arXiv:1912.01106v1 [cs.CV] 2 Dec 2019

MnasFPN : Learning Latency-aware Pyramid Architecture for Object Detection on Mobile Devices

Bo Chen[1] **Golnaz Ghiasi**[2] **Hanxiao Liu**[2] **Tsung-Yi Lin**[2] **Dmitry Kalenichenko**[1] **Hartwig Adam**[1] **Quoc V. Le**[2]
[1]Google AI, [2]Google Brain
{bochen, golnazg, hanxiaol, tsungyi, dkalenichenko, hadam, qvl}@google.com

Abstract

Despite the blooming success of architecture search for vision tasks in resource-constrained environments, the design of on-device object detection architectures have mostly been manual. The few automated search efforts are either centered around non-mobile-friendly search spaces or not guided by on-device latency. We propose MnasFPN , a mobile-friendly search space for the detection head, and combine it with latency-aware architecture search to produce efficient object detection models. The learned MnasFPN head, when paired with MobileNetV2 body, outperforms MobileNetV3+SSDLite by 1.8 *mAP at similar latency on Pixel. It is both* 1 *mAP more accurate and* 10% *faster than NAS-FPNLite. Ablation studies show that the majority of the performance gain comes from innovations in the search space. Further explorations reveal an interesting coupling between the search space design and the search algorithm, and that the complexity of MnasFPN search space may be at a local optimum.*

1. Introduction

Designing neural network architectures for efficient deployment on mobile devices is not an easy task: one has to judiciously trade off the amount of computation with accuracy, while taking into consideration the set of operations that are supported and favored by the devices. Neural architecture search (NAS, [33]) provides the framework to automate the design process, where a RL controller will learn to generate fast and accuracy models within a user-specified search space. While the focus of NAS papers have been on improving the search algorithm, the search space design remains a critical performance factor that is less visited.

Despite the significant advances on NAS for image classification both in the server setting [33, 25] and in the mobile setting [24, 3, 9, 28, 6], relatively fewer attempts [7, 4, 26] focus on object detection. This is in part because the additional complexity in the search space of the detection head relative to the backbone. The backbone is a feature extractor that sequentially extracts features at increasingly finer scales, which behaves the same way as the feature extractor for image classification. Therefore, current NAS approaches either repurpose classification feature extractors for detection [9, 24, 25], or search the backbone while fixing the detection head [4]. Since the backbone is composed of a sequence of layers, its search space is sequential. In contrast, a detection head could be highly non-sequential. It needs to fuse and regenerate features across multiple scales for better class prediction and localization. The search space therefore includes what features to fuse, as well as how often and in what order to fuse them. This is a challenging task that few NAS frameworks have demonstrated the ability to handle.

One exception is NAS-FPN [7], which was the first NAS paper that tackles the non-sequential search space of the detection head. It demonstrates state-of-the-art performance when optimized for accuracy only, and provides NAS-FPNLite, a mobile variant that performs competitively on mobile devices. However, NAS-FPNLite is limited in three aspects. 1) The search process that produces the architecture is not guided by computational complexity or on-device latency; 2) The architecture was manually adapted to work with mobile devices, of which the process may be further optimized; 3) The original NAS-FPN search space was designed for accuracy-only and not tailored towards mobile use cases.

Our work addresses the above limitations. We propose a search space called MnasFPN , which is specifically designed for mobile devices where depthwise convolutions are reasonably optimized. Our search space re-introduces the inverted residual block [23], which is proven to be effective for mobile CPUs, into the detection head. We conduct NAS on the search space that is guided by on-device latency signals. The search found an architecture that is remarkably simple yet highly performant.

Our contributions include: 1) A **mobile-specific search space** for the detection head; 2) The **first attempt** to conduct latency-aware search for object detection; 3) A set of **detection head architectures** that outperform SS-

DLite [23] and NAS-FPNLite [7]. 4) **Ablation studies** showing that our search space design is locally optimal for the current NAS controller.

2. Related Work

2.1. Mobile Object Detection Models

The most common detection models on mobile devices are manually designed by experts. Among them are single-shot detectors such as YOLO [21], SqueezeDet [29], and Pelee [27] as well as two-stage detectors, such as Faster RCNN [22], R-FCN [5], and ThunderNet [20].

SSDLite [23] is the most popular light-weight detection head architecture. It replaces the expensive 3×3 full convolutions in the SSD head [17] with separable convolutions to reduce computational burden on mobile devices. This technique is also employed by NAS-FPNLite [7] to adapt NAS-FPN to mobile devices. SSDLite and NAS-FPNLite are paired with efficient backbones such as MobileNetV3 [9] to produce state-of-the-art mobile detectors. Since we design mobile-friendly detection heads, both SSDLite and NAS-FPNLite are crucial baselines to showcase our effectiveness.

2.2. Architecture Search for Mobile Models

Our NAS search is guided by latency signals that come from on-device measurements. Latency-aware NAS was first popularized by NetAdapt [31] and AMC [8] to learn channel sizes for a pre-trained model. A look-up table (LUT) was used to efficiently estimate the end-to-end latency of a network based on the latency sum of its parts. This idea was then extended in MnasNet [24] to search for generic architecture parameters using the NAS framework [33], where a RL controller learns to generate efficient architectures after observing the latency and accuracy of thousands of architectures. This framework was successfully adopted by MobileNetV3 [9] to produce the current state-of-the-art architectures for mobile CPU.

The MnasNet-style search was not accessible to researchers with limited resources. Therefore a large body of the NAS literature [3, 28, 2] focus on improving the search efficiency. These methods capitalize on the idea of hypernetwork and weight-sharing [16, 1, 19] to boost search efficiency. Despite the success in mobile classification, these efficient search techniques have not been extended to highly non-sequential search spaces, hence have not seen many applications in mobile object detection.

2.3. Architecture Search for Object Detection

Due to the above-mentioned non-sequential nature of search in object detection, NAS work on object detection has generally been limited.

NAS-FPN [7] was the pioneering work that tackles detection head search. It proposes an overarching search space based on feature pyramid networks [14]. The design covers many popular detection heads. Our work is primarily inspired by NAS-FPN, but with the goal of innovating a search space that is more mobile-friendly.

Another pioneering work was Auto-Deeplab [15], which extended NAS searches to semantic segmentation. Our work faces the similar challenge of learning the connectivity pattern across feature resolutions.

DetNAS [4] focuses on improving the efficiency of searching for the detection body. It deals with the unmanageable computation caused by the need for ImageNet pre-training for every sampled architecture during search. Our work instead searches for the head only.

More recently, NAS-FCOS [26] extends weight-sharing to the detection head in order to accelerate the search process for object detection. Similar to NAS-FPN, their search space for the detection head is based on full convolutions and not targeted for mobile. Our work is complementary to theirs, in that our latency-aware search based on a mobile-friendly search space could be accelerated with their weight-sharing search strategy.

On the mobile side, object detection architectures are rarely optimized as a primary target. Rather, they are composed of a light-weight backbone designed for classification and a predefined detection head. A partial list of work that follows this design strategy is [9, 24, 23, 32]. Our work takes a first step towards directly optimizing object detection architectures for mobile deployment.

3. MnasFPN

We overload the term MnasFPN to mean both our proposed search space and the family of architectures found via NAS, and leave disambiguation to context. Similar to NAS-FPN, MnasFPN constructs a detection network from a feature extractor backbone and a repeatable cell structure that iteratively generates new features by merging pairs of existing features. Relative to NAS-FPN, the innovations of MnasFPN are in the feature generation process, which we describe below.

3.1. Generalized Inverted Residual Block (IRB)

IRBs are well known block architectures to exploit light-weight depthwise convolutions in mobile CPUs. Ever since their original introduction in [23], IRBs have been widely used in NAS search spaces [24, 3, 9, 28].

IRBs mark a new era in block architecture design for image classification. The block architectures for classification feature extractors have evolved from regular convolutions to depthwise-separable convolutions in MobileNetV1 [10], and to IRBs in MobileNetV2 and V3 [9]. By comparison, current mobile detection head architectures, such as SSDLite and NAS-FPNLite, have only evolved to depthwise-separable convolutions, which is analogous to

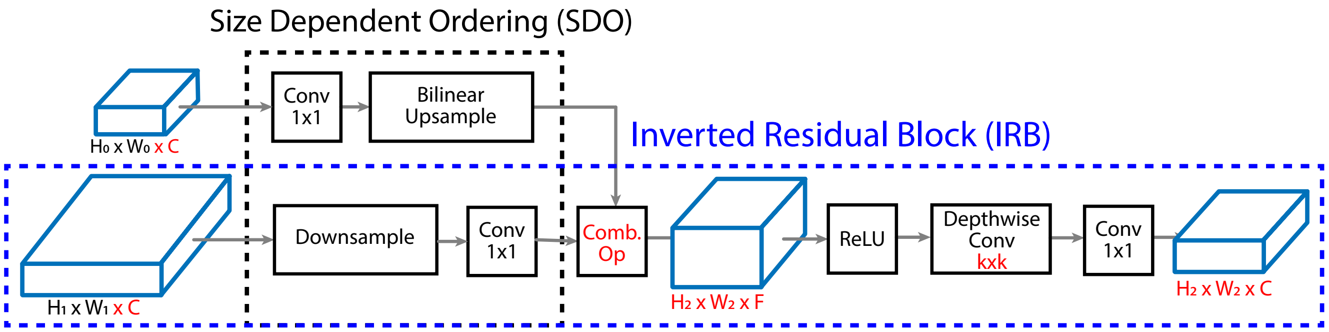

Figure 1. MnasFPN search space re-introduces the Inverted Residual Block (IRB) into the NAS-FPN head (Sec. 3.1). Any path connecting an input and a new feature, as highlighted in blue dashed rectangle, resembles an IRB. The proposed search space also employs Size Dependent Ordering (SDO) shown in black rectangle to re-order the resizing operation and the 1×1 convolution prior to feature merging (Sec. 3.2). Search-able components are highlighted in red (Sec. 3.3).

MobileNetV1 for classification. This analogy prompted us to explore the possibility of evolving the NAS-FPN search space to incorporate IRBs.

The core of NAS-FPN is the recursive merging of feature maps. All feature maps share the same, fixed channel size C by design. Once two feature maps are reshaped to the same, search-able resolution, they are added or SE-ed together to produce a merged feature. The merged feature is followed by separable convolution in NAS-FPNLite.

In MnasFPN, we allow the merged feature and the input features to have different channel sizes. Every merged feature can have its own channel size F. F can be larger or smaller than C, which means the merged feature could be a feature **expansion** or **bottleneck**. Specifically, a 1×1 convolution is applied on each input feature, if necessary, to match their channel count from C to F. Networks with unequal input and internal feature sizes are instances of **asymmetric FPNs**, as defined in [13].

Additionally, we engineered MnasFPN to dynamically choose the number of internal features. While NAS-FPN recycles all merged feature maps if they are not connected to the output, MnasFPN's merged feature maps almost never get recycled because their channel count F typically differs from C. Therefore an internal feature map in MnasFPN will be pruned away if it is not used in the merging process towards an output feature. This mechanism gives the controller additional flexibility in navigating the latency-accuracy trade-off. As the connectivity gets thinner, we found that it's helpful to augment the flow of information by adding **cell-wide residuals** between every output feature and the input feature of matching sizes, a design that is reminiscent of the residual connections in IRB.

Finally, we add *relu* non-linearity only after each combining operation, but not after generating new features. This is because the feature channel size C is intended to be small to lessen the burden on memory. Adding lossy non-linearities such as *relu*'s may unnecessarily throttle the information flow.

Given the design above, one can traverse a connected path between an input feature and an output feature and see that it resembles an IRB, as shown in Fig. 1.

We have not experimented with the MobileNetV3-styled IRB with hard-swish in the search space because their implementations were not optimized at the time of the experiment design for this paper. They are worth re-visiting once efficient kernels for hard-swish become available. We have explored Squeeze-Excite (SE) [11], but much to our surprise, it was not chosen by our NAS controller for top-performing candidates.

3.2. Size Dependent Ordering (SDO)

MnasFPN intelligently orders the reshaping ops and the convolution ops in order to reduce computation.

For notation simplicity we assume the feature maps are square, and use R to represent both the height and width. When merging feature maps, we need to apply reshaping and 1x1 convolutions when the resolution R_0 and channel count C of the input feature do not match the resolution R and channel count F of the merged feature. The order to apply the two operations depends on the relative sizes of the two feature maps.

If $R_0 > R$ and we are down-sampling, it makes sense to first down-sample and then apply 1×1 conv, as this strictly reduces the number of computes in the 1×1 conv.

Quantitatively, let $R_0 = kR$ where $k \geq 2$, and assume down-sampling is performed with $k \times k$ convolution with stride k, the cost (in MACs) of down-sample-then-1×1 is:

$$Cost_1 = R \times R \times k \times k \times C + R \times R \times C \times F \quad (1)$$

whereas the cost of 1×1-then-down-sample is:

$$Cost_2 = kR \times kR \times C \times F + R \times R \times k \times k \times F \quad (2)$$

Assume reasonably that $F \geq 2$, we have $k^2 C(F-1) \geq$

$k^2CF/2 \geq CF$, therefore:

$$Cost_2 - Cost_1 = R^2\left(k^2C(F-1) + R^2F - CF\right) > 0 \tag{3}$$

hence pruning that the down-sample-then-1×1 is more economical.

Similarly, if $R_0 < R$ and we are up-sampling, then we should first apply 1×1 before performing up-sampling. We refer to this dynamically ordering of the reshaping operation and the 1×1 convolution as Size Dependent Ordering (SDO), and examine its effectiveness in ablation studies.

3.3. MnasFPN Search

The feature generation process of MnasFPN and all the searchable components are illustrated in Fig. 1. For each feature generation block, we search for which two input features to merge, the target resolution R and channel count F of the merged feature, the merging operation (addition or SE), and the kernel sizes for the depthwise convolution post merging. For the entire network, we mandate that the input, output and generated features all share the same channel count C, which is also searched.

We adopt the architecture search framework in MnasNet [24] to incorporate latency measurements into the search objective. We train an RL controller to propose network architectures to maximize a reward function defined as follows. An architecture m is trained and evaluated on a proxy task. The proxy task is a scaled-down version of the real task, with details in Sec. 4.1. The proxy task performance, measured in mean average precision $mAP(m)$, as well as the network latency on-device $LAT(m)$ are combined into the following reward function:

$$Reward(m) = \ mAP(m) \times LAT(m)^w \tag{4}$$

where $w < 0$ controls the tradeoff point between latency and accuracy. In theory, w is the slope of the tangent line that cuts the performance trade-off curve at the desired latency. In practice, we observe that architectures around the desired latency will also be optimized, and the performance frontier of our search spaces have similar curvatures, suggesting that w needs to be set only once.

$LAT(m)$ was estimated using the latency look-up table approach similar to [31, 24]. Our particular adaptation for MnasFPN is that since MnasFPN can learn the number of merged features, we need to compute the connectivity diagram for each model at run-time to determine the layers to be included in the look-up.

The controller repeatedly proposes candidate architecture m, and trains itself based on reward feedback $Reward(m)$. After every search experiment, all the architectures sampled by the controller trace a performance frontier, as shown in Fig. 7. We can then deploy promising architectures along the frontier to the real task.

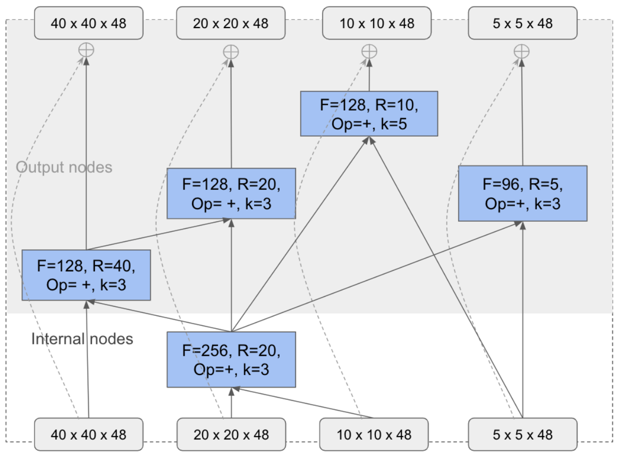

Figure 2. Visualization of a MnasFPN cell architecture found via latency-aware search. Both the inputs and outputs, represented as boxes with rounded edges, consist of four feature maps at C_3 to C_6, respectively. Each rectangle box represents a feature generation process whose internal structure is outlined in Fig. 1. The box also contains architectural parameters such as channel size F and resolution R for the merged feature, the merging operation Op, and the kernel size k of the depthwise convolution. Finally, all outputs receive cell-wide residuals (dashed arrows) from the input with the corresponding resolution. Note that although the search allows for a maximum of five internal nodes, only one was chosen.

3.4. Discovered Architectures

We compare models discovered via latency-aware search on the MnasFPN search space, and a controlled search space called "NAS-FPNLite-S".

NAS-FPNLite-S: Similar to the NAS-FPN search space, except that all models in the search spaces use separable convolutions in the head instead of full convolutions. The difference between NAS-FPNLite-S and NAS-FPNLite is that the former is searched with latency signals (hence the "S"), whereas the latter only performs the depthwise convolution replacement on a pre-discovered architecture.

We inspect a top-performing MnasFPN architecture in 2 and a NAS-FPNLite-S architecture in Fig. 3. Both models have similar latencies as MobileNetV2+NAS-FPNLite. The comparison shows that:

- We allow for at most 5 internal nodes for both searches. MnasFPN only chose one internal node, whereas NAS-FPNLite-S requires all 5 nodes at the same resolution. We suspect the compactness of MnasFPN comes from two sources: 1) the ability of MnasFPN to prune internal nodes, and 2) the expansions in IRB that increase the representational capacity for each node.

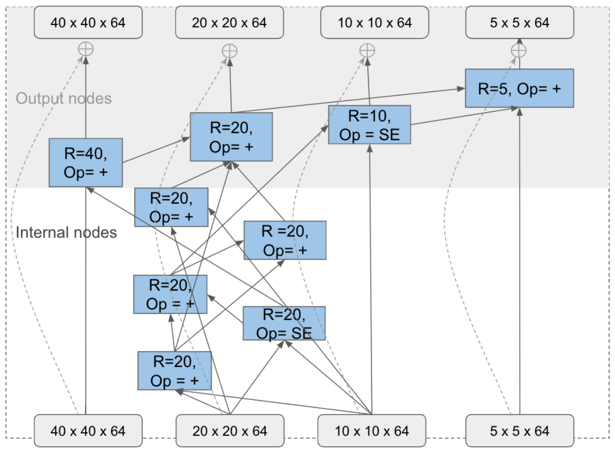

Figure 3. Visualization of a NAS-FPNLite-S cell architecture found via latency-aware search on the NAS-FPNLite search space. Each rectangle describes the resolution R and merge operation (sum or SE) for the feature generation process. The channel sizes and kernel sizes are fixed to 64 and 3, respectively, according to NAS-FPNLite [7].

- The Squeeze-and-excite (SE) option to merge features is never used. This is an interesting discovery as SE was quite popular in the classification backbone.
- Both MnasFPN and NAS-FPNLite-S chose internal nodes at 20×20 resolution. This choice was also persistent among multiple search runs and multiple variations of search spaces.

3.5. Connectivity Search

In this section we describe an attempt to expand the MnasFPN search space to include more general connectivity patterns, but are unfruitful due to limitations of current architecture search algorithms.

As recent work on randomly-wired networks [30] suggests, search quality may be hampered as much by design biases in network connectivity as by search efficiency. MnasFPN have inherited the connection rule from NAS-FPN, apart from the cell-wide residuals. Specifically, only two features are chosen to be merged each time. It is potentially inconvenient to merge $N > 2$ features across multiple scales as doing so requires $N/2$ internal nodes, which could be computationally expensive. Therefore, we propose the “ Conn-Search“ search space that allows higher in-degrees for merging.

Conn-Search: With everything else being the same as MnasFPN, each merge operation can select between 2 to $D \geq 2$ distinct feature maps as input. When $D = 2$ this degenerates to MnasFPN . Merge operation is limited to addition only as SE is both non-trivial to generalize to multiple inputs and never selected by the controller in MnasFPN.

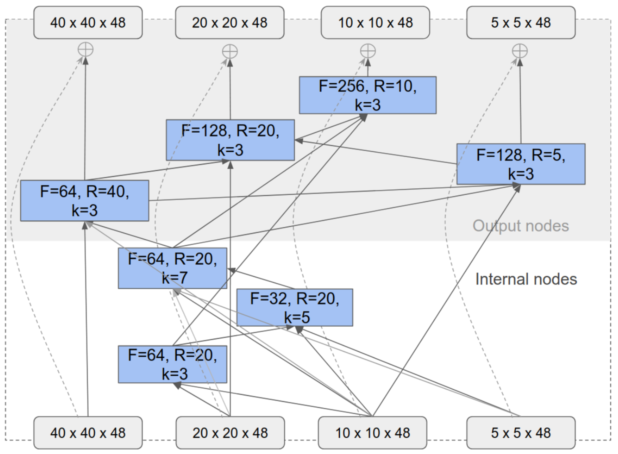

Figure 4. Visualization of a Conn-Search cell architecture with maximum in-degree $D = 4$. Each rectangle describes the expansion size F, resolution R, and kernel size k for the feature generation process. The merge operation is fixed to be summation.

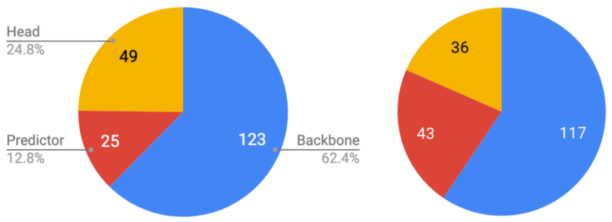

Figure 5. Latency breakdown of MnasFPN and NAS-FPNLite. Both models have around 200ms latency, out of which only less than 40% are reserved for the detection head, which we optimize in this paper, as well as the box and class predictors.

Fig. 4 shows a learned architecture with $D = 4$. Despite the larger search space, this model does not outperform MnasFPN at the same latency. Since the modified search space completely subsumes the MnasFPN search space, we conclude that the problem lies in the search process, not the search space design. Nonetheless, two trends are evident in the connectivity search. First, similar to MnasFPN, the resolutions of the intermediate features all concentrate around 20×20. Second, the controller learns to merge only 2 or 3 features almost all the time. Therefore, either allowing 4 input connections was already excessive, or the current search space is at the limit of what the controller can handle that the architecture is still at a local optimum.

3.6. Performance Ceiling

To put the improvement on the MnasFPN detection head into perspective, we plot the latency breakdown of two models at around 200 ms latency, namely MnasFPN with 5 repeats and NAS-FPNLite with 6 repeats.

As shown in Fig. 5, the majority of the computation cost is incurred at the backbone: totaling around 120ms, or 60%

of the total running time. We further separate the remaining computation into the "head", which is the focus of architecture search in this paper, and the "predictor", which is the module that predicts class probabilities and box locations from the output of the detection head. The predictor is only partially searched, as C, the channel size of the output features, dictates the sizes of the predictor's 1×1 convolutions. MnasFPN uses $C = 48$ compared to NAS-FPNLite's $C = 64$, hence giving more computational budget towards the head. However, note that this balancing act is jammed within less than 40% of the total budget.

The analysis above indicates that as the detection head becomes more efficient with MnasFPN , the backbone now becomes the performance bottleneck. Since joint search of backbone and head is outside the scope of this paper, it is reasonable to assess all improvements in the paper relative to the latency budget excluding the backbone.

4. Experiments

We present experimental results to showcase the effectiveness of the proposed MnasFPN search space. We report results on COCO object detection. We also added ablation studies to isolate the effectiveness of every component of the search space design as well as latency-aware search.

4.1. Experimental Setup

To ensure comparability we train all detection models with the same configuration and hyper-parameters. Ablation study results are reported on the 5k COCO *val2017* dataset, whereas the final comparison is reported on the COCO *test-dev* dataset.

Training setup: Training setup for COCO *val2017*: Each detection model is trained for 150 epochs, or 277k steps with a batch size of 64 on COCO *train2017* dataset. Training is synchronized with 8 replicas. Learning rate follows a step-wise procedure: it increases linearly from 0 to 0.04 in the first epoch then holds its value; The learning rate drops sharply to 0.1 of its value at epoch 120 and 140, respectively. Gradient-norm clipping at 10 was used to stabilize training. In ablation studies, models that use MobileNetV2 as the backbone are warm-started from an ImageNet pre-trained checkpoint.

Training setup for COCO *test-dev*: Each model is trained for 100k steps from scratch with a batch size of 1024 on the combined COCO *train2017* and *val2017* data over 32 synchronized replicas. We use a cosine schedule for the learning rate [18], which is decayed from a maximum value of 4 to 0. The schedule also comes with a linear warmup phase at the first 2k steps.

All training and evaluation use 320×320 input images. We do not employ drop-block or auto-augmentation or hyper-parameter tuning to avoid favoring a particular class of models in our comparison studies, and for fair comparison with some previous results in the literature.

Timing setup: All timing was performed on a Pixel 1 device with single-thread and a batch size of one using TensorflowLite's latency benchmarker[1]. Following the convention in MobileNetV2[23], each detection model is converted into TensorflowLite flatbuffer format where the outputs are the box and class predictors immediately before non-max-suppression.

Architecture Search Setup: We follow the same controller setup as used in MNASNet [24]. The controller samples about 10K child models, each taking $\sim$ 1 hour of a TPUv2 device. To train a child model, we split COCO *train2017* randomly into a 111k-*search-train* dataset and a 7k-*search-val* dataset. We train for 20 epochs with a batch size of 64 on *search-train* and evaluate its mAP on *search-val*. Learning rate increases linearly from 0 to 0.04 in the first epoch, the follows a step-wise procedure that decays to 0.1 of its value at epoch 16. We used the same 320×320 resolution for proxy task training to ensure that the estimated latency between the proxy task and the main task are identical. For the reward objective (Eq. 4), we use $w = -0.3$, estimated from a few trial runs, for all search experiments.

After training, for MnasFPN we compute the performance frontier over all the sampled models, and fetch the top models at 166 ms, 173 ms and 180 ms simulated latency. Then we increase the repeats from 3 and 5 to generate a total of $3 \times 3 = 9$ models. Among them we extract the performance frontier by only keeping models that are not dominated in both latency and mAP by any other model.

All Search Experiments: A detailed comparison of all the search spaces in the ablation studies are listed in Table. 1. Their performance frontiers are shown in Fig. 7.

4.2. Ablation on Latency-aware Search

Our work is the first to introduce latency-aware training in architecture search for object detection. To investigate the gain of the latency signal, we compare MnasFPN with two baselines: NAS-FPNLite and NAS-FPNLite-S. NAS-FPN applies depthwise convolution to an architecture not search with latency signals, whereas NAS-FPNLite-S searches using latency signals and with depthwise convolution built into the search space.

According to Fig. 6, MnasFPN shows a superior latency-accuracy tradeoff than NAS-FPNLite. At 187 ms, MnasFPN achieves 24.9 mAP that is unmatched even by the NAS-FPNLite model at 205 ms. While the latency differential constitutes a mere 9% in terms of end-to-end latency, it amounts to around 22% improvement considering the latency portion excluding the backbone.

[1] https://www.tensorflow.org/lite/performance/benchmarks

Search spaces	Kernel sizes	Filter sizes C	Expansion sizes F	SDO	Maximum in-degree	Cardinality
NAS-FPNLite	3	64	-	N	2	2×10^{22}
No-Expand	$\{3,5,7\}$	$\{16,32,48,64,80,96\}$	-	Y	2	2.4×10^{27}
MnasFPN	$\{3,5,7\}$	$\{16,32,48,64,80,96\}$	$\{16,32,64,96,128,256,512\}$	Y	2	10^{31}
Conn-Search	$\{3,5,7\}$	$\{16,32,48,64,80,96\}$	$\{16,32,64,96,128,256,512\}$	Y	4	3×10^{42}

Table 1. Search space comparisons. The common search parameters such as merge operations, feature resolutions and connectivity are omitted. See Appendix for detailed search space cardinality calculations.

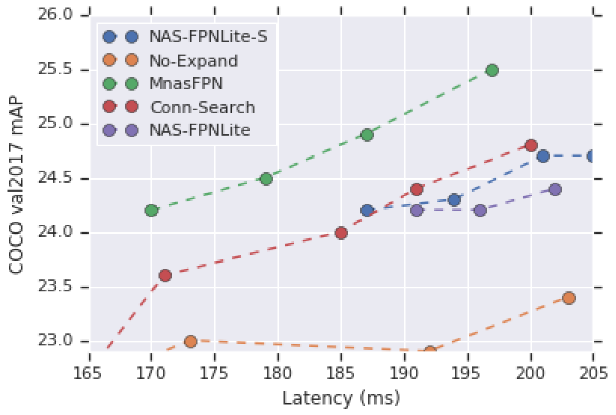

Figure 6. Performance comparisons between MnasFPN and various ablation designs. Latency is measured on Pixel 1 and mAP is computed on COCO *val2017*.

NAS-FPNLite-S also performs better than NAS-FPNLite, but only by a moderate amount. This indicates that the MNASNet-styled latency-aware search is an effective strategy overall, but the primary factor of MnasFPN's success is instead the search space design.

4.3. Ablation on IRB

Our primary contribution is to re-introduce IRB into the detection head. To evaluate the effectiveness of this design, we compare in Fig. 6 MnasFPN with NAS-FPNLite-S and another baseline defined below:

No-Exand: We remove and only remove expansion from the MnasFPN search space. Specifically, we forces $F = C$ for all merged features.

Comparing MnasFPN with NAS-FPNLite-S, where both are products of latency-aware search, we see that the overall design of the MnasFPN search space contributes to almost all the improvements over NAS-FPNLite.

We also see that No-Expand introduces a large bias that it does not even match the performance of NAS-FPNLite. A closer inspection of the learned architectures shows that the model reduces the channel size C to 16 while increasing the number of intermediate nodes. This is a sub-optimal design strategy, on which the RL controllers got stuck repeatedly. As a result, the entire performance frontier (during search) seems sub-optimal compared to those of other searches (Fig. 7).

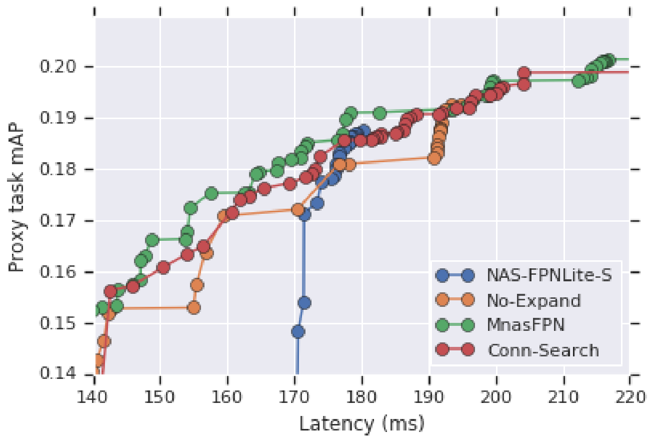

Figure 7. **Proxy** task performance vs. **simulated** latency frontiers of various search spaces. This figure represents the NAS controller's view on the problem, where latency is simulated using LUT and quality is computed on the proxy task, which correlates with but is not directly comparable to mAPs of the real task.

4.4. Connectivity Search

To assess whether the MnasFPN search space was sufficiently large, we compare with Conn-Search (Sec. 3.5) where each merging operation can take a maximum of $D = 4$ inputs.

As shown in Fig. 6, despite having a larger search space that subsumes MnasFPN, Conn-Search has a suboptimal latency-accuracy tradeoff. In Fig. 7 we see that its performance frontier on the proxy task is slightly worse than that of MnasFPN, suggesting that the controller is unable to sufficiently explore the search space. Table. 1 shows that the cardinality of Conn-Search is roughly 10^{42}, greatly surpasses the cardinalities of the two known successful applications of the MnasNet framework: MnasNet (10^{13}) and NAS-FPN (10^{22}).

This result reiterates the significance of the co-adaptation of search spaces and search algorithms. While it is tempting to believe that NAS eliminates the need for manual tuning, and that one only needs to innovate a sufficiently powerful search space that subsumes all search spaces, the reality is that the search algorithm is not yet powerful enough to address arbitrarily large search spaces. Therefore, iterative shrinking and co-adaptation of search spaces, as practiced in the original NAS paper [33], are still relevant.

Model	Repeats	mAP	Latency (ms)	MAdds (B)	Params (M)
MnasFPN	4	24.9	187	0.90	2.5
	5	25.5	196	0.94	2.6
No SDO	4	24.9	195	0.94	2.5
	5	25.5	207	1.0	2.6

Table 2. Ablation study of SDO. SDO does not affects parameters that much but reduces both MAdds and latency.

4.5. Ablation on SDO

To understand the impact of SDO, we disable SDO of the MnasFPN architectures with 4 and 5 repeats, respectively. Models with no SDO will perform 1×1 convolution before resizing, which will be less economical for down-sampling operations, and the discovered MnasFPN architecture as it is dominated by down-sampling operations (Fig. 2).

Unsurprisingly, we see from Table. 2 that disabling SDO does not affect the mAP, but would lead to a 8 to 11ms ($4-6\%$) latency regression. Similarly if we consider the portion of the network without the backbone, this amounts to 12% to 14% of the latency that is "optimizable". Given this strict dominance we conclude that the effectiveness of SDO is sufficiently evident and do not conduct search experiments without SDO for the ablation study.

4.6. Performance Comparison on COCO Test-dev

We compare MnasFPN under different backbones and with other SOTA on-device detection heads. We re-implement MnasFPN in the Tensorflow Object Detection API [12], so the actual model instantiation differs slightly from the one used in our internal comparisons on COCO *val2017*.

Model	**Test-dev mAP**	**Latency**	**MAdds**	**Params**
MnasNet-A1 + SSDLite	23.0 [24]	174*	0.8B	4.9M
MobileNetV2 + SSDLite	22.1 [23]	163*	0.8B	4.3M
MobileNetV2 + NAS-FPNLite	25.1 [7]	202*	0.98B	2.02M
MobileNetV3 + SSDLite	22.0 [9]	137*	0.62B	4.97M
MobileNetV3† + SSDLite	22.0 [9]	119*	0.51B	3.22M
MNASNet-B1 + MnasFPN	24.6	184	0.89B	2.74M
MobileNetV2 + MnasFPN	26.1	183	0.92B	2.50M
MobileNetV2 + MnasFPN ‡	23.8	121	0.53B	1.29M
MobileNetV3 + MnasFPN	25.5	168	0.77B	3.46M

Table 3. MnasFPN variations compared with other mobile detection models on COCO *test-dev*. Latency numbers with '*' are re-measured in the same configuration (same benchmarker binary and same device) as MnasFPN models to ensure fairness of comparison. Models with † employs the channel-halving trick [9]. Models with ‡ was obtained with a depth multiplier of 0.7 on both head and backbone

As shown in Table. 3, with the same MobileNetV2 backbone, MnasFPN achieves **1.0 mAP improvement over NAS-FPNLite**. Furthermore, MnasFPN is 10% faster in end-to-end latency, or **25% faster** in terms of latency incurred outside the backbone.

Since SSDLite is generally much faster than MnasFPN , we compare the two either by applying width-multipler or changing the backbone. With a 0.7 width-multiplier on both head and backbone, MnasFPN with MobileNetV2 achieves **1.8 higher mAP compared with SSDLite** with MobileNetV3 at around 120 ms. Here the MobileNetV3 results use the channel-halving trick, which tends to reduce latency with no mAP degradation, while our results do not. Removing this trick for both shows a further 20 ms latency advantage for MnasFPN.

When paired with MobileNetV3 backbone, MnasFPN is 3.4 mAP higher than SSDLite with MobileNetV2 at around 165 ms. It is both faster and 2.5 mAP higher than SSDLite with MnasNet-A1 backbone.

Therefore, we conclude that MnasFPN compares favorably to both SSDLite and NAS-FPNLite head in its ability to trade off latency with accuracy.

5. Conclusion

Object detection should be treated as a first-class citizen in NAS. In addition, the search process, and more importantly, the search space should both be designed to incorporate knowledge about the targeted platform. Our work searches architectures directly for object detection, and the search is guided by simulated signals of on-device latency. We have also proposed the MnasFPN search space with two innovations. First, MnasFPN incorporates inverted residual blocks into the detection head, which is proven to be favored on mobile CPUs. Second, MnasFPN restructured the reshaping and convolution operations in the head to facilitate efficient merging of information across scales.

Through detailed ablation studies, we've discovered that both innovations in the search space are necessary for the performance boost. On the other hand, further expanding the search space in feature map connectivity seems to overwhelm the NAS framework. As a result, we conclude that the proposed MnasFPN search space is at a local optimum *for this controller*. Note that the point of optimality could change as the controller becomes more powerful, in which case the MnasFPN with connectivity search could become viable again.

On COCO *test-dev* MnasFPN leads to a 25% improvement in non-backbone latency over NAS-FPNLite. The improvements are encouraging but at the same time also suggests that performance bottleneck is somewhere else. The backbone currently occupies over 60% of the total latency, which over-shadows the improvement in the head. Therefore, a promising direction is to search for the head and the body jointly. While the cardinality of MnasFPN is challenging for our current controller, recent one-shot NAS methods are opening avenues for more ambitious search spaces, of which MnasFPN could be an ideal component.

References

[1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018. 2

[2] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 2

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1, 2

[4] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Chunhong Pan, and Jian Sun. Detnas: Neural architecture search on object detection. *arXiv preprint arXiv:1903.10979*, 2019. 1, 2

[5] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 2

[6] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11398–11407, 2019. 1

[7] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019. 1, 2, 5, 8

[8] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 2

[9] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019. 1, 2, 8

[10] 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. 2

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 3

[12] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017. 8

[13] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019. 3

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 2

[15] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019. 2

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2

[17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2

[18] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 6

[19] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 2

[20] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. Thundernet: Towards real-time generic object detection. *arXiv preprint arXiv:1903.11752*, 2019. 2

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 2

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1, 2, 6, 8

[24] 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 *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1, 2, 4, 6, 8

[25] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 1

[26] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, and Chunhua Shen. Nas-fcos: Fast neural architecture search for object detection. *arXiv preprint arXiv:1906.04423*, 2019. 1, 2

[27] Robert J Wang, Xiang Li, and Charles X Ling. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*, pages 1963–1972, 2018. 2

[28] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 1, 2

[29] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017. 2

[30] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019. 5

[31] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 2, 4

[32] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 2

[33] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 2, 7

A. Appendix

A.1. Search space cardinality comparison

NAS-FPNLite-S: There are 9 nodes in total, where the i-th node has two choices for the combine operation, and $choose(i+4,2)$ choices for picking a pair of inputs. The first 5 are internal nodes, and each have 4 resolutions choices. The last 4 are output nodes, whose orders are permuted with $permute(4)$ possibilities. This gives a total search space size of:

$$2^9 4^5 permute(4) \prod_{i=0}^{8} choose(i+4,2) \approx 2 \times 10^{22}$$

No-Expand: In addition to the NAS-FPNLite-S search space, No-Expand additionally grants 3 kernel sizes for each node. It also have 6 choices for the globally-shared channel size C, giving a total search space size of:

$$2 \times 10^{22} \times 3^9 \times 6 \approx 2.4 \times 10^{27}$$

MnasFPN : In addition to the No-Expand search space, MnasFPN additionally searches for channel sizes for the merged features for all 9 nodes, each with 7 choices. The total search space size is:

$$2 \times 10^{27} \times 7^9 \approx 10^{31}$$

Conn-Search: Finally, connectivity search allows for $choose(i+4,4)$ choices for each node, which is $(i+2)(i+1)\times$ more possibilities than that in MnasFPN. It does not search for combine operations, so each node has $2\times$ fewer choices. Therefore the total search space size is:

$$10^{31} \prod_{i=0}^{8} (i+2)(i+1)/2^9 \approx 3 \times 10^{42}$$