# Towards The Co-design of Neural Networks and Accelerators

Yanqi Zhou [1] Xuanyi Dong [1] Tianjian Meng [2] Mingxing Tan [3] Berkin Akin [1] Daiyi Peng [1]
Amir Yazdanbakhsh [1] Da Huang [1] Ravi Narayanaswami [4] James Laudon [1]

## Abstract

Better neural architectures and new hardware accelerators are two driving forces for the progress in deep learning. Previous works typically focus on one aspect: they either design new neural architectures for fixed hardware like GPUs or customize hardware (often on FPGAs) for a fixed set of neural models like ResNets or Transformers. In this work, we aim to jointly optimize neural architecture and hardware configurations for Google's Edge TPUs. Through extensive studies, we observe that: 1) the neural architecture search space has to be customized to fully leverage the targeted hardware, 2) neural architecture and hardware accelerator should be jointly searched to achieve the best of both worlds, and 3) conventional metrics such as FLOPs and parameter size often do not well represent model efficiency in real accelerators. Our experiments show that our joint search approach, named NaaS, consistently outperforms previous state-of-the-art results, such as EfficientNet, on both image classification and segmentation tasks. Furthermore, our approach reduces energy consumption by up to 2x under the same accuracy on Edge TPUs.

## 1 Introduction

The optimization of traditional hardware designs is typically done by evaluating the performance of the designs on a set of predefined benchmarks (e.g., SPEC (SPE, 2006)), which act as a proxy for the full set of workloads. In addition to the difficulty for the benchmarks to capture all the salient characteristics of the workload set, the benchmarks tend to stay fixed for a substantial period of time, which can result in the hardware design lagging behind the algorithmic changes, particularly in a rapidly changing field such as machine learning.

In the recent decade, hardware specialization has become prevalent as a result of hitting the end of Moore's Law. Google's TPU (TPU, 2016), Intel's Nervana NNP (Yang, 2019), and Apple's Neural Engine in the M1 SoC (app, 2020) are representative accelerators specialized for deep learning primitives. MLPerf (MLP, 2019) becomes the dominating benchmark for the state-of-the-art design of machine learning (ML) accelerators. However, rapid progress in deep learning has given birth to numerous more expressive and efficient models in a short time, which results in both benchmarking and accelerator development lagging behind. For example, squeeze-and-excite with global pooling and SiLU/Swish non-linearity (Ramachandran et al., 2017; Elfwing et al., 2018) are found to be useful in EfficientNet (Tan & Le, 2019); however, they are often not supported or inefficient in many specialized accelerators like GPUs, EdgeTPUs and DSPs.

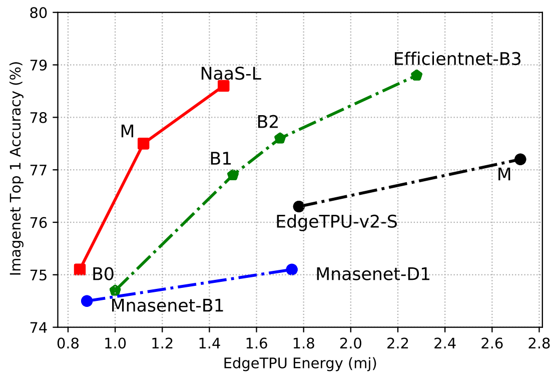


*Figure 1.* ImageNet Top 1 Accuracy vs. Energy (mj) comparing NaaS and related work.

Platform-aware neural architecture search (NAS) (Tan et al., 2019; Wu et al., 2019; Cai et al., 2019) and hardware design space exploration (Parsa et al., 2020; Iqbal et al., 2020; Sun et al., 2020; Nardi et al., 2019; Cong et al., 2018; Koeplinger et al., 2018; Balaprakash et al., 2016; Ansel et al., 2014) optimizes only one end of the space (either neural networks or hardware), which can significantly limits application performance. For example, the target device may have a sub-optimal compute and memory ratio for the target application

[1]Google Research, Brain Team [2]The University of Texas at Austin, work done while at Google [3]Waymo [4]Cruise, work done while at Google. Correspondence to: Yanqi Zhou <yanqiz@google.com>.

*Proceedings of the $5^{th}$ MLSys Conference*, Santa Clara, CA, USA, 2022. Copyright 2022 by the author(s).

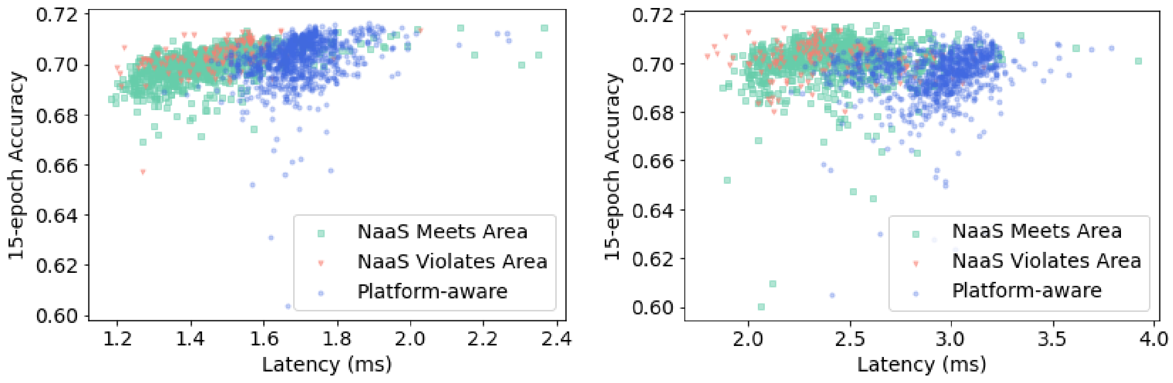


*Figure 2.* Network architecture distributions comparing NaaS and platform-aware NAS (fixed accelerator) on (1) Efficientnet-B0 (left) and (2) Efficientnet-B1 (right) search spaces: During the search, we set the inference latency target to be 1.0 ms and 2.0 ms for the two search spaces, and set chip area constraint to be the same as the default Edge TPU. Note that we don't remove inefficient Swish or Squeeze-and-Excite layers from the search spaces, thus the latency is higher than later presented in Section 4.

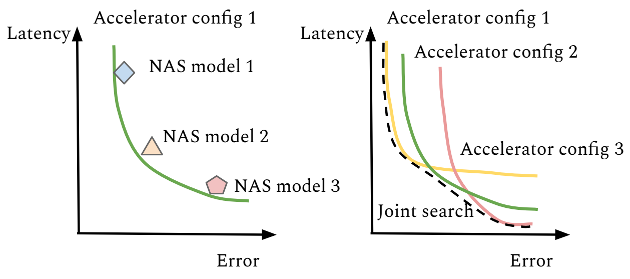


*Figure 3.* NaaS expands the pareto frontiers.

combined with an inference latency target. The change of accelerator configuration can shift the optimal NAS model distributions and fixing the accelerator configuration can result in underperformance of NAS. Jointly searching accelerator configuration and neural architectures, as shown in Figure 2, unleashes more performance opportunities, by identifying neural architectures with lower latencies and higher accuracies compared to fixed accelerator neural architecture search. For smaller models with lower inference latencies, the joint search benefits from adding more on-chip computational units, getting a further boost on inference speed. For larger model regime, adding more on-chip memory enables identifying more expressive models while not increasing inference time.

Building on this prior work, we propose to study the joint search of hardware accelerator and model architectures. We generalize the above ideas into a framework called NaaS – Neural Architecture and Accelerator Search. In NaaS, we parameterize neural architecture search and hardware accelerator search in a unified joint search space, using a symbolic programming library named PyGlove (Peng et al., 2020). We use Google's Edge TPUs including Coral (Google, 2020) and Tensor SoC on Pixel 6 (Osterloh, 2021), as the target devices in this work. An Edge TPU is a highly parameterized accelerator that the tuning of its searchable parameters yields valid architectures always. However, whether a neural network can be compiled successfully onto an accelerator is determined by various factors such as the compatibility of neural network operations, model parameters, accelerator on-chip memory, etc. Figure 3 conceptually shows that while conventional platform-aware NAS selects models along the Pareto frontier with different latency and accuracy tradeoffs for one target device, NaaS further expands the Pareto frontier by enabling different hardware accelerator configurations.

Unlike the conventional search approach, NaaS is *task driven*, where the task is a problem (e.g., image classification, object detection) or a domain of problems (e.g., vision, NLP), not a set of fixed programs or graphs (e.g., ResNet, Transformers). This effectively creates generalization across the vertical stack, making the hardware evolve with the applications. NaaS can be practically used to design customized accelerators for autonomous driving where there is a stringent inference latency requirement and model architecture search can benefit from tuning accelerator parameters, as demonstrated in Figure 2. NaaS can be economically used, targeting mobile SoC (system-on-chip) where a set of highly optimized accelerators are combined into a system. To summarize our contributions:

- We develop a fully automated framework that can jointly optimize neural architectures and hardware accelerators and demonstrate the effectiveness of co-optimizing NAS with the parameterization of Edge TPUs.

- We constrained the search on accelerator performance such as chip latency and energy and compared across different search spaces. In addition, we evolve the search space and create an Edge TPU friendly search space for image classification and segmentation tasks.

- NaaS outperforms platform-aware NAS including MnasNet (Tan et al., 2019) and EfficientNet (Tan & Le, 2019), and manually optimized models based on TuNAS search (Bender et al., 2020) (The network is called Manual-EdgeTPU (Xiong et al., 2021; Gupta & Akin, 2020)), by around 1% in ImageNet top-1 accuracy or around 20% in inference latency.
- NaaS segmentation models are 3% more accurate with the same latency or 30% faster with the same accuracy, compared to the SoTA models on Edge TPUs. Our results are based on real hardware evaluation and are consistent on both Coral and Tensor SoC.
- We compare different optimization strategies and find that joint search consistently outperforms optimizing each space with coordinate descent or set coordinate descent.
- We compare multi-trial search with oneshot search where sampled neural networks share parameters. We find that even though oneshot search can significantly reduce search cost, it is less suitable for large models when constructing the super network is too costly.

## 2 Related Work

**Platform-Aware:** There has been growing interest in leveraging NAS to automate the design process of edge models (Howard et al., 2019; Tan et al., 2019; Dai et al., 2019). ProxylessNAS directly searches for ImageNet by proposing a gradient-based approach to train binarized parameters (Cai et al., 2019). FBNet proposed a differentiable platform-aware NAS using Gumbel Softmax (Wu et al., 2019). NetAdapt (Yang et al., 2018) and AMC (He et al., 2018) utilize a direct metric like latency to finetune the number of channels of a pre-trained model. However, none of these works optimize the underlying hardware accelerators together with NAS.

**Efficient NAS:** There are many recent efforts to reduce the search cost of NAS (Brock et al., 2018; Bender et al., 2018; Pham et al., 2018; Liu et al., 2019; Cai et al., 2020; Dong & Yang, 2019; Cai et al., 2019; Dong et al., 2021; Wu et al., 2019). Existing works are heavily focusing on highly regularized search spaces consisting of mainly Inverted Bottleneck (IBN) layers, for FLOPs on mobile CPUs. Less attention has been paid to the adaptation of search space or a latency/power driven search for modern accelerators such as DSPs or Edge TPUs.

**Accelerator performance:** Table 1 enumerates neural architectural characteristics that more fundamentally affect accelerator performance. For example, Edge TPUs has comparatively smaller memory-to-compute ratio on chip compared to GPUs or TPUs, thus are more sensitive to model parameter size, activation memory, data reuse, global pooling, and upsampling. Residual connections in ResNet, a bi-FPN head increases activation memory, thus could hurt accelerator inference time if not carefully optimize the neural architectures. The additional ability to tune accelerator configuration can unleash more performance opportunities for models with above listed characteristics. A naive way is to slightly increase the memory-to-compute ratio on chip.

**Hardware metrics:** Platform-aware NAS typically searches for efficient models with lower parameter counts and FLOPs. However, optimizing *indirect metrics* like parameter counts or FLOPs won't necessarily improve the *direct metrics* of latency (Bender et al., 2020; Wu et al., 2019). For example, a multi-branch network like NASNet (Zoph et al., 2018) has lower parameter counts and FLOPs compared to the layer-wise network such as ResNet and MobileNet (Sandler et al., 2018), but its fragmented cell-level structure is not hardware friendly and is often slow on real-world devices and accelerators. The effects of *indirect metrics* like parameter counts and FLOPs do not have a simple correlation with *direct metrics* such as latency and power. For example, the model can run out of memory if the number of parameters is too large.

**Co-design:** There is a growing body of work exploring neural architecture search and hardware design (Lin et al., 2021; Jiang et al., 2020a; Choi et al., 2021; Jiang et al., 2020b; Achararit et al., 2020; Yang et al., 2020; Kwon et al., 2018; Yang et al., 2019). However, most of the work targets FPGAs or academic accelerators (Chen et al., 2016). During the search, most of the work based their evaluations on a performance simulator or a less accurate regression model. In addition, none of the existing work has customized the NAS search space targeting the accelerator. To our best knowledge, we are the first to: 1. Search based on a cycle-accurate accelerator simulator that has been validated across a wide range of industry standard workloads. 2. NaaS demonstrates effectiveness on a real ImageNet workload, on multiple search spaces including an adapted search space for EdgeTPU. NaaS is the first to adapt the NAS search space for the co-design of NAS and accelerators. 3. We evaluate NaaS on more than just one task. The presented results on the segmentation task were based on real hardware evaluation.

## 3 Methodology

### 3.1 Formulation

The objective for NaaS is to find a neural architecture configuration in the space of $\alpha$ and hardware accelerator config-

| Architectures | Large params | Large activation | Low data reuse | Global Pooling | Expensive upsampling | Unoptimized ops |
|---|---|---|---|---|---|---|
| ResNet (He et al., 2016) | Yes | Yes | No | No | No | No |
| NASNet (Zoph et al., 2018) | No | Yes | Yes | No | No | No |
| EfficientNet (Tan & Le, 2019) | No | No | Yes | Yes | No | Yes (swish, SE) |
| Deeplab-v3 (Chen et al., 2017) | No | Yes | Yes | No | Yes | Yes (atrous conv) |
| EfficientSeg (Tan et al., 2020) | No | No | Yes | Yes | No | Yes (bi-FPN) |

*Table 1.* Neural architectural characteristics that affect accelerator performance on Edge Devices. “No” is better than “Yes” for accelerator performance. A good metric for data reuse is algorithm’s computation intensity–computations per data loading. Most neural networks have lower computation intensity than a modern accelerator (e.g. GPU, TPU) provisions.

uration in the space of $h$ such that the validation accuracy can be maximized while meeting a chip area and latency target.

$$\begin{aligned} &\min_{\mathbf{h},\alpha} \mathcal{L}_{val}(\mathbf{h}, \alpha, \mathbf{w}^*(\alpha, \mathbf{h})) \\ &\quad \text{s.t. } \mathcal{L}_{la}(\alpha, \mathbf{h}) \le t_{la},\ \mathcal{L}_a(\mathbf{h}) \le t_a, \\ &\text{where } \ \mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}|\alpha, \mathbf{h}) \end{aligned} \tag{1}$$

To optimize the target, we first applies direct optimizations on a joint, flattened search space using a RL controller network. We then compare the flattened search with 1) coordination descent and 2) Set coordinate descent.

Original objective can be simplified as two sub objective, and alternative the steps which guarantees its minimization to a local minimum:

$$\begin{aligned} \alpha_{i+1} &= \min_{\alpha} \mathcal{L}_{val}(\alpha, \mathbf{w}^*(\alpha, \mathbf{h}_i)|\mathbf{h}_i), \\ \mathbf{h}_{i+1} &= \min_{\mathbf{h}} \mathcal{L}_{val}(\alpha_{i+1}, \mathbf{w}^*(\alpha_{i+1}, \mathbf{h})|\alpha_{i+1}) \end{aligned}$$

Original objective may suffer from overfitting of hardware configuration to certain targets, and easy to affected by noisy data. Therefore, we add a regularization to the original objective in a set coordinate descent, which let hardware optimize a set of $k$ best performing architectures.

Let

$$\mathcal{L}_{\alpha,i} = \mathcal{L}_{val}(\alpha, \mathbf{w}^*(\alpha, \mathbf{h}_i)|\mathbf{h}_i)$$

We have a set of top k performing architectures.

$$\begin{aligned} \mathrm{A}_{i+1} &= \{\alpha\} \quad s.t. \mathcal{L}_{\alpha,i} \le \mathcal{L}_{\alpha_k,i} \\ \mathbf{h}_{i+1} &= \min_{\mathbf{h}} \sum_{\alpha \in \mathrm{A}_{i+1}} \mathcal{L}_{val}(\alpha, \mathbf{w}^*(\alpha, \mathbf{h})|\alpha) \end{aligned}$$

where $\alpha_k$ is the $k$-th best performing architecture, $\mathcal{L}$ indicates the objective function of the tasks (e.g., cross-entropy for classification) and $\omega_\alpha$ denotes the weights of the architecture $\alpha$. We empirically compare different optimization strategies in Section 4.2.

## 3.2 NAS Search Space

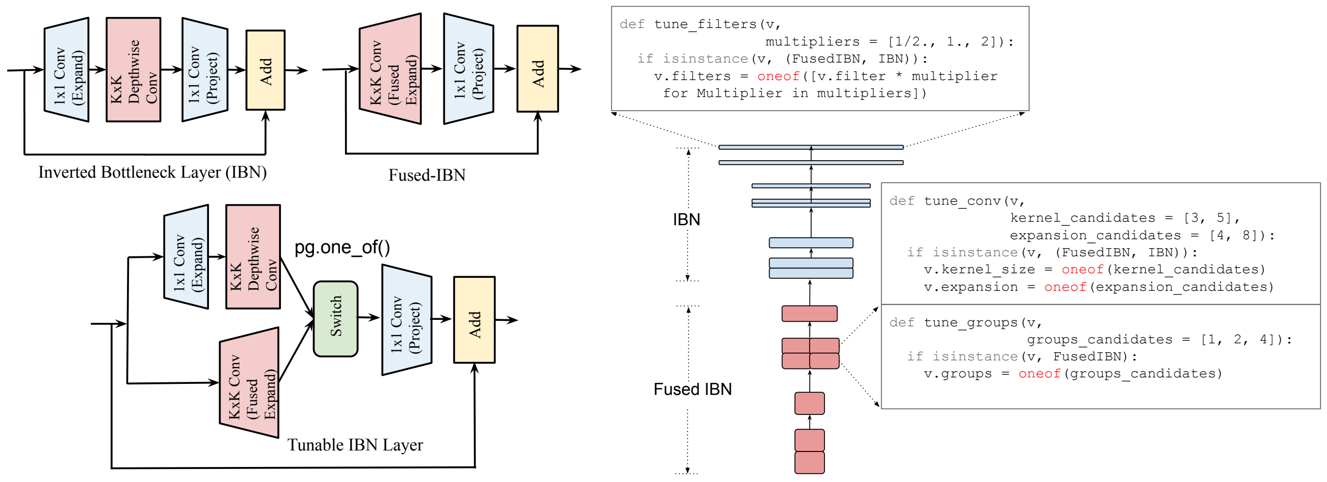
### *3.2.1 IBN-based Search Space*

**MobileNetV2:** We build the architecture search space $\mathcal{S}_1$ based on the standard MobileNetV2. The search space is tailored for mobile edge processors, and therefore consists mostly of efficient operations such as mobile IBN. Specifically, we search for the kernel size from $\{3, 5, 7\}$ for each IBN layer, and we also search for the expansion ratio from $\{3, 6\}$ for each block except for the first one, which has the default expansion ratio of 1. In MobileNetV2, there are 17 inverted residual blocks, and thus the cardinality of $\mathcal{S}_1$ is about 8.4e12.

**EfficientNet:** In order to create larger NAS models and to better leverage modern edge accelerators which have larger number of compute units and memory capacities, we build the architecture search space $\mathcal{S}_2$ based on the standard EfficientNet-B0. Similar to $\mathcal{S}_1$, we also search for the kernel size from $\{3, 5, 7\}$ and the expansion ratio from $\{3, 6\}$. Since there are 16 inverted residual blocks in EfficientNet-B0, the cardinality of $\mathcal{S}_2$ is about 1.4e12. Different from the MobileNet-v2 search space, EfficientNet search enables scaling the model capacity with optimized compounding factors in depth and width. For our presented models with various latency targets, we applied the same compounding factors as EfficientNet.

### *3.2.2 Evolved NAS Search Space*

IBNs remain the predominant building blocks in many SoTA mobile models (adopted in the previous two search spaces). Although IBN layers are good at reducing the parameter count and FLOPs, depthwise-separable convolutions in IBN can significantly increase inference latency or reduce accelerator utilization. For example, many industry standard edge accelerators, such as Google’s Edge TPU and Qualcomm DSPs, have small on-chip memory therefore benefit from high compute intensity (more computations per data transfer) (Zhou et al., 2018) models. For certain tensor shapes and kernel dimensions, a regular convolution can utilize the hardware up to 3x more efficiently than the depth-wise variation on an Edge TPU despite the much larger computation cost (7x more FLOPs) (Xiong et al., 2021). This leads the



```
def tune_filters(v,
                 multipliers = [1/2., 1., 2]):
  if isinstance(v, (FusedIBN, IBN)):
    v.filters = oneof([v.filter * multiplier
     for Multiplier in multipliers])
```

```
def tune_conv(v,
              kernel_candidates = [3, 5],
              expansion_candidates = [4, 8]):
  if isinstance(v, (FusedIBN, IBN)):
    v.kernel_size = oneof(kernel_candidates)
    v.expansion = oneof(expansion_candidates)
```

```
def tune_groups(v,
                groups_candidates = [1, 2, 4]):
  if isinstance(v, FusedIBN):
    v.groups = oneof(groups_candidates)
```

*Figure 4.* (a) Tunable Layers for Edge Accelerators. In a Fused-IBN layer, we use a $one\_of$ operation in PyGlove to select whether to use a regular depthwise convolution or a fused convolution. (b) Evolved Search Space: NaaS respects Efficientnet's compound scaling ratios, while introducing edge efficient Fused-IBN with tunable parameters. We use a symbolic programming library named PyGlove (Peng et al., 2020) to tune **filter size, kernel size, expansion factor, and groups**.

design of an evolved search space.

**Fused IBN Layers:** We adopted the fused IBN layer from MobileDets (Xiong et al., 2021) that modifies an IBN layer by fusing together the first 1x1 convolution (which usually comes with an expansion ratio) and its subsequent $K$x$K$ depthwise convolution with a single $K$x$K$ full convolution. We keep the expansion factor as conventional MobileNet models to enable channel size expansion. We name this *Fused-IBN*. Fused-IBN provides more trainable parameters over baseline IBN but runs faster on edge accelerators for some tensor shapes.

**Tunable Layers via Symbolic Programming:** Intuitively, Fused-IBN becomes less efficient when the network gets deeper and the channel dimension gets larger, due to the cost of full convolution over large input channels. We introduce a more flexible, tunable search space via PyGlove (Peng et al., 2020) symbolic library, to select ops type at each layer as well as selecting layer parameters such as kernel size, expansion factor, filter scaling multiplier, and number of groups. This flexibility provides a middle ground between IBN and Fused-IBN layers.

Figure 4 (a) demonstrates the idea of a switched Fused-IBN layer. For some tensor shapes, Fused-IBN can be more effective when running on edge accelerators, such as early layers with smaller channel sizes. However, as the network goes deeper and channel size becomes larger, we may still adopt the conventional IBN to reduce latency and cost. Figure 4 (b) demonstrates our evolved search space, which consists of a mixture of Fused-IBN, and conventional IBN layers. In our manually crafted model (*Manual-EdgeTPU* presented in our evaluation results) based on the evolved search space, we use a fixed number of fused-IBN in the early layers. In NaaS, we use PyGlove (Peng et al., 2020) to symbolize each layer to create a tunable search space. For example, we can replace any static node in a computational graph, with a tunable node with a predefined search space (e.g., filter scaling, number of groups, kernel size, and expansion factor, etc.). Therefore, we can essentially transform any static neural network into a tunable search space.

## 3.3 Accelerator Search Space

We target Google's Edge TPUs, a domain of accelerators for low-power, edge devices that are widely used in various Google products such as Coral (Google, 2020) and Pixel (Osterloh, 2021) devices. Edge TPUs are parameterized and contain a large design space with tradeoffs between performance, power, area, and cost. The accelerator in Figure 5 features a set of parallel processing elements (PE) organized in a 2D tile. The number of PEs in each dimension determines the aspect ratio of the chip. In each PE there are multiple compute lanes that shares a local memory and each lane has a register file and a series of single-instruction multiple-data (SIMD) style multiply-accumulate (MAC) compute units. Each of these architecture components provide a degree of parallelism and a corresponding area cost.

With a fixed chip area budget, the optimal accelerator parameters $h$ essentially provisions a balanced mixture of computation and memory resources for a targeted application or a domain. We also found that the aspect ratio of processing elements, the hierarchy of parallelism also affect model performance. **Note that the target accelerator is a highly**

| parameters | type | search space |
|---|---|---|
| PEs_in_x_dimension | int | 1, 2, 4, 6, 8 |
| PEs_in_y_dimension | int | 1, 2, 4, 6, 8 |
| SIMD_units | int | 16, 32, 64, 128 |
| compute_lanes | int | 1, 2, 4, 8 |
| local_memory_MB | int | 0.5, 1, 2, 3, 4 |
| register_file_KB | int | 8, 16, 32, 64, 128 |
| io_bandwidth_gbps | int | 5, 10, 15, 20, 25 |

*Table 2.* Edge TPU Search Space. The search space contains only valid points in the design space.

**parameterized design and our search space presented in Table 2 contains only valid architectures.** Enlarging the accelerator search space that could results in sampling invalid design points will be a future direction but not the focus of this paper. However, a sampled neural architecture in combined with an accelerator configuration can be invalid because compilation can fail due to the incompatibility between the sampled neural architecture and the sampled accelerator configuration. For instance, the sampled neural architecture can be too large to execute on an accelerator with small on-chip memory.

As our baseline accelerator configuration used by the compared related work, we take the default accelerator configuration optimized over a series of production workloads from multiple domains. The baseline configuration features 4x4 PEs where each PE has 2 MB local memory and 4 compute lanes. Each compute lane has a 32 KB register file and 64 4-way SIMD units. Since the accelerator is targeted for edge use cases, SIMD units can sustain the peak throughput for 8-bit quantized operations. This baseline configuration can deliver a peak throughput of 26 TOPS/s at 0.8 GHz.

## 3.4 Search Objective

Chip power and area are two important efficiency metrics for accelerator design. In this work, we focus on maximizing model accuracy while meeting an inference latency constraint on a target device. Chip latency is determined by both neural architectures as explained in Table 1, and accelerator configurations (e.g. how many PEs, memory-to-compute ratio, bandwidth, etc.). The latency constraint can be easily swapped with an energy (average power times latency) constraint that considers both power and latency. We also impose a chip area constraint, which is set equal to the baseline accelerator design. This paper does not focus on minimizing chip area or latency beyond these set constraints.

Similar to MnasNet (Tan et al., 2019), we use a customized weighted product to encourage Pareto optimal solutions. More specifically, we have a *optimization metric* of model accuracy and two *hardware constrain metrics* of model latency and chip area. The optimization goal is to maximize model accuracy while meeting the latency and area constraints.

$$\max_{\alpha,h} Accuracy(\alpha,h) \times [\frac{Latency(\alpha,h)}{T_{latency}}]^{w_0} \times [\frac{Area(h)}{T_{area}}]^{w_1}$$

where $w_0, w_1$ are the weight factors:

$$w_0 = \begin{cases} p, \text{ if } Latency(\alpha,h) \le T_{latency} \\ q, \text{ otherwise} \end{cases}$$

$$w_1 = \begin{cases} p, \text{ if } Area(h) \le T_{area} \\ q, \text{ otherwise} \end{cases}$$

When $p = 0, q = -1$, the reward function imposes a hard latency constraint and we simply use accuracy as the objective if the measured latency is meeting the latency target $T$ and only sharply penalize the objective value if the sample violates the latency constraint. When $p = q = -0.07$[1], the reward becomes a soft constraint function.

## 3.5 Search Strategy

### 3.5.1 *Joint search w/o weight sharing*

Similarly as NASNet (Zoph et al., 2018) and MnasNet (Tan et al., 2019), we use PPO (Schulman et al., 2017) as the controller algorithm to optimize the joint search space from NAS and HAS. The controller samples the search space using a recurrent network, each sample is trained by a child program. For the MobileNetV2 search space, we use a proxy task that trains each sample on ImageNet for only 5 epochs and it takes 5000 samples for the controller to converge. For larger models using EfficientNet search space, training the proxy task for 15 epochs while reducing the total number of samples to 2000 improves the results.

### 3.5.2 *Joint search with weight sharing*

To further reduce the search cost, we employ an efficient search method with weight sharing. Similarly as Proxyless-NAS (Cai et al., 2019) and TuNAS (Bender et al., 2020), we use the controller decisions from the NAS space to construct a super-network for optimizing the architecture, meanwhile using the decisions from the HAS space to create a sub-graph for computing the cost. Decision points from both spaces are optimized by a RL algorithm within the same graph. For each training step, we train the model weights and the controller decision points in an interleaved way. To achieve better results, we apply the absolute reward function and RL warm-up procedure introduced in TuNAS.

To estimate the latency of the model on a given accelerator configuration, we train a cost model with random generated samples using an in-house accelerator simulator. We need

[1](Tan et al., 2019) found -0.07 to empirically ensure Pareto-optimal solutions have similar reward under different accuracy-latency trade-offs.

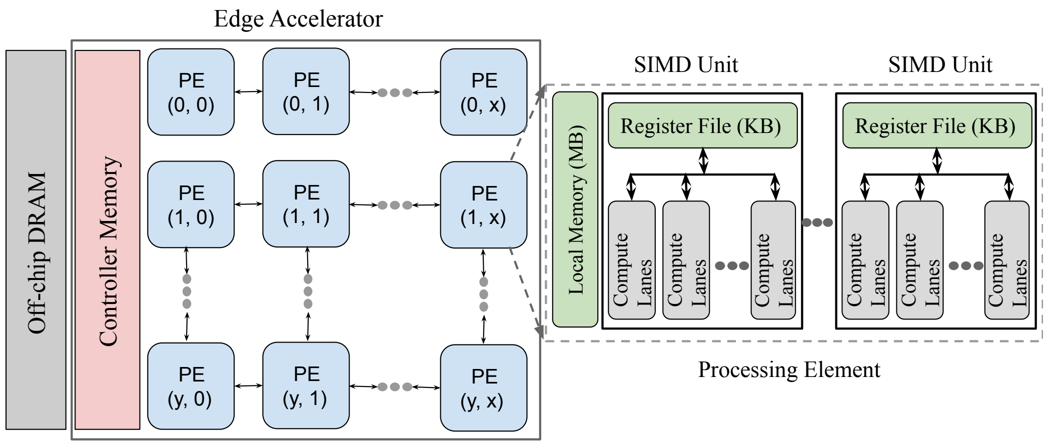


*Figure 5.* Edge TPU High-level Architecture: A set of parallel processing elements (PE) organized in a 2D tile. In each PE there are multiple compute lanes that shares a local memory and each lane has a register file and a series of single-instruction multiple-data (SIMD) style multiply-accumulate (MAC) compute units.

a cost model because as NAS becomes much faster with oneshot search, the query to the accelerator performance simulator for chip area and inference latency becomes the new bottleneck for NaaS oneshot search. Given a sampled neural architecture configuration and an accelerator configuration, the cost model predicts the accelerator area $f_a(h)$ and model accuracy $f_l(\alpha, h)$. We use an MLP network with ReLU to encapsulate the non-linearity in the latency prediction. The area predictor and latency predictor largely share parameters with only separate parameterization in the prediction heads.

$$Loss = MSE(L_a, f_a(h)) + \lambda MSE(L_l, f_l(\alpha, h)), \quad (2)$$

The cost model was trained with 500k labeled data randomly generated by permuting the neural architecture configurations and accelerator configurations. Because labeled data for accelerator performance is much cheaper than labeled data for NAS accuracy, and the collection of data can utilize the vast amount of CPU resources, we do not consider the cost of training a cost model. We use a 3-layer MLP of hidden size 256 and apply a dropout of 0.1 to mitigate overfitting at each layer.

## 4 EVALUATION

In this section, we evaluate NaaS on its latency/energy efficiency compared to highly competitive baselines on ImageNet classification and Citystacpe and ADE20K segmentation. We evaluate the searched models on real hardware.

### 4.1 Experiment Setup

**Accelerator Performance Simulator:** Evaluating the candidate neural and hardware architectures accurately is a key requirement for the NaaS framework. Simply using the number of MACs or parameters of the neural model as a proxy for performance can be highly inaccurate as its performance highly depends on how the neural network is mapped on the hardware architecture and its unique compute characteristics. To this end, we have utilized an in-house cycle-accurate performance simulator and an analytical area model based on hardware synthesis. We deployed both of these estimators as a service where multiple NaaS clients can send parallel requests. This provides a flexible way to scale-up the performance and area evaluations.

**Controller:** For the multi-trial search, we choose PPO as it is tested by time. We use the average performance of 10 trials as a reward. We use Adam optimizer with the learning rate of 0.0005 to update the controller, where the policy gradients are clipped by 1.0. For the oneshot search, we utilize REINFORCE to optimize the controller following TuNAS. We use Adam with a learning rate of 0.0048 to optimize it and use the momentum as 0.95 for baseline. In addition, we use RMSProp to optimize the shared weights following the same learning rate schedule as TuNAS. The latency predictor is pre-trained. **Proxy task:** We use a proxy task with five training epochs for the multi-trail search on ImageNet classification, using RMSProp. For these five epochs, we will first warm up the model by two epochs using the learning rate from 0 to 0.66, and then cosine decay it from 0.66 to 0 for the rest three epochs. On CityScapes segmentation (Cordts et al., 2016), we train the sampled candidate from scratch with a SGD optimizer and a cosine learning rate of 0.08 and warmup proportion of 0.01. We use a training batch size of 64 and evaluation batch size of 32. During the NaaS search, we use a proxy training of 20 epochs. In the final evaluation, we train the models for 1000 epochs, warming start from a model pretrained on

| Model | Top-1 Acc. | Latency in **ms** (Ratio-to-best) | Energy in **mJ** (Ratio-to-best) |
|---|---|---|---|
| EfficientNet-B0 (Tan & Le, 2019) wo SE/Swish | 74.7% | 0.35 (1.17x) | 1.00 (1.64x) |
| MobileNetV2 (Sandler et al., 2018) | 74.4% | 0.30 (1.00x) | 0.70 (1.15x) |
| MnasNet-B1 (Tan et al., 2019) | 74.5% | 0.41 (1.37x) | 0.88 (1.44x) |
| ProxylessNAS (Cai et al., 2019) | 74.8% | 0.42 (1.40x) | 0.98 (1.61x) |
| Manual-EdgeTPU-small | 76.2% | 0.42 (1.40x) | 1.78 (2.91x) |
| IBN-only fixed accelerator | 74.6% | 0.38 (1.27x) | 0.82 (1.34x) |
| **IBN-only NaaS multi-trial** | 74.9% | **0.30** | 0.75 (1.23x) |
| **IBN-only NaaS oneshot** | **76.5%** | 0.35 (1.17x) | **0.61** |
| EfficientNet-B1 (Tan & Le, 2019) wo SE/Swish | 76.9% | 0.51 (1.04x) | 1.50 (1.53x) |
| MnasNet-D1 (Tan et al., 2019) | 75.1% | 0.55 (1.12x) | 1.75 (1.78x) |
| Fixed accelerator multi-trial w fused-IBN | 75.3% | 0.52 (1.06x) | 1.32 (1.35x) |
| IBN-only NaaS multi-trial | 77.4% | 0.52 (1.06x) | 1.50 (1.53x) |
| **NaaS multi-trial w fused-IBN** | **77.9%** | 0.51 (1.04x) | 1.12 (1.14x) |
| **IBN-only NaaS oneshot** | 76.8% | **0.49** | **0.98** |
| EfficientNet-B3 (Tan & Le, 2019) wo SE/Swish | 78.8% | 0.72 (1.12x) | 2.28 (1.56x) |
| Manual-EdgeTPU-medium | 77.2% | **0.62** | 2.72 (1.86x) |
| MobilenetV3 w SE | 76.8% | 1.44 (2.32x) | 4.00 (2.74x) |
| Fixed accelerator multi-trial w fused-IBN | 78.2% | 0.74 (1.19x) | 1.75 (1.20x) |
| **NaaS multi-trial w fused-IBN** | **79.5%** | 0.72 (1.12x) | **1.46** |

*Table 3.* Comparison on ImageNet classification, on accuracy v.s. latency with previous approaches. Models are grouped into different regimes and sorted by accuracy. On the small latency regime, "NaaS oneshot" has the best performance, while on the high accuracy regime, "NaaS multi-trial" achieves the best performance. Note we remove Swish and Sqeeze-and-Excite layers from our search space and the re-implemented EfficientNet baselines, as they consume too much on-chip memory leading to 2x slower inference on Edge TPUs.

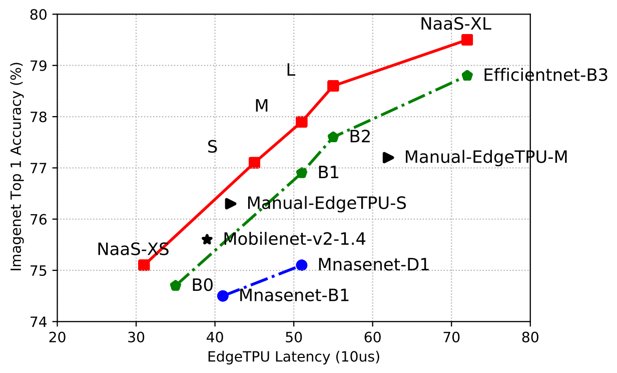


*Figure 6.* **Inference Latency vs. ImageNet Accuracy.** NaaS significantly outperform other platform-aware, efficient ConvNets. All the related work are evaluated on a fixed accelerator described in Section 3.3.

ImageNet.

Inference latency can be a critical metric for applications with stringent service-level-agreement (SLA) requirements (e.g. mobile face recognition) or real-time requirements (e.g. autonomous driving). In this section, we compare a latency-driven NaaS based on both search spaces defined in Section 3.2.1 and in Section 3.2.2 with platform-aware NAS and manually crafted models for Edge TPUs. Manutal-EdgeTPU-S and Manual-EdgeTPU-M are manually designed models targeting Edge TPU on the evolved search space described in Section 3.2.2.

#### *4.1.1 ImageNet Classification*

**Accuracy vs. Latency**: Demonstrated in Figure 6, NaaS consistently outperforms related work by around 1% ImageNet top-1 accuracy at all latency targets. With the same accuracy, NaaS on average reduces inference latency by around 20%. We empirically found that an IBN-only search space is good for identifying small, low-latency models (e.g. NaaS-XS, NaaS-S) while the proposed evolved search space is good for identifying larger, more accurate models (e.g. NaaS-M, NaaS-L, NaaS-XL) with more relaxed latency targets. Manually crafted models can be very efficient at a particular latency target (e.g. Manual-EdgeTPU-S), however, they can be less efficient when comparing the entire latency spectrum (e.g. Manual-EdgeTPU-M), as model latency is a complex function of accelerator resources, compute-to-memory ratio, and neural model characteristics (op types and tensor shapes).

**Accuracy vs. Energy**: Mobile processors have a stringent power budget due to limited battery life, therefore, maximizing model accuracy while meeting a power budget is another useful way to co-design neural architectures and accelerators. We evaluated an energy-driven NaaS, where energy is a product of power and latency, comparing to related mobile-efficient NAS and Manual-EdgeTPU. We found that **our method can improve energy efficiency by up to 2x comparing to related work at the same ImageNet top-1 accuracy**, as shown in Figure 1.

| Backbone | Head | mIOU Acc. | Latency in **ms** (Ratio-to-best) | Energy in **mj** (Ratio-to-best) |
|---|---|---|---|---|
| EfficientNet-B0 wo SE/Swish | bi-FPN | 73.8% | 3.29 (1.10x) | 6.71 (1.03x) |
| EfficientNet-B1 wo SE/Swish | bi-FPN | 72.8% | 3.66 (1.22x) | 8.04 (1.23) |
| EfficientNet-B2 wo SE/Swish | bi-FPN | 72.6% | 3.71 (1.24x) | 8.48 (1.30x) |
| Manual-EdgeTPU-S w groups | Deeplab-v3 | 74.8% | 4.82 (1.60x) | 6.75 (1.03x) |
| Manual-EdgeTPU-S wo groups | Deeplab-v3 | 74.1% | 4.30 (1.43x) | 6.93 (1.06x) |
| Manual-EdgeTPU-S | bi-FPN | 71.2% | 4.15 (1.38x) | 7.82 (1.20x) |
| Manual-EdgeTPU-M | bi-FPN | 74.4% | 4.75 (1.58x) | 8.91 (1.36x) |
| IBN-only NaaS multi-trial | bi-FPN | 73.6% | 3.06 (1.02x) | **6.54** |
| **NaaS multi-trial w fused-IBN wo groups** | bi-FPN | 74.7% | **3** | 7.68 (1.17x) |
| **NaaS multi-trial w fused-IBN w groups** | bi-FPN | **75.5%** | 3.01 (1.0x) | 7.27 (1.11x) |

*Table 4.* Comparison on Cityscape Segmentation Dataset (Cordts et al., 2016). NaaS on customized bi-FPN head achieves the best mIOU and lowest accelerator latency and energy.

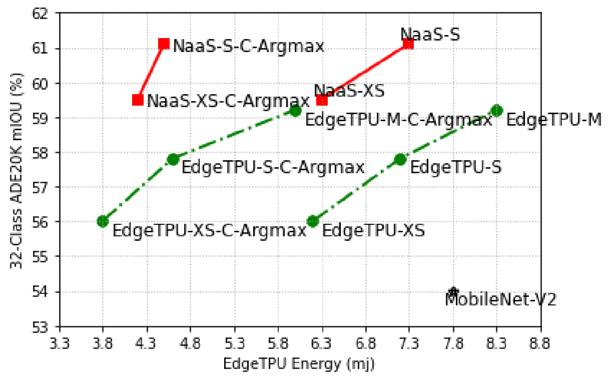


*Figure 7.* Comparison on 32-Class ADE20K. EdgeTPU models are based on TuNAS (Bender et al., 2020) with manual optimizations, thus are so far the best baselines on Edge TPUs. We apply a customized argmax to further improve on-device performance (C-Argmax).

**Comparing to Related Work**: In Table 3, We compare NaaS with mobile-efficient models (including NAS-based and manually crafted). We also compare NaaS variants by 1) Fixing the accelerator to the default accelerator. 2) Studying multi-trial NaaS and oneshot NaaS. 3) Comparing the IBN-only search space and IBN+fused-IBN search space. We categorize the models into three: small, medium, and large. We use a latency target of 0.3ms, 0.5ms, and 0.7ms and use a energy target of 0.7mJ, 1.0mJ, and 1.5mJ for these three categories. NaaS significantly outperforms related work on all three perspectives (accuracy, latency, and energy), except for latency when comparing to medium sized manually crafted Manual-EdgeTPU. However, Manual-EdgeTPU consumes much more (1.82x) energy than NaaS.

### *4.1.2 Semantic Segmentation*

The image classification models are often employed as feature extractors for other vision tasks such as semantic segmentation. In this experiment, we compare NaaS search with related segmentation models. We invoke architecture search to jointly search for an efficient feature extractor and an edge-efficient Bi-FPN segmentation head (first proposed in EfficientDet (Tan et al., 2020)).

**Cityscape Performance Targeting Coral Edge TPU:** Table 4 compares NaaS with related work consisting of SoTA backbones and heads on Cityscape. NaaS with bi-FPN head demonstrates the highest mIOU, and improves latency and energy by up to 60% and up to 36% respectively, compared to all baselines. The on-device measurement aligns with the performance simulator. On Coral Edge TPU, we observe 1% accuracy gain and 27% speedup compared to EfficientNet+bi-FPN, and 3% accuracy gain and 30% speedup compared to Manual-EdgeTPU+Deeplab-v3.

**32-Class ADE20K Targeting Tensor's Edge TPU (Osterloh, 2021):** Figure 7 demonstrates a real-chip evaluation on NaaS targeting Tensor Edge TPU for Pixel 6. The final layers of the segmentation model consists of upsampling the features to the desired resolution and applying argmax to determine the per-pixel class labels. As an approximation, we upsample the features to some intermediate resolution, apply argmax and finally perform nearest neighbor upsampling to the final desired solution. This approximation helps improve the on-device latency by nearly 2x without degrading the mIOU or introducing significant visual artifacts. We compare NaaS with manually optimized models based off TuNAS search (Bender et al., 2020) (EdgeTPU-Series), and find that NaaS yields 3% better mIOU at the same on-device inference latency.

## 4.2 Search Method

Figure 8 compares flattened search with coordinate descent as explained in Section 3.1. More particularly, we start with an initial accelerator configuration and conduct NAS in the inner loop and identify top-k neural models to aggregate the reward, then we do gradient descent on the accelerator configuration. We compare coordinate descent using 1x and 2x total searched samples, compared to the flattened search

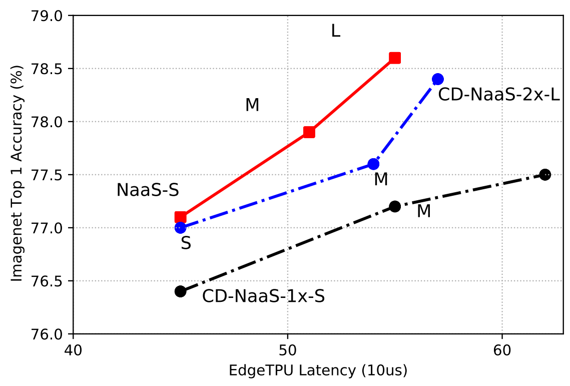


*Figure 8.* Comparisons across different search strategies. Flattened search is more sample efficient than coordinate descent based search.

baseline. NaaS coordinate descent with the same number of searched samples performs much worse than the NaaS flattened search. Doubling the total searched samples thus the search time, improves the results of coordinate descent.

### 4.3 Discussion

**-Can neural architectures targeting one hardware run efficiently on another hardware?**
Not necessary. Squeeze-and-excite with global pooling has been found effective on GPUs and TPUs in EfficientNet (Howard & Gupta, 2020), however, it consumes too much memory on Edge TPUs where on-chip memory is scarce. Similarly, Sigmoid in Swish is expensive in computation on edge accelerators. Removing both improves latency by more than 2x without significantly impairing model accuracy.

**-Expensive operations with more parameters and Flops can be more effective with certain tensor shape.**
Some expensive ops such as fused-IBN, can be more effective, when used in early layers with small channel sizes. Fused-IBN increase kernel compute intensity, therefore increases data reuse. It can be critical to edge accelerators where on-chip memory is scarce.

**-Accelerator configuration should be optimized for different applications and latency targets.**
Larger models require a higher memory-to-compute ratio in the accelerator design. NaaS identifies edge accelerator configurations with larger number of processing elements (PE) and smaller memory capacity, for small models with very tight latency/energy target, compared to the baseline edge accelerator; It identifies accelerator configurations with larger local memory and register file size for models containing fused-IBN, which gives additional performance boost for models with fused-IBN. This provides incentives to customize accelerators for different application scenarios such as low latency models in autonomous vehicles.

**-Oneshot search is not panacea.**
Oneshot neural architecture search with parameter sharing is more effective than a multi-trial search without parameter sharing for smaller models with a lower latency. However, it becomes less effective when the model search space becomes larger or the latency/energy target is more relaxed. For example, oneshot only allows a cell-based search which with restricted search space, however, multi-trial enables flexible layer-wise search. In addition, constructing a supernetwork in an oneshot search can be impractically too expensive when the search space gets bigger or the model becomes larger (EfficientNet-B4).

**-Bi-FPN reduces tensor sharding compared to deeplab-v3 segmentation head, thus significantly improves inference time on Edge TPUs.**
Upsampling in deeplab-v3 introduces additional tensor sharding that significantly hurts inference time on edge accelerators. This can be hardly captured by a performance model using FLOPs or parameter size. Bi-FPN introduces overhead in activation memory, but can be mitigated by feature selection.

## 5 Conclusion

We systematically study the importance and strategies of co-designing neural architectures and hardware accelerators. Our experiments show that the joint search method consistently outperforms related work. Our method can reduce energy consumption of an edge accelerator by up to 2x under the same accuracy constraint.

## References

Spec cpu 2006, 2006. URL https://www.spec.org/cpu2006/.

Tpu, 2016. URL https://cloud.google.com/tpu/.

Mlperf, 2019. URL https://mlperf.org.

Apple m1, 2020. URL https://www.apple.com/mac/m1/.

Achararit, P., Hanif, M. A., Putra, R. V. W., Shafique, M., and Hara-Azumi, Y. Apnas: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators. *IEEE Access*, 8:165319–165334, 2020.

Ansel, J., Kamil, S., Veeramachaneni, K., Ragan-Kelley, J., Bosboom, J., O'Reilly, U.-M., and Amarasinghe, S. OpenTuner: An extensible framework for program autotuning. In *PACT*, 2014.

Balaprakash, P., Tiwari, A., Wild, S. M., Carrington, L., and

Hovland, P. D. AutoMOMML: Automatic multi-objective modeling with machine learning. In *HiPC*, 2016.

Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.-J., and Le, Q. V. Can weight sharing outperform random architecture search? an investigation with tunas. In *CVPR*, 2020.

Bender, G. M., jan Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *ICML*, 2018.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: One-shot model architecture search through hypernetworks. In *ICLR*, 2018.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Chen, Y., Emer, J. S., and Sze, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ISCA*, 2016.

Choi, K., Hong, D., Yoon, H., Yu, J., Kim, Y., and Lee, J. Dance: Differentiable accelerator/network co-exploration. In *DAC*, 2021.

Cong, J., Wei, P., Yu, C. H., and Zhang, P. Automated accelerator generation and optimization with composable, parallel and pipeline architecture. In *DAC*, 2018.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., and Jha, N. K. Chamnet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019.

Dong, X., Tan, M., Yu, A. W., Peng, D., Gabrys, B., and Le, Q. V. AutoHAS: Efficient hyperparameter and architecture search. In *NAS Workshop at ICLR*, 2021.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

Google. Helping you bring local ai to applications from prototype to production, 2020. URL `https://coral.ai/products/`.

Gupta, S. and Akin, B. Accelerator-aware neural network design using automl. In *MLSys On-device Intelligence Workshop*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

Howard, A. and Gupta, S. Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU. `https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html`, 2020.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *ICCV*, 2019.

Iqbal, M. S., Su, J., Kotthoff, L., and Jamshidi, P. Flexibo: Cost-aware multi-objective optimization of deep neural networks. *arXiv preprint arXiv:2001.06588*, 2020.

Jiang, W., Yang, L., Dasgupta, S., Hu, J., and Shi, Y. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):4154–4165, 2020a.

Jiang, W., Yang, L., Sha, E. H.-M., Zhuge, Q., Gu, S., Dasgupta, S., Shi, Y., and Hu, J. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020b.

Koeplinger, D., Feldman, M., Prabhakar, R., Zhang, Y., Hadjis, S., Fiszel, R., Zhao, T., Nardi, L., Pedram, A., Kozyrakis, C., et al. Spatial: A language and compiler for application accelerators. In *PLDI*, 2018.

Kwon, K., Amid, A., Gholami, A., Wu, B., Asanovic, K., and Keutzer, K. Co-design of deep neural nets and neural net accelerators for embedded vision applications. In *DAC*, 2018.

Lin, Y., Yang, M., and Han, S. Naas: Neural accelerator architecture search. In *DAC*, 2021.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *ICLR*, 2019.

Nardi, L., Koeplinger, D., and Olukotun, K. Practical design space exploration. In *MASCOTS*, 2019.

Osterloh, R. Google tensor debuts on the new pixel 6 this fall, 2021. URL https://blog.google/products/pixel/google-tensor-debuts-new-pixel-6-fall/.

Parsa, M., Mitchell, J. P., Schuman, C. D., Patton, R. M., Potok, T. E., and Roy, K. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, 14:667, 2020.

Peng, D., Dong, X., Real, E., Tan, M., Lu, Y., Bender, G., Liu, H., Kraft, A., Liang, C., and Le, Q. Pyglove: Symbolic programming for automated machine learning. In *NeurIPS*, 2020.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *ICML*, 2018.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sun, Y., Xue, B., Zhang, M., Yen, G. G., and Lv, J. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 2020.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Tan, M., Pang, R., and Le, Q. V. Efficientdet: Scalable and efficient object detection. In *CVPR*, 2020.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Xiong, Y., Liu, H., Gupta, S., Akin, B., Bender, G., Wang, Y., Kindermans, P.-J., Tan, M., Singh, V., and Chen, B. Mobiledets: Searching for object detection architectures for mobile accelerators. In *CVPR*, 2021.

Yang, A. Deep learning training at scale spring crest deep learning accelerator (intel® nervana$^{TM}$ nnp-t). In *Proceedings of the Hot Chips*, 2019.

Yang, L., Yan, Z., Li, M., Kwon, H., Lai, L., Krishna, T., Chandra, V., Jiang, W., and Shi, Y. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *DAC*, 2020.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018.

Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., Blott, M., Lavagno, L., Vissers, K., Wawrzynek, J., et al. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In *FPGA*, 2019.

Zhou, Y., Ebrahimi, S., Arık, S. Ö., Yu, H., Liu, H., and Diamos, G. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*, 2018.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.