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

Thanh Vu¹* Yanqi Zhou² Chunfeng Wen³† Yueqi Li³ Jan-Michael Frahm¹

¹UNC at Chapel Hill ²Google Research ³X, The Moonshot Factory
{tvu,jmf}@cs.unc.edu {yanqiz}@google.com {fannywen,yueqili}@google.com

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-

*Work done during an internship at X. †Work done while at X.
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 upsample 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 bottleneck 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 \(L_1\) 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 share 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”
Learning to branch \[22, 4, 58\]  
Learning to skip layers \[53\]  
Searching for layers (ours) \[50x81\]

the number of tasks due to separate per-task sub-networks, (SPS) \[20\], whose multi-task model size scales linearly with (HPS) \[36, 66\]. Compared to soft parameter sharing that accomplishes this goal is called hard parameter sharing or detection \[57, 64\]. A popular formulation of MTL for their counterparts such as classification \[24, 26, 48, 55, 56\] speed. This is particularly useful for dense predictions between tasks to avoid multiple inference runs and improve the overall inference speed across several dense prediction tasks, while utilizing 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-

**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-
pothezize 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 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.

$$\text{max}_{a \in A} \text{Rwd}(a, T, h, w_a^*)$$

(1)

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

(2)

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

(3)

with $\text{Rwd}()$ 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 $\text{Rwd}()$ 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 $\text{Lat}(a, h)$ as the main hardware constraint.

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

(4)

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

(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 $\text{Acc}()$ to multi-task setting using a simple-yet-effective nested weighted product of metrics and tasks. Let $M_t = \{m_{i,1}, m_{i,2}, ..., m_{i,K}\}$ be the set of metrics of interest for tasks $T_i$, e.g. $\{\text{mIoU}, \text{PixelAcc}\}$ for semantic segmentation. Our multi-task $\text{Acc}()$ can be expressed as:

$$\text{Acc}(a, T, w_a) = \left[ \prod_{i} m_{i} \right]^{1/N}$$

(6)

s.t. $m_i = \left[ \prod_{j} m_{i,j}^{w_{i,j}} \right]^{1/\sum_{j} w_{i,j}}$

(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 EfficientNet [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.
<table>
<thead>
<tr>
<th>Depth Loss</th>
<th>mIoU</th>
<th>σ%</th>
<th>AbsE</th>
<th>σ%</th>
<th>RelE</th>
<th>σ%</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{L}_1$</td>
<td>38.6</td>
<td>2.5</td>
<td>0.01763</td>
<td>4.4</td>
<td>0.3541</td>
<td>4.1</td>
</tr>
<tr>
<td>JAReD</td>
<td>38.9</td>
<td>1.6</td>
<td>0.01680</td>
<td>1.9</td>
<td>0.3237</td>
<td>0.5</td>
</tr>
<tr>
<td>% improved</td>
<td>↑10.8</td>
<td>↓36.0</td>
<td>↓4.7</td>
<td>↓56.8</td>
<td>↓8.6</td>
<td>↓87.8</td>
</tr>
</tbody>
</table>

**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 EfficientNet backbone into a tunable search space by replacing any standard computational node with a PyGlove’s mutable object, e.g. converting $\text{Conv2d(kernel=3)}$ into $\text{Conv2d(kernel=oneof([3,5,7])}$). 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 $\left(2 \times 2 \times 4 \times 2\right)^{16} = 2^{80} \approx 1.2e24$.

### 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%, $\times 2$ 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:

$$
\mathcal{L}_{JAReD} = \frac{1}{N} \sum_{y} |y - \hat{y}| + \lambda \frac{1}{N} \sum_{y} \left| \frac{y - \hat{y}}{y} \right|
$$

(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,
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 (ReE) 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 $\theta \in \{11.25^\circ, 22.5^\circ, 30^\circ\}$, denoted as $\{\theta_{11}, \theta_{22}, \theta_{30}\}$ respectively. Following other works [40, 53, 59], we calculate a single evaluation score $\Delta T$ averaging over all relative gains $\Delta T_i$ of all tasks $T_i$ relative to the Single-Task baseline. A formal definition of these metrics are provided in our supplementary materials.

### Table 2: Two-task CityScapes results.

Best numbers are in **bold**, the second best are underlined. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Seg mIoU</th>
<th>PAcc</th>
<th>Depth AbsE</th>
<th>ReE</th>
<th>Depth MeanE</th>
<th>Surface Normal 011</th>
<th>022</th>
<th>030</th>
<th>Avg ΔT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT edge baseline</td>
<td>23.1</td>
<td>58.3</td>
<td>0.50</td>
<td>0.20</td>
<td>13.8</td>
<td>50.8</td>
<td>81.2</td>
<td>90.8</td>
<td></td>
</tr>
<tr>
<td>EDNAS</td>
<td>4.3</td>
<td>54.8</td>
<td>0.50</td>
<td>0.22</td>
<td>16.5</td>
<td>41.9</td>
<td>73.0</td>
<td>85.1</td>
<td>-11.3</td>
</tr>
<tr>
<td>EDNAS+JARed</td>
<td>4.3</td>
<td>54.8</td>
<td>0.50</td>
<td>0.22</td>
<td>16.5</td>
<td>41.9</td>
<td>73.0</td>
<td>85.1</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Seg mIoU</th>
<th>PAcc</th>
<th>Depth AbsE</th>
<th>ReE</th>
<th>Surface Normal 011</th>
<th>022</th>
<th>030</th>
<th>Avg ΔT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST baseline</td>
<td>27.5</td>
<td>58.9</td>
<td>0.62</td>
<td>0.25</td>
<td>17.5</td>
<td>34.9</td>
<td>73.3</td>
<td>85.7</td>
</tr>
<tr>
<td>MT baseline</td>
<td>24.1</td>
<td>57.2</td>
<td>0.58</td>
<td>0.23</td>
<td>16.6</td>
<td>42.5</td>
<td>73.2</td>
<td>84.6</td>
</tr>
<tr>
<td>MT edge</td>
<td>19.5</td>
<td>54.8</td>
<td>0.55</td>
<td>0.22</td>
<td>16.5</td>
<td>41.9</td>
<td>73.0</td>
<td>85.1</td>
</tr>
</tbody>
</table>

### 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 $\Delta T$ 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 $\Delta T$=$+8.5$, 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 $\Delta T_D$=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 ($\Delta T$=$+8.5$) of EDNAS. Together, our two proposed techniques outperform all previous approaches on 3 out of 4 individual metrics, namely mIoU, PAcc, and ReE, as well as on all the average metrics, which are $\Delta T_S$, $\Delta T_D$, and $\Delta T$.

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-
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 underlined. *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].

<table>
<thead>
<tr>
<th>Method</th>
<th>Seg mIoU</th>
<th>Seg PAcc</th>
<th>Depth AbsE</th>
<th>Surface Normal MeanE</th>
<th>θ11</th>
<th>θ22</th>
<th>θ30</th>
<th>ΔTS</th>
<th>ΔTD</th>
<th>ΔTsw</th>
<th>Avg Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT baseline [53]</td>
<td>24.1</td>
<td>57.2</td>
<td>0.58</td>
<td>0.23</td>
<td>16.6</td>
<td>42.5</td>
<td>73.2</td>
<td>84.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-Stitch [41]</td>
<td>25.4</td>
<td>57.6</td>
<td>0.58</td>
<td>0.23</td>
<td>17.2</td>
<td>41.4</td>
<td>70.5</td>
<td>82.9</td>
<td>+3.0</td>
<td>+0.0</td>
<td>-3.0</td>
</tr>
<tr>
<td>Slique [47]</td>
<td>23.8</td>
<td>56.9</td>
<td>0.58</td>
<td>0.24</td>
<td>17.2</td>
<td>38.9</td>
<td>71.8</td>
<td>83.9</td>
<td>-0.9</td>
<td>-2.2</td>
<td>-3.7</td>
</tr>
<tr>
<td>NDDR-CNN [21]</td>
<td>21.6</td>
<td>53.9</td>
<td>0.66</td>
<td>0.26</td>
<td>17.1</td>
<td>37.4</td>
<td>73.7</td>
<td>85.6</td>
<td>-8.1</td>
<td>-13.4</td>
<td>-3.3</td>
</tr>
<tr>
<td>MTAN [36]</td>
<td>26.0</td>
<td>57.2</td>
<td>0.57</td>
<td>0.25</td>
<td>16.6</td>
<td>43.7</td>
<td>73.3</td>
<td>84.4</td>
<td>+3.9</td>
<td>-3.5</td>
<td>+0.7</td>
</tr>
<tr>
<td>DEN [1]</td>
<td>23.9</td>
<td>54.9</td>
<td>0.97</td>
<td>0.31</td>
<td>17.1</td>
<td>36.0</td>
<td>73.4</td>
<td>85.9</td>
<td>-2.4</td>
<td>-51.0</td>
<td>-4.1</td>
</tr>
<tr>
<td>AdaShare [53]</td>
<td>30.2</td>
<td>62.4</td>
<td>0.55</td>
<td>0.20</td>
<td>16.6</td>
<td>45.0</td>
<td>71.7</td>
<td>83.0</td>
<td>+17.2</td>
<td>+9.1</td>
<td>+0.5</td>
</tr>
<tr>
<td>MT edge baseline</td>
<td>19.5</td>
<td>54.8</td>
<td>0.55</td>
<td>0.22</td>
<td>16.5</td>
<td>41.9</td>
<td>73.0</td>
<td>85.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDNAS</td>
<td>22.1</td>
<td>57.7</td>
<td>0.51</td>
<td>0.20</td>
<td>14.3</td>
<td>49.5</td>
<td>79.2</td>
<td>89.4</td>
<td>+9.3</td>
<td>+8.2</td>
<td>+11.3</td>
</tr>
<tr>
<td>EDNAS+JAReD</td>
<td>22.1</td>
<td>58.1</td>
<td>0.51</td>
<td>0.20</td>
<td>12.6</td>
<td>56.1</td>
<td>83.9</td>
<td>92.4</td>
<td>+9.7</td>
<td>+8.2</td>
<td>+20.3</td>
</tr>
</tbody>
</table>

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.

### 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. Our result of training the EDNAS-found architecture 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!

### Robustness to Stronger Baselines
To further demonstrate the robustness of EDNAS as a solution for discovering better multi-task architectures for dense predictions on edge platforms.

*Robustness to Stronger Baselines* demonstrates the benefits of our proposed joint learning for discovering and training better multi-task architectures for dense predictions on edge platforms.
### Table 6: Backbone Architecture found by EDNAS

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

**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.

### Table 7: Backbone Architecture found by Single-task NAS

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

**Analysis of EDNAS-Found Architectures.** Tab. 6 gives a summary of the backbone architecture found by EDNAS 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 architecture 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 FusedIBN 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


[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.


