Toward Edge-Efficient Dense Predictions with Synergistic Multi-Task Neural Architecture Search

Thanh Vu[1*]　　Yanqi Zhou[2]　　Chunfeng Wen[3†]　　Yueqi Li[3]　　Jan-Michael Frahm[1]

[1]UNC at Chapel Hill　　　　[2]Google Research　　　　[3]X, The Moonshot Factory

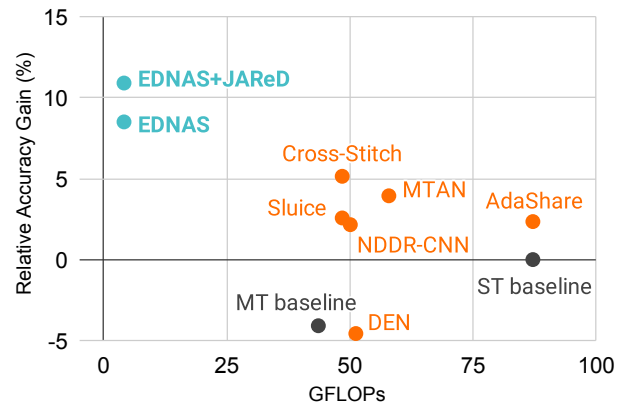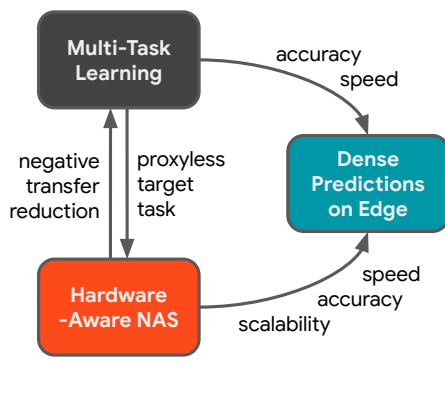{tvu,jmf}@cs.unc.edu　　{yanqiz}@google.com　　{fannywen,yueqili}@google.com

Figure 1: An overview of our proposed methods. First, EDNAS framework leverages the synergy and joint learning of multi-task dense prediction (MT-DP) and hardware-aware NAS to both complement each component and boost on-device performance. On the left is an illustration of the synergistic relationship of these components. Second, JAReD loss reduces depth estimation noise and further improves accuracy. On the right is the performance of our proposed techniques on CityScapes compared to state-of-the-art MT-DP approaches.

Abstract

In this work, we propose a novel and scalable solution to address the challenges of developing efficient dense predictions on edge platforms. Our first key insight is that Multi-Task Learning (MTL) and hardware-aware Neural Architecture Search (NAS) can work in synergy to greatly benefit on-device Dense Predictions (DP). Empirical results reveal that the joint learning of the two paradigms is surprisingly effective at improving DP accuracy, achieving superior performance over both the transfer learning of single-task NAS and prior state-of-the-art approaches in MTL, all with just 1/10th of the computation. To the best of our knowledge, our framework, named EDNAS, is the first to successfully leverage the synergistic relationship of NAS and MTL for DP. Our second key insight is that the standard depth training for multi-task DP can cause significant instability and noise to MTL evaluation. Instead, we propose JAReD, an improved, easy-to-adopt Joint Absolute-Relative Depth loss, that reduces up to 88% of the undesired noise while simultaneously boosting accuracy. We conduct extensive evaluations on standard datasets, benchmark against strong baselines and state-of-the-art approaches, as well as provide an analysis of the discovered optimal architectures.

1. Introduction

Recent years have witnessed a strong integration of computer vision in many downstream edge applications such as autonomous driving [2, 11, 38, 44, 52, 65, 68], mobile vision [16, 24, 25, 60, 61, 63], robotics [27, 35, 42], and even computational agriculture [12, 28, 37], fueled by rapid innovations of deep neural networks. In many of these applications, pixel-level dense prediction tasks such as semantic segmentation or depth estimation can play a critical role. For example, self-driving agents are using semantic and depth information to detect lanes, avoid obstacles, and locate their own positions. In precision agriculture, the output of these tasks can be used for crop analysis, yield predic-

---

tion, in-field robot navigation, etc. As more and more neural models are being deployed into the real world, there has been a continuously growing interest in developing edge-efficient architectures for dense predictions over the years.

However, designing fast and efficient dense prediction models for edge devices is challenging. First of all, pixel-level predictions such as semantic segmentation and depth estimation are fundamentally slower than some other popular vision tasks, including image classification or object detection. This is because after encoding the input images into low-spatial resolution features, these networks need to up-sample them back to produce high-resolution output masks. In fact, dense estimation can be several times or even an order of magnitude slower than their counterparts, depending on the specific model, hardware, and target resolution. Thus, real-time dense prediction models are not only non-trivial to design, they can easily become a latency bottle-neck in systems that utilize their outputs. Such problems are intensified for edge applications on platforms like the Coral TPU [13] due to the limited computational resources, despite the need for low latency, e.g., to inform the users or process subsequent tasks in real time.

Second, developing models for these edge environments is costly and hard to scale in practice. On one hand, the architectural design process requires a significant amount of time, human labor, and expertise, with the development process ranging from a few months to a couple of years. On the other hand, edge applications may require deployment on various platforms, including cell phones, robots, drones, and more. Unfortunately, optimal designs discovered for one hardware may not generalize to another. All of these together pose challenges to the development of fast and efficient models for on-edge dense predictions.

To tackle these problems, our first key insight is that Multi-Task Learning of Dense Predictions (MTL-DP or MT-DP) and hardware-aware Neural Architecture Search (h-NAS) can work in synergy to not only mutually benefit but also significantly improve accuracy and computation. To the best of our knowledge, our framework, named *EDNAS*[1], is the first to successfully exploit such a synergistic relationship of NAS and MTL for dense predictions. Indeed, on one hand, state-of-the-art methods for multi-task dense predictions [4, 22, 36, 40, 53, 58, 66], in which related tasks are learned jointly together, mostly focus on learning *how to share a fixed set* of model components effectively among tasks but do not consider if such a set itself is *optimal* for MTL to begin with. Moreover, these works typically study large models targeting powerful graphic accelerators such as V100 GPU for inference and are not readily suitable for edge applications. On the other hand, NAS methods aim to automatically learn an optimal set of neural components and their connections. However, the current

literature often focuses on either simpler tasks such as classification [7, 33, 62] or single-task training setup [19, 34]. In contrast, we jointly learn MTL-DP and NAS and leverage their strengths to tackle the aforementioned issues simultaneously, resulting in a novel and improved approach to efficient dense predictions for edge.

Our second key insight is that the standard depth estimation training used in MTL-DP can produce significant fluctuation in the evaluation accuracy. Indeed, our analysis reveals a potential for undesirably large variance in both absolute and relative depth. We hypothesize that this is caused by the standard depth training practice that relies solely on $\mathcal{L}_1$ loss function. This can significantly and negatively affect the accuracy of MT-DP evaluation as arbitrary "improvement" (or "degradation") can manifest purely because of random fluctuation in the relative error. It is important that we raise awareness of and appropriately address this issue as segmentation and depth information are arguably two of the most commonly jointly learned and used tasks in edge applications. To this end, we propose *JAReD*, an easy-to-adopt augmented loss that jointly and directly optimizes for both relative and absolute depth errors. The proposed loss is highly effective at simultaneously reducing noisy fluctuations and boosting overall prediction accuracy.

We conduct extensive evaluations on CityScapes [14] and NYUv2 [50] to demonstrate the effectiveness and robustness of EDNAS and JAReD loss. Experimental results indicate that our methods can yield significant gains, up to +8.5% and +10.9% DP accuracy respectively, considerably higher than the previous state of the art, with only 1/10th of the parameter and FLOP counts (Fig. 1).

2. Background and Related Works

In general, dense prediction models are often designed manually, in isolation, or not necessarily constrained by limited edge computation [10, 27, 34, 35]. Specifically, works on multi-task learning for dense predictions (MTL-DP) [4, 5, 20, 22, 53, 58] often take a fixed base architecture such as DeepLab [9] and focus on learning to effectively shared components, e.g. by cross-task communication modules [5, 20], adaptive tree-like branching [4, 22, 58], layer skipping [53], etc. (Fig. 2). On the other hand, neural architecture search (NAS) studies up until recently have focused mostly on either image classification problems[1, 7, 29, 33, 39, 62] or learning tasks in isolation [19, 34, 54, 67]. Few have explored architecture search for joint training of dense prediction tasks. However, as mentioned earlier, edge efficiency can potentially benefit both MTL-DP and NAS. To the best of our knowledge, our study is the first to report successful joint optimization of these two learning paradigms for dense predictions. Next, we give an overview of the most relevant efforts in the two domains of MTL and NAS. For more details, please refer to
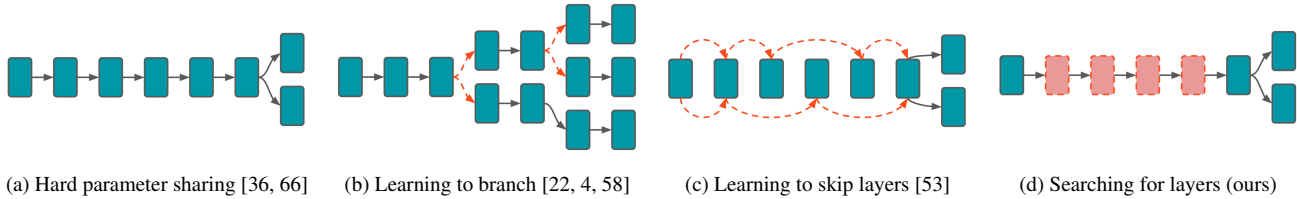
---

[1] short for "Edge-Efficient Dense Predictions via Multi-Task NAS"

(a) Hard parameter sharing [36, 66]   (b) Learning to branch [22, 4, 58]   (c) Learning to skip layers [53]   (d) Searching for layers (ours)

Figure 2: Conceptual comparison with existing approaches. While current MT-DP methods focus on how to *better share a fixed set of layers*, we instead learn *better sets of layers to share*. Components in red are learnable while others are fixed

these comprehensive surveys: MTL [8, 15], MTL for dense predictions [59], NAS [46], and hardware-aware NAS [3], .

Neural Architecture Search (NAS). In the past few years, neural architecture search (NAS) has emerged as a solution to automate parts of the network design process. NAS methods have shown remarkable progress and outperformed many handcrafted models [34, 54, 55, 56]. In our case, we are interested in hardware-aware NAS [6, 63, 67] which can discover efficient architectures suitable for one or multiple targeted edge platforms. This is typically done by casting hardware-aware NAS as a multi-objective optimization problem [6, 54, 63] and adding hardware cost, e.g. latency, memory, and energy, alongside prediction accuracy, to guide the search. However, current studies often focus on image classification [1, 7, 29, 33, 39, 62] or learning tasks in isolation [54, 67]. However, performing multiple dense prediction tasks simultaneously can have significant benefits for both inference speed and accuracy since tasks can leverage each other's training signals as inductive biases to improve their own learning and the model's generalization [8]. Thus, we are interested in combining hardware-aware NAS with multi-task learning of dense prediction tasks to achieve both better accuracy and better inference speed on edge devices. To this end, there have been only a limited number of studies [4, 22, 53, 58] that started to explore similar problems, which we will discuss next.

MTL for Dense Predictions. The goal of Multi-Task Learning (MTL) [8, 15] is to jointly learn multiple tasks together to leverage cross-task information to improve per-task prediction quality. In the context of edge applications, we are also interested in the property of MTL that lets tasks share computation and output multiple task predictions in one pass, thereby improving the overall inference speed. This is particularly useful for dense predictions because they tend to be more computationally expensive than their counterparts such as classification [24, 26, 48, 55, 56] or detection [57, 64]. A popular formulation of MTL that accomplishes this goal is called hard parameter sharing (HPS) [36, 66]. Compared to soft parameter sharing (SPS) [20], whose multi-task model size scales linearly with the number of tasks due to separate per-task sub-networks,

HPS models are more edge-friendly due to their compact architectural structure. Specifically, HPS architectures are typically composed of a shared trunk that extracts joint features for all tasks and multiple per-task heads or branches that take the extracted features as input and produce specific task prediction. The most standard setup is to have all task heads branch off at the same point [36]. This is also our setup of choice for the scope of this work. In addition, recent studies have begun to explore strategies to learn adaptive sharing architectures from data [4, 22, 40, 53, 58]. Attention [40] and Layer-skipping[53] have been used to efficiently learn a single shared model while modifying their behaviors to output the desired task-specific prediction, given a task. Other studies [4, 22, 58] opt to augment the HPS architectures by learning the branching of tasks. In other words, the learned models may have multiple splitting points, where some tasks can branch off earlier while some others share more layers. A common theme of these approaches is that given a fixed starting architecture, the focus is on learning *which components* of such network should be shared. Our work shifts the focus to the base network and instead asks *what components* should be included in such architecture to best benefit multi-task dense predictions.

3. Methodology

3.1. EDNAS: Joint MTL-DP and h-NAS

Synergistic Joint Learning. Our key idea is that we can leverage multi-task inference to significantly reduce computation across several dense prediction tasks, while utilizing hardware-aware NAS to simultaneously improve edge latency, design scalability, and multi-task learning. Combining these two paradigms, MT-DP and NAS, is beneficial not only to edge inference but also to each other. Fig. 1 illustrates these relationships. First, regarding edge applications, multi-task models [59] that output several predictions at once are attractive since they share computation across tasks to avoid multiple inference runs and improve the overall latency linearly by design. However, this multi-task setup also leads to performance degradation, known as negative transfer. While most current works attribute this problem to improper sharing of neural components, we hy-

pothesize that components of popular base networks such as DeepLab [9] -ResNet [23] may be well-tuned for their original individual task, but not necessarily optimal for multi-task setting. It is possible that certain layers, for example, may need more channels to capture nuanced features required when the number of tasks increases. Moreover, these models may need to be deployed on different edge platforms and thus, their components need to be optimized accordingly. This motivates us to explore NAS as a systematic and scalable method to discover components that could be more suitable for multi-task learning and edge inference. Second, from the perspective of NAS, directly searching for multi-task architectures can potentially yield better results than transferring single-task searched architectures to multi-task settings post NAS. In a way, we are removing a proxy target and its assumption that architectures, which are good for an individual task such as segmentation, are also optimal for multi-task learning.

Hardware-Aware Multi-Task Objective. Given a fixed set of N tasks $T = \{T_1, T_2, ...T_N\}$, we formulate the problem of multi-task NAS as a multi-objective search. Our goal is to discover optimal models with both high accuracy for all tasks in T and low inference latency on specific edge devices. Let a be an architecture with weights w_a sampled from the search space A and h be a target edge hardware. Our optimization can then be expressed as follows:

$$\max_{a \subset A} Rwd(a, T, h, w_a^*) \qquad (1)$$

$$\text{s.t.} \quad w_a^* = arg \min_{w_a} Loss(a, T, w_a) \qquad (2)$$

$$\text{and} \quad Lat(a, h) \leq l_h \qquad (3)$$

with $Rwd()$ being the objective or reward function and l_h being the target edge latency dependent on the hardware and application domain. Inspired by [54], we use a weighted product for the reward function $Rwd()$ to jointly optimize for models' accuracy and latency constrained by hardware-dependent requirements such as inference latency, chip area, energy usage, etc. This allows for flexible customization and encourages Pareto optimal solutions of multi-objective learning [17]. In this work, we focus on inference latency $Lat(a, h)$ as the main hardware constraint.

$$Rwd(a, T, h, w_a) = Acc(a, T, w_a) \left[\frac{Lat(a, h)}{l_h} \right]^\beta \qquad (4)$$

$$\text{s.t.} \quad \beta = \begin{cases} p \text{ if } Lat(a, h) \leq l_h \\ q \text{ otherwise} \end{cases} \qquad (5)$$

We use an in-house cycle-accurate performance simulator to estimate the on-device latency of sampled architectures during NAS. This offers a middle ground between the accurate-but-expensive benchmarking methods that use real, physical devices and the cheap-but-inaccurate one that use proxy metrics like FLOPs, MACs, or number of parameters. Moreover, by configuring such a simulator differently, we can inject hardware-specific information and bias the search to adapt to different targeted edge platforms.

Unlike prior works [54, 67], we extend the notion of $Acc()$ to multi-task setting using a simple-yet-effective nested weighted product of metrics and tasks. Let $M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,K}\}$ be the set of metrics of interest for tasks T_i, e.g. {mIoU, PixelAcc} for semantic segmentation. Our multi-task $Acc()$ can be expressed as:

$$Acc(a, T, w_a) = \left[\prod_i m_i \right]^{1/N} \qquad (6)$$

$$\text{s.t.} \quad m_i = \left[\prod_j m_{i,j}^{w_{i,j}} \right]^{1/\sum_j w_{i,j}} \qquad (7)$$

This extended formulation is straightforward and scalable even when the number of tasks or metrics increases. Since our goal is to discover multi-task networks that can perform well across all tasks without bias to individual tasks, we treat all task rewards equally in our formulation.

Edge-Friendly Base Architecture. Previously works [4, 22, 36, 53, 58] typically use bigger networks such as ResNet [23] or VGG [51] backbone with ASPP [9] decoder. Such models, however, are not suitable for edge platforms like the Coral TPU [13] due to their limited computational resources. To this end, we propose the use of Efficient-Net [55, 56] backbone and BiFPN fusion modules [57], which have been shown to have significantly better FLOPs and parameter efficiency (e.g. an order of magnitude lower) compared to their counterparts [32, 55, 57, 67]. These advantages make them promising candidate modules to build edge-friendly models. To generate multi-task outputs while saving computation, we share the majority of the network, including both the EfficientNet backbone and BiFPN modules, across all tasks and use only small per-task heads. This keeps our model compact and avoids a significant increase in size as the number of tasks goes up . We also replace Swish activation and attention-based fusion with ReLU6 and Sum operations in [55] to further improve efficiency on edge. We balance the compact EfficientNet backbone with 4 BiFPN fusion modules instead of 3 like [57] to boost accuracy. The multi-scale fusion modules take features $\{P_3, P_4, P_5, P_6, P_7\}$ from levels 3-7 of the backbone. These components together make up our edge-friendly base architecture, which we will use as both the seed for our NAS and the baseline model for evaluating MTL performance.

Depth Loss	mIoU	$\sigma\%$	AbsE	$\sigma\%$	RelE	$\sigma\%$
$\mathcal{L}_1$	38.6	2.5	0.01763	4.4	0.3541	4.1
JAReD	38.9	1.6	0.01680	1.9	0.3237	0.5
% improved	↑0.8	↓36.0	↓4.7	↓56.8	↓8.6	↓87.8

Table 1: Depth estimation noise and JAReD loss. We train a model for segmentation and depth prediction on CityScapes with the standard $\mathcal{L}_1$ and proposed JAReD loss. The mean and standard deviation are taken across 3 runs. Except for mIoU, lower is better.

Edge-Friendly Search Space. Modern NAS usually retains some aspect of the base architecture in order to keep the search space tractable and to reduce the computational cost. Thus, it is important to have a good initialization architecture to seed the search. For this, we leverage the base architecture designed above and Pyglove [43], a Python AutoML library that supports flexible layer-level mutation for NAS components via symbolic programming. This allows us to transform the static Efficient-Net backbone into a tunable search space by replacing any standard computational node with a PyGlove's mutable object, e.g. converting `Conv2d(kernel=3)` into `Conv2d(kernel=oneof([3,5,7]))`. Furthermore, we expand the search space to include Fused-IBN [56, 64, 67] modules alongside the standard Inverted Bottleneck (IBN) [48]. Despite inciting more trainable parameters, Fused-IBN can potentially offer better efficiency on edge devices if strategically placed, e.g. via NAS. This is because industry accelerators are better tuned for regular convolution than their depthwise counterparts, e.g. resulting in 3× speedup for certain tensor shapes and kernel dimensions [64]. Our final search space is defined by the following per-layer decisions:

- Layer type: {IBN, Fused-IBN}
- Kernel size: {3, 5}
- Output channel multiplier: {0.5, 0.75, 1.0, 1.5}
- Expansion ratio: {3, 6}

The search is performed for all 16 IBN blocks of our base EfficientNet backbone, together with the other search parameters, producing an expressive search space of size $(2 * 2 * 4 * 2)^{16} = 2^{80} \approx 1.2e24$.

3.2. Depth Estimation Noise and JAReD Loss

Instability in Depth Estimation. During our study, we discover that depth prediction accuracy can vary greatly across different training runs of the same setting. This is illustrated in Tab. 1 by the results of standard depth training with $\mathcal{L}_1$ loss. Note that the standard deviation of depth errors across identical runs are fairly large at 4.4% and 4.1%, ×2 higher than that of segmentation mIoU. Such large variation

is problematic for the multi-task evaluation as one model could potentially arbitrarily and falsely "improve" or "degrade" purely by chance. Moreover, this may even interfere with the joint learning MT-DP and NAS through noisy task accuracy in the objective function in Eq 4. In other words, it would be challenging for NAS to identify good architectures if training accuracy itself is unstable and unreliable.

Joint Absolute-Relative Depth. We hypothesize that the noisy depth result is due to the fact that popular MT-DP training [36, 53, 59] relies only $\mathcal{L}_1$ loss, which focuses on optimizing for absolute depth and only implicitly learn relative depth. For monocular setting, learning absolute depth directly is ill-posed and challenging due to the scale ambiguity [18, 31]. Instead, we propose to augment the standard loss using a weighted relative-error component, resulting in a Joint Absolute-Relative Depth loss, or *JAReD*:

$$\mathcal{L}_{JAReD} = \frac{1}{N}\Sigma|y - \hat{y}| + \lambda\frac{1}{N}\Sigma\left|\frac{y - \hat{y}}{y}\right| \qquad (8)$$

Tab. 1 shows that JAReD can help significantly reduce depth estimation noise—the STDs of all tasks decrease, especially for relative error with 87.8% lower fluctuation. Moreover, JAReD can simultaneously improve accuracy, with both absolute and relative errors dropping by 4.7% and 8.6%.

4. Experiments

4.1. Setup

Datasets and Tasks. We evaluate our proposed method using two popular datasets for multi-task dense predictions: CityScapes [14] and NYU-v2 [50]. CityScapes contains 2975 training images and 500 validation images of driving scenes while NYU-v2 is composed of 1449 densely labeled RGBD indoor images, with a stand training-to-validation split of 795 to 654. We use the preprocessed versions provided by AdaShare [53]. We jointly learn semantic segmentation (19 classes) and depth prediction for CityScapes. For NYU-v2, we study 3-task learning of segmentation, depth prediction, and surface normal estimation.

Baselines. We adopt the standard practice of evaluating our proposed techniques against the Single-Task (ST) and vanilla Multi-Task (MT) versions, which are EfficientNet-based in our case. We refer to these as *edge baselines*. For fair comparisons, we consult the training hyperparameters used by AdaShare [53] to match their baseline performance and only compare the relative improvements.

Implementation Details. For all experiments, we use EfficientNet-B0 [55] as our backbone. We use Regularized Evolution [45] as our search controller as it can produce compact and accurate models with less search time,

Method	#P	Model GFLOP	Speed	Seg mIoU	PAcc	Depth AbsE	RelE	ΔSeg ΔmIoU	ΔPAcc	ΔDepth ΔAbsE	ΔRelE	ΔT_S	Avg ΔT_D	ΔT
ST baseline [53]	42.6	87.1	—	40.20	74.70	.0170	.330	—	—	—	—	—	—	—
MT baseline [53]	21.3	43.6	—	37.70	73.80	.0180	.340	-6.2	-1.2	-5.9	-3.0	-3.7	-4.5	-4.1
Cross-Stitch [41]	42.6	48.4	—	40.30	74.30	.0150	.300	+0.2	-0.5	**+11.8**	<u>+9.1</u>	-0.1	<u>+10.4</u>	+5.1
Sluice [47]	42.6	48.4	—	39.80	74.20	.0160	.310	-1.0	-0.7	5.9	6.1	-0.8	+6.0	+2.6
NDDR-CNN [21]	44.1	50.1	—	41.50	74.20	.0170	.310	3.2	-0.7	0.0	6.1	+1.3	+3.0	+2.2
MTAN [36]	51.3	57.9	—	40.80	74.30	.0150	.320	+1.5	-0.5	**+11.8**	+3.0	+0.5	+7.4	+3.9
DEN [1]	23.9	51.2	—	38.00	74.20	.0170	.370	-5.5	-0.7	0.0	-12.1	-3.1	-6.1	-4.6
AdaShare [53]	21.3	87.1	—	41.50	74.90	.0160	.330	3.2	0.3	5.9	0.0	+1.8	+2.9	+2.3
ST edge baseline	3.4	2.3	×1.0	40.04	88.68	.0157	.340	—	—	—	—	—	—	—
MT edge baseline	3.4	1.2	×1.2	38.64	88.49	.0171	.354	-3.5	-0.2	-8.5	-4.1	-1.9	-6.3	-4.1
EDNAS	4.3	4.1	×1.3	46.52	90.61	.0143	.316	**+16.2**	**+2.2**	+8.9	+6.9	**+9.2**	+7.9	<u>+8.5</u>
EDNAS+JAReD	4.3	4.1	×1.3	46.11	90.47	.0143	.281	<u>+15.1</u>	<u>+2.0</u>	<u>+9.1</u>	**+17.4**	<u>+8.6</u>	**+13.3**	**+10.9**

Table 2: Two-task CityScapes results. Best numbers are in **bold**, the second best are <u>underlined</u>. *ST* stands for single-tasks. *MT* stands for multi-task. We multiply the FLOPs by the number of tasks for methods that need multiple runs to get different per-task predictions. FLOP counts are in gigas(G) and parameter counts are in millions(M). Both of these, along with our model's edge latency, are measured for 256x256 resolution. We consult Table 8 and Table 11 in [53] as well as its first author to acquire the full measurements of prior works

Method	Seg mIoU	PAcc	Depth AbsE	RelE	Surface Normal MeanE	θ11	θ22	θ30	Avg ΔT
ST edge	23.1	58.3	0.50	0.20	13.8	50.8	81.2	90.8	—
MT edge	19.5	54.8	0.55	0.22	16.5	41.9	73.0	85.1	-11.3
ST [53]	27.5	58.9	0.62	0.25	17.5	34.9	73.3	85.7	—
MT [53]	24.1	57.2	0.58	0.23	16.6	42.5	73.2	84.6	+2.0
MT edge	19.5	54.8	0.55	0.22	16.5	41.9	73.0	85.1	-0.1

Table 3: NYUv2 baselines. *ST* and *MT* are prior large-scale models from [53] while *edge* denotes our edge-friendly baselines

thus shortening the experimentation cycle. Nonetheless, we expect other controllers, e.g. PPO [49] as used by prior works [54, 67], to also work. We use Adam [30] optimizer and cosine learning rate scheduler for all our training, including both the proxy task during NAS and the final training of the best candidates, to reduce hyperparameter tuning effort. For full training, we train each model 3 times and take the average results similar to Table 1 to reduce noise. All models are trained from scratch without any pretrained weights. We acquire wall-clock latency measurements by benchmarking models on a Coral EdgeTPU [13]. Further details are included in the supplementary.

Evaluation Metrics. We use mean Intersection over Union (mIoU) and pixel accuracy (PAcc) for semantic segmentation, and mean absolute error (AbsE) and mean relative error (RelE) for depth prediction. For surface normal estimation on NYU-v2, we use mean angle distance error (MeanE) across all pixels, as well as the percentage of pixels with angle distances less a threshold $\theta \in \{11.25°, 22.5°, 30°\}$, denoted as $\{\theta11, \theta22, \theta30\}$ respectively. Following other works [40, 53, 59], we calculate a single evaluation score ΔT averaging over all relative gains ΔT_i of all tasks T_i relative to the Single-Task baseline. A formal definition of these metrics are provided in our supplementary materials.

4.2. Results

EDNAS for 2-task CityScapes. Tab. 2 shows our experiments for the 2-task learning of 19-class semantic segmentation and depth estimation on CityScapes dataset. In this experiment, the same ΔT of -4.1 is shared by the MT edge baseline and its large-scale counterpart, indicating that they both experience a similar level of negative transfer and MTL difficulty. Following [53], we present MTL gains relative to the ST baseline model. The proposed EDNAS exhibits a strong multi-task performance with ΔT=+8.5, outperforming all prior methods. Since the full training of MT edge baseline and EDNAS-found architecture are *identical*, it shows that joint MTL-DP and NAS can produce a superior relative improvement of +8.5 - (-4.1) = +12.6 compared to the vanilla multi-task model.

JAReD Loss. From Tab. 2, we see that the proposed JAReD loss is able to greatly improve depth estimation with a relative gain of ΔT_D=13.3%. This in turn further strengthens the overall multi-task performance by a significant margin of +2.4 on top of the already-strong result (ΔT=+8.5) of EDNAS. Together, our two proposed techniques outperform all previous approaches on 3 out of 4 individual metrics, namely ΔmIoU, ΔPAcc, and ΔRelE, as well as on all the average metrics, which are ΔT_S, ΔT_D, and ΔT.

Edge-Efficient Inference. Regarding edge efficiency, EDNAS and EDNAS+JAReD use only 1/5th of the parameters and 1/10th of the FLOPs compared to prior ResNet-based methods. More importantly, the EDNAS-found model is able to practically maintain the same on-device speed as the vanilla MT baseline, if not slightly faster, despite the +12.6 improvement. This equates to a 30% improvement in latency compared to separate single-task inferences, and fur-

	Seg		Depth		Surface Normal				Avg			
Method	mIoU	PAcc	AbsE	RelE	MeanE	$\theta 11$	$\theta 22$	$\theta 30$	ΔT_S	ΔT_D	ΔT_{SN}	ΔT
MT baseline [53]	24.1	57.2	0.58	0.23	16.6	42.5	73.2	84.6	—	—	—	—
Cross-Stitch [41]	25.4	57.6	0.58	0.23	17.2	41.4	70.5	82.9	+3.0	+0.0	-3.0	+0.0
Sluice [47]	23.8	56.9	0.58	0.24	17.2	38.9	71.8	83.9	-0.9	-2.2	-3.7	-2.3
NDDR-CNN [21]	21.6	53.9	0.66	0.26	17.1	37.4	73.7	85.6	-8.1	-13.4	-3.3	-8.3
MTAN [36]	26.0	57.2	0.57	0.25	16.6	43.7	73.3	84.4	+3.9	-3.5	+0.7	+0.4
DEN [1]	23.9	54.9	0.97	0.31	17.1	36.0	73.4	85.9	-2.4	-51.0	-4.1	-19.2
AdaShare [53]	30.2	62.4	0.55	0.20	16.6	45.0	71.7	83.0	**+17.2**	**+9.1**	+0.5	+8.9
MT edge baseline	19.5	54.8	0.55	0.22	16.5	41.9	73.0	85.1	—	—	—	—
EDNAS	22.1	57.7	0.51	0.20	14.3	49.5	79.2	89.4	+9.3	<u>+8.2</u>	<u>+11.3</u>	<u>+9.6</u>
EDNAS+JAReD	22.1	58.1	0.51	0.20	12.6	56.1	83.9	92.4	<u>+9.7</u>	<u>+8.2</u>	**+20.3**	**+12.7**

Table 4: Three-task NYUv2 results . Our tasks of interest include 40-class semantic segmentation, depth estimation, and surface normal estimation. Best numbers are in **bold**, the second best are <u>underlined</u>. *ST* stands for single-task and *MT* stands for multi-task. We multiply the FLOPs by the number of tasks for methods that need multiple runs to get different per-task predictions. The measurements of prior works are from Table 9 and Table 11 in [53]

		Model		Seg		Depth		SegΔ		DepthΔ			AvgΔ		
Method	#P	GFLOP	Speed	mIoU	PAcc	AbsE	RelE	ΔmIoU	ΔPAcc	ΔAbsE	ΔRelE	ΔT_S	ΔT_D	ΔT
ST edge baseline	3.4	2.3	×1.0	40.04	88.68	.0157	.340	—	—	—	—	—	—	—
ST edge+maxLR	3.4	2.3	×1.0	55.02	92.29	.0121	.288	+37.4	+4.1	+23.2	+15.3	+20.7	+19.3	+20.0
ST edge+maxLR+JAReD	3.4	2.3	×1.0	55.02	92.29	**.0116**	.168	+37.4	+4.1	+26.7	+50.5	+20.7	+38.6	+29.7
ST edge+maxLR+JAReD	3.4	2.3	×1.0	55.02	92.29	**.0116**	.168	—	—	—	—	—	—	—
MT edge+maxLR+JAReD	3.4	1.2	×1.2	53.80	91.94	.0124	.159	-2.2	-0.4	-7.4	+5.7	-1.3	-0.9	-1.1
Transfer: NAS-Seg→MT	4.1	2.5	×1.3	<u>58.17</u>	**92.78**	.0118	**.156**	+5.4	<u>+0.5</u>	-2.9	<u>+6.2</u>	<u>+3.1</u>	+2.8	<u>+3.0</u>
Transfer: NAS-Dep→MT	3.6	2.5	×1.3	57.97	<u>92.73</u>	.0119	<u>.158</u>	<u>+5.7</u>	<u>+0.5</u>	<u>-1.8</u>	**+7.4**	+2.9	+1.6	+2.3
EDNAS+maxLR+JAReD	4.3	4.1	×1.3	**58.54**	**92.78**	<u>.0117</u>	**.156**	**+6.4**	**+0.5**	-1.3	**+7.4**	**+3.5**	**+3.1**	**+3.3**

Table 5: Stronger baselines on CityScapes. *ST edge baseline* and *ST edge+maxLR* have identical training setting with the only exception of their learning rate being 3e-4 and 1e-2 respectively

ther demonstrates the benefits of our proposed joint learning for discovering and training better multi-task architectures for dense predictions on edge platforms.

Generalization to 3-Task NYUv2. Unlike with Cityscapes where the MT baselines have similar accuracy drop, for NYUv2, we notice a large difference between the amount of negative transfer in MT edge baseline (ΔT=-11.3) and in the large-scale MT model (ΔT=+2.0), as shown in Tab. 3. This indicates that multi-task training on NYUv2 data may be more challenging for edge models with limited computation. Because of such discrepancy in the level of MTL difficulty, we directly use the MT models (instead of ST models) as the baselines to benchmark the improvement gained. Note that despite such a large gap compared to the ST edge setting, our MT edge model is still comparable to the computationally heavy ST baselines of prior studies, with a negligible ΔT=-0.1. The NYUv2 results from Tab. 4 show that EDNAS and JAReD continue to achieve consistent and significant improvements (ΔT of +9.6 and +12.7) over the baseline, similar to what we observed for Cityscapes.

Robustness to Stronger Baselines. To further demonstrate the robustness of EDNAS as a solution for discovering better multi-task architectures for dense predictions, we are interested in examining its performance with stronger baselines (Tab. 5). Although prior work [53] only uses learning rates in the order of 1e-4 to 1e-3, we also experiment with other rates and observe a huge jump of ΔT=+20.3 in performance when simply increasing the learning rate while holding other settings the same. We utilize this simple adjustment to obtain our stronger edge baseline with the largest learning rate of maxLR=1e-2. Taking a step further, we add JAReD loss to our ST edge baseline both to demonstrate the effectiveness of JAReD loss even for single-task depth estimation and to acquire our strongest baseline for evaluation. Our result of training the EDNAS-found architecture with similar setup (+maxLR and +JAReD) illustrates the strength of our proposed method with a relative multi-task gain of ΔT=+3.3. We emphasize that +3.3, despite being smaller than the improvements we have seen so far, is *still* comparable to the majority of state-of-the-art methods shown in Tab. 2, and that is on top of a *+30% stronger* ST baseline!

Index	Layer	Stride	Kernel	Filters	Expansion
0	Conv2D	2	3	32	–
1	FusedIBN	1	3	16	1
2	IBN	2	5	36	6
3	FusedIBN	1	5	24	6
4	FusedIBN	2	3	60	6
5	FusedIBN	1	3	40	3
6	FusedIBN	2	5	120	3
7	IBN	1	3	120	3
8	FusedIBN	1	5	80	6
9	FusedIBN	1	5	168	6
10	FusedIBN	1	5	84	3
11	FusedIBN	1	5	84	6
12	FusedIBN	2	5	288	3
13	FusedIBN	1	3	96	3
14	FusedIBN	1	3	96	6
15	FusedIBN	1	3	96	3
16	FusedIBN	1	5	160	6

Table 6: **Backbone Architecture found by EDNAS** – Backbone architecture found EDNAS for multi-task segmentation and depth estimation on CityScapes, same model as presented in Tab. 2.

Index	Layer	Stride	Kernel	Filters	Expansion
0	Conv2D	2	3	32	–
1	FusedIBN	1	3	24	1
2	IBN	2	3	36	6
3	IBN	1	3	36	6
4	FusedIBN	2	5	40	6
5	FusedIBN	1	5	40	3
6	IBN	2	3	80	6
7	FusedIBN	1	3	120	3
8	FusedIBN	1	3	80	6
9	FusedIBN	1	3	168	3
10	FusedIBN	1	3	56	6
11	FusedIBN	1	3	112	3
12	FusedIBN	2	5	192	6
13	FusedIBN	1	3	192	6
14	IBN	1	5	192	3
15	IBN	1	5	192	3
16	FusedIBN	1	5	240	3

Table 7: **Backbone Architecture found by Single-task NAS** – An example of the backbone architecture found the single-task NAS targeting depth estimation on CityScapes . We suspect that multi-task learning can benefit from more expressive layers such as FusedIBN; thus, fewer of such layers compared Tab. 6 may correlate to the lower accuracy as seen in the previous experiments.

Joint Learning vs Transfer Learning. Tab. 5 also shows the performance of EDNAS when compared to the transferring of NAS-found single-task models to the multi-task setting. Although transferred architectures can bring a considerable amount of improvement compared to our baseline ST and MT models, EDNAS' joint learning of multi-task dense predictions and hardware-aware NAS evidently offers the optimal performance among these models, achieving either the best or second best scores in all categories. Moreover, it is also important to note that there is a significant difference in the performance gains of the transferred depth estimation network compared to that of the transferred segmentation model. Therefore, we may not know in advance which specific tasks transfer better than the other, further illustrating the power and benefits of our EDNAS.

Analysis of EDNAS-Found Architectures. Tab. 6 gives a summary of the backbone architecture found by ED-NAS for multi-task segmentation and depth estimation on CityScapes. This is the same model as presented in CityScapes experiment section. Except for the first Conv2D layer, which is a fixed stem, the following 16 layers (1-16) are all tunable. Our first observation is that FusedIBN is heavily favored by the search algorithm over regular IBN, occupying 14 out of 16 tunable layers. This is likely due to the fact that modern edge accelerators such as the Coral Edge TPU [13] are more optimized for normal convolution than for depthwise separable convolution. Therefore, they can leverage the dense computations to improve both accuracy and inference latency. Second, we notice that 4 out of our top 5 searched models have an IBN module at layer 2 and 7, including the one in Tab. 6. The remaining architec-

ture also has IBN for layer 7 but not for layer 2. Hence, we believe that even though sparsely used, IBN layers can still be beneficial if placed strategically, e.g. via EDNAS.

Tab. 7 provides an example of architectures found by our single-task NAS for depth estimation. We observe that there are consistently and considerably lower numbers of Fused-IBN modules, namely 11 compared to 14 in Table Tab. 6, which is produced by EDNAS, a multi-task NAS algorithm. Similar observation also applies to the single-task NAS for segmentation, which has 12 FusedIBN layers. We conjecture that multi-task learning might require more powerful and expressive layers to capture cross-task nuances. As a result, single-task NAS, which performs an indirect search using individual tasks, may fail to recognize and meet these needs, leading to fewer FusedIBN blocks and poorer accuracy as seen in the transferring experiments.

5. Conclusion

In this work, our two main contributions include EDNAS and JAReD loss. The former is a novel and scalable solution that exploits the synergy of MTL and h-NAS to improve both accuracy and speed for dense prediction task on edge platforms. The latter is an easy-to-adopt augmented depth loss that simultaneously mitigates noise and further boosts accuracy. Through extensive experimentation, we show that the proposed techniques can outperform state-of-the-art methods, minimize on-device computational cost, generalize to different data and training settings, as well as discover meaningful and effective architectures.

References

[1] Chanho Ahn, Eunwoo Kim, and Songhwai Oh. Deep Elastic Networks with Model Selection for Multi-Task Learning. In *ICCV*, 2019.

[2] Tanya Amert, Ming Yang, Saujas Nandi, Thanh Vu, James H. Anderson, and F. Donelson Smith. The price of schedulability in multi-object tracking: The history- vs.-accuracy trade-off. In *Proceedings of the IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2020.

[3] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. Hardware-Aware Neural Architecture Search: Survey and Taxonomy. In *IJCAI*, 2021.

[4] David Bruggemann, Menelaos Kanakis, Stamatios Georgoulis, and Luc Van Gool. Automated Search for Resource-Efficient Branched Multi-Task Networks. In *BMVC*, 2020.

[5] David Bruggemann, Menelaos Kanakis, Anton Obukhov, Stamatios Georgoulis, and Luc Van Gool. Exploring Relational Context for Multi-Task Dense Prediction. In *ICCV*, 2021.

[6] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[7] Ronghong Cai and Jianping Luo. Multi-Task Learning for Multi-Objective Evolutionary Neural Architecture Search. In *CEC*, 2021.

[8] Rich Caruana. Multitask Learning. In *Machine learning*, 1997.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, , and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets and atrous convolution and and fully connected CRFs. In *TPAMI*, 2017.

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin P. Murphy, and Alan Loddon Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2015.

[11] Yuntao Chen, Chenxia Han, Yanghao Li, Zehao Huang, Yi Jiang, Naiyan Wang, and Zhaoxiang Zhang. Simpledet: A simple and versatile distributed framework for object detection and instance recognition, 2019.

[12] Mang Tik Chiu, Xingqian Xu, Yunchao Wei, Zilong Huang, Alexander G. Schwing, Robert Brunner, Hrant Khachatrian, Hovnatan Karapetyan, Ivan Dozier, Greg Rose, David Wilson, Adrian Tudor, Naira Hovakimyan, Thomas S. Huang, and Honghui Shi. Agriculture-vision: A large aerial image database for agricultural pattern analysis. In *CVPR*, 2020.

[13] Coral Edge TPU. https://coral.ai/.

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, , and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. License: Cityscapes is freely available to academic and non-academic entities for non-commercial purposes.

[15] Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. *arXiv preprint arXiv:2009.09796*, 2020.

[16] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation, 2018.

[17] K. Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449, 2014.

[18] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NeurIPS*, 2014.

[19] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *CVPR*, 2020.

[20] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. MTL-NAS: Task-Agnostic Neural Architecture Search towards General-Purpose Multi-Task Learning. In *CVPR*, 2020.

[21] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, , and Alan L Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *CVPR*, 2019.

[22] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to Branch for Multi-Task Learning. In *ICML*, 2020.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[26] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

[27] Ping Hu, Federico Perazzi, Fabian Heilbron, Oliver Wang, Zhe Lin, Kate Saenko, and Stan Sclaroff. Real-time semantic segmentation with fast attention. In *IEEE Robotics and Automation Letters*, volume 6, pages 1–1, 01 2020.

[28] Andreas Kamilaris and Francesc Xavier Prenafeta-Boldú. Deep learning in agriculture: A survey. *Comput. Electron. Agric.*, 147:70–90, 2018.

[29] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. Auto-VirtualNet: Cost-adaptive dynamic architecture search for multi-task learnin. In *Neurocomputing*, 2021.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

[31] Jae-Han Lee and Chang-Su Kim. Monocular depth estimation using relative depth maps. In *CVPR*, 2019.

[32] Yanwei Li, Lin Song, Yukang Chen, Zeming Li, X. Zhang, Xingang Wang, and Jian Sun. Learning dynamic routing for semantic segmentation. In *CVPR*, 2020.

[33] Jason Liang, Elliot Meyerson, and Risto Miikkulainen. Evolutionary Architecture Search For Deep Multitask Networks. In *GECCO*, 2018.

[34] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019.

[35] Haotian Liu, Rafael A. Rivera Soto, Fanyi Xiao, and Yong Jae Lee. Yolactedge: Real-time instance segmentation on the edge. In *ICRA*, 2021.

[36] Shikun Liu, Edward Johns, and Andrew J. Davison. End-to-End Multi-Task Learning with Attention. In *CVPR*, 2019.

[37] Nils Lüling, David Reiser, Alexander Stana, and Hans W. Griepentrog. Using depth information and colour space variations for improving outdoor robustness for instance segmentation of cabbage. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[38] Chenxu Luo, Xiaodong Yang, and Alan Yuille. Self-supervised pillar motion learning for autonomous driving. In *CVPR*, 2021.

[39] Jiaqi Ma, Zhe Zhao, Jilin Chen, Ang Li, L. Hong, and Ed H. Chi. SNR: Sub-Network Routing for Flexible Parameter Sharing in Multi-task Learning. In *AAAI*, 2019.

[40] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive Single-Tasking of Multiple Tasks. In *CVPR*, 2019.

[41] I. Misra, A. Shrivastava, A. Gupta, , and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016.

[42] Kiru Park, Timothy Patten, Johann Prankl, and Markus Vincze. Multi-task template matching for object detection, segmentation and pose estimation using depth images. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7207–7213, 2019.

[43] Daiyi Peng, Xuanyi Dong, Esteban Real, Mingxing Tan, Yifeng Lu, Hanxiao Liu, Gabriel Bender, Adam Kraft, Chen Liang, and Quoc V. Le. PyGlove: Symbolic Programming for Automated Machine Learning. In *NeurIPS*, 2020.

[44] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *CVPR*, 2021.

[45] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 2019.

[46] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. In *ACM Computing Surveys*, 2021.

[47] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, , and Anders Søgaard. Latent multi-task architecture learning. In *AAAI*, 2019.

[48] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv preprint arXiv:1801.04381*, 2018.

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[50] Pushmeet Kohli Nathan Silberman, Derek Hoiem, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *CVPR*, 2012.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[52] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2020.

[53] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. AdaShare: Learning What To Share For Efficient Deep Multi-Task Learning. In *NeurIPS*, 2020.

[54] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*, 2019.

[55] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML*, 2019.

[56] Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller Models and Faster Training. In *ICML*, 2021.

[57] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. In *CVPR*, 2020.

[58] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. Branched Multi-Task Networks: Deciding What Layers To Share. In *BMVC*, 2020.

[59] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-Task Learning for Dense Prediction Tasks: A Survey. In *TPAMI*, 2021.

[60] Thanh Vu, Marc Eder, True Price, and Jan-Michael Frahm. Any-width networks. In *CVPRW*, 2020.

[61] Thanh Vu, Daniel Piros, and Amir Sadovnik. How your phone recognizes your home: An investigation of mobile object recognition. In *National Conference on Undergraduate Research (NCUR)*, 2016.

[62] Qifei Wang, Junjie Ke, Joshua Greaves, Grace Chu, Gabriel Bender, Luciano Sbaiz, Alec Go, Andrew Howard, Feng Yang, Ming-Hsuan Yang, Jeff Gilbert, and Peyman Milanfar. Multi-path Neural Networks for On-device Multi-domain Visual Classification. In *WACV*, 2021.

[63] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *CVPR*, 2019.

[64] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. In *CVPR*, 2021.

[65] Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F. Donelson Smith, James H. Anderson, and Jan-Michael Frahm. Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.

[66] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient Surgery for Multi-Task Learning. In *NeurIPS*, 2020.

[67] Yanqi Zhou, Xuanyi Dong, Berkin Akin, Mingxing Tan, Daiyi Peng, Tianjian Meng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. Rethinking co-design of neural architectures and hardware accelerators. *arXiv preprint arXiv:2102.08619*, 2021.

[68] Ming Zhu, Chao Ma, Pan Ji, and Xiaokang Yang. Cross-modality 3d object detection. In *WACV*, 2021.