SplitNets: Designing Neural Architectures for Efficient Distributed Computing on Head-Mounted Systems

Xin Dong¹, Barbara De Salvo², Meng Li², Chiao Liu², Zhongnan Qu³, H.T. Kung¹ and Ziyun Li²
¹Harvard University, ²Meta Reality Labs, ³ETH Zurich
xindong@g.harvard.edu

Abstract

We design deep neural networks (DNNs) and corresponding networks' splittings to distribute DNNs' workload to camera sensors and a centralized aggregator on head mounted devices to meet system performance targets in inference accuracy and latency under the given hardware resource constraints. To achieve an optimal balance among computation, communication, and performance, a split-aware neural architecture search framework, SplitNets, is introduced to conduct model designing, splitting, and communication reduction simultaneously. We further extend the framework to multi-view systems for learning to fuse inputs from multiple camera sensors with optimal performance and systemic efficiency. We validate SplitNets for single-view system on ImageNet as well as multi-view system on 3D classification, and show that the SplitNets framework achieves state-of-the-art (SOTA) performance and system latency compared with existing approaches.

1. Introduction

Virtual Reality (VR) and Augmented Reality (AR) are becoming increasingly prevailing as one of the next-generation computing platforms [3]. Head mounted devices (HMD) for AR/VR feature multiple cameras to support various computer vision (CV) / machine learning (ML) powered human-computer interaction functions, such as object classification [40, 61], hand-tracking [19] and SLAM [38].

Due to the recent advances of camera technologies, tiny multi-layer stacked cameras with AI computing capability arise [32, 33], as depicted in Figure 1. Because of the small form factor, these intelligent cameras have highly constrained resources compared with general purpose mobile or data center grade AI processors. However, each camera is still capable of performing pre-processing directly after image acquisition, significantly reducing expensive raw image data movement. In a modern HMD system, the distributed intelligent stacked image sensors and a central AR/VR processor (aggregator) form the hardware backbone to realize complex CV/ML functions on device [3, 13, 14].

Within such systems, it's natural to split the machine learning workload for an application between the sensors and the centralized computer (aggregator). As is shown in Figure 2, for an application that requires multiple layers of convolutional neural networks and fusion of multiple input sources, the early layers can be distributed to the on-sensor processing. The feature fusion and rest of processing are on the aggregator. This way, overall system latency can be improved by leveraging direct parallel processing on sensors and reduced sensor-aggregator communication.

The success of distributed computing for DNNs between sensors and aggregator heavily relies on the network architecture to satisfy the application and hardware system constraints such as memory, communication bandwidth, latency target and etc. Prior work [10, 27, 28, 39] searches network partitions (i.e., the splitting points) for existing models in either exhaustive or heuristic manners. Some work [4, 11, 36, 45, 46] manually injects a bottleneck mod-
We design SplitNets, a split-aware NAS framework, for efficient and flexible searching of the splitting module in the network in the distributed computing context where the splitting module is able to split the network, compress features along the channel dimension for communication saving as well as view fusion for multi-view tasks.

We introduce a series of techniques for module initialization and sampling to stabilize the training with information bottlenecks and mitigate introduced accuracy degradation. We further extend SplitNets to support searching of splitting modules with view fusion for multi-view tasks. To our best knowledge, it is the first framework supporting the position searching of information compression / fusion in a multi-input neural network. Overall, our contributions are summarized as follows:

- We propose SplitNets, a split-aware NAS framework, for efficient and flexible position searching of splitting modules for single / multi-view task.
- We introduce splitting modules for single- and multi-view tasks which can achieve model splitting, feature compression, as well as view fusion simultaneously.
- To search the optimal position of the splitting module, we propose to use separate supernets for sensors and the aggregator respectively, and stitch them together to form the split-aware model of interest, using the shared splitting module as the joint point. In addition, we combine the compression- / recovery- / fusion- based splitting module design with custom weights initialization, and a novel candidate networks sampling strategy to mitigate the accuracy drop due to model partition.
- We evaluate SplitNets with single-view classification and multi-view 3D classification. Empirical observations validate the importance of joint model and splitting position search to improve both task and system performance. Our results show that optimized network architectures and model partitions discovered by SplitNets significantly outperform existing solutions and fit the distributed computing system well on AR/VR glasses.

### 2. Background and Related Work

We consider a system that consists of three kinds of components, V sensors, one aggregator, and communication interfaces between any of sensors and the aggregator. For simplicity, we assume that sensors are homogeneous.

A DNN is a composition of layers / computation which can be distributed to sensors and aggregator. Compared

### Table 1. Comparison of different DNN computing offload paradigms.

| Components | System factors to be considered |
|------------|----------------------------------|
| Sensor     | ![Computation capability](✓)    |
|            | ![Peak memory constraint](✓)    |
|            | ![Number of sensors (i.e., parallelism)](✓) |
| Comm.      | ![Communication bandwidth](✓)  |
| Aggregator | ![Computation capability](✓)    |
| Whole system | ![Task performance (e.g., accuracy)](✓) |
|            | ![Overall latency = on-sen. latency + comm. latency + on-agg. latency](✓) |

Table 2. Factors need to be balanced when distributing DNN computation. The listed factors are interwoven with each other and have to be considered holistically.

The challenge of splitting DNNs comes from the complicated mutual-impact of many model and hardware factors. For example, existing (hand-crafted and searched) model architectures are designed without considering distributed computing over multiple computing modalities, and thus not suitable for splitting in the first place. In addition, the position of the splitting point as well as the inserted compression module will simultaneously impact model performance, computation offload, communication, and hardware utilization in different directions. Heuristic and rule-based methods are limited in this context.

In this work, we adapt the neural architecture search (NAS) approach to automatically search split-aware model architectures targeting the distributed computing systems on AR/VR glasses. We propose the SplitNets framework that jointly optimizes the task performance and system efficiency subject to resource constraints of the mobile AR/VR system featuring smart sensors. We specifically answer the following two questions:

1. Can we jointly search for optimal network architectures and their network splitting solution between sensors and the aggregator while satisfying resource constraints imposed by the underlining hardware?
2. Can we learn an optimal network architecture to compress and fuse features from multiple sensors to the aggregator and achieve SOTA performance and efficiency compared with conventional centralized models?
with on-sensor computing (All-on-sensor) and conventional mobile computing (All-on-aggregator), distributed computing has the flexibility to achieve a better balance between computation and communication, and enable high utilization of hardware, as summarized in Table 1.

A challenge is to determine a splitting point in the DNN. The on-sensor (on-sen.) part $f_{\text{sen}}$, which is composed of all layers of the DNN before the splitting point, is executed on sensors’ processors. The feature $z$ generated by each sensor will be uploaded to the aggregator. The on-aggregator (on-agg.) part $f_{\text{agg}}$, which consists of all remaining layers of the DNN after the splitting point, receives $z$ and finishes its computation on the computationally more capable aggregator processor.

The problem of finding the optimal splitting point can be summarized as follows,

$$
\min_{f_{\text{sen}}, f_{\text{agg}}} \quad T_{\text{sen}}(f_{\text{sen}}, x) + T_{\text{comm}}(z) + T_{\text{agg}}(f_{\text{agg}}, z)
$$

where $z = f_{\text{sen}}(x)$

s.t.

$$
L(f_{\text{agg}} \circ f_{\text{sen}}; D_{\text{val}}) \leq \text{Loss target}
$$

$$
\text{PeakMem}(f_{\text{sen}}, x) \leq \text{Mem}_{\text{sen}},
$$

where $T(\cdot, \cdot)$ is latency measurement function and $L$ is the loss function, e.g., cross entropy. PeakMem$(\cdot, \cdot)$ measures the peak memory consumption of the on-sensor part, and Mem$_{\text{sen}}$ is the memory size of a sensor processor.

To find the optimal splitting point, one has to take several factors into consideration simultaneously as summarized in Table 2. Existing solutions only consider a fraction of these factors via heuristic or exhaustive approaches [10, 27, 28, 39]. DNNs typically have tens or even hundreds of layers. Therefore, exhaustive searching with retraining is costly. Some literature proposes injecting a hand-crafted compression module to reduce the feature communication [4, 11, 36, 45, 46]. However, manually designing and inserting the compression module not only lead to high engineering effort and training cost, but also result in sub-optimal system performance and accuracy degradation.

In addition, previous methods all consider to split existing models like VGG [47] and ResNet [22]. However, those models may not be suitable for splitting. For example, Kang et al. [28] profiles seven models and finds that their best splitting points typically fall in the first or the last layers. Furthermore, injecting compression module to existing models usually leads to significant accuracy drop [36].

Other work proposes dedicated lossy (or lossless) compression techniques to reduce intermediate activation and save communication, including compressive sensing [59], pruning [26] and quantization [6, 8, 29]. SplitNets are orthogonal to, and can benefit from these prior approaches.

### 3. Split-Aware NAS

We adapt NAS to solve the constrained optimization in Equation (1). The goal is to minimize the systemic latency $T_{\text{sen}} + T_{\text{comm}} + T_{\text{agg}}$, and improve task performance while satisfying the hardware constraints.

We search the whole model $f_{\text{sen}} \circ f_{\text{agg}}$ and the position of splitting point, denoted as Split-Aware- NAS (or SA-NAS). This way, all layers before (or after) the splitting point will compose $f_{\text{sen}}$ or $f_{\text{agg}}$, respectively. In addition, we also consider a splitting module which is inserted at the position of splitting point. The feature compression in the splitting module can reduce communication cost by reducing the channel size $c$ of feature tensor $z \in \mathbb{R}^{c \times s \times s}$, but could affect training stability and result in loss of task performance. We propose several approaches to mitigate the issue. The splitting module evolves along with $f_{\text{sen}}$ and $f_{\text{agg}}$ during SA-NAS as elaborated in Section 3.1.3. Eventually, the searched network must have exactly one splitting module.

#### Preliminary: Two-stage NAS

Previous NAS approaches that leverage evolutionary search [42, 43, 50, 58] or reinforcement learning [52, 62–64] require excessive training due to the large number of models to be trained in a single experiment. Recent advancement of NAS decouples the model training and architecture search into two separate stages [5, 7, 18, 55, 60] that significantly reduces the training cost. In a two-stage NAS framework, the model training stage optimizes a supernet and its all sub-networks with weight-sharing. In the subsequent architecture optimization stage, sub-networks are searched based on system constraints to yield the best task and system performance trade-off. Due to the above advantages, we construct our SA-NAS based on two-stage NAS framework.

#### 3.1. Building and Training Supernet (Stage 1)

** Backbone of Supernet ** We build the supernet’s search space following FBNet-V3 [9] which uses the MobileNetv2 block (denoted as MB, a.k.a., inverted residual block) [44] as meta architecture. Check Appendix C for more details. We augment the supernet with SA-NAS specific search space (Figure 3) for our experiments as discussed in the following two sections.

#### 3.1.1 SA-NAS for Single-View SplitNets

Now we elaborate how to search SplitNets for a single-view task (e.g., ImageNet classification), on searching the position of a splitting module, as shown in Figure 3.

The splitting module for single-view tasks consists of two parts, **Conv-Reduce** and **Conv-Recover**, which will be discussed in Section 3.1.3 for their architectures and initialization strategy. The splitting module could be inserted...
Figure 3. Architecture sampling space in training SA-NAS. Both on-sen. and on-agg. networks consist of multiple phases with each containing a split module, flexible blocks before stride, a stride-2 block, and flexible blocks after stride. The splitting module can be a reduce operator (on-sen., green) or a recover / fuse operator (on-agg., purple). SA-NAS also includes standard NAS search space (blue) such as input resolution, channel widths, and etc.

after any of inverted residual block (MB) in a supernet theoretically. However, instead of equipping every MB with a candidate splitting module, we divide a supernet into several phases and insert one splitting module for each phase to reduce the number of splitting positions to be optimized.

Suppose a supernet has $N$ stride-2 MB blocks. We divide the whole model into $N$ phases where each phase has one stride-2 MB block. Within a phase, MB blocks before the stride-2 block have the same spatial size and similar channel sizes. We then insert one Conv-Reduce (green trapezium, Figure 3) layer $\phi'$ : $\mathbb{R}^{c_1 \times s_1 \times s_1} \rightarrow \mathbb{R}^{d_1 \times s_1 \times s_1}$ in each phase for the on-sensor supernet, associated with a gate variable $g_i \in \{0, 1\}$ indicating whether this layer is selected. A corresponding Conv-Recover (purple trapezium, Figure 3) layer $\phi'' : \mathbb{R}^{d_1 \times s_1 \times s_1} \rightarrow \mathbb{R}^{c_1 \times s_1 \times s_1}$ will be inserted into the on-aggregator supernet. Conv-Recover layer shares the same gate variable with Conv-Reduce, meaning that they are always selected, or not, at the same time. Using Conv-Recover allows weight-sharing of a block between different sampled sub-networks specified by different Conv-Reduce selections during training.

The number of MB blocks before (variable $M$) and after (variable $K$) the splitting module can be adjusted freely and searched through SA-NAS. This provides us with full search space of the splitting point at fine granularity, as well as a small amount of candidate networks to train. In particular, on-sen. and on-agg. blocks at the same depth learn similar features and can share weights for single-view problems.

In sub-network sampling, once a splitting module (i.e., $g_i = 1$) is selected, we take blocks before splitting from the on-sensor supernet and blocks after splitting from the on-aggregator supernet, and stitch them together using splitting module as the joint point to form a sampled network as illustrated in Figure 4.

Eventually, we need exactly one splitting module. Therefore, gate vector $g = (g_1, \ldots, g_M)$ is restricted to a one-hot vector in the resource-constrained architecture search stage. However, during supernet training, we are free to insert more splitting modules to help reduce the maximum performance loss (see Appendix D).

Figure 5. SA-NAS for multi-view SplitNets. A convolution layer (Conv-Reduce, purple trapezium) to reduce input’s channel size on the sensor side. The reduced input from $V$ views will be concatenated together (View-Fuse, green trapezium) on the aggregator. In this example, 3-A and 3-S are learning features for local view and mixed views respectively, thus their weights cannot be shared.

3.1.2 SA-NAS for Multi-View SplitNets

With $V$ sensors in the system, $V$ images are captured from different perspectives. Each sensor processes its captured image with the on-sen. model. Afterwards, a fusion module on the aggregator will fuse the compressed $V$ features from sensors and the on-agg. model further transforms the fused features to the final result.

Similar to searching the position of splitting module (i.e., Conv-Reduce and Conv-Recover) for single-view tasks, we
now use SA-NAS to search the position of splitting module (i.e., Conv-Reduce and View-Fuse) for multi-view tasks. Compared with single-view case, we keep the Conv-Reduce layers in the on-sen. supernet, but replace the Conv-Recover layers in the on-agg. supernet with View-Fuse layers to aggregate features across multiple views as illustrated in Figure 5.

Different from single-view case, no weight sharing can be performed between the on-sen. and on-agg. supernets because on-agg. network learns to process fused representation which contains mixed information from V views, while on-sen. supernet processes only local information. In other words, blocks on-sen. and on-agg. have radically different functionalities, and it is sub-optimal to enforce a shared block to play these two roles simultaneously.

### 3.1.3 Architecture of Splitting Module

#### Single-View: Reduce and Recover

Inspired by [4, 11, 36, 45, 46], we adopt a straightforward implementation for the splitting module in SA-NAS. As illustrated in Figure 3, we use two convolution layers to compress features: \( \phi: \mathbb{R}^{c \times s \times s} \rightarrow \mathbb{R}^{d \times s \times s} \) and recover features: \( \phi': \mathbb{R}^{d \times s \times s} \rightarrow \mathbb{R}^{c \times s \times s} \) on channel dimension, where \( d \) can be searched by SA-NAS. In SA-NAS, \( \phi \) and \( \phi' \) are always selected simultaneously.

#### Multi-View: Reduce and Fuse

Prior work has proposed intricate fusion architecture like recurrent and graph neural networks targeting different applications [16, 20, 34, 49].

Similar to the single-view, multi-view splitting module uses a convolution layer \( \phi: \mathbb{R}^{c \times s \times s} \rightarrow \mathbb{R}^{d \times s \times s} \) for communication saving, and then fuses these reduced features from \( V \) views together with the View-Fuse layer. Without loss of generality, we use a highly simplified fusion operation - concatenation (See Appendix B). View-Fuse will concatenate \( V \) views together along the channel dimension \( \text{concat}(V \times \mathbb{R}^{d \times s \times s}) \rightarrow \mathbb{R}^{V \times d \times s \times s} \).

Despite of the extreme simplicity of fusion architecture, we empirically show that searching the splitting (i.e., fusion) position jointly with network architecture using SA-NAS achieves considerable accuracy improvement, compared with SOTA approaches with dedicated fusion layers (see Section 4).

#### Split-aware Weights Initialization

As is also observed by [36], adding compression / fusion modules into supernet training process degrades the training stability and yields suboptimal training performance. We find that the issue is caused by the gradient of the compression module. In practice, the gradients of the compression layer are \( >10 \times \) bigger than regular layers, starting from initialization.

Take Kaiming initialization as an example [21]. Its Fan-In mode is to ensure that the output of each layer has zero mean and unit variance, so the output magnitude will not explode in the forward pass. Similarly, the backward pass of a convolution layer is also a convolution but with the transposed weight matrix \( W^T \), and applying the same idea will lead to Fan-Out mode.

Either Fan In or Fan Out works well for standard neural architectures because most of layers’ input channel size \( c_{in} \) and output channel size \( c_{out} \) are similar (see Table 3). However, for Conv-Reduce and Conv-Recover, \( c_{in} \) and \( c_{out} \) are very different, thus Kaiming and Xavier cannot reconcile both forward and backward at the same time.

In order to mitigate this conflict, we use a new initialization strategy by replacing \( c_{in} \) or \( c_{out} \) with their geometric average \( \sqrt{c_{in} \cdot c_{out}} \). Using geometric average allows us to find a better compromise between forward and backward passes and significantly improve training stability and final accuracy as shown in Appendix F.

### 3.2. Resource-constrained searching (Stage 2)

After the supernet training stage, all candidate networks are fully optimized. The next step is to search the optimal network which satisfies system constraints. Since two-stage NAS decouples the training and searching, the system constraints are fully configurable. In addition, one does not have to redo the costly supernet training when the system constraints change.

Since the searching space of supernet is extremely huge, evolution algorithm is adopted to accelerate the searching process [5, 7, 18, 55, 60]. Specifically, we mutate and crossover pairs of current generation’s networks to generate their children networks. All children networks are evaluated and selected according to their hardware and task performance. The selected children networks will be used to populate candidate networks for the next generation. After several generations, optimal networks can be found.

We consider two kinds of hardware constraints simultaneously. 1) Soft constraints, such as overall latency under the same task accuracy: Children networks with top-\( k \) smallest latency’s will be selected. 2) Hard constraints, such as peak memory usage: Regardless of accuracy and performance, only children networks that satisfy all hard constraints can be selected to the next generation.

#### System Hardware Modeling

Given a sampled network, its hardware performance is modeled through a hardware simulator. In this work, we use a hardware simulator customized for a realistic HMD system [40, 61] with smart sen-

| Initialization Method | Weights’ Variance | Average |
|-----------------------|--------------------|---------|
| Kaiming Fan-In or -Out [21] | \( \frac{2^{k \cdot c_{in}}}{c_{in}^{2}} \) or \( \frac{2^{k \cdot c_{out}}}{c_{out}^{2}} \) | Arithmetic |
| Xavier [17] | \( \frac{2^{k \cdot (c_{in} + c_{out})/2}}{c_{in} \cdot c_{out}} \) | Geometric |

Table 3. Comparison of different initialization approaches. Equations in this table assume the activation function as ReLU.
Figure 6. Hardware model for the target system. On-sensor compute uses a less advanced technology node (16nm) with limited on-chip memory. Smart sensors are connected to an aggregator using a shared 1.2 Gb/s bus. Aggregator uses advanced process (7nm) and has faster compute with larger memory.

The communication between the sensors and the aggregator is modeled with a high-performance shared bus for HMD computing. To make the evaluation more realistic to real-time applications [32], we assume each of the four sensors is allocated 25% bandwidth of the shared bus. We also assume that all models’ weights and activation are quantized to 8-bit without accuracy degradation. (And literature has shown this assumption is reasonable. [12,15, 30, 56]) If models have larger bitwidth, SplitNets will benefit more because the communication saving from feature compression is more pronounced.

We compare SA-NAS approach with four baseline model split methods:
• “All-on-sen.”: All computation happens on the sensor. The communication overhead is negligible but the peak memory often exceeds sensor’s memory capacity. In our experiments, we discover that all existing models, even lightweight ones, require >5MBs peak memory, which are too large for on-sensor deployment.
• “All-on-agg.”: Transmit raw image to aggregator and perform all computation on aggregator. Communication becomes the bottleneck, and the system is not scalable with increasing number of sensors.
• “Neurosurgeon [28]”: a heuristic method which profiles each layer and exhaustively searches every possible splitting position to find the optimal partition. Neurosurgeon performs the model split for many networks and finds that the optimal split often falls in the beginning or the last layer, which degenerates to “All-on-agg.” or “All-on-sen.”. We apply “Neurosurgeon [28]” on recent efficient architectures and observe similar results. This validates the necessity of joint designing network and the split.
• “Head Distill [23]”: a method that first manually determines the position of splitting point and then replaces the head part with a smaller networks through knowledge distillation. This approach is able to significantly

4. Results

We validate that SplitNets can satisfy all system hardware constraints and find an optimized model with competitive accuracy and the best system performance compared with SOTA models crafted by hands or searched by standard NAS methods. Specifically, we evaluate our SplitNets for single-view system on ImageNet classification (Section 4.1) and SplitNets for multi-view system on 3D classification (Section 4.2).
reduce the communication cost compared with “All-on-agg.” and satisfy the hardware constraints. However, “Head Distill [23]” introduces 1.5%-3.6% accuracy drop even for parameter-redundant architectures like ResNet-152, DenseNet-169, and Inception-v3. Also, exhaustively searching the splitting point and hyper-parameters of the on-sen. network in [23] also requires redoing the costly training, which is inefficient for deployment.

In the bottom of Table 4, we report a series of networks searched by SA-NAS. Under different accuracy targets, SplitNets consistently outperforms prior methods (“Neurosurgeon” [28] and “Head Distill” [23]) on overall latency while satisfying all hardware constraints. In addition, SplitNets are able to balance the workload of sensor and aggregator and make trade-off among communication, accuracy and computation when system hardware configuration changes (See Appendix H).

### 4.2 Multi-view Task: 3D Classification

We also evaluate SplitNets on a multi-view system with ModelNet 3D Warehouse classification dataset that consists of 3D CAD models from 40 categories. CAD models in this benchmark come from 3D Warehouse [1]. Each 3D object is captured from 12 different views by 12 sensors, where each view is a 2D image of size $3 \times 224 \times 224$.

#### Searching Fusion Module’s Position Benefits Accuracy

In Table 5, we compare the task performance of SplitNets models against existing models with various back-
bones [22, 24, 47, 52, 53] with dedicated fusion modules [31]. We assume all models are trained from scratch. Prior methods do not have the flexibility in adjusting the position of fusion module, and typically fuse features after the last convolution. SplitNets are able to explore all possible fusion positions (Figure 9, Right) and learn to pick the optimal split. Despite the simplicity of fusion used, SplitNets significantly outperforms prior methods with fewer parameters and less computation.

**Superior Overall Latency with SOTA Accuracy** We further demonstrate the benefits of SplitNets for distributed computing on a 12-sensor system in Table 6. Compared to single-view system, multi-view system has heavier communication burden because multiple sensors have to share a single serial communication bus. As a result, the communication advantage of SplitNets is more pronounced. We visualize networks found by SplitNets in Figure 9. The position of fusion module keeps changing given different latency and accuracy targets. Interestingly, the best accuracy is not strictly correlated with the largest model which further validates the necessity of searching the best model instead of increasing model size.

### Table 6. Overall latency comparison on a multi-sensor system with a 3D classification task. Since 12 sensors are assumed in this task, each sensor is allocated 12.5MB/s bandwidth. Green (or red) numbers mean the on-sensor memory consumption is less (or more) than a sensor’s memory constraint.

| Split | Backbone | Top-1 | #Params | On-each-sensor | Comm. | On-aggregator | Overall |
|-------|----------|-------|---------|----------------|-------|--------------|---------|
|       |          |       |         | #OPs | Latency | #OPs | Size | Latency | #OPs | Size | Latency |
| Sensor × 12 | VGG-11 | 88.7 | 0 | 0 | 0 | 0 | 3 | 224² | 12.0ms | 132 M | 90.0G | 73.2ms | 84.9ms |
| Aggregator × 1 | ResNet-18 | 89.1 | 0 | 0 | 0 | 0 | 3 | 224² | 12.0ms | 11.7M | 21.8G | 17.7ms | 29.7ms |
| | MobileNet-v3 | 86.0 | 0 | 0 | 0 | 0 | 3 | 224² | 12.0ms | 2.55M | 661M | 0.58ms | 12.6ms |
| | MNASNet-0.5 | 91.4 | 0 | 0 | 0 | 0 | 3 | 224² | 12.0ms | 2.22M | 1.24G | 1.01ms | 13.0ms |
| | EfficientNet-B0 | 92.1 | 0 | 0 | 0 | 0 | 3 | 224² | 12.0ms | 5.29M | 4.62G | 3.76ms | 15.8ms |
| BW_{comm} = 12.5 MB/s | VGG-11 | 88.7 | 9.22M | 7.49G | 12.4MB | 59.9ms | 512 | 7² | 2.00ms | 123M | 123M | 0.10ms | 62.1ms |
| | ResNet-18 | 89.1 | 11.2M | 1.81G | 12.8MB | 14.5ms | 512 | 7² | 2.00ms | 0.51M | 0.51M | 0.40ms | 16.5ms |
| | MobileNet-v3 | 86.0 | 0.93M | 54.9M | 1.38MB | 0.44ms | 576 | 7² | 2.26ms | 1.62M | 1.62M | 1.29ms | 2.71ms |
| | MNASNet-0.5 | 91.4 | 0.94M | 103M | 1.54MB | 0.83ms | 1280 | 7² | 5.03ms | 1.28M | 1.28M | 1.03ms | 5.83ms |
| | EfficientNet-B0 | 92.1 | 4.01M | 385M | 6.42MB | 3.08ms | 1280 | 7² | 5.03ms | 1.28M | 1.28M | 1.03ms | 8.03ms |
| | SplitNets | A³ | 92.3 | 0.14M | 41.8M | 0.06MB | 0.33ms | 16 | 6² | 0.05ms | 1.25M | 17.4M | 0.01ms | 0.39ms |
| | B³ | 93.0 | 0.17M | 58.7M | 0.06MB | 0.47ms | 6 | 12² | 0.07ms | 2.27M | 55.4M | 0.06ms | 0.59ms |
| | C³ | 93.8 | 0.18M | 147M | 0.11MB | 1.17ms | 6 | 14² | 0.09ms | 2.15M | 73.3M | 0.07ms | 1.34ms |

Figure 9. **Left:** Accuracy (stars and the right y-axis) and overall latency breakdown (the stacking bar plot and the left y-axis) of searched networks by SplitNets. Each searched model’s overall latency is represented by a vertical stacked bar. From bottom to top of each bar, the length’s of three colors represent T_vnet, T_comm, T_agg respectively. **Right:** Visualization of three searched networks by SplitNets. The length of vertical lines represents the output channel size of a convolution layer. Dash means that the stride size is two. The green and purple trapezoids are **Conv-Reduce** and **View-Fuse.** Among the three networks, the first and third networks conduct fusion either too ‘late’ or too ‘early’, leading sub-optimal task and system performance.

5. **Conclusion and Discussion**

In summary, we introduce SplitNets with SA-NAS for efficient partition and inference of ML models on-device with distributed smart sensors. The proposed SA-NAS approach enables end-to-end model search with flexible positioning of a splitting module for single-view and multi-view problems. The resource-constrained searching successfully identifies model architectures that reduce overall system latency, satisfy hardware constraints, while maintaining high accuracy. Empirical results on ImageNet (single-view) and 3D Classification (multi-view) show that our approach can discover SOTA network architectures and model splitting solutions for distributed computing systems on AR/VR glasses.

**Limitations and future work.** The hardware model we adopt leverages an analytical model combing different hardware modalities, which are calibrated separately. Full system simulation with cycle accurate hardware models could be deployed for more precise latency evaluation.
References

[1] 3d warehouse. https://3dwarehouse.sketchup.com/, 7, 12

[2] Oculus quest 2: Our most advanced new all-in-one vr headset. 14

[3] Micheal Ambrash. Creating the future: Augmented reality, the next human-machine interface, pages 1–4, 2021, 1

[4] Juliano S Assine, Eduardo Valle, et al. Single-training collaborative object detectors adaptive to bandwidth and computation. arXiv preprint arXiv:2105.00591, 2021, 1, 3, 5

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 12239–12248, 2021, 3, 5

[6] Hyomin Choi and Ivan V Bajić. Deep feature compression for collaborative object detection. In 2018 25th IEEE International Conference on Image Processing (ICIP), pages 3743–3747. IEEE, 2018, 3

[7] Robert A Cohen, Hyomin Choi, and Ivan V Bajić. Lightweight compression of neural network feature tensors for collaborative intelligence. In 2020 IEEE International Conference on Multimedia and Expo (ICME), pages 1–6. IEEE, 2020, 3

[8] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using neural acquisition function. CoRR, abs/2006.02049, 2020, 3, 6, 12

[9] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. IEEE Transactions on Mobile Computing, 2019, 1, 3

[10] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. Bottlenet: A deep learning architecture for intelligent mobile cloud computing services. In 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pages 1–6. IEEE, 2019, 1, 3, 5

[11] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. ICLR, 2020, 6

[12] Abrash et al. Creating the future: Augmented reality, the next human-machine interface. IEEE IEDM, 2021, 1

[13] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. ICLR, 2021, 6

[14] Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. Gvcn: Group-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 264–272, 2018, 5, 12

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256. JMLR Workshop and Conference Proceedings, 2010, 5, 14

[16] Fang Zhang, Ziheng Jiang, Yu Chen, Bo Liu, and Yizhe Wang. Network Pruning via Learnable Vector Quantization. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2017, 1, 2, 3, 6, 7

[17] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Min Wang. Learning Representations with quantization noise for extreme model compression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016, 3, 6, 8, 14

[18] 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, 5, 14

[19] 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, 3, 6, 8, 14

[20] Geoff Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015, 6, 7, 12

[21] 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, 8

[22] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. arXiv preprint arXiv:1608.06993, 2016, 1, 3, 4, 5

[23] 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 conference on computer vision and pattern recognition, pages 770–778, 2016, 3, 6, 8, 14

[24] 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, 8

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017, 6

[26] Mikolaj Jankowski, Deniz Gündüz, and Krystian Mikolajczyk. Joint device-edge inference over wireless links with pruning. In 2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), pages 1–5. IEEE, 2020, 3

[27] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. Computation offloading for machine learning web apps in the edge server environment. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 1492–1499. IEEE, 2018, 1, 3

[28] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2017, 1, 2, 3, 6, 7
and Xiaobing Feng. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In International Conference on Artificial Neural Networks, pages 402–411. Springer, 2018.

30] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. ICLR, 2019.

31] Zhaosun Li, Hongren Wang, and Jinxing Li. Auto-mvcnn: Neural architecture search for multi-view 3d shape recognition. arXiv preprint arXiv:2012.05493, 2020.

32] Chiao Liu, Andrew Berkovich, Qing Chao, Song Chen, Ziyun Li, Hans Reyserhove, Syed Shabik Sarwar, and Tsung-Hsien Tsai. Intelligent vision sensors for ar/vr. In Imaging Systems and Applications, pages ITu5G–1. Optical Society of America, 2020.

33] Chiao Liu, Andrew Berkovich, Song Chen, Hans Reyserhove, Syed Shabik Sarwar, and Tsung-Hsien Tsai. Intelligent vision systems—bringing human-machine interface to ar/vr. In 2019 IEEE International Electron Devices Meeting (IEDM), pages 10–5. IEEE, 2019.

34] Chao Ma, Yulan Guo, Jungang Yang, and Wei An. Learning multi-view representation with lstm for 3-d shape recognition and retrieval. IEEE Transactions on Multimedia, 21(5):1169–1182, 2018.

35] Yoshito Matusbara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled split deep neural networks for edge-assisted real-time systems. In Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges, pages 21–26, 2019.

36] Yoshito Matusbara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. IEEE Access, 8:212177–212193, 2020.

37] Yoshito Matusbara, Marco Levorato, and Francesco Restuccia. Split computing and early exiting for deep learning applications: Survey and research challenges. arXiv preprint arXiv:2103.04505, 2021.

38] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. IEEE transactions on robotics, 31(5):1147–1163, 2015.

39] Daniele Jahier Pagliari, Roberta Chiaro, Enrico Macii, and Massimo Poncino. Crime: Input-dependent collaborative inference for recurrent neural networks. IEEE Transactions on Computers, 2020.

40] Guangfan Qin and Guoliang Qin. Virtual reality video image classification based on texture features. Complexity, 2021.

41] 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.

42] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In Proceedings of the aacai conference on artificial intelligence, volume 33, pages 4780–4789, 2019.

43] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In International Conference on Machine Learning, pages 2902–2911. PMLR, 2017.

44] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, pages 4510–4520. Computer Vision Foundation / IEEE Computer Society, 2018.

45] Marion Sbai, Muhamad Risqi U Saputra, Niki Trigoni, and Andrew Markham. Cut, distil and encode (cde): Split cloud-edge deep inference. In 2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pages 1–9. IEEE, 2021.

46] Jiawei Shao and Jun Zhang. Bottleneck++: An end-to-end approach for feature compression in device-edge co-inference systems. In 2020 IEEE International Conference on Communications Workshops (ICC Workshops), pages 1–6. IEEE, 2020.

47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In International Conference on Learning Representations, 2015.

48] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision, pages 945–953, 2015.

49] Jong-Chyi Su, Matheus Gadelha, Rui Wang, and Subhransu Maji. A deeper look at 3d shape classifiers. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pages 0–0, 2018.

50] Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In International Conference on Machine Learning, pages 4771–4780. PMLR, 2018.

51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

52] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pages 2820–2828. Computer Vision Foundation / IEEE, 2019.

53] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kama- lika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 6105–6114. PMLR, 2019.

54] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra.
Attentivenas: Improving neural architecture search via attentive sampling. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021.

[55] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. Hat: Hardware-aware transformers for efficient natural language processing. In Annual Conference of the Association for Computational Linguistics, 2020.

[56] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[57] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1912–1920, 2015.

[58] Lingxi Xie and Alan Yuille. Genetic cnn. In Proceedings of the IEEE international conference on computer vision, pages 1379–1388, 2017.

[59] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems, pages 476–488, 2020.

[60] 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.

[61] Lim Jia Zheng, James Mountstevens, and Jason Teo. Four-class emotion classification in virtual reality using pupillometry. Journal of Big Data, 7(1):1–9, 2020.

[62] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2423–2432, 2018.

[63] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.

[64] 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.