2022 Fall @ POSTECH

EECE695D: Efficient ML Systems

Neural Architecture Search

(part 3)

Announcements

RC reviews. Thanks for early submissions.
Please try to be as detailed as possible.

No class next week! Sorry I’m going to New Orleans.

Today

Zero-shot NAS

Efficiency-aware NAS

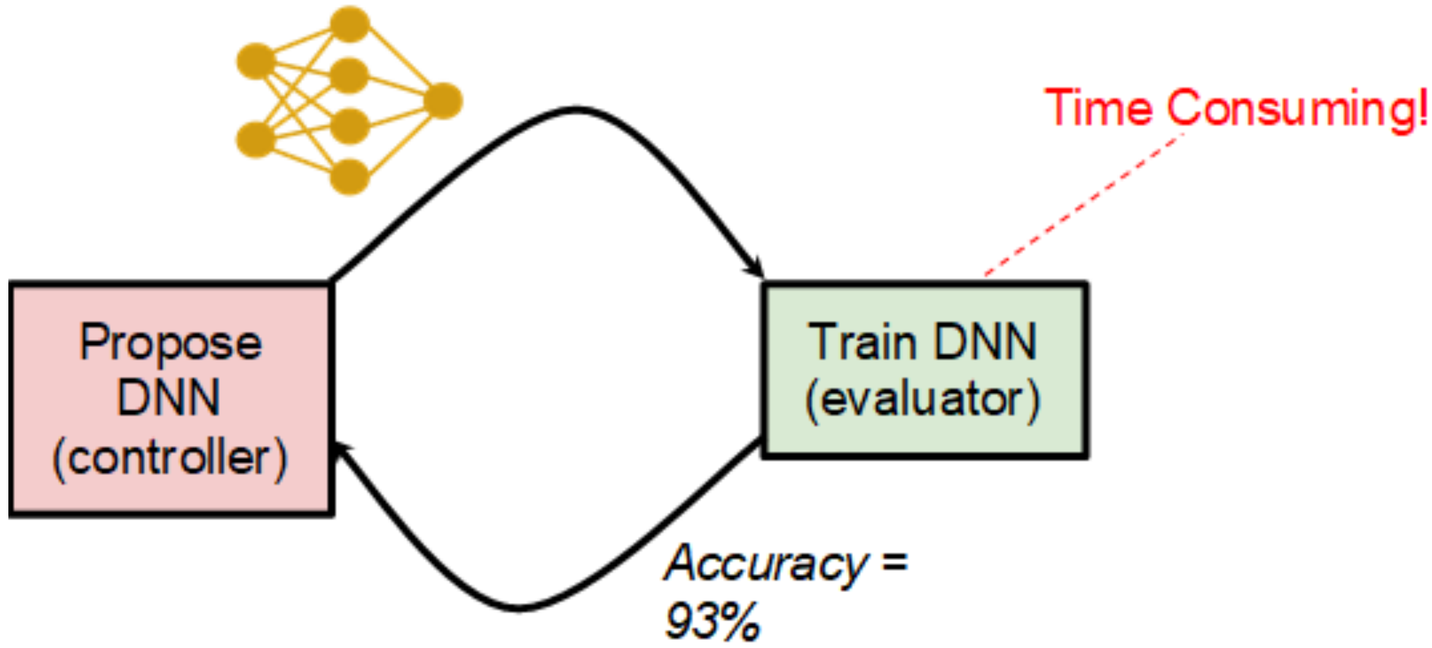
Zero-shot NAS

We talked about three building blocks of NAS: Search space, Search strategy, and Performance Estimation.

The main challenge in performance estimation was that we need to train models to evaluate them, where training these models take a lot of time and compute.

Idea. We had “pruning-at-init score”, which evaluate the “goodness” of each weight without a trained model.

Can we do the same for NNs, instead of weights?



Zero-Cost Proxies. Mellor et al. (2021) proposes one of such "zero-cost proxies" to evaluate NN architecture without performing any training.

One can either do a direct search with such proxy as an optimand (Alg. 1),

Or use the proxy to construct a nice "starting point," e.g., by constraining the search space or use as a parent pool for evoluationary algorithm (Alg 2)

What proxy? A bit later...

Algorithm 1 NASWOT

```
generator = RandomGenerator()
best_net, best_score = None, 0
for i=1:N do
    net = generator.generate()
    score = net.score()
    if score > best_score then
        best_net, best_score = net, score
chosen_net = best_network
```

Algorithm 2 Assisted Regularised EA — AREA

```
population = []
generator = RandomGenerator()
for i=1:M do
    net = generator.generate()
    scored_net = net.score()
    population.append(scored_net)
Keep the top N scored networks in the population
history = []
for net in population do
    trained_net = net.train()
    history.append(trained_net)
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)
chosen_net = Network in history with highest accuracy
```

Mellor et al. (2020), "NAS without training" ICML 2021

Method	Search (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
(a) NAS-Bench-201							
Non-weight sharing							
REA	12000	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS	12000	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE	12000	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB	12000	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
Weight sharing							
RSPS	7587	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1	10890	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	29902	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	28926	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN	31010	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS	13315	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
Training-free							
NASWOT (N=10)	3.05	89.14 ± 1.14	92.44 ± 1.13	68.50 ± 2.03	68.62 ± 2.04	41.09 ± 3.97	41.31 ± 4.11
NASWOT (N=100)	30.01	89.55 ± 0.89	92.81 ± 0.99	69.35 ± 1.70	69.48 ± 1.70	42.81 ± 3.05	43.10 ± 3.16
NASWOT (N=1000)	306.19	89.69 ± 0.73	92.96 ± 0.81	69.86 ± 1.21	69.98 ± 1.22	43.95 ± 2.05	44.44 ± 2.10
Random	N/A	83.20 ± 13.28	86.61 ± 13.46	60.70 ± 12.55	60.83 ± 12.58	33.34 ± 9.39	33.13 ± 9.66
Optimal (N=10)	N/A	89.92 ± 0.75	93.06 ± 0.59	69.61 ± 1.21	69.76 ± 1.25	43.11 ± 1.85	43.30 ± 1.87
Optimal (N=100)	N/A	91.05 ± 0.28	93.84 ± 0.23	71.45 ± 0.79	71.56 ± 0.78	45.37 ± 0.61	45.67 ± 0.64
AREA	12000	91.20 ± 0.27	-	71.95 ± 0.99	-	45.70 ± 1.05	-

Mellor et al. (2020), "NAS without training" ICML 2021

Roughly, these zero-cost proxies can be divided into two categories.

Data-Independent. Does not require data; entirely model-based.

- **#Param.** More parameters means a better model (caveat 1: blind to “where to connect” issues)
  (caveat 2: resource-constrained scenarios?)

- **SynFlow.** A data-free pruning-at-initialization algorithm (Tanaka et al., 2020)
  In a nutshell, it evaluates the importance of each connection as a Hadamard product

$$S(\theta) = \frac{\partial R}{\partial \theta} \odot \theta$$

where R is a data-free loss that represents the overall connectivity, defined as

$$R = \mathbf{1}^\top \left(\prod_{i=1}^{L} |\, \theta^{[i]} \,| \right) \mathbf{1}$$

For model evaluation, just sum all saliency scores, i.e., $\sum_{i,j} S(\theta_{ij})$

+ GenNAS, Zen-NAS...

Tanaka et al., “Pruning neural networks without any data by iteratively conserving synaptic flow” NeurIPS 2020

Data-Dependent. Use some mini-batch of data to evaluate data-specific quantities, or dependent on the characteristis of data instances.

- **FLOPs.** More compute means a better-performing model, in many cases.
- **Gradients.** Just take the sums of the Euclidean norm of all gradients.
- **SNIP.** Use the pruning-at-initialization quantities that depend on loss gradient.

$$S(\theta) = \left| \frac{\partial L}{\partial \theta} \odot \theta \right|, \qquad \text{score} = \sum_i [S(\theta)]_i$$

(similar: GraSP)

- **Fisher.** Similar to SNIP, but for activation maps (Turner et al., 2020).
- **Jacobian Covariance.** Construct a binary code, based on whether a

Turner et al., "BlockSwap: Fisher-guided block substitution for network compression on a budget," ICLR 2020

- **Jacobian Covariance.** Construct binary codes, based on the activation statistics of a neural net for each data inside a mini-batch.

  Then, construct an $N \times N$ normalized similarity matrix of binary codes.
  Use the log-determinant of the matrix as a diversity measure;
  lower the similarity, more expressive the network!

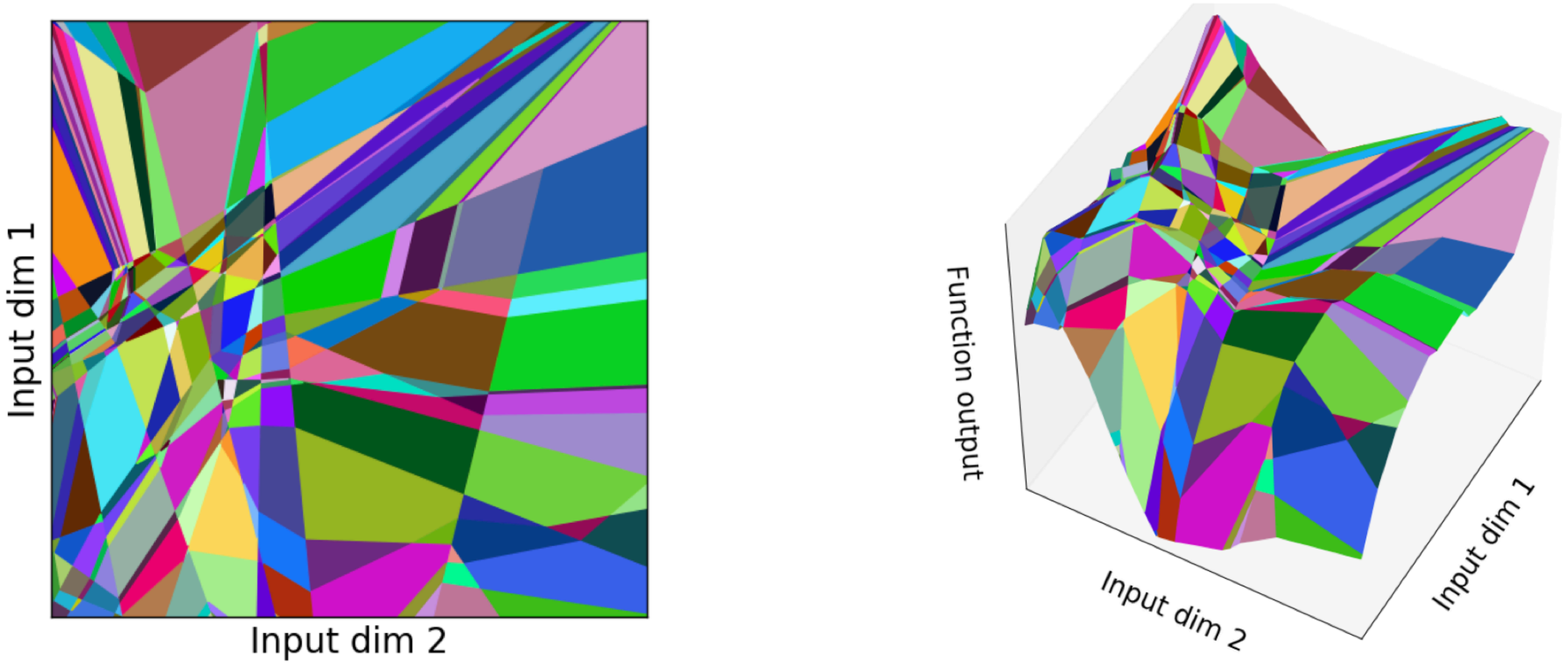


Figure 2: Function defined by a ReLU network of depth 5 and width 8 at initialization. Left: Partition of the input space into regions, on each of which the activation pattern of neurons is constant. Right: the function computed by the network, which is linear on each activation region.

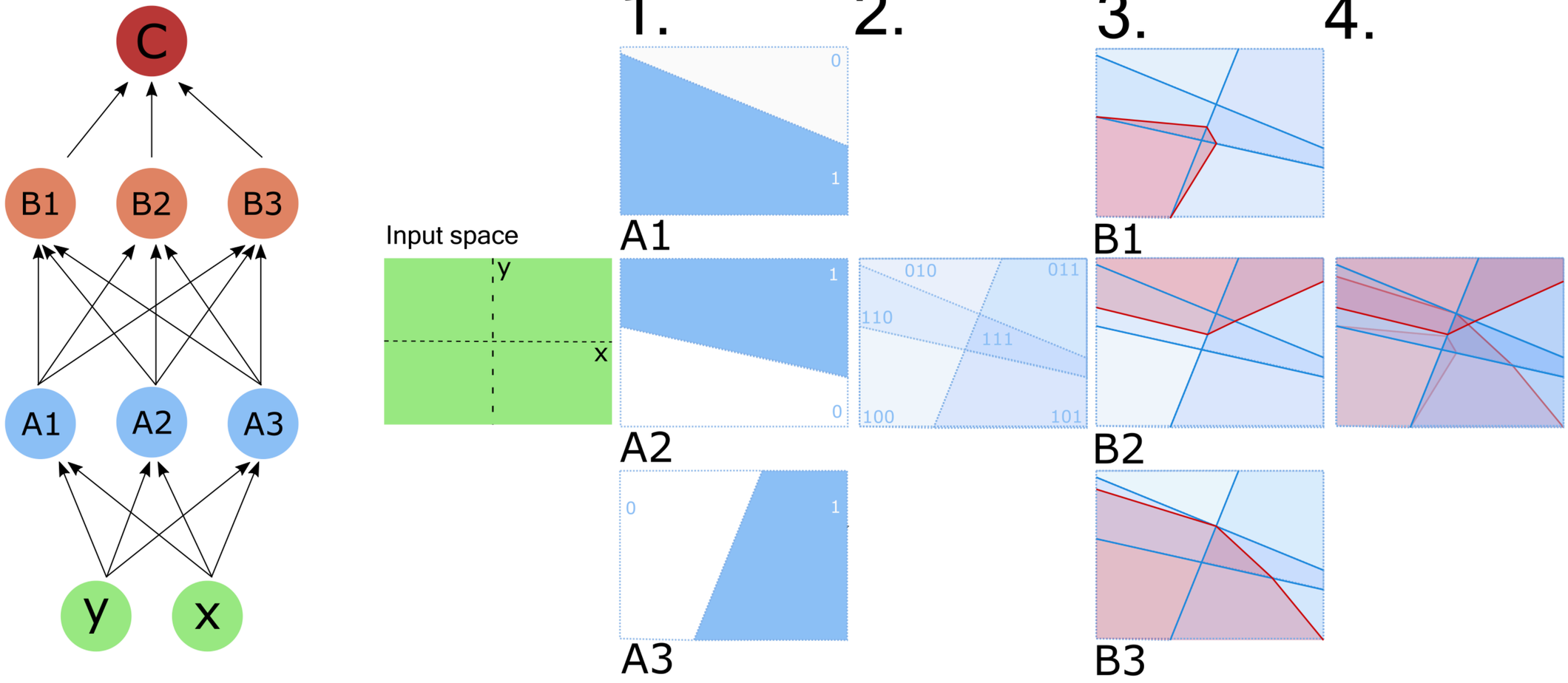


Figure 2. Visualising how binary activation codes of ReLU units correspond to linear regions. **1:** Each ReLU node Ai splits the input into an active (> 0) and inactive region We label the active region 1 and inactive 0. **2:** The active/inactive regions associated with each node Ai intersect. Areas of the input space with the same activation pattern are co-linear. Here we show the intersection of the A nodes and give the code for the linear regions. Bit i of the code corresponds to whether node Ai is active. **3:** The ReLU nodes B of the next layer divides the space further into active and inactive regions. **4:** Each linear region at a given node can be uniquely defined by the activation pattern of all the ReLU nodes that preceded it.

Empirically. There is no clear winner!

White et al. (2022) performed a large-scale study on ~15 zero-shot NAS frameworks, and measured the Spearman's rank correlation (i.e., correlation for rank variables).

Table 1: Spearman's rank correlation between each ZC proxy and ground-truth evaluations for NATS-Bench Topological Search Space (TSS).

Proxies	NATS-Bench TSS Spherical CIFAR-100	NATS-Bench TSS CIFAR-100	NATS-Bench TSS Synthetic CIFAR-10
`fisher`	0.0745	0.6104	-0.4281
`grad_norm`	0.0056	0.6779	-0.3140
`grasp`	0.0327	0.6319	-0.2568
`jacob_cov`	-0.2512	0.7194	**0.1988**
`snip`	0.0075	0.6789	-0.3194
`synflow`	**0.1758**	**0.7938**	-0.0004
`flops`	0.0239	0.7142	-0.0701
`params`	-0.0017	0.7346	-0.0426

White et al., "A deeper look at zero-cost proxies for lightweight NAS" ICLR Blog Track 2022

Table 2: Spearman's rank correlation between each ZC proxy and ground-truth evaluations for DARTS.

Proxies	DARTS Spherical CIFAR-100	DARTS CIFAR-100	DARTS NinaPro	DARTS Synthetic CIFAR-10	DARTS Darcy Flow
fisher	**0.4986**	-0.0161	-0.1181	-0.1685	0.1540
grad_norm	0.2450	0.2669	-0.1436	0.0105	**0.1788**
grasp	-0.4754	0.2301	**0.0107**	0.0523	-0.0970
jacob_cov	0.3538	-0.1337	0.0277	-0.1235	-0.1232
snip	0.2675	0.2303	-0.1458	0.0234	0.1419
synflow	-0.0560	0.3935	-0.1729	0.0552	-0.3978
flops	-0.2074	0.5625	-0.1085	0.2635	-0.1971
params	-0.2389	**0.5630**	-0.0888	**0.2644**	-0.2275

#Param rules...?

White et al., "A deeper look at zero-cost proxies for lightweight NAS" ICLR Blog Track 2022

Table 3: Spearman's rank correlation between each ZC proxy and ground-truth evaluations for TransNAS-Bench-101.

Proxies	TransNAS-Bench-101 Jigsaw	TransNAS-Bench-101 Object Classification	TransNAS-Bench-101 Scene Classification	TransNAS-Bench-101 Autoencoder
fisher	0.4361	**0.7998**	0.7522	**0.5611**
grad_norm	0.4933	0.7286	0.6756	0.4380
grasp	0.5085	0.6233	0.5034	0.4646
jacob_cov	0.3733	0.3969	0.6964	-0.1569
snip	**0.5367**	0.7582	0.7162	0.3671
synflow	0.4853	0.6331	**0.7582**	-0.0850
flops	0.5161	0.5686	0.7360	0.0650
params	0.5068	0.5614	0.7181	0.0517

Fisher rules...?

White et al., "A deeper look at zero-cost proxies for lightweight NAS" ICLR Blog Track 2022

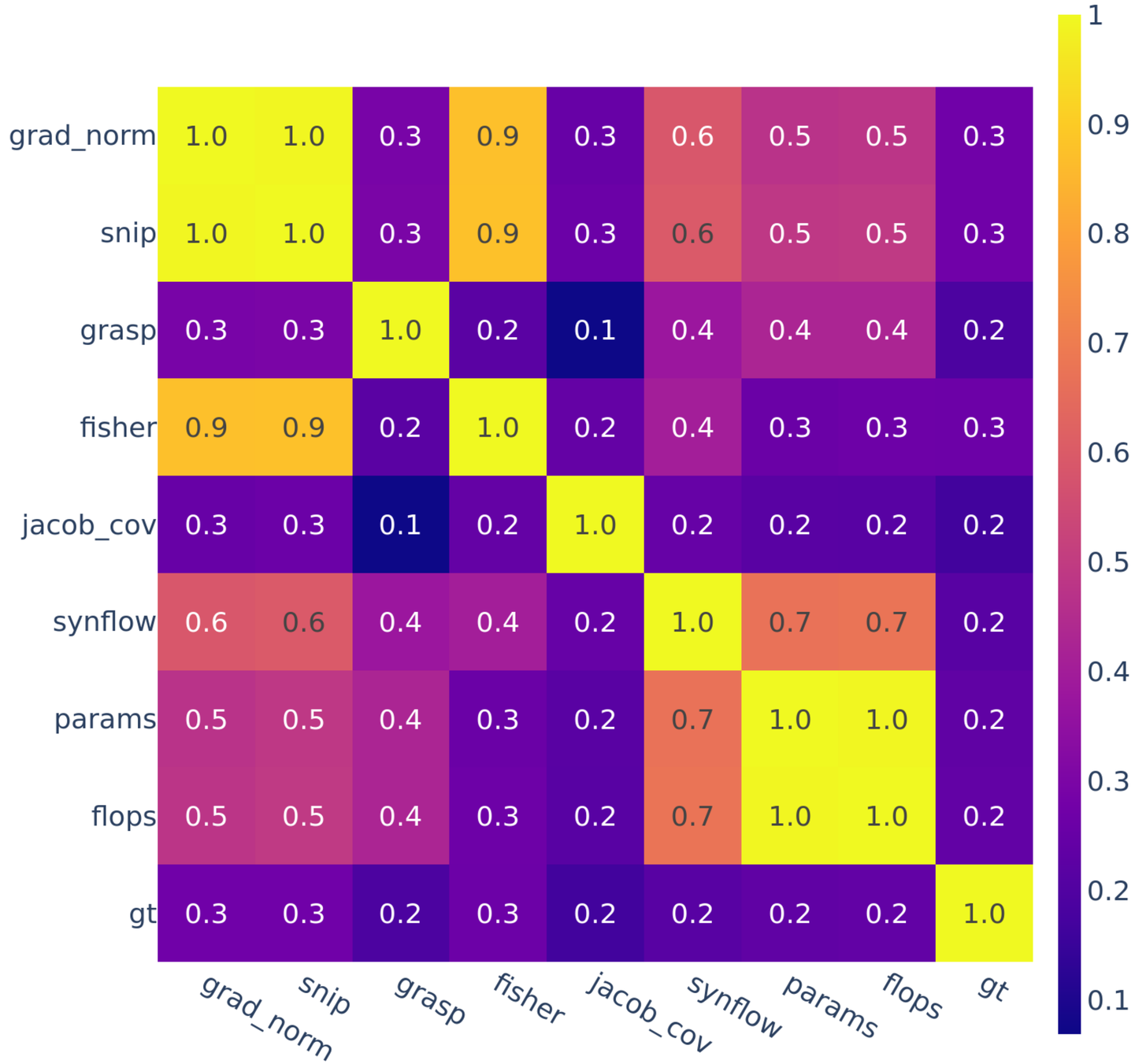
Table 4: Average ranking of each of the ZC proxies on each search space, and over all search spaces.

	fisher	grad_norm	grasp	jacob_cov	snip	synflow	flops	params
NATS-Bench TSS	6.0	6.0	5.0	4.0	5.67	**1.33**	4.0	4.0
DARTS	4.6	4.2	4.6	4.8	4.6	5.4	4.0	**3.8**
TransNAS-Bench-101	**2.75**	4.5	4.5	7.5	3.0	4.5	4.0	5.25
Overall	4.33	4.75	4.67	5.5	4.33	4.08	**4.0**	4.33

No clear winner... FLOPs maybe... and SynFlow too?

White et al., "A deeper look at zero-cost proxies for lightweight NAS" ICLR Blog Track 2022

SynFlow good,
only because it is highly
correlated with FLOPs?



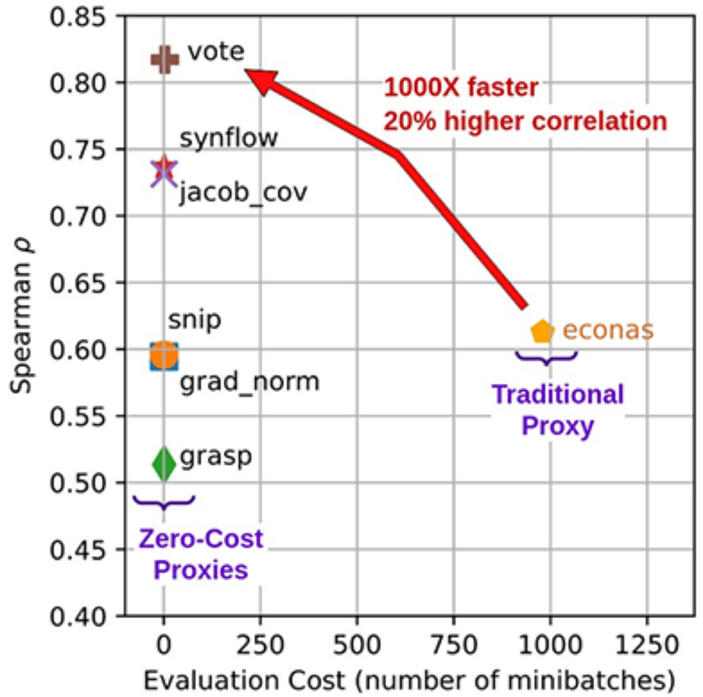
White et al., "A deeper look at zero-cost proxies for lightweight NAS" ICLR Blog Track 2022

Some hopes. (1) Ensembling these may be a nice thing.

Table 1: Spearman ρ of zero-cost proxies on NAS-Bench-201.

Dataset	grad_norm	snip	grasp	fisher	synflow	jacob_cov	vote
CIFAR-10	0.58	0.58	0.48	0.36	0.74	0.73	0.82
CIFAR-100	0.64	0.63	0.54	0.39	0.76	0.71	0.83
ImageNet16-120	0.58	0.58	0.56	0.33	0.75	0.71	0.82

(2) Still faster than non-zero-shots.



+ As mentioned earlier, can be combined with other NAS

Abdelfattah et al., "Zero-cost proxies for lightweight NAS" ICLR 2021

Efficiency-aware NAS

These NAS frameworks are mostly about performance maximization only—ignores efficiency or HW constraint!

Let us additionally discuss some works that explicitly focus on **efficiency/HW.**

Search Space. Tan et al. (2019) proposes MNAS with a "factorized hierarchical search space."
✅ Efficient Modules ✅ Efficiency-critical HPs

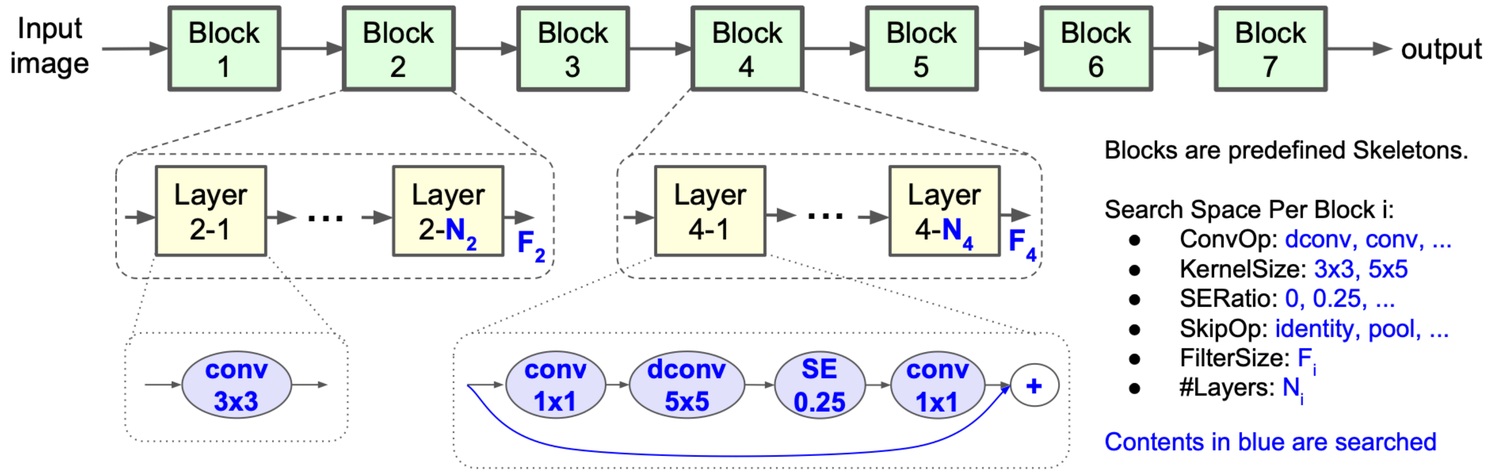


Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N, then the same layer is repeated N times (e.g., Layer 4-1 to 4-N_4 are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

Tan et al., "MNASNet:Platform-aware NAS for mobile" CVPR 2019.

Lin et al. (2020) proposes TinyNAS that uses *multiple versions* of the MNAS search space.

In particular, they tune (1) input resolution (2) width multiplier
to generate multiple search spaces with similar peak memory requirements.

Then, instead of running NAS over all spaces, they select the space with most FLOPs!

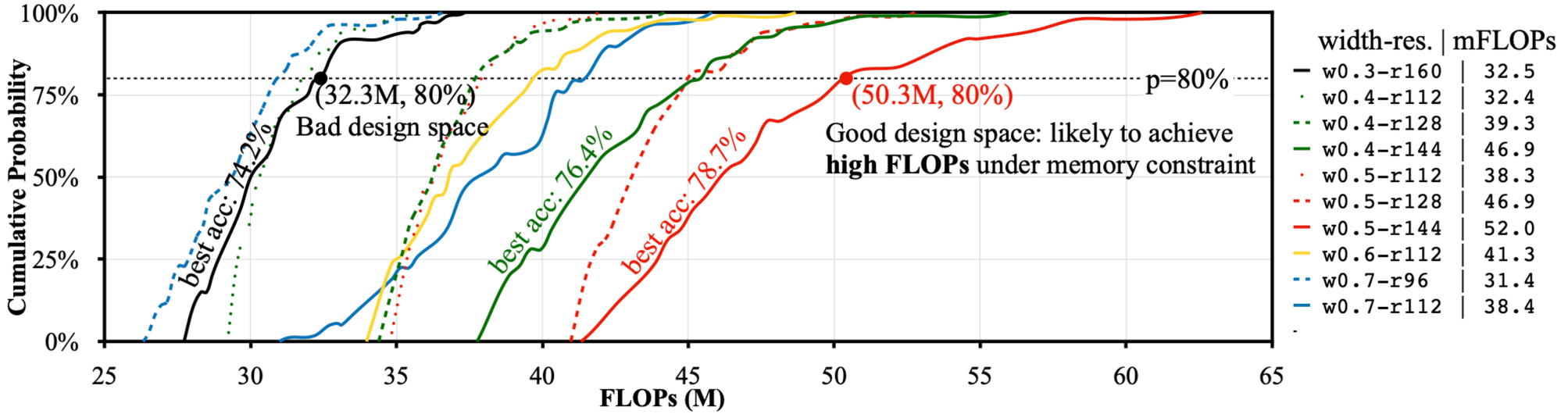


Figure 3. TinyNAS selects the best search space by analyzing the FLOPs CDF of different search spaces. Each curve represents a design space. Our insight is that the design space that is more likely to produce *high FLOPs* models under the memory constraint gives higher model capacity, thus more likely to achieve high accuracy. For the solid **red** space, the top 20% of the models have >50.3M FLOPs, while for the solid **black** space, the top 20% of the models only have >32.3M FLOPs. Using the solid **red** space for neural architecture search achieves 78.7% final accuracy, which is 4.5% higher compared to using the **black** space. The legend is in format: `w{width}-r{resolution}|{mean FLOPs}.`

Lin et al., "MCUNet: Tiny Deep Learning on IoT Devices" NeurIPS 2020.

Search Strategy. Recall that RL was (and still is) a popular strategy to search over NN architectures, where the reward was to maximize the "validation performance after training."

$$\max_a \text{Acc}(a) \qquad \Rightarrow \qquad \text{Reward} : \text{Acc}(a)$$

In efficiency-aware NAS, one may want to solve

$$\max_a \text{Acc}(a) \qquad \text{subject to} \qquad \text{Latency}(a) \leq \tau$$

Some natural approaches may be:

$$\text{Reward} : \text{Acc}(a) \cdot \mathbf{1}\{\text{Latency}(a) \leq \tau\}$$

$$\text{Reward} : \text{Acc}(a) \cdot \left(\frac{\text{Latency}(a)}{\tau}\right)^{\gamma}$$

$$\text{Reward} : \alpha \cdot \text{Acc}(a) + (1-\alpha) \cdot (\tau - \text{Latency}(a))$$

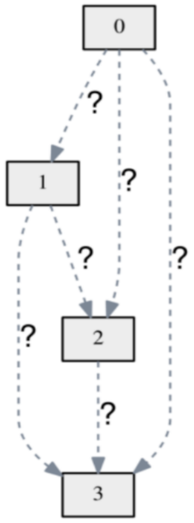
(see, e.g., MONAS by Hsu et al. (2019))

Hsu et al., "MONAS: Multi-Objective NAS" AAAI 2019.

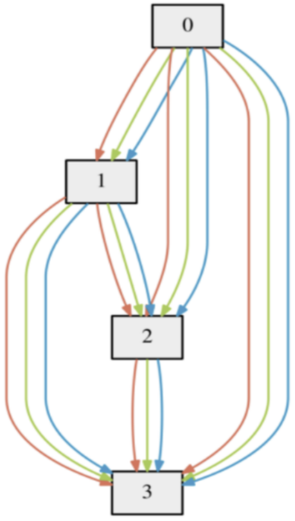
One can do a similar thing for DARTS: $$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

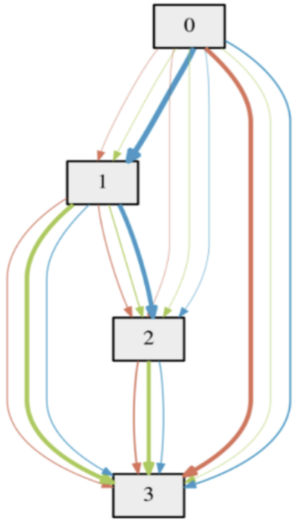
$$\mathscr{L}_{\text{FLOP}} = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} \text{FLOP}^{i,j}$$



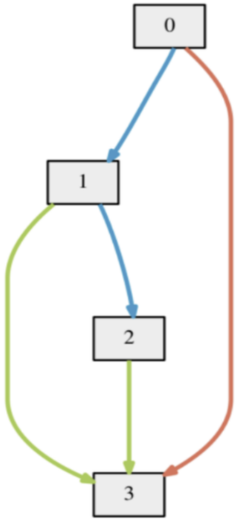
(a) Initially unknown operations on the edges.



(b) Continuous relaxation by placing a mixture of operations on each edge.



(c) Bilevel optimization to jointly train mixing probabilities and weights.



(d) Finalized the model based on the learned mixing probabilities.

Hsu et al., "MONAS: Multi-Objective NAS" AAAI 2019.

Performance Estimation. Latency is a very HW-dependent quantity—should be measured for all NN!

To avoid measuring latency for each generated network,
one typically jointly trains an auxiliary latency/energy predictor for the HW
e.g., ChamNet (Dai et al., 2019)

Note: One can use Graph NNs for this!
Note: Use meta-learning to reduce #samples

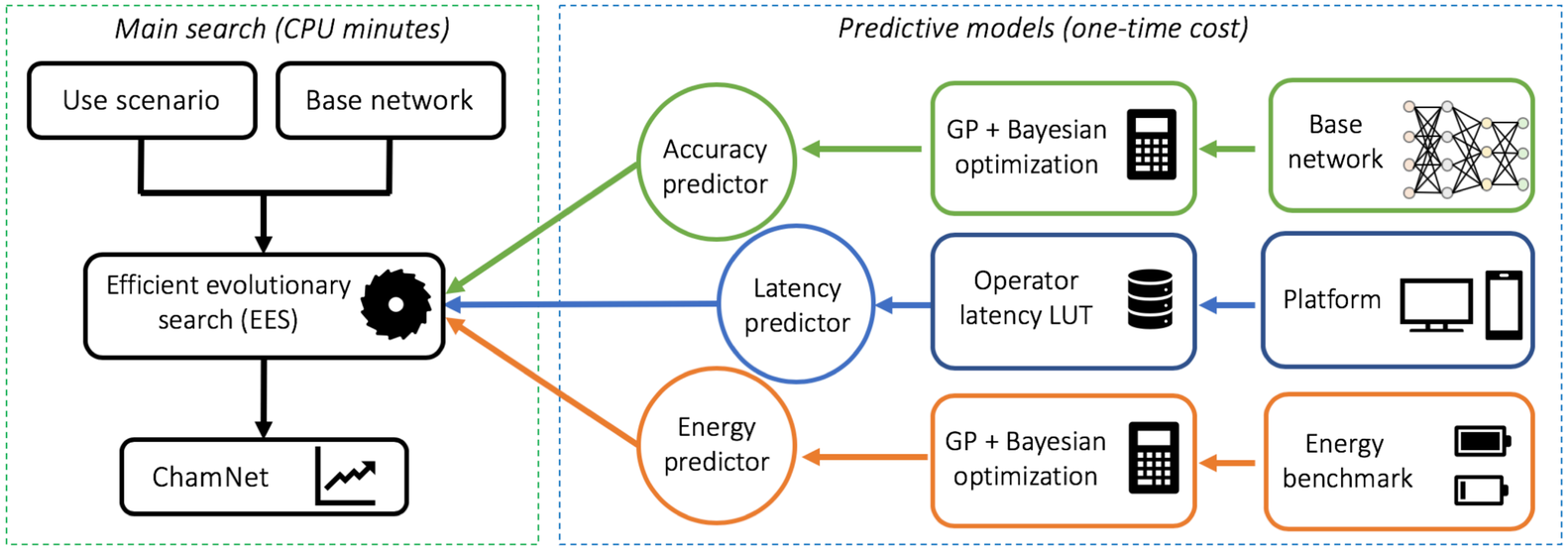


Figure 1. An illustration of the Chameleon adaptation framework

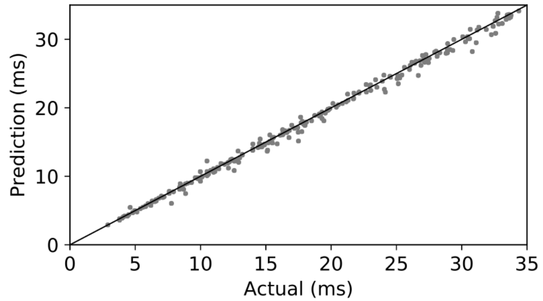


Figure 5. Latency predictor evaluation on Snapdragon 835 CPU.

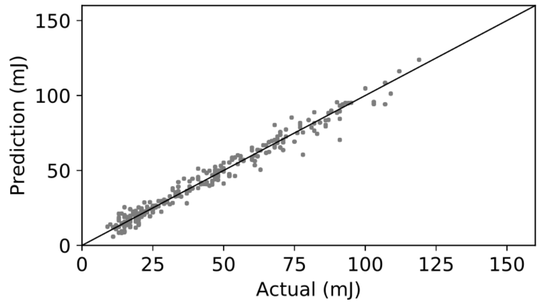


Figure 6. Energy predictor evaluation on Snapdragon 835 CPU.

Dai et al., "ChamNet: Toward efficient network design through platform-aware model adaptation" CVPR 2019.