SqueezeNAS: Fast Neural Architecture Search for Faster Semantic Segmentation

Albert Shaw, Daniel Hunter, Forrest Iandola and Sammy Sidhu
DeepScale Inc.
{albert,daniel,forrest,sammy}@deepscale.ai

Abstract

For real time applications utilizing Deep Neural Networks (DNNs), it is critical that the models achieve high-accuracy on the target task and low-latency inference on the target computing platform. While Neural Architecture Search (NAS) has been effectively used to develop low-latency networks for image classification, there has been relatively little effort to use NAS to optimize DNN architectures for other vision tasks. In this work, we present what we believe to be the first proxyless hardware-aware search targeted for dense semantic segmentation. With this approach, we advance the state-of-the-art accuracy for latency-optimized networks on the Cityscapes semantic segmentation dataset. Our latency-optimized small SqueezeNAS network achieves 68.02% validation class mIOU with less than 35 ms inference times on the NVIDIA Xavier. Our latency-optimized large SqueezeNAS network achieves 73.62% class mIOU with less than 100 ms inference times. We demonstrate that significant performance gains are possible by utilizing NAS to find networks optimized for both the specific task and inference hardware. We also present detailed analysis comparing our networks to recent state-of-the-art architectures. The SqueezeNAS models are available for download here: https://github.com/ashaw596/squeezenas

1. Introduction and Motivation

In recent years, Deep Neural Networks (DNNs) have become a dominant approach for solving numerous problems in computer vision. Image classification tasks such as ImageNet [1] and CIFAR10 [2] are the de facto "playground" for designing DNN model architectures. When developing DNNs for a target task other than image classification (e.g. semantic segmentation or object detection), a popular approach is to use *architecture-transfer*: start with an image classification network and append a few task-specific layers to the end of the network. [1]

We believe architecture-transfer has become mainstream because of a number of conventional-wisdom assumptions that have permeated the computer vision community. In the following, we enumerate these assumptions and present evidence for why these assumptions are becoming outdated.

- **Assumption 1: The most accurate neural network for ImageNet image classification will also be the most accurate backbone for the target task.**
  Reality: ImageNet accuracy is only loosely correlated with accuracy on a target task. For example, SqueezeNet is a small neural network that achieves significantly lower ImageNet classification accuracy than VGG [3] [4]. However, SqueezeNet is more accurate than VGG when used for the task of identifying similar patches in a set of images [5]. Thus, the right DNN design varies depending on the target task.
- **Assumption 2: Neural Architecture Search (NAS) is prohibitively expensive.**
  Reality: It is true that some NAS methods based on genetic algorithms (e.g. [6]) or reinforcement learning (e.g. [7]) often require thousands of GPU days to converge on a good DNN design because they train hundreds or thousands of different DNNs before converging. However, recent "supernetwork" approaches such as DARTS [8] and FBNet [9] have turned the problem inside out. They can train one supernetwork that contains millions of DNN designs, but it still converges on an optimal DNN design within 10 GPU days.

So, the "right" DNN design depends on the target task, and modern NAS methods can quickly converge on the right DNN for a task. A similar issue arises when we look at choosing the right DNN for a target computing platform (e.g. a specific version of a CPU, GPU, or TPU):

- **Assumption 3: Fewer multiply-accumulate (MAC) operations will yield lower latency on a target computing platform.**
  Reality: In a recent study, Almeida *et al*. showed that two DNNs with the same number of MACs can have a 10x difference in latency on the same computing platform [10]. Further, when the FBNet authors optimized networks for different smartphones, they found a DNN that ran fast on the iPhone X, but slow on the Samsung Galaxy S8; as well as a DNN ran fast on the iPhone, but slow on the Samsung [9]. Depending on the processor

[1] In our terminology, we refer to the task-specific end of the network as the *head*, and we refer to the portion of the network that was originally designed for image classification as the *backbone*.

and the kernel implementations, different convolution dimensions run faster or slower, even when the number of MACs is held constant.

To make use of these new realities, we propose a playbook for producing the lowest-latency, highest-accuracy DNNs on a target task and a target computing platform:

1. Run Neural Architecture Search directly on the target task (e.g. object detection or semantic segmentation), and not on a proxy task (e.g. image classification).[2]
2. Use modern supernetwork-based NAS, and enjoy the fact the search converges quickly.
3. Configure the NAS to optimize for both accuracy (on the target task) and latency (on the target platform).

In the rest of this paper, we investigate the effectiveness of this playbook by doing a proxyless search using the Cityscapes semantic segmentation dataset [11], targeting low-latency inference on the NVIDIA Xavier embedded GPU computing platform [12], and producing fast and accurate DNNs. We refer to the optimized DNNs generated in this study as *SqueezeNAS* networks.

2. Related work

2.1. Semantic Segmentation

Semantic segmentation is the computer vision task of assigning a class for each pixel in a given image. It is a workhorse in many computer vision applications areas, from automotive (segmenting the road and lane lines) to aerial imagery analysis. To train and evaluate semantic segmentation models, a number of datasets have been developed such as Cityscapes[11], ADE20k[13], NYUDv2[14], and PASCAL VOC[15] which have made the research in semantic segmentation algorithms much more accessible.

DNNs initially found success with image classification tasks; AlexNet[16] and its successors dramatically increased the state-of-the-art accuracies on the ImageNet and CIFAR10 classification tasks. Following this success, Long et al. developed Fully Convolutional Networks for Semantic Segmentation[17] (FCN) by utilizing an Imagenet backbone - achieving then state-of-the-art performance on VOC PASCAL and NYUDv2. DeepLab[18] later leveraged dilated convolutions to further increase the accuracy on segmentation benchmarks. The typical workflow of these approaches is to start with an image classification DNN and then adapt it for higher resolution, increasing the compute proportionally to the number of pixels. This part is usually called the encoder or backbone. The semantic segmentation network's decoder uses the low resolution feature maps from the encoder to perform more computation and generates an output prediction for each pixel that is the same size as the input resolution. This decoder or "head" can be a series of deconvolutions like in FCN, or something much more complex like the dilated Spatial Pyramid Pooling (ASPP) module seen in the DeepLab[18, 19, 20] Family.

Semantic segmentation, however, is a very different task from image classification. One way semantic segmentation networks differs from image classification networks is that they usually requires much higher resolution inputs to get good results. Image classification networks commonly use an input at a $224x224$ resolution, while segmentation networks often use more than 40 times the number of pixels. Segmentation networks also typically have exotic architectures due to the fact that they have a dense high resolution output. Large input resolutions also means that segmentation networks often use trillions of Multiply-Accumulates (MACs) for a single image prediction, whereas accurate image classification networks are usually in the tens of billions. Many early deep learning approaches focused on maximizing accuracy, without a regard to the number of operations or latency.

2.2. Efficient Network Design

From 2012 to 2016, a substantial portion of the computer vision research community focused on designing DNNs that achieved the highest possible accuracy on image classification. These networks were then modified and finetuned to perform other tasks such as object detection and semantic segmentation. This led to significant year-over-year improvements in accuracy on image classification (from AlexNet[16], to ZFNet[21], to VGGNet[4], to ResNet[22]), which further led into improved accuracy on the other computer vision tasks. This also led to an upward trend in computation time as well as parameter count. To mitigate this, starting in 2016 with SqueezeNet[3], Iandola et al. were successfully able to design networks that were 50 times smaller in parameters compared to AlexNet[16]. MobileNets[23] and ShuffleNet[24] came soon after, optimizing their networks to have fewer computational operations, with the goal of reducing latency. The problem of reducing the size, the number of operations, and ultimately the latency of DNN inference became a widely-studied problem in computer vision research. One thing to note is that this research typically requires expertise in both computer vision as well as computer architecture.

2.3. Neural Architecture Search (NAS)

Since classification networks have commonly been used as the encoder for other computer vision tasks [25, 26, 19, 27, 28], they are often a target of NAS searches[8, 9, 29, 30, 31, 32, 33, 34] in efforts to exceed the performance of expert designed networks. However, many prior NAS works such as some that use Reinforcement Learning or Evolutionary search algorithms can often require thousands of GPU days

[2] If you wish to use outside data from an other task for pretraining, first perform a proxyless search to produce the DNN architecture, then reset the weights and do pretraining on outside data, and finally finetune on the target task.

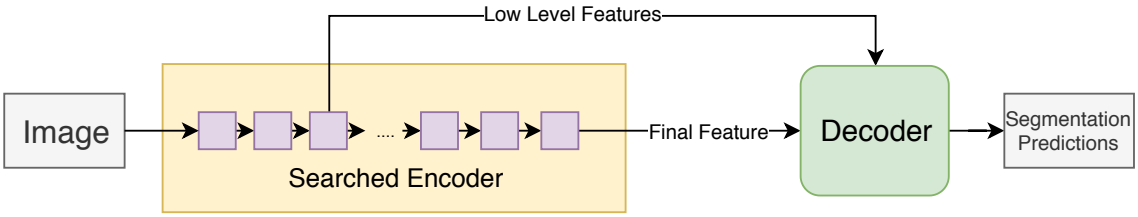

Figure 1: General Encoder-Decoder Structure of our Segmentation Networks. We search the architecture space of the "Searched Encoder". We use either an ASPP[19] inspired decoder or the LR-ASPP Decoder depending on the search space.

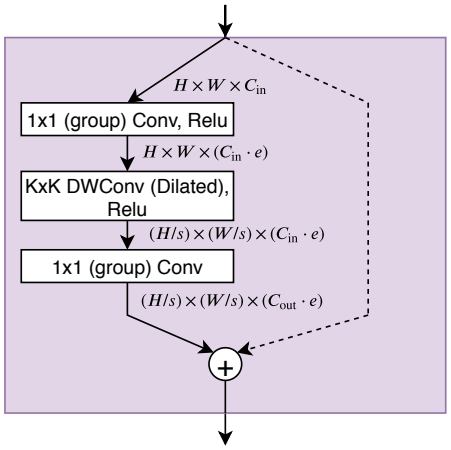

Figure 2: Diagram showing the architecture of the Inverted Residual blocks we use in our search space. They are parameterized so that the number of groups ($g \in \{1, 2\}$) in the $1x1$ convolutions, the dilation rate of the depthwise convolution ($d \in \{1, 2\}$), the kernel size ($k \in \{3, 5\}$), and the expansion ratio ($e \in 1, 3, 6$) may vary for different candidate blocks. The 12 possible configurations are shown in shown in Figure 7 and Appendix B. C_{in}, C_{out}, and stride ($s \in \{1, 2\}$) are defined by the macro level parameters shown in Appendix C. A residual connection is used if $C_{\text{in}} = C_{\text{out}}$ and $s = 1$.

per search[34, 35, 6]. The compute time of these searches would further increase if they were run directly on these high resolution vision tasks. Howard et al. in MobileNetV3[36] created networks for semantic segmentation by modifying classification networks that were produced by NAS. The NAS in that work had the objective of minimizing latency of the low resolution image classification network for mobile phones, and not for our ultimate goal of semantic segmentation at high resolution.

Many works have developed methods to greatly reduce the search time of NAS[37, 32, 33]. Recently, supernetwork-based NAS approaches have been proposed which have led to search times that are orders of magnitude faster by searching over millions of potential DNN designs while training just one supernetwork[8, 9, 29, 30, 31, 38]. While there has been some work searching directly on other vision tasks, most of these do not also directly optimize for hardware latency[26, 38, 25]. In our work described later in this paper, a gradient-based NAS method optimizes a supernetwork for both high semantic segmentation accuracy as well as low latency on our target hardware. Our particular NAS algorithm utilizes the Gumbel-Softmax[39] approximation of the categorical choice distribution which is also used in [9, 29, 31].

3. Architecture Search Space

In this work, we explore the space of encoders for semantic segmentation networks consisting of sequential Inverted Residual Blocks[40]. The blocks are parameterized as shown in Figure 2. In each architecture search, we constrain the macro-architecture and find optimal parameters for each block. This search space was chosen to be similar to the FBNet[9], MobileNetV2[40], and MobileNetV3[36] network families which allows us to directly compare our segmentation optimized networks to their classification optimized networks.

The general structure of all our networks is shown in Figure 1. We follow a common structure of some segmentation networks[19, 36] where the decoder uses both the final output features from the encoder as well as a low level feature map from an earlier layer in the encoder.

3.1. Constrained Macro-Architecture

In our experiments we searched 3 search spaces: `Small`, `Large`, and `XLarge`. To define each of these architecture spaces, we first constrain the macro-architecture of the encoder networks. The macro-architectures describe the total number of blocks N in the encoder, which decoder is used, and which layer our lower level features come from. For each block, we fix the input and output channels(C_{in} and C_{out}) and whether each block uses a stride of $s = 1$ or $s = 2$ in the depthwise convolution layer. It should be noted that since we allow each block to choose a no-op skip connection, the final layer count can be less than N.

The specifics of each of the three search spaces are shown in Appendix C. They were chosen to be comparable to the MobileNetV2[40] and MobileNetV3[36] segmentation networks. In the `Small` and `Large` search spaces, we use the LR-ASPP[36] decoder. In the `XLarge` search space, we use the variation of the ASPP decoder with fully depthwise convolutions proposed in Chen et al. [20].

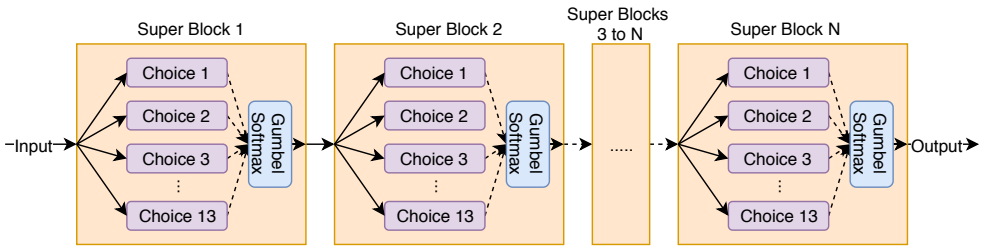

Figure 3: Diagram of a supernetwork with N superblocks, which each contain 13 possible candidate block choices.

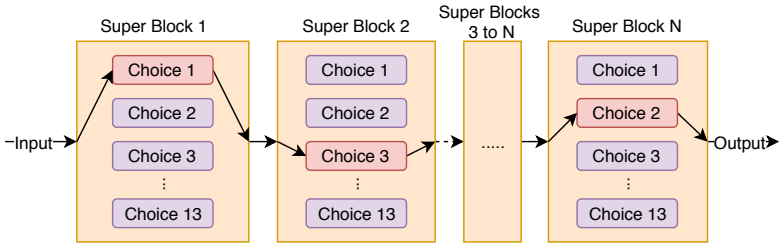

Figure 4: Diagram of an architecture path of a sampled architecture from a supernetwork. In this example, the 1st superblock uses candidate block 1, the 2nd superblock uses candidate block 3, and the Nth superblock uses candidate block 2.

3.2. Block Search Space

Within each macro-architecture space, our NAS picks the optimal hyperparameters for each block or replaces it with a no-op skip connection. As shown in Figure 2, these hyperparameters define whether the 1x1 convolutions are grouped, whether the depthwise convolution is dilated with a rate 2, the size of the kernel k for the depthwise convolution, and the expansion ratio e. We choose from 12 possible configurations as shown in Figure 7 and Appendix B as well as the skip connection.

4. Neural Architecture Search Algorithm

The particular approach and search space we use is similar to those used in [9]. We consider architecture search as a path-selection problem within a stochastic supernetwork such that any particular architecture in our search space is represented by some path through our supernetwork. As illustrated in Figure 3, we define our supernetwork to be a sequence of superblocks that each contain the candidate block choices. Running inference for a sampled architecture of the stochastic supernetwork is shown in Figure 4.

We simultaneously co-optimize the convolutional weights (w) and architecture parameters (θ) of the stochastic supernetwork to minimize our loss function which is defined as

$$L(\theta, w) = L_P(\theta, w) + \alpha * L_E(\theta) \tag{1}$$

where L_P represents the problem-specific loss, L_E is resource aware-loss term, and the hyperparameter α controls the tradeoff made between the two. As this work focuses on semantic segmentation, L_P is a pixel-level cross-entropy loss. For L_E we experiment with both the estimated total inference latency on our target-platform as well as the estimated number of Multiply-Accumulates for the network.

4.1. Gumbel-Softmax

In order to make computation and optimization of the stochastic supernetwork tractable, each superblock picks a candidate block independent of the choices of other superblocks. Thus, we can model the choice of a candidate block as sampling from an independent categorical distribution where the probability of choosing candidate block j for superblock i in the network is $p(i, j)$. We define this probability using the softmax function on our architecture parameters (θ) for each superblock.

$$p(i, j|\theta) = \frac{e^{\theta_{i,j}}}{\sum_j^{13} e^{\theta_{i,j}}} \tag{2}$$

The categorical distribution is difficult to directly optimize efficiently, so we use the Gumbel-Softmax relaxation of the categorical distribution proposed in Jang et al. [39]. Sampling from the Gumbel-Softmax distribution allows us to efficiently optimize the architecture distribution by using gradient descent on the stochastic supernetwork. The Gumbel-Softmax distribution is controlled by a temperature parameter t. As t approaches zero, the Gumbel-Softmax distribution becomes equivalent to the categorical distribution. The temperature parameter is annealed from 5.0 to 1.0 during our search.

4.2. Early Stopping

A caveat of our supernetwork approach is that the optimization requires computation through every single candidate block for every iteration regardless of the learned architecture distribution. As optimal network architectures converge, the probability that a low performing candidate block is chosen decreases, but it still continues to use compute. So we use a compute optimization when the estimated

probability of a candidate block being chosen is less than 0.5%. We simply remove it from the supernetwork. While there is some low probability that a removed candidate block could be optimal later in the search process, we have not seen this in practice. This compute optimization can cut search time in half.

4.3. Resource-Aware Architecture Search

We define our resource aware loss as follows:

$$L_E(\theta) = \sum_{j}^{N} \sum_{i}^{13} p(i,j|\theta_i) C(i,j) \tag{3}$$

$C(i,j)$ represents the network resource cost of choosing candidate j in block i of the network. We model the resource cost of each block to be independent of others. C can also be implemented as a lookup table similar to FBNet[9] so the resource costs only needs to be calculated once. Depending on how we build the lookup table, we can optimize for many different objectives ranging from hardware-agnostic metrics such as MACs or parameter size to hardware-aware costs like inference-time, memory accesses, or energy usage.

5. Experiments and Results

We demonstrate two key ideas: first, Neural Architecture Search (NAS) is a powerful tool that can yield high-accuracy, low-latency networks. The second idea is that optimizing for hardware-agnostic metrics such as Multiply-Accumulates (MACs) is not an ideal proxy and can lead to sub-optimal latency results.

To demonstrate this, we use search spaces similar to prior work: the `Small`, `Large`, `XLarge` search spaces, which we define in Section 3.1. We first use our NAS method along with a hardware-agnostic objective (MACs) to generate a semantic segmentation network in each of our search spaces. These networks are comparable with current state-of-the-art networks on the MACs/Accuracy trade-off curve. We then measure the latency of these low-MAC networks on an embedded platform (NVIDIA Xaiver) as a baseline. Finally, we use our NAS method again on the same search spaces, but optimize with a hardware-aware objective (latency) to find 3 new networks targeted at similar latencies of the networks generated in the previous search.

All search experiments are done on the Cityscapes[11] semantic segmentation dataset.

5.1. Hardware-Agnostic Search

For our hardware-agnostic architecture searches, we apply our NAS method with a Multiply-Accumulates (MACs) minimization objective to create networks that are on the pareto-optimal tradeoff curve of MACs vs mIOU. To implement this, for each block i in the network, we compute the number of Multiply-Accumulates for each candidate block j and store the results in the lookup table C such that $C(i,j) = MACS_{i,j}$.

We then perform an independent search in each of the three search spaces and obtain three MAC-optimized *SqueezeNAS-MAC* networks. As shown in Table 1, we achieve results that exceed the performance of prior work without NAS. We also achieve comparable results with MobileNetV3[36] w.r.t the number of MACs. We finally measure the inference time of the 3 networks on a NVIDIA Xavier using cuDNN 7.3.1. As in many applications requiring real-time inference, we use **batch size = 1** for all of our latency tests throughout the paper. The results can be seen in Table 1.

5.2. Hardware-Aware Search

Our hardware-aware searches use the same NAS algorithm and architectural search space as the hardware-agnostic approach, but now we use a latency minimization objective for the resource-aware loss; formulated as $C(i,j) = Latency_{i,j}$. To compute the latency of every candidate j in each block i, we measure the inference time of all candidates on our target platform. We conduct 3 new independent hardware-aware searches that target the latencies measured from the hardware-agnostic networks. The results of these searches yield the three *SqueezeNAS-LAT* networks. Our hardware-aware searches find networks that have significantly higher accuracies at the same or lower latency compared to the hardware-agnostic networks seen in Table 1. The latency-optimized networks have a higher number of MACs, but they still run faster on our target device.

5.3. Implementation

5.3.1 Architecture Search

In our supernetwork-based architecture search, we train directly on the Cityscapes training set, without using any proxy task. After we finish optimizing the supernetwork, we sample 200 discrete architectures from the optimal architecture distribution. We estimate the performance of each architecture by running inference on the Cityscapes fine validation dataset using the architecture path within the supernetwork as shown in Figure 4. After validating the 200 architectures, we choose one from this estimated pareto-optimal frontier and retrain the singular architecture. The MAC-optimized networks are chosen to have comparable MACs to the MobileNetV3 segmentation networks, and the Latency-optimized networks are chosen to have inference latencies comparable with our MAC-optimized baseline networks.

5.3.2 Training Details

For comparability with other results, we follow a similar pretraining scheme to that used in [20]. After the architecture search is complete, we pretrain our sampled networks on ImageNet classification using the training regime used in ResNet[22]. We then do a stage of training on COCO [43] segmentation masks using the scheme used in

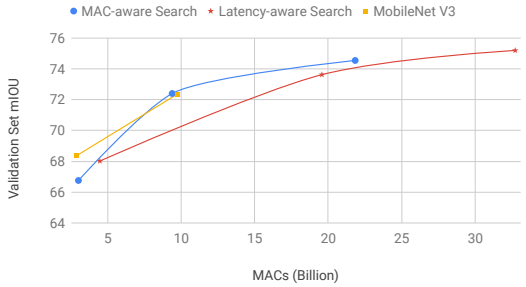

Figure 5: **MACs** vs mIOU on Cityscapes validation set. SqueezeNAS MAC-optimized and latency-optimized models compared to MobileNetV3[36] segmentation models.

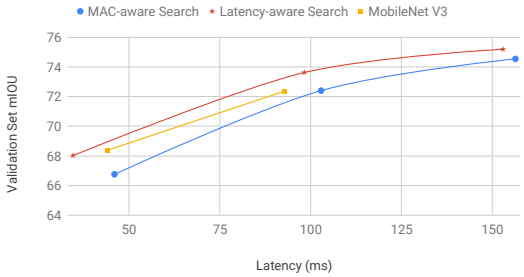

Figure 6: **Latency** vs mIOU on Cityscapes validation set. SqueezeNAS MAC-optimized and latency-optimized models compared to MobileNetV3[36] segmentation models.

Architecture	Class mIOU	Latency (ms)	MACs (G)	MACs/sec (G)	Params (M)
C3[41]	61.96	-	6.29	-	0.19
EDANet[42]	65.11	-	8.97	-	0.68
MobileNetV2[40]	70.71	-	21.27	-	5.75
MobileNetV3-Small[36]	68.38	44.01	2.90	65.89	0.47
MobileNetV3-Large[36]	72.36	92.78	9.74	104.97	1.51
SqueezeNAS MAC Small	66.76	46.01	3.01	65.37	0.30
SqueezeNAS MAC Large	72.40	102.90	9.39	91.21	0.73
SqueezeNAS MAC XLarge	74.62	156.41	21.84	139.63	1.80
SqueezeNAS LAT Small	68.02	34.57	4.47	129.17	0.48
SqueezeNAS LAT Large	73.62	98.28	19.57	199.17	1.90
SqueezeNAS LAT XLarge	75.19	152.98	32.73	213.94	3.00

Table 1: Cityscapes Validation mIOU of MAC-Aware Searched, Latency-Aware Searched, and published state-of-the-art models. The latency values were benchmarked on the NVIDIA Xavier on the 30 watt power mode. Latency values for the MobileNetV3[36] segmentation networks were obtained using an open source re-implementation.

DeepLabV3+[20]. Then, we train on the Cityscapes coarse training set annotations for 40 epochs, and finally we train on the Cityscapes fine training set annotations for 100 epochs, cutting the learning rate by 10 at 50 and 75 epochs. All segmentation training uses patch sizes of 768x768 pixels and are optimized with SGD with momentum, using a base learning rate of 0.05 and a weight-decay of 1e-5.

We use servers with 8 Nvidia Turing GPUs with 24GB of VRAM and train in mixed precision, allowing us to both leverage the tensor cores on the GPUs and fit a larger batch in VRAM. When we search larger supernetworks, we employ Synchronized BatchNorm[44] to keep our BatchNorm[45] batch sizes large enough for training stability.

5.4. Results

First, our hardware-agnostic NAS method is able to produce networks that are competitive with the state-of-the-art with respect to both MACs and latency. Compared to expert designed networks found without NAS such as EDANet [42] and MobileNetV2 [40], our MAC-optimized networks achieve higher accuracy at a fraction of the MACs, as shown in Table 1. Our *SqueezeNAS-MAC-Small* network achieves more than 3% higher absolute mIOU compared to the EDANet [42] segmentation network, which has three times more MACs than ours. Our *SqueezeNAS-MAC-Large* network achieves more than 2.5% higher absolute mIOU compared to the MobileNetV2[40] segmentation network, which has more than double the MACs of our network.

Our hardware-aware networks all have higher accuracy while having less latency compared to their hardware-agnostic counterparts. The *SqueezeNAS-LAT-Small* network is 1.3% more accurate, 35% faster, and has 50% more MACs compared to *SqueezeNAS-MAC-Small*. The *SqueezeNAS-LAT-Large* network is 1.2% more accurate, 4% faster, and has more than double the number of MACs compared to *SqueezeNAS-MAC-Large*. This means that we're able to achieve double the number of operations in the same inference time window, as seen in Figure 10. This allows us to have much more expressive models that yield better accuracy while running at the same framerate.

We also compare our networks to the efficient segmentation networks proposed in MobileNetV3[36]. These networks were optimized for image classification using NAS and were then modified for the semantic segmentation task. The *SqueezeNAS-MAC-Large* network is able to match the accuracy of the *MobileNetV3-Large* network while using

Architecture	Class mIOU	Latency (ms)	MACs (Giga)	Params (M)
MobileNet V3-Small[36]	69.4	44.01	2.90	0.47
MobileNet V3-Large[36]	72.6	92.78	9.74	1.51
SqueezeNAS LAT Small	66.8	34.57	4.47	0.48
SqueezeNAS LAT Large	72.5	98.28	19.57	1.90

Table 2: Test mIOU of Different Architectures on Cityscapes. The latency values were benchmarked on the NVIDIA Xavier on the 30 Watt power setting.

Architecture	Search Time (GPU Days)
NAS with RL[7]	22,400
NASNet[34]	2,000
MnasNet[35]	2,000 [3]
MobileNetV3[36]	$>$ 2,000 [4]
AmoebaNet[6]	3,150
FBNet[9]	9
DARTS[8]	4
SqueezeNAS MAC Small	7.0
SqueezeNAS MAC Large	9.7
SqueezeNAS MAC XLarge	14.6
SqueezeNAS LAT Small	8.7
SqueezeNAS LAT Large	9.4
SqueezeNAS LAT XLarge	11.5

Table 3: Search times of SqueezeNAS Networks compared to other NAS methods.

less MACs as seen in Table 1. It should be noted that the *SqueezeNAS-MAC-Small* network does perform worse than MobileNetV3-Small. However, the MobileNetV3 networks do use Squeeze-Excitation[46] and Hard Swish[47] activations which our networks do not. *SqueezeNAS-LAT-Small* runs 20% faster than *MobileNetV3-Small* while achieving an mIOU that is only 0.26% lower. *SqueezeNAS-LAT-Large* achieves over 1.2% higher accuracy with less than 6% higher latency.

We have noticed a small gap in our validation and test accuracies. This may be due to the small size of the Cityscapes dataset or the lack of our use of test-time augmentations.

The full validation set results are shown in Table 1. Test set results are shown in Table 2. Each network was found in less than 15 GPU-days, which is more than 100 times less than some reinforcement learning and genetic search methods as shown in Table 3.

6. Network Analysis

We now compare the block choices of the hardware-agnostic, hardware-aware, and MobileNetV3 segmentation networks. Since the three families all use the same Inverted Residual blocks, we can place MobileNetV3's building blocks into our 13 candidate blocks which can be seen

[3] Approximated from TPUv2 Hours. In the literature, it has been suggested that one 8-core TPUv2 is comparable to 8 NVIDIA V100s [48].

[4] Starts with a MnasNet network (search time is approximated from TPUv2 Hours) and adapts it with the NetAdapt NAS algorithm. The NetAdapt search time is not included since it is not reported in the paper[36].

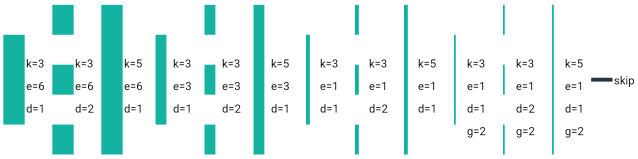

Figure 7: Visualization of the search space. Each of these blocks represent a MobileNetV2[40] Inverted Residual block as seen in Figure 2. *k* represents the kernel size of the middle depthwise convolution layer. *e* represents the expansion multiple for the depthwise convolution. *d* represents the dilation rate of the depthwise convolution. *g* represents the number of groups(1 if not listed) in the 1x1 convolutions. Finally we have a no-op *skip* connection that can be chosen.

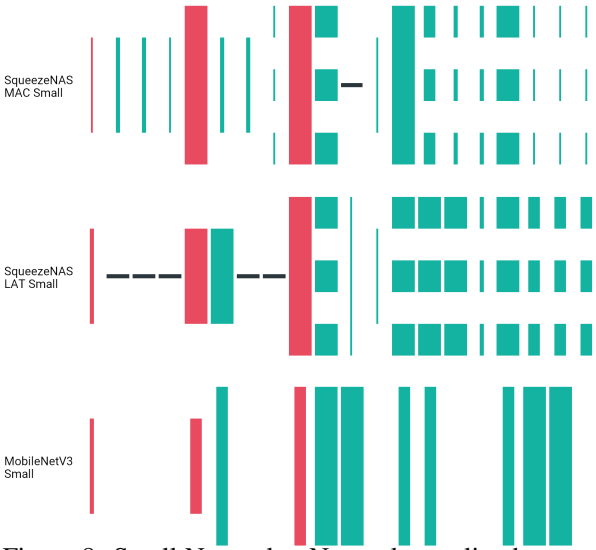

Figure 8: Small Networks. Networks are lined up at their down-sampling block represented by the color red.

in Figure 7. One caveat to note is that we are not accounting for the Squeeze-Excitation[46] blocks that are in some MobileNetV3 blocks for visualization, and the expansion ratios are approximated to be either 1, 3, or 6.

We visualize the small networks in Figure 8. We first examine our *SqueezeNAS-MAC-Small* network and see that it uses a mix of low and high expansion blocks. It also uses the highest compute candidate block possible for its second and third downsampling blocks. The last thing to note is that our NAS method chose to use dilated *3x3* blocks for the last stage of the network. This is a very common trend that we see

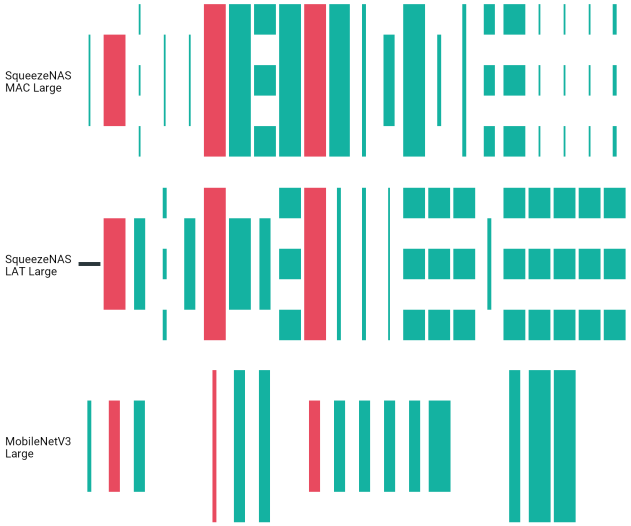

Figure 9: Large Networks. Networks are lined up at their down-sampling block represented by the color red.

in expert designed, high resolution semantic segmentation networks such as DeepLabV3[19] and PSPNet[49].

The next small network we examine is our *SqueezeNAS-LAT-Small*, which is more accurate and lower latency than the previous network. A radical difference that we immediately see is that the network uses many more skip connections instead of low expansion blocks. This makes the macro-architecture look very similar to that of *MobileNetV3-Small*, also visualized in Figure 8. Both networks do aggressive down-sampling and push their compute (via higher expansion ratios) later in the network, where the resolution is lower and the base channel count is higher. This yields a higher arithmetic-intensity.[5] On devices like GPUs, which are typically memory bandwidth bound, higher arithmetic-intensity allows for more operations for the same memory bandwidth. It is interesting to see how both of the latency optimizing NAS methods produce similar networks that follow intuition from a computer architecture perspective. The networks differ in that our network uses more blocks but with a smaller kernel sizes near the end of the network. (*3x3* dilated vs *5x5*). Which is consistent with our hardware-agnostic network and other related segmentation work.

We now visually compare the large networks in Figure 9. Both *SqueezeNAS-MAC-Large* and *SqueezeNAS-LAT-Large*, follow the a trend similar to our smaller networks where they all have high compute down-sampling blocks, as well as heavy use of dilated convolutions in the second half of the networks. If we compare the MAC and latency networks, we see that the MAC network has the majority of its compute in the middle, whereas the latency network pushes its compute towards the end where it would yield a higher overall arithmetic-intensity for the network. This also has the side-effect of more than doubling the total number of MACs but still decreasing latency. We can conclude with saying that our NAS method is effective at producing high-throughput networks while maintaining low latency as seen in Figure 10.

[5] Arithmetic Intensity is the ratio of MACs to memory traffic [50]. When arithmetic intensity drops below a certain threshold, the latency is dominated by the time to access data from memory.

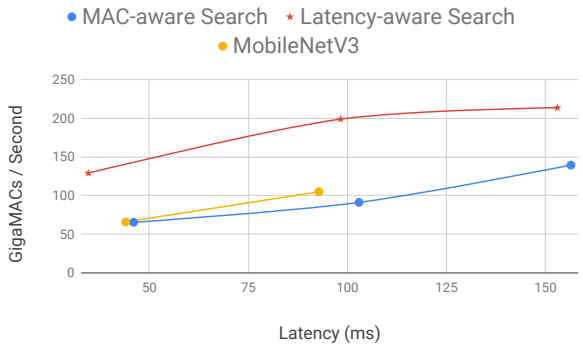

Figure 10: Comparison of Throughput (GigaMACs per second) vs Latency, of SqueezeNAS networks and MobileNetV3[36] segmentation networks.

7. Conclusion

In Section 1, we presented a playbook for replacing architecture-transfer with neural architecture search to develop DNNs that are optimized for specific tasks and for specific computing platforms. After following this playbook throughout this paper, we have learned the following.

First, by doing a proxyless search on a semantic segmentation dataset, our NAS produced the *SqueezeNAS* family of models, which achieve superior latency-accuracy tradeoffs relative to MobileNetV3 on the semantic segmentation validation set. We attribute our superior results, at least in part, to the fact that the backbone of the MobileNetV3 semantic segmentation network was designed by NAS for the proxy task of image classification on mobile phones (that is to say, it was not designed in a proxyless manner for semantic segmentation on embedded GPU devices).

Second, while the MobileNetV3 authors searched for thousands of GPU days, our approach produced these results in 7 to 15 GPU days per search. In other words, modern supernetwork-based NAS can now produce state-of-the-art results in less than a weekend of search time on an 8 GPU server.

Third, recall that we did two sets of NAS experiments: one in which we searched for low-MAC models, and one where we searched for low-latency models on a target computing platform. We achieved substantially faster and more accurate models when searching for latency on the target platform. Finally, given the growing diversity of chips and computing platforms designed for deep neural networks, we believe that using NAS to optimize for low latency on a target computing platform will continue to grow in importance.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.

[2] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[3] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018.

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, 2019.

[7] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[8] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[9] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[10] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "EmBench: Quantifying performance variations of deep neural networks across modern commodity devices," in *International Workshop on Embedded and Mobile Deep Learning (EMDL)*, 2019.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] NVIDIA, "Jetson AGX Xavier developer kit," 2018. [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[13] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[14] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[17] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, Apr. 2017. [Online]. Available: https://doi.org/10.1109/TPAMI.2016.2572683

[18] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *CoRR*, vol. abs/1606.00915, 2016. [Online]. Available: http://arxiv.org/abs/1606.00915

[19] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017. [Online]. Available: http://arxiv.org/abs/1706.05587

[20] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision*, 2018, pp. 801–818.

[21] M. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014, pp. 818–833.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[24] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[25] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for efficient multi-scale architectures for dense image prediction," in *Advances in Neural Information Processing Systems*, 2018, pp. 8699–8710.

[26] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7036–7045.

[27] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *CVPR*, 2017, pp. 2980–2988.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 91–99.

[29] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rylqooRqK7

[30] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HylVB3AqYm

[31] A. Shaw, B. Dai, W. Liu, and L. Song, "Bayesian meta-network architecture learning," *CoRR*, vol. abs/1812.09584, 2018. [Online]. Available: http://arxiv.org/abs/1812.09584

[32] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*, 2018, pp. 4095–4104. [Online]. Available: http://proceedings.mlr.press/v80/pham18a.html

[33] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 19–34.

[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[35] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[36] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," *arXiv:1905.02244*, 2019.

[37] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI Conference on Artificial Intelligence*, 2018.

[38] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 82–92.

[39] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2017.

[40] M. B. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," 2018, pp. 4510–4520.

[41] H. Park, Y. Yoo, G. Seo, D. Han, S. Yun, and N. Kwak, "Concentrated-comprehensive convolutions for lightweight semantic segmentation," *CoRR*, vol. abs/1812.04920, 2018. [Online]. Available: http://arxiv.org/abs/1812.04920

[42] S. Lo, H. Hang, S. Chan, and J. Lin, "Efficient dense modules of asymmetric convolution for real-time semantic segmentation," *CoRR*, vol. abs/1809.06323, 2018. [Online]. Available: http://arxiv.org/abs/1809.06323

[43] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *ECCV*, 2014.

[44] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun, "Megdet: A large mini-batch object detector," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[45] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[46] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[47] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017. [Online]. Available: http://arxiv.org/abs/1710.05941

[48] S. Reitsma, "Cost comparison of deep learning hardware: Google TPUv2 vs Nvidia Tesla V100," 2019. [Online]. Available: https://medium.com/bigdatarepublic/cost-comparison-of-deep-learning-hardware-google-tpuv2-vs-nvidia-tesla-v100-3c63fe56c20f

[49] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890.

[50] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), Tech. Rep., 2009.