Searching the Deployable Convolution Neural Networks for GPUs

Linnan Wang, Chenhan Yu, Satish Salian, Slawomir Kierat, Szymon Migacz
Alex Fit Florea
NVIDIA
{linnanw, chenhany, ssalian, skierat, smigacz, afitflorea}@nvidia.com

Abstract

Customizing Convolution Neural Networks (CNN) for production use has been a challenging task for DL practitioners. This paper intends to expedite the model customization with a model hub that contains the optimized models tiered by their inference latency using Neural Architecture Search (NAS). To achieve this goal, we build a distributed NAS system to search on a novel search space that consists of prominent factors to impact latency and accuracy. Since we target GPU, we name the NAS optimized models as GPUNet, which establishes a new SOTA Pareto frontier in inference latency and accuracy. Within $1ms$, GPUNet is 2x faster than EfficientNet-X and FBNetV3 with even better accuracy. We also validate GPUNet on detection tasks, and GPUNet consistently outperforms EfficientNet-X and FB-NetV3 on COCO detection tasks in both latency and accuracy. All of these data validate that our NAS system is effective and generic to handle different design tasks. With this NAS system, we expand GPUNet to cover a wide range of latency targets such that DL practitioners can deploy our models directly in different scenarios.

1. Introduction

The progress of neural networks has decoupled from the actual deployment for a long time. Deep Learning (DL) researchers have been dedicated to inventing new building blocks, while DL engineers deploy these building blocks in real-world tasks, painstakingly recombine them to find architectures that meet the design requirements. Most of the time, we can simplify these requirements to find the best-performing architecture on the target device (e.g., GPUs) within a specific latency budget. Though there are many exciting advancements in the neural network designs, e.g., the residual connection [13], Inverted Residual Block (IRB) [28] and the attention [32], deploying these network designs remains challenging and laborious; and this is the problem to be addressed in this paper.

Our solution to alleviate the gap between the DL research
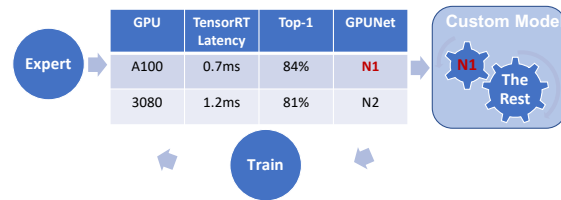
(A) Accelerate the model customization using GPUNet.
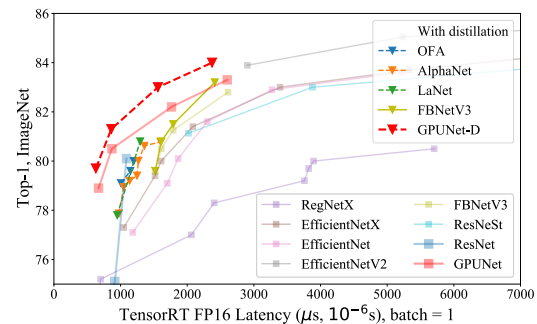
(B) The performance of GPUNet w.r.t SOTA networks in the inference latency and the top 1 accuracy on ImageNet.

Figure 1. GPUNet establishes the new SOTA Pareto frontier in the accuracy and inference latency.

and the actual deployment is to propose a set of optimized Convolution Neural Networks for each type of GPUs tiered by their optimized inference latency (e.g., post-processed by TensorRT [3] or OpenVINO [2]). Specifically, we deliver a table of models, an entry of which is the result of model optimization from maximizing the accuracy subject to the limit of inference latency on a GPU. This table enables DL engineers to directly query the optimized neural architecture w.r.t the design requirements to expedite the customization process on expensive models.

We resort to Neural Architecture Search (NAS) to design models in this table. Recently NAS has shown promising results to automate the design of network architectures in many tasks [19, 36, 37]. Therefore, NAS can be a handy tool since we need to design many models for many latency limits for different GPUs. When models are ready to de-

ploy, we measure post-processed TensorRT engine latency, i.e., including quantization, layer/tensor fusion, kernel tuning, and other system side model optimizations. Finally, we design our model toward NVIDIA enterprise GPU products for their broad adoption by the community today.

We built a novel distributed NAS system to achieve our goal. Following the prior works, our NAS consists of 3 modules, a search space, an evaluation module, and a search method. The search space provisions networks following the predefined patterns; the search method proposes the most promising network based on the priors. The evaluation module returns the performance of the proposed network either by training or estimation from a supernet [45]. Our search space constructs a network by stacking convolution layers, IRBs, and Fused-IRBs used in EfficientNet [31]. However, our search space is the most comprehensive by far that includes filter numbers (#filters), kernel sizes, the number of layers (#layers) or IRBs (#IRBs) in a stage, and the input resolution. Within an IRB or Fused-IRB, we also search for the expansion ratio, the activation type, with or without the Squeeze-Excitation (SE) layer. All of these factors are identified as prominent factors to affect latency and accuracy. Therefore, this search space enables us to better leverage the accuracy and latency than prior works, e.g., the fixed filter pattern in NASNet [46] and the fixed activation and SE pattern in FBNetV3 [10]; and the search also enables us to find a better policy than the fixed scaling strategy in EfficientNet [18, 30]. To support such a complex search space, we choose to evaluate a network candidate by training. Although this approach is far more expensive than the supernet approach, the evaluation is more accurate in ranking the architectures [42, 45]. And we can avoid many unresolved problems in building a supernet for our search space, e.g., supporting multiple types of activation, activating/deactivating SE, and variable filter sizes. We built a client-server-style distributed system to tackle the computation challenges, and it has robustly scaled to 300 A100 GPUs (40 DGX-A100 nodes) in our experiments. Finally, we adopt the LA-MCTS guided Bayesian Optimization (BO) [35] as the search method for its superior sample efficiency demonstrated in the recent black-box optimization challenges [1].

We name the NAS optimized CNNs as GPUNet, and GPUNet has established a new SOTA Pareto front in the latency and accuracy in Fig. 1. We measure the latency of GPUNet using TensorRT, so GPUNet is directly reusable to DL practitioners. Particularly, GPUNet-1 is nearly 2x faster and 0.5% better in accuracy than FBNetV3-B and EfficientNet-X-B2-GPU, respectively. We also validate GPUNet on COCO detection tasks, and GPUNet still consistently outperforms EfficientNet-X and FBNetV3. All of these data validate that our NAS system is effective and generic in designing various tasks. Although this paper only

shows a few GPUNet for comparisons, the complete model hub tiered by the inference latency is still ongoing, and we will release them with the paper.

2. Related Works

Today running DL models locally on an edge device or hosting the model as a service on the enterprise-level GPUs in data centers are two major ways for the model deployment. Edge devices, such as your smartphones or the embedded system for the self-driving, are often equipped with a small CPU, limited RAM, and slow inter-connect. Given the constrained resource, a model's #FLOPS or MACs can correlate well with the latency on such devices, driving many works to propose low FLOPS operators for the faster deployment on the edge devices [15, 26, 28], e.g., depthwise separable convolutions. Accelerating the inference on mobile devices has been a hot research topic in recent years [8, 14, 38].

Deploying models on the GPU requires different optimizations from the edge devices. Since GPU has a far more powerful architecture than edge devices, we need to consider the device saturation, parallelism and compute/memory-bound operators, etc., for the deployment [18]. Although GPU has dominated the MLPerf inference benchmark for years, only a few works optimize the CNN deployment on GPUs. One line of these works is to propose fast operators. RegNet [25] accelerates the inference by optimizing the search space to select simple, GPU-friendly operators. ResNetXt proposes the split attention operator to improve the accuracy and inference latency [44]. Another line of work is to optimize the structure. TResNet [27] optimizes operators in ResNet-50, including SE layers and BatchNorm, to improve the inference on GPU. At the same time, EfficientNet-X [18] proposes a latency-aware scaling method for designing the fast EfficientNet to the GPU/TPU and uses a roofline model to explain the gap between the FLOPS and latency on GPUs. Rather than using a fixed scaling policy, our work treats the model optimization as a black box, searching for the fast architectures for GPUs using the TensorRT optimized inference latency. Therefore, we can better trade-off the latency and accuracy than EfficientNet-X, and our final networks are directly deployable on GPUs.

We use NAS to build proposed networks, and here we review the recent advances in NAS. Early works in NAS, e.g., NASNet [46], models CNN as a Direct Acycle Graph (DAG). While EfficientNet quickly gains popularity for its good performance on ImageNet [39]. This paper reuses the building blocks from EfficientNetV2 to find fast architectures. Recently transformer [21] and Multi-Layer Perception (MLP) [20] starts to emerge as promising alternatives to ConvNet; we leave the NAS on these search spaces as future work. Our paper evaluates each network independently

by training end-to-end despite the popularity of the super-net approach. ENAS [24] proposed the supernet, which is an over-parameterized network to approximate the performance of sub-networks. Although the supernet significantly reduces the computation requirement for NAS, the rank predicted by supernet can be inaccurate [45]. Besides, training the supernet is non-trivial [41] and the construction of supernet to support variable activation, expansion ratio, and filter sizes, etc., still remains an open problem. Therefore, we perform NAS using a distributed system to avoid unresolved issues of supernet.

3. Methodology

We built a novel distributed NAS framework to automate the model design for GPUs. Our NAS system consists of a search space, a search algorithm, and an evaluation method following the existing NAS framework. First, the search algorithm selects networks from the search space, querying its performance using the evaluation method. Then the search algorithm refines its decision in the next iteration by leveraging all the evaluated network accuracy pairs.

Our NAS consists of two stages, 1) categorizing networks by the inference latency and 2) performing NAS on networks within a latency group to optimize the accuracy. In the first stage (Fig. 2.A), we use Sobol sampling [29] to draw network candidates from the high-dimensional search space evenly, approximate the network latency by using the latency look-up table, then categorize the network into a sub-search space, e.g., networks < 0.5ms. We approximate the inference latency by summing up the latency of each layer from a latency lookup table. The latency table uses the input data shape and layer configurations as the key to the latency of a layer. In the second stage (Fig. 2.B), Bayesian optimization consumes a sub-space to find the best performing network within the latency range of the sub-space. We built a client-server distributed framework to perform NAS. The search algorithm runs on the server, proposing network for a client. The client will return the accuracy and network after training. The following elaborates each components and its design justifications.

3.1. Search Space

The search space prescribes the general structure of network candidates, and our search space is inspired by EfficientNet [31]. Please note our search framework is generic to support various search spaces, e.g., designing visual transformer or MLP for visual tasks. Although transformer has shown excellent performance recently [7, 21], here we focus on ConvNet due to the better support from current TensorRT that performs critical performance optimizations for the fast inference on GPUs. We leave the NAS on the transformer- or MLP-based vision models as future work.

Stage	Type	Stride	Kernel	#Layers	Act	E‡	Filters	SE
0	Conv	2	[3, 5]	1	[R,S]†		[24, 32, 8]*	
1	Conv	1	[3, 5]	[1, 4]	[R,S]		[24, 32, 8]	
2	F-IRB◇	2	[3, 5]	[1, 8]	[R,S]	[2, 6]	[32, 80, 16]	[0, 1]
3	F-IRB	2	[3, 5]	[1, 8]	[R,S]	[2, 6]	[48, 112, 16]	[0, 1]
4	IRB	2	[3, 5]	[1, 10]	[R,S]	[2, 6]	[96, 192, 16]	[0, 1]
5	IRB	1	[3, 5]	[0, 15]	[R,S]	[2, 6]	[112, 224, 16]	[0, 1]
6	IRB	2	[3, 5]	[1, 15]	[R,S]	[2, 6]	[128, 416, 32]	[0, 1]
7	IRB	1	[3, 5]	[0, 15]	[R,S]	[2, 6]	[256, 832, 64]	[0, 1]
8	Conv1x1 & Pooling & FC						1792	
Res	[224, 256, 288, 320, 352, 384, 416, 448, 480, 512]							

†: R is ReLU and S is Swish.
◇: F-IRB indicates Fused-Inverse-Residual-Block (Fused-IRB).
‡: E indicates the range of IRB expansion rate.
∗: number of filters increase from 24 to 32 at the step of 8.

Table 1. The proposed convnet search space.

Table 1 demonstrates the details of our search space used in this paper. Our search space consists of 8 stages. Here we search for the configurations of each stage, and the layers within a stage share the same configurations. The first two stages are to search for the head configurations using convolutions. Inspired by EfficientNet-V2 [31], the 2 and 3 stages uses Fused-IRB [31]. But we observed the increasing latency after replacing the rest IRB with Fused-IRB. From the stage 4 to 7, we use IRB as the basic layers. The column *#Layers* shows the range of #layers in the stage, e.g., [3, 10] at stage 4 means that the stage can have 3 to 10 IRBs. And the column *Filters* shows the range of filters for the layers in the stage (see table note for details). Our search space also tunes the expansion ratio, activation types, kernel sizes, and the Squeeze Excitation(SE) [16] layer inside the IRB/Fused-IRB. Finally, the dimensions of the input image increase from 224 to 512 at the step of 32.

3.1.1 Justifications of the Search Space

Unlike prior works, our search is guided by the accuracy and the TensorRT optimized inference latency. In a good experiment design, we should identify the most relevant factors [17] to the design targets, i.e., fast and accurate networks. Table. 1 demonstrates the several prominent factors found by us to impact the latency and accuracy. Here we provide the empirical data to support our decisions.

- *#Layers* and *Filters*: extensive evidence from published results demonstrates that a deep or wide network can perform better than the shallow or narrow variants [13, 22] while adding layers or increasing filters slows down the inference. *#Layers* and *Filters* are essential design choices to the accuracy and latency.

- *Activation*: many past works have demonstrated that a good activation design can notably improve the fi-

(A) NAS workflow (B) Curating the search space by the latency (C) Distributed NAS
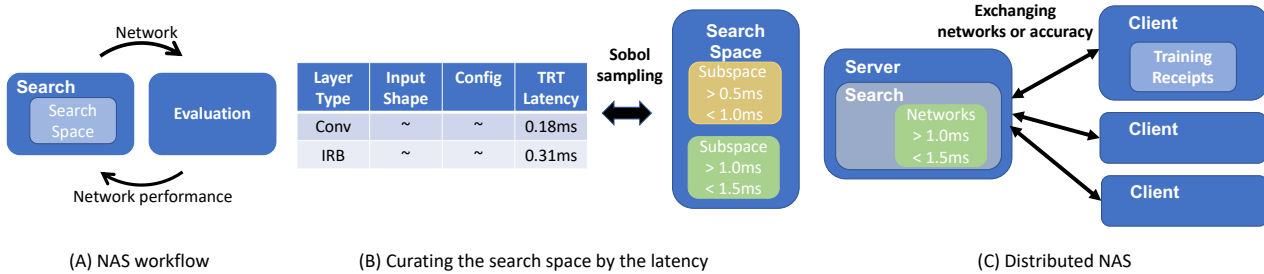
Figure 2. The work flow of proposed NAS framework. A proposed search space is first pruned by TensorRT inference latency in (B). Then we use a black box optimizer to iteratively explore the search space in (A). We implement a distributed search framework to exploit the parallelism in (C).
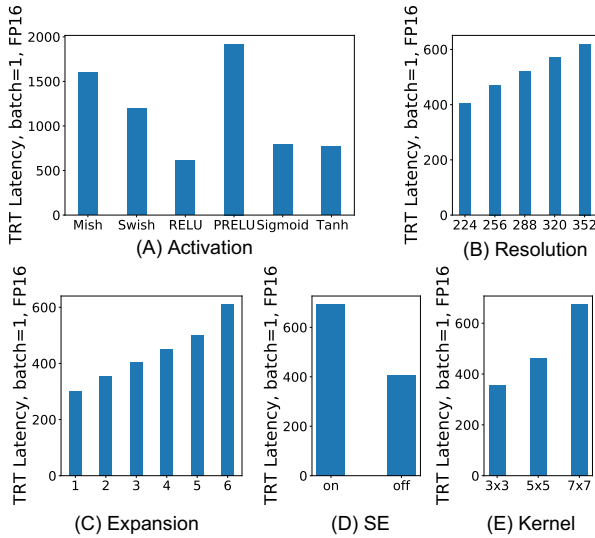

Figure 3. The latency impact of different hyper-parameters and configurations on EfficentNet-B0.

Stage	Type	Hyper-parameters	Length
	Resolution	[*Resolution*]	1
0	Conv	[*#Filters*]	1
1	Conv	[*Kernel, Activation, #Layers*]	3
2	Fused-IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
3	Fused-IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
4	IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
5	IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
6	IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
7	IRB	[*#Filters, Kernel, E, SE, Act, #Layers*]	6
	total		41

Table 2. The encoding scheme of networks in the search space.

search.

- *Kernel*: a large convolution kernel can increase the receipt field to improve the accuracy (more details in [6]). Still, it also increases the latency (Fig. 3.D), which validates the choice of kernel size into the search space.

- *SE*: Squeeze-Excitation [16] was introduced by the winning entry to ILSVRC 2017 that improved 25% over the previous year. After adding SE, Fig. 3 shows the latency significantly increases. This justifies SE to be a factor in the search space.

- *Image Resolution*: EfficientNet [30] clearly demonstrates the accuracy improvement by increasing the resolution, and Fig. 3.B shows the latency also significantly increases. So we search for the input image resolution for better accuracy and latency trade-off.

3.1.2 Network and Search Space Representations

Now we have the general picture of search space; the next is to find the proper representation that embodies the design. We use a vector of integers to encode a network sampled from the search space described in Table. 1. The length of the vector is 41, and Table. 2 elaborates the hyper-parameters represented by each digit. Stage 1 shares the same filter number as stage 0, and we only search the filter size for the first 3x3 convolution (stage 0). Because layers

nal accuracy on ImageNet [12, 26], whereas Fig. 3.A shows a network using ReLU can be 4x faster than using PReLU. In general, the activation is the memory-bound operation. And TensorRT supports ReLU, Sigmoid, and Tanh in the layer fusion, which explains the speed gap. Therefore, the choice of activation is an important factor in the latency and accuracy trade-off.

- *Expansion*: IRB or Fused-IRB internally expands the channel size using a 1x1 convolution, and the expansion ratio controls the size of the internal channel, i.e., expansion ratio x the input channel. The MobileNet [28] paper claims that the larger channel expansion will help improve the capacity of the network and expressiveness. Our empirical results are also consistent with the claim. For example, the accuracy of a network drops 4 points on ImageNet top-1 after reducing the expansion from 6 to 2, whereas increasing the expansion incurs non-negligible costs(Fig 3.C). These data suggest the expansion ratio is an important factor to
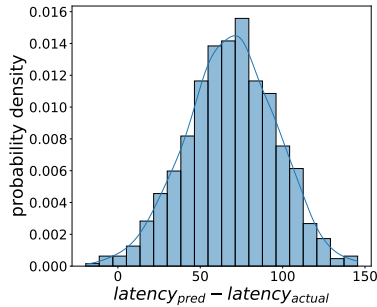
Figure 4. The error between the actual measured TensorRT latency and the table predicted latency.

within a stage share the configurations, we use 6 integers to represent the filter size, kernel size, expansion ratio, using SE or not, the types of activation, and the number of IRBs for the stages from 2 to 7. So a network is an instance of the vector described in Table. 2, and the range of every digit collectively define the search space in Table. 1.

3.2. Stratify Networks by Inference Latency

To design networks tiered by the inference latency, we choose to directly measure the latency of networks in the search space. Because the size of search space is exponentially large, we approximate the search space by sampling millions of networks from it. The sampling techniques is critical to capture the true distribution of search space, and here we use the Sobol sequence [5], the advantages of which is straightforward in Fig. 5. The sampling is a low cost operations that we can get millions of samples within a minute. The challenge is to measure the latency of sampled networks. Since TensorRT has dominated the MLPerf inference benchmark, we want to measure the inference latency optimized by TensorRT. Whereas, TensorRT takes minutes to build the inference engine for the measurement, which makes it infeasible to measure all the sampled networks.

We approximate a network's latency by adding up the latency of each layer. Although the search space renders 10^{30} networks, the layers have limited configurations, e.g., 10^4 in our case. Therefore we can significantly speed up the latency measurement by building a latency table with the input data shape and the layer configurations as the key. Given a network, we iterate over layers to look up the latency. If a layer does not exist in the table, we only benchmark it and record its latency in the table. Finally, the network latency is the sum of the latency of all the layers. Fig. 4 demonstrates that the table estimated latency is close to the network's actual latency, and the table estimation is on average $75\mu s$ higher than the actual end-to-end measurement. Because a whole network subjects more opportunities for the layer fusion to TensorRT than the single layer. Benchmarking $\sim 10^4$ layers is still an expensive task, and we parallelize the curation of the latency table over multi-GPUs to speed up the process from weeks to days.
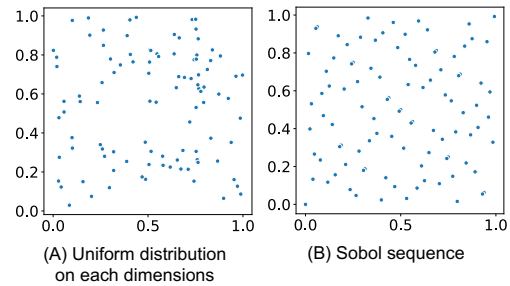

Figure 5. The distribution sampled by (A) the uniform random on each dimensions and (B) sobol sequence.

3.3. Distributed Neural Architecture Search

We treat the network design as a black box. Guided by the reward, e.g. the validation accuracy, the search tunes the hyper-parameters prescribed in the search space to optimize the model performance (Fig.2.A). Here we elaborate the details of search algorithms and the evaluation method.

3.3.1 Search Algorithms

We choose LA-MCTS boosted Bayesian optimization (BO) [35] as the search algorithm, which is one of the top entries to the 2020 NeurIPS black-box optimization competition [1]. Since we evaluate a network by training, the sample efficiency is critical to the overall cost. The competition results show that LA-MCTS boosted BO has demonstrated the leading sample efficiency among other BO variants and evolutionary algorithms; therefore, we adopt it in our experiments, and Fig. 2 depicts the workflow.

Some prior works [23] define the problem as a Multi-Objective Optimization (MOO) to find the optimal Pareto frontier to the latency and accuracy. However, finding the Pareto Optimality is too fine-grained to the practice. For example, two solutions located on the Pareto frontier may have trivial differences in accuracy and latency, but finding these Pareto solutions is very expensive. Therefore, we organize the search space by latency before maximizing the accuracy. This also allows us to build a table of networks tiered by their inference latency.

3.3.2 Evaluation

We choose to evaluate each proposed network by training and apply the early-stopping if the training curve is not promising. For networks from the same search space with similar latency, we use the same training receipt, as our practical experience suggests that tuning the training receipt brings up to 1% accuracy improvement at a tremendous cost. The details of our training receipts can be found at sec **??** in the supplemental material. The training will return the best validation accuracy to the search algorithm after 450 epochs.

Rather than democratizing NAS, this paper intends to maintain a set of NAS-optimized models using hundreds of GPUs for the community. We believe the training approach is necessary, although it is far more expensive than recent supernet approaches [24]. First, extensive evidence in [42, 45] demonstrates that supernet can be inaccurate in ranking the network candidates, and training a good supernet is non-trivial [41]. Second, no supernet properly supports variable expansion ratios, image resolutions, and with/without SE. The training approach can circumvent all these problems at additional costs.

3.3.3 Distributed NAS

Now we're ready to put everything together. Fig. 2 demonstrates that we implement a client and server distributed system to run NAS. Following sec 3.2, we start with generating networks in a latency range as the search space by sampling. Then we integrate the pruned search space into the search algorithm to run on the server. The server and clients exchange data via sockets. The clients will request a network from the server to evaluate if they are free and return the network and the best validation accuracy to the server. The search algorithm can leverage this information to propose the next network candidate. To validate the framework, we test the system with a few synthetic functions to ensure the performance metric increases along with the #samples. This framework is also generic to different search problems, and we can also use the same framework to search the architecture for Transformer.

4. Experiments

This section demonstrates the details of using the proposed search space and NAS system in designing GPUNet. Compared to existing works, GPUNet significantly improves the SOTA Pareto frontier in both the accuracy and inference latency (Fig. 1). With a similar latency of 1.8ms, the accuracy of GPUNet is 1% better than the corresponding FBNet-V3 on ImageNet. With a similar 80.5 accuracy, GPUNet is 1.6x faster than FBNet. We start with describing the experiment setup, then discuss the main results. Finally, we show that GPUNet also effectively improves the downstream tasks.

4.1. Experiment Setup

Software Setup: we perform NAS directly on ImageNet [11] that contains 1.28 million training and 50000 validation images in 1000 classes. Each network candidate is pre-trained with 300 epochs for the performance ranking with automatic mixed precision (AMP), then we fine-tune the top network for another 150 epochs. We use a modified training script from Pytorch Image Models [4]
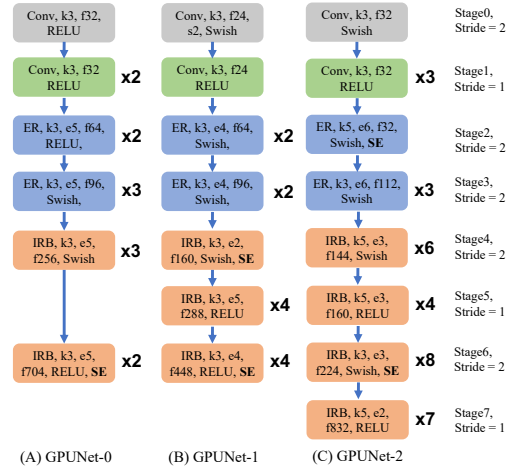

Figure 6. The architecture of searched GPUNet in Table. 3. For stages 2, 3, 4 and 6, only the stride of first layer is 2 and the stride of rest layers is 1.

to train models. The training only uses random augmentation at the magnitude of 9 and a standard deviation of 0.5. The learning rate decays by .97 for every 2.4 epochs. Exponential Moving Average (EMA) is also in use with a decay rate of 0.9999. The crop percentage is set to 1, and the optimizer is RMSprop. We set NAS to focus on models with FP32+FP16 TensorRT GPU compute time[1] $< 2ms$, which is more relevant in practice. For latency measurement, we use TensorRT-8.0.1. We export the onnx model and measure FP16 GPU compute latency using the `trtexec --fp16` command-line on a standalone PCI-E NVIDIA GV100 GPU.

Machine Setup: we perform the training on DGX A100 with 8x A100 80 GB. Our system is flexible to allow training on preemptible (spot) instances. We launch the server at a dedicated node to propose networks and launch clients at preemptible instances. Each client checkpoints per epoch during the training in case of preemption. The server also consistently checkpoints its state for fault tolerance.

4.2. Main Results

4.2.1 Preparation of Baselines

Table. 3 lists the recent SOTA baselines used in comparisons; There are two set of baselines that include and exclude distillation, respectively. The accuracy and the inference latency are two key metrics in our evaluations. Comparing the accuracy is easy to be fair but not on the inference latency, since the latency can be impacted by the software stack (e.g. runtime efficiency and system optimizations), GPUs, batch size and etc.. To ensure fairness, we transform

---

[1]TensorRT also reports throughput, end-to-end, device-to-host, host-to-device time. We use GPU compute time to better capture the latency impact on architecture difference.

Models	Top1 ImageNet	TensorRT Latency[†] FP16 GV100 (ms)	#Params (Million)	#FLOPS (Billion)	GPUNet Speedup ↑	GPUNet Accuracy ↑
RegNet-X [25]	77.0	2.06	9.19.	1.6	3.3x	1.9
EfficientNet-B0 [30]	77.1	1.18	5.28	0.38	1.9x	1.8
EfficientNetX-B0-GPU [18]	77.3	1.05	7.6	0.91	1.69x	1.6
FBNetV2-L1 [33]	77.2	1.13	x	0.32	1.82x	1.7
GPUNet-0	**78.9**	**0.62**	11.9	3.25		
RegNet-X	80.0	3.9.	54.3.	15.	2.72x	0.5
EfficientNet-B2	80.3	1.86	9.2	1	2.26x	0.2
EfficientNetX-B2-GPU	80.0	1.61	10	2.3	1.96x	0.5
FBNetV3-B [10]	79.8	1.55	8.6.	0.46	1.89x	0.7
ResNet-50 [13]	80.3	1.1	28.09	4.	1.34x	0.2
GPUNet-1	**80.5**	**0.82**	12.7	3.3		
RegNet-X	80.5	5.7	107.	31.7	3.24x	1.7
EfficientNet-B3	81.6	2.3	12	1.8	1.3x	0.6
EfficientNetX-B3-GPU	81.4	2.1	13.3	4.3	1.2x	0.8
ResNeSt-50 [44]	81.1	2.3	27.5	5.4	1.27x	1.1
FBNetV3-F	82.1	2.26	13.9	1.18	1.28x	0.1
GPUNet-2	**82.2**	**1.76**	25.8	8.38		

Models	Top1 ImageNet	TensorRT Latency FP16 GV100 (ms)	#Params (Million)	#FLOPS (Billion)	GPUNet Speedup ↑	GPUNet Accuracy ↑
AlphaNet-a2 [34]	79.2	1.14	11.3	0.32	1.8x	0.5
FBNetV3-A	79.6	1.52	8.6	0.35	2.4x	0.1
BigNAS-L [40]	79.5	1.55	6.4	0.58	2.46	0.2
LaNet-200M	77.8	0.95	5.76	0.24	1.5x	1.9
GPUNet-D0	**79.7**	**0.63**	6.2	0.72		
AlphaNet-a6	80.8	1.6	15.4	0.71	1.28x	1.7
FBNetV3-D	81.1	1.7	10.3	0.64	1.36x	1.4
BigNAS-XL	80.9	1.64	9.5	1.04	1.31x	1.6
LaNet-600M	80.8	1.29	8.67	0.24	1.03x	1.7
GPUNet-D1	**82.5**	**1.25**	10.6	3.66		
FBNetV3-G	83.2	2.43	10.8	2.1	1.0x	0.4
GPUNet-D3	**83.6**	**2.4**	19	15.6	(Only random augmentation)	

[†]: We measure the latency (`FP16` GPU compute time) using an explicit-shape at batch size 1.

Table 3. Comparisons of GPUNet to SOTA results. Fig. 1 visualizes the table and shows that GPUNet-D dominates the baseline models in both the accuracy and inference latency.

all the baseline models into ONNX models for benchmarking with TensorRT. The inference benchmark exclusively runs on an NVIDIA GV100; and the workspace of TensorRT is fixed at 10G across all runs. We benchmark the latency at the batch size = 1 with a explicit-shape, and report the average latency from 1000 runs. We take baseline models from either their original implementations or from PyTorch Image Models [4]. Appendix Sec. 6.1.1 provides the details.

4.2.2 Results on ImageNet1K

Table. 3 compares the performance of searched GPUNet to baselines, and Fig. 1 visualizes the Pareto frontier of different models in the inference latency and accuracy. Please note we prepare two sets of different networks GPUNet

and GPUNet-D, for the cases of with/without distillation. Fig. 1.B clearly shows GPUNet-D dominates other models in both target objectives, and Table. 3 suggests both GPUNet and GPUNet-D are significantly faster than SOTA networks while maintaining similar accuracy. At the similar 80.5 top-1 accuracy, GPUNet-1 is nearly 2 times faster than EfficientNet-B2, EfficientNetX-B2-GPU, and FBNetV3-B. For other accuracy groups, GPUNet consistently demonstrates the speedup from $1.27\times$ to $3.24\times$ than baselines.

While EfficientNet-V2 [31] shows slightly better results than GPUNet when latency $> 3ms$ in Fig. 1.B, EfficientNet-V2 utilizes a far more sophisticated training scheme that includes Mixup [43] and progressive training in regularizing the network for better accuracy. These regularizations are orthogonal to NAS, and they can be an excellent future work to improve the accuracy of GPUNet further.

Interestingly, we also note that the `#FLOPS` and #Parameters of GPUNet are larger than baselines, though GPUNet is significantly faster. These results indicate that low `FLOPS` models are not necessarily fast on GPUs. EfficientNet-X explains this with the roofline model [18], and we will provide more results in Sec. 4.2.4.

Please note that our ultimate goal is to provide a table of models tiered by their inference latency to expedite the customization. Table. 3 and Fig. 1 only show a few models to demonstrate that our NAS system can effectively design fast and accurate networks on the proposed search space in Table. 1. We will release this table of models after the paper.

4.2.3 GPUNet Architecture

Fig. 6 shows that the architectures of NAS optimized GPUNet are too irregular to be human design. For example, the two adjacent stride=2 ER (Fused-IRB) blocks in GPUNet-2 consecutively halve H and W twice, while the human-designed networks usually have multiple stride = 1 layer between two stride = 2 layers. There is no obvious pattern for the activation functions and expansions in IRB as well. However, these NAS optimized networks show one common characteristic in the filter distribution, which are skinny in the beginning/middle stages and very wide in the last few stages though the search space in Table. 1 permits large filters at the beginning and small filters in the end. For example, the filters of GPUNet-2 follow the pattern of $32 \rightarrow 32 \rightarrow 116 \rightarrow 144 \rightarrow 160 \rightarrow 224 \rightarrow 832$; GPUNet-0 and GPUNet-1 also follow a similar filter pattern.

4.2.4 Why GPUNet Are Faster and Better?

We also compare the architecture of FBNet and EfficientNet to GPUNet. Here are a few key differences found by us that explain the GPUNet performance. Let's use GPUNet-1 as an example.

- *Mixed types of activation*: Fig.6 suggests that GPUNet switches between ReLU and Swish, but EfficientNet and FBNet use Swish across all the layers. Fig.3 suggests Swish greatly increases the latency. Some layers of GPUNet uses ReLU to reduce the latency for other opportunities to improve the accuracy, e.g., larger filters.

- *Fewer expansions in IRB*: Fig.3.C shows the network latency almost doubles by increasing the expansions in all IRB from 1 to 6. The expansion is part of our search space, so some GPUNet layers tend to have small expansions to save the latency.

- *Wider and Deeper*: the filters (wide) and the number of layers (deep) in a stage are part of our search space. Because of the latency saving from mixed activation

Backbone	ImageNet top1	Method	TRT Latency(ms)	mAP
GPUNet-2	82.2	Cascade RCNN	5.2	40.0
ResNet-50	80.3	Cascade RCNN	5.8	40.4
FBNetV3-F	82.1	Cascade RCNN	7.90	26.5
EfficientNet-B3	81.6	Cascade RCNN	10.65	28.40

Table 4. Applying GPUNet to COCO object detection tasks. The latency was measured using the resolution of 1333x800.

and fewer expansions, GPUNet tends to be wider and deeper than FBNet and EfficientNet. In the same accuracy group, the filters of FBNetV3-B follow the pattern of $16 \rightarrow 24 \rightarrow 40 \rightarrow 72 \rightarrow 120 \rightarrow 183 \rightarrow 224$, and the filter pattern of EfficientNet-B2 is $32 \rightarrow 16 \rightarrow 24 \rightarrow 48 \rightarrow 88 \rightarrow 120 \rightarrow 208 \rightarrow 352$, but GPUNet-1 is a lot wider than FBNetV3-B and EfficientNet-B2 that has a pattern of $24 \rightarrow 64 \rightarrow 96 \rightarrow 160 \rightarrow 288 \rightarrow 448$. Besides, GPUNet-2 has 33 layers, 2 more than FBNetV3-F and 5 more than EfficientNet-B3. It is known that deep and wide networks have better accuracy; therefore, the accuracy of GPUNet is better than baselines within each group.

- *Larger Resolution*: GPUNet-(1 and 2) are 32 and 64 larger than EfficientNet-B2 and B3 in resolutions, 72 and 120 larger than FBNetV3-B and FBNetV3-F, respectively. Using large resolution generally improves the accuracy; therefore, GPUNet shows better accuracy and higher `FLOPS` than baselines.

4.3. Evaluating on Detection tasks

We test GPUNet on COCO detection tasks. We evaluate GPUNet, FBNetV3-F, and EfficientNet-B3 on COCO detection tasks by replacing the backbone in the cascade RCNN [9]. Table. 4 shows GPUNet-2 is not only faster, but also delivers higher mAP than baseline models.

5. Conclusions

Model customization is challenging for DL engineering. In this work, we proposes to build a model hub tiered by their inference latency to expedite model customization, which facilitates practitioners to reuse the our pretrained models with known TensorRT latency and compatibility in mind. With a novel distributed NAS system and an enhanced search space to design a set of fast and accurate GPUNet, we establish a new SOTA Pareto frontier in latency and accuracy, validating the effectiveness of our NAS system. Although this paper primarily focus on EfficientNet search space, our NAS system is generic to support various tasks and search spaces. Ultimately we intend to maintain a hub of NAS optimized models that track the latest technology so that ML practitioners can directly reuse them.

References

[1] Black box optimization challenges. http://bbochallenge.com/leaderboard/. 2, 5

[2] Intel's openvino. https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html. 1

[3] Nvidia's tensorrt. https://developer.nvidia.com/tensorrt. 1

[4] Pytorch image models. https://github.com/rwightman/pytorch-image-models. 6, 7

[5] The sobol sequence implemented in the pytorch. https://pytorch.org/docs/stable/generated/torch.quasirandom.SobolEngine.html. 5

[6] André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. https://distill.pub/2019/computing-receptive-fields. 4

[7] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021. 3

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

[9] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 8

[10] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16276–16285, 2021. 2, 7

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 3, 7

[14] 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. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 2

[15] 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

[16] 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, 4

[17] Oscar Kempthorne. The design and analysis of experiments. 1952. 3

[18] Sheng Li, Mingxing Tan, Ruoming Pang, Andrew Li, Liqun Cheng, Quoc V Le, and Norman P Jouppi. Searching for fast model families on datacenter accelerators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8085–8095, 2021. 2, 7, 8

[19] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Autodeeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019. 1

[20] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021. 2

[21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021. 2, 3

[22] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6232–6240, 2017. 3

[23] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 419–427, 2019. 5

[24] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018. 3, 6

[25] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020. 2, 7

[26] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017. 2, 4

[27] Tal Ridnik, Hussam Lawen, Asaf Noy, Emanuel Ben Baruch, Gilad Sharir, and Itamar Friedman. Tresnet: High performance gpu-dedicated architecture. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1400–1409, 2021. 2

[28] 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, 4

[29] Il'ya Meerovich Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 7(4):784–802, 1967. 3

[30] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 2, 4, 7

[31] Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021. 2, 3, 7

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 1

[33] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020. 7

[34] Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. Alphanet: Improved training of supernet with alpha-divergence. *arXiv preprint arXiv:2102.07954*, 2021. 7

[35] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. *arXiv preprint arXiv:2007.00708*, 2020. 2, 5

[36] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1

[37] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019. 1

[38] 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/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 2

[39] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020. 2

[40] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020. 7

[41] Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. How to train your super-net: An analysis of training heuristics in weight-sharing nas. *arXiv preprint arXiv:2003.04276*, 2020. 3, 6

[42] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neu-ral architecture search. *arXiv preprint arXiv:1902.08142*, 2019. 2, 6

[43] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 7

[44] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020. 2, 7

[45] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR, 2021. 2, 3, 6

[46] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 2