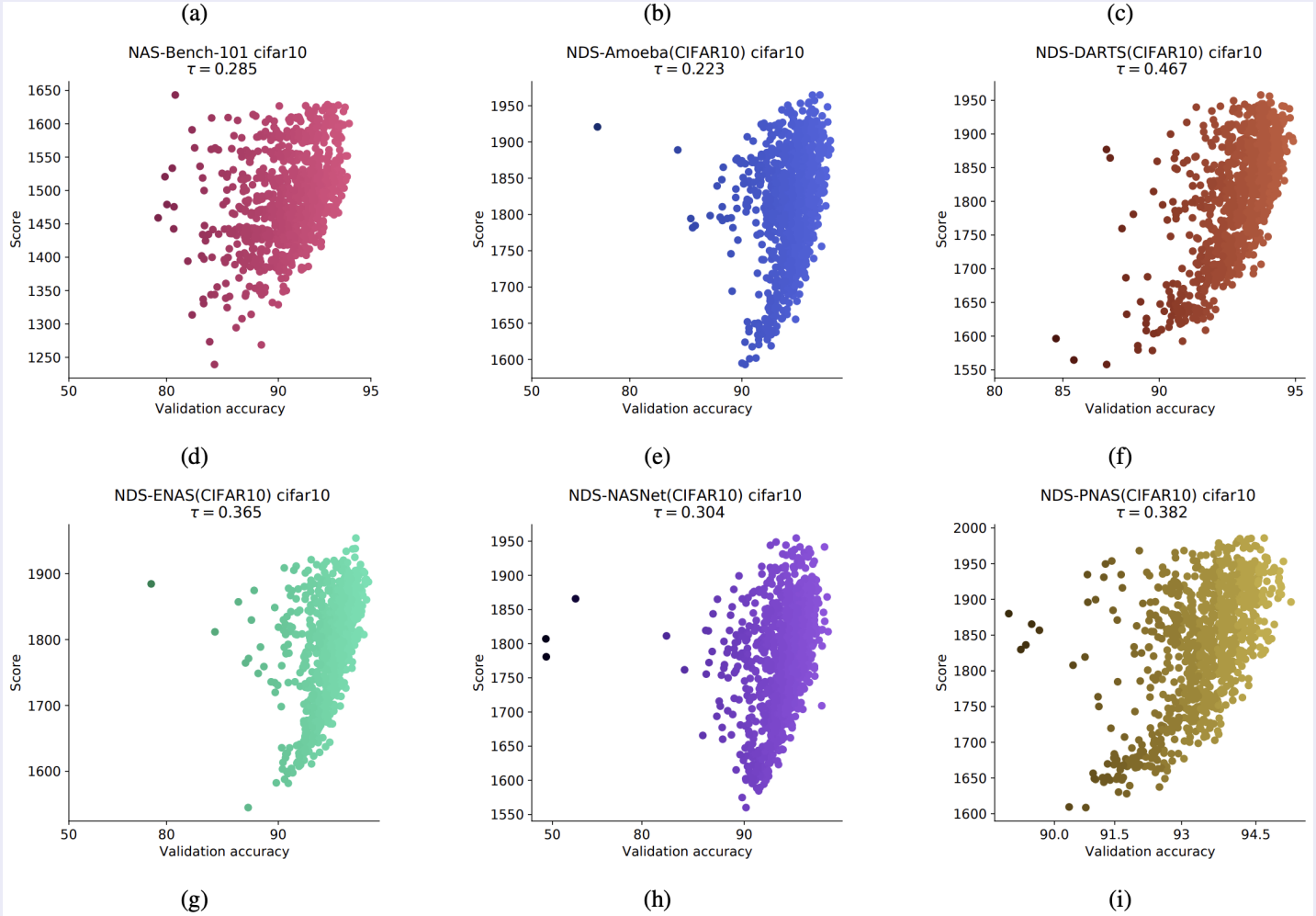

Figure: Plots of our score for randomly sampled untrained architectures against validation accuracy when trained

Scoring Networks at Initialisation

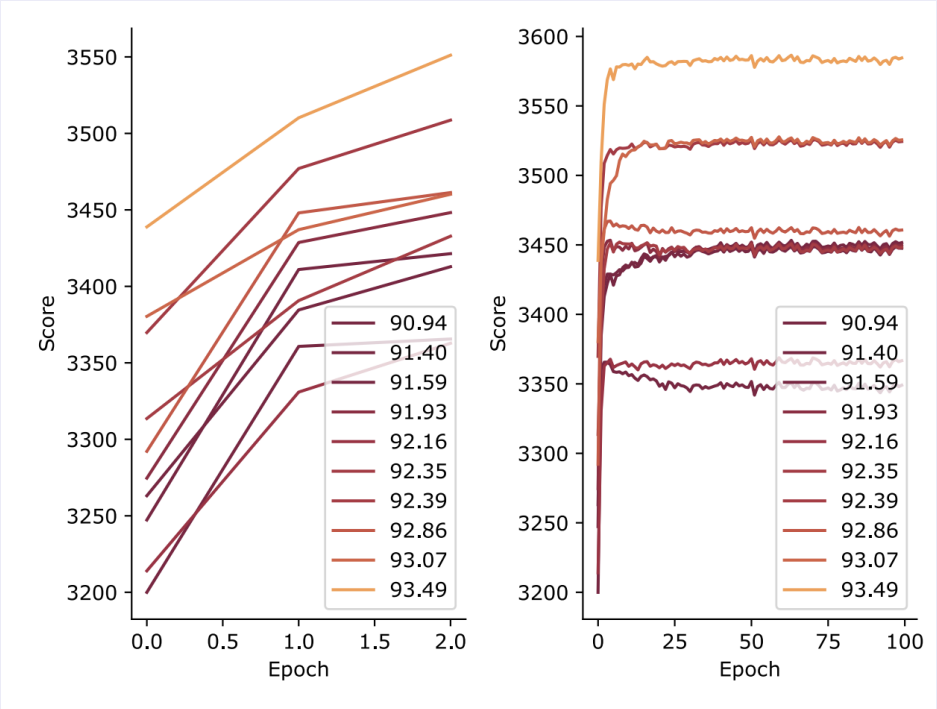

Figure: Plots of our score during training for 10 networks from NAS-Bench-201 using the CIFAR-10 dataset. For all 10 networks the score increases sharply in the first few epochs and then flattens. The ranking of the scores between networks remains relatively stable throughout training

NASWOT Algorithm

Algorithm 1 NASWOT

```
generator ← RandomGenerator()
best_net, best_score ← None, 0
for i ← 1 to N do
    net ← generator.generate()
    score ← net.score()
    if score > best_score then
        best_net, best_score ← net, score
    end if
end for
chosen_net ← best_net
```

Algorithm 2: Assisted Regularised EA — AREA

Algorithm 2 Assisted Regularised EA — AREA

population ← []
generator ← RandomGenerator()
for *i* ← 1 to *M* **do**
    net ← *generator.generate*()
    scored_net ← *net.score*()
    population.append(*scored_net*)
end for
Keep the top *N* scored networks in the population
history ← []
for *net* in *population* **do**
    trained_net ← *net.train*()
    history.append(*trained_net*)
end for
while time limit not exceeded **do**
    Sample sub-population, *S*, without replacement from population
    Select network in *S* with highest accuracy as parent
    Mutate parent network to produce child
    Train child network
    Remove oldest network from population
    population.append(*child network*)
    history.append(*child network*)
end while
chosen_net ← Network in history with highest accuracy

Results

Method	Search (s)	CIFAR-10
Random	N/A	90.38±5.51
NASWOT (N=100)	23	91.77±0.05
REA	12000	93.87±0.22
AREA (Ours)	12000	93.91±0.29

Figure: Table 1. Mean ± std. accuracy from NAS-Bench-101. NASWOT is our training-free algorithm (across 500 runs). REA uses evolutionary search to select an architecture (50 runs), Random selects one architecture (500 runs). AREA (assisted-REA) uses our score to select the starting population for REA (50 runs). Search times for REA and AREA were calculated using the NASBench-101 API

Results

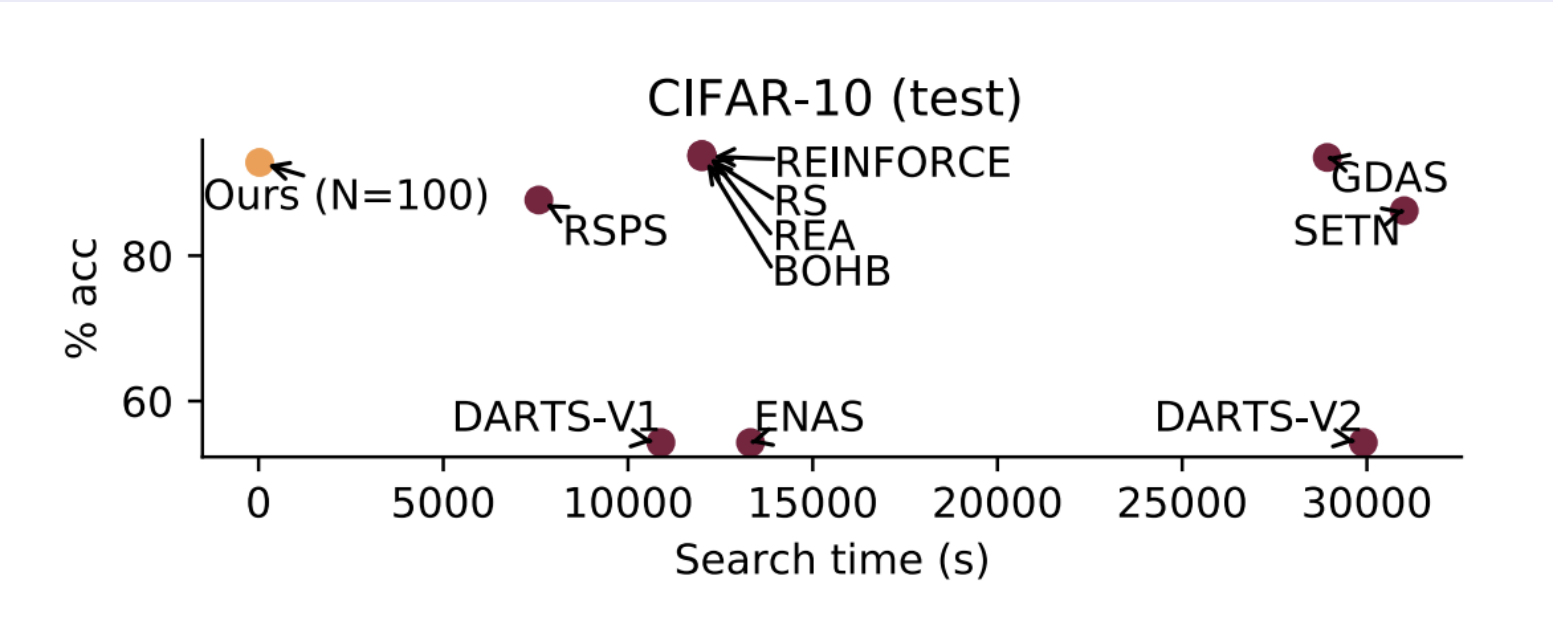

Figure: Plot showing the search time (as measured using a 1080Ti) against final accuracy of the proposed NAS-Bench-201 network for a number of search strategies

Literature

1. **Main article** Neural Architecture Search without Training.