Dual Dynamic Inference: Enabling Multi-Grained and More Controllable Inference Adaptivity in Complex Deep Networks

Anonymous ICCV submission

Paper ID 4989

Abstract

State-of-the-art convolutional neural networks (CNNs) yield record-breaking predictive performance, yet at the cost of high-energy-consumption inference, that prohibits their widely deployments in resource-constrained Internet of Things (IoT) applications. We propose a dual dynamic inference (DDI) framework that highlights the following aspects: 1) we integrate both input-dependent and resource-dependent dynamic inference mechanisms under a unified framework in order to fit the varying IoT resource requirements in practice; 2) we propose a flexible multi-grained learning to skip (MGL2S) approach for input-dependent inference which allows simultaneous layer-wise and channel-wise skipping; 3) we extend DDI to complex CNN backbones such as DenseNet and show that DDI can be applied towards optimizing any specific resource goal including inference latency or energy cost. Extensive experiments are discussed to demonstrate the superior inference accuracy-resource trade-off achieved by DDI, as well as the flexibility to control such trade-offs compared to existing peer methods. Specifically, DDI can achieve up to 4× computational savings with the same or even higher accuracy as compared to the most competitive baseline SkipNet.

1. Introduction

The increasing penetration of intelligent visual sensors has clearly revolutionized the way Internet of Things (IoT) works. For visual data analytics, we witness the record-breaking predictive performance achieved by convolutional neural networks (CNNs) [1, 2, 3]. To this end, there has been a growing demand to bring CNN-powered intelligence into IoT devices, ranging from drones, to security surveillance, to self-driving cars, to wearables and many more, for enabling intelligent "Internet-of-Eyes". This demand is in line with the recent surge of edge computing where raw data are processed *locally* in edge devices using their embedded inference algorithms [4]. Such local processing avoids the necessity of transferring data back and forth between data centers and edge devices, reducing communication cost and latency, and enhancing privacy, compared to traditional cloud computing.

Despite the promise of CNN-powered "Internet-of-Eyes", deploying CNNs into resource-constrained IoT devices is a non-trivial task because IoT devices, such as smart phones and wearables, have limited energy, computation and storage resources. Meanwhile, the excellent performance of CNN algorithms comes at a cost of very high complexity. Some of these algorithms require around one billion multiply-accumulate (MAC) operations [5] during the inference. This mismatch between the limited resources of IoT devices and the high complexity of CNNs is only getting worse because the network architectures are getting more complex as they are designed to solve harder and larger-scale tasks [6]. To close the gap between the stringently constrained resources of IoT devices and the increasingly growing complexity of CNNs, there is a pressing need to develop innovative techniques that can achieve orders of magnitude savings in CNN inference resources.

For more resource-efficient implementations, CNNs are mostly compressed before being deployed, thus are "static" and unable to adjust their own complexity at inference. As [7, 8, 9] pointed out, the continuous improvements in accuracy, while significant, are small relative to the growth in model complexity. This implies that 1) computationally intensive models may only be necessary to classify a handful of "hard tail" examples correctly, 2) computationally intensive models are wasteful for many simple and "canonical" examples, and 3) IoT applications often have dynamic time or energy constraints over time, due to time-varying system requirements or resource allocations. Ideally, the deployed CNN should adaptively and automatically use "smaller" networks when test images are easy to recognize or the computational resources are limited, and only perform full inference when necessary.

Lately, a handful of works have considered the problem of adaptively controlling the amount of computations for dynamic inference, by either enabling early prediction from intermediate layers, or dynamically bypassing unnecessary intermediate layers and only executing sub-network infer-

ences [10, 11, 12, 13, 7]. However, there seem to be no effort to unify the two directions (skipping and early exiting). We argue that the integration of both is not only beneficial, but even necessary, to fit CNNs for practical IoT deployments. Moreover, current dynamic layer-skipping methods only allow a "coarse-grained" choice to execute each layer or not, while the potential power of finer-grained dynamic selections over channels or filter in a layer has not been jointly considered. Last but not least, dynamic inference has so far only been explored in simple chain-like backbones such as ResNet [9]. While more complicated connectivity [14] or tree-like topology [15] has proven to improve accuracy much further, it remains unclear how dynamic inference could benefit their inference efficiency.

This paper makes multi-fold efforts to address the above unsolved challenges. We propose a novel *dual dynamic inference* (**DDI**) framework, that is motivated to address the practical IoT needs for resource-efficient CNN inference. Our main contributions are as follows:

- We formulate two dynamic inference mechanisms, i.e., **input-dependent** and **resource-dependent**, and for the first time unify them in one framework. The two mechanisms in DDI together ensure boosting and controlling the energy efficiency, by both removing unnecessary costs for easy samples, and halting inference when hard resource constraints are enforced.
- For input-dependent dynamic inference, DDI goes beyond the existing layer-skipping scheme, and incorporates a novel multi-grained learning to skip (**MGL2S**) approach. Specifically, MGL2S simultaneously allows for layer-wise and channel-wise skipping, enabling superior flexibility in striking a more favorable accuracy-resource balance.
- Beyond ResNet where DDI can be straightforwardly integrated, we demonstrate how DDI could be readily applied to more complicated backbones such as DenseNet, which we observe further gains. Furthermore, DDI could be optimized with any specific resource goal, such as inference latency or energy cost.

Extensive experiments on CIFAR 10 and ImageNet datasets demonstrate the superior performance (in terms of accuracy-resource trade-off) and flexibility of DDI, over existing dynamic inference or early existing methods.

2. Related Work

Model Compression. Model compression has been extensively studied for reducing model sizes and speeding up inference. Early works [16, 17] reduce the number of parameters by element-wise pruning unimportant weights. More structured pruning was exploited by enforcing group sparsity, such as filter or channel pruning [18, 19, 20, 21, 22, 23, 24, 25]. [26] first proposed multi-grained pruning by grouping weights into structured groups with each employing a Lasso regularization. [27] proposed to stack element-wise pruning on top of the filter-wise pruned model. Lately, [28] proposed to train a multi-grained pruned network by introducing a multi-task objective. A comprehensive review of model compression can be found in [29].

Dynamic Inference. Model compression presents "static" solutions for improving inference efficiency, i.e., the compressed models cannot adaptively adjust their complexity at inference. In contrast, the rising direction of dynamic inference reveals a different option to execute partial inference, conditioned on input complicacy or resource constraints. Our work is deeply rooted in this field.

Dynamic Layer Skipping. Many dynamic inference methods [9, 30, 31] propose to selectively execute subsets of layers in the network conditioned on each input, framed as sequential decision making. Most of them used gating networks to skip within chain-like, ResNet-style models [32]. SkipNet [9] introduced a hybrid learning algorithm which combines supervised learning with reinforcement learning to learn the layer-wise skipping policy based on the input, enabling greater computational savings and supporting deep architectures. BlockDrop [30] trained one global policy network to skip residual blocks.

Channel Selection or Pruning. The smallest "skippable" unit in the above methods is a residual block. Hence, the above layer skipping methods can only be applied to the networks with residual skips. In comparison, many input-adaptive filter pruning or attention works could also be viewed as finer-grained channel skipping ideas. [33] modeled channel skipping as a Markov decision process, and used RNN gating networks to adaptively prune convolutional layer channels. GaterNet [34] trained a separate network to calculate the routing policy. The slimmable neural network [35] was recently proposed to train networks with varying channel widths while sharing parameters. [36] proposed an architecture that contains distinct components each of which computes features for similar classes, and executed only a small number of components for each image.

Early Exiting. To meet the stringent resource constraints, a few prior works introduced "early exit" into CNN inference. BranchyNet [37] augmented CNNs with additional branch classifiers for forcing a large portion of inputs to exit at these branches in order to meet the resource demands. In a similar flavor,

3. The Proposed Framework

In IoT applications, it is apparent that one always desires to save resources whenever possible, without incurring considerable inference performance loss: that is considered as a "soft" constraint for efficient inference. Meanwhile, due to system-level scheduling and coordination, the edge devices often have to perform "approximate computing" [38] in order to output the best possible result with a stringent and potentially time-varying resource limit (even that re-

sult considerably degrades compared to the full inference performance): that could be in contrast viewed as a "hard" constraint for efficient inference.

The practical need in IoT applications has motivated us to develop and integrate two different adaptive inference schemes: 1) **input-dependent** dynamic inference: the model will execute only a small subset of computations (e.g., a simpler submodel) for the inference of simple inputs, and more computations will be activated only for harder inputs as needed; 2) **resource-dependent** dynamic inference: regardless of specific input samples, the model has to terminate its inference and output a good prediction, within certain resource limits that may potentially vary over time. We hereby propose a unified *Dual Dynamic Inference* (**DDI**) framework to embed the following two capabilities into one network:

- **Input-Adaptive Dynamic Inference (IADI)**: the network learns to dynamically choose which subset of computations to execute during inference so as to best reduce total inference comp/energy cost with minimal degradation of the prediction accuracy. Then a multi-grained skipping policy will be learned together with the network training.
- **Resource-Adaptive Dynamic Inference (RADI)**: for learning under hard resource constraints (such constraints can be varied over time), a deep network could admit multiple early exits in addition to the final output, to enable "anytime classification", where its prediction for a test example is progressively updated, facilitating the output of a prediction at any time.

To our best knowledge, DDI represents the first effort to unify the above two mechanisms in one framework. The two mechanisms together ensure boosting and controlling the comp/energy efficiency, by both saving unnecessary costs, and halting inference when there are hard constraints. DDI could be optimized for different specific forms of resources, such as computational latency or energy cost.

3.1. Input-Adaptive Dynamic Inference

3.1.1 MGL2S for ResNet

Input-Adaptive Dynamic Inference (IADI) will selectively execute a subset of inference computation based on the input complexity. A baseline for IADI would be the dynamic layer skipping method as described in [9]. However, that only chooses to execute a layer or not. In comparison, enabling finer-grained options, such as whether or not executing a channel or filter, would be more flexible and potentially yield better savings. It would also lead to a vastly larger routing decision space which makes it non-trivial to implement.

We propose **MGL2S** for both finer-grained and efficient implementation of IADI. MGL2S allows for skipping both layers and channels in a CNN inference, and performs so in a *coarse-to-fine* fashion. Overall, it first examines whether a layer shall be entirely skipped; and if not, it will consider

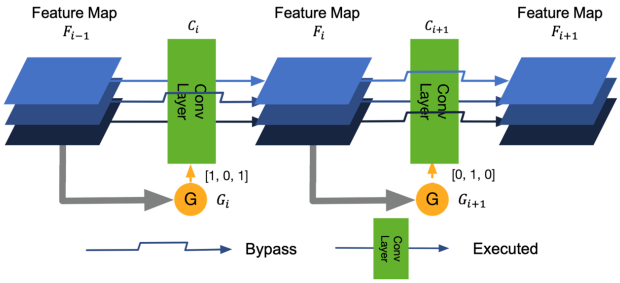


Figure 1: Illustration of channel skipping for ResNet.

skipping part of channels in that layer. The skipping policies are *jointly learned* by compact supervised gating networks (rather than as two sequential steps) together with the base network.

Next, we introduce how to incorporate MGL2S into ResNet inference, which has been the most popular testbed for dynamic inference [9, 30] due to its skipping connection and chain-like simple structure. For the i-th layer, we let $F_i \in \mathbb{R}^{s\times s\times k}$ denote its output feature map and therefore F_{i-1} as its input, where k denotes the channel number of the i-th layer. C_i denotes the convolutional filtering operation in that layer. We assume two gating networks: G_i^L for layer skipping, and G_i^C for channel skipping. The layer skipping during inference could be formulated as:

$$F_i = G_i^L(F_{i-1})C_i(F_{i-1}) + (1 - G_i^L(F_{i-1}))F_{i-1} \quad (1)$$

Note that $G_i^L(F_{i-1})$ outputs a scalar $\in \{0, 1\}$: 0 denotes skipping the i-th layer computation C_i and let F_{i-1} directly pass on to F_i. That implicitly requires F_{i-1} and F_i to have the same dimension, which is another reason why ResNet has been preferred. Similarly, channel skipping can also be expressed as (also depicted in Fig. 1):

$$F_i = G_i^C(F_{i-1})C_i(F_{i-1}) + (1 - G_i^C(F_{i-1}))F_{i-1} \quad (2)$$

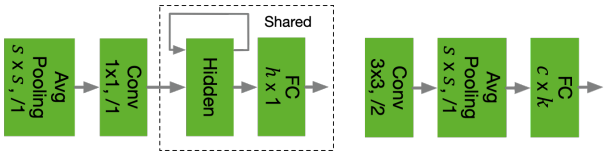
However, as a critical difference from layer skipping, $G_i^C(F_{i-1})$ outputs a length-k vector $\{0, 1\}^k$, where a zero value denotes that corresponding channel (indexed from 1 to k) should be skipped. MGL2S is defined as:

$$\begin{aligned} F_i = {} & G_i^L(F_{i-1})G_i^C(F_{i-1})C_i(F_{i-1}) \\ & +(1 - G_i^L(F_{i-1})G_i^C(F_{i-1}))F_{i-1} \end{aligned} \quad (3)$$

In practice, to reduce the computation overhead, we first compute the G_i^L output, and if it is zero, we do not compute G^C since all channels are by default skipped.

3.1.2 Heterogeneous Gating Design in MLG2S

Design of G^L: As discussed in [9], recurrent neural networks (RNN) can serve as a gating function and find routing for all layers as a sequential decision-making process. It is computationally efficient due to weight reuses, and can better capture the conditional relevance between different



(a) RNN Gate for G^L (b) CNN Gate for G^C

Figure 2: The two gating designs in MLG2S. s, c, k, h depend on different backbones and will be specified in Sec. 4. "/1" means a stride of 1.

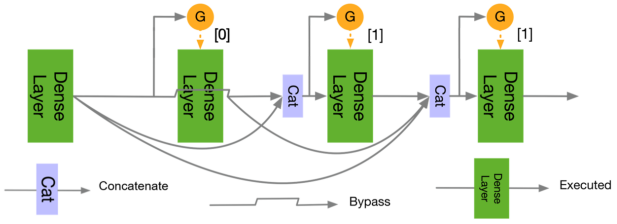


Figure 3: Illustration of layer skipping for DenseNet.

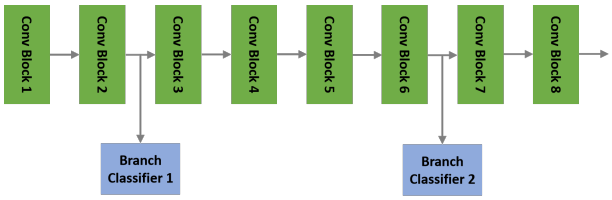
layers. We adopt this convention and implement G^L as an Long Short Term Memory (LSTM) network, as depicted in Fig. 2(a). At each LSTM stage, we project its output to a scalar between [0, 1] using a Sigmoid function, and then quantize it to either 0 or 1.

Design of G^C: When skipping channels, the inter-channel relevance is usually more significant a consideration than cross-layer correlations. Moreover, different layers normally have different output channel numbers k, making a recurrent design difficult. Motivated by the two observations, we turn to design a CNN gating function G^C for each layer. Each CNN gate is associated with one convolutional layer in the base model. The CNN gate structure is depicted in Fig. 2(b). Its output is a k-dimensional vector (k is the output channel number of the current layer), that is fed into a Sigmoid function to be element-wise rescaled and quantized to 0 or 1.

To calculate the final computational savings, we need to take the overhead of gating networks into account. Based on the above light-weight design, the computation overhead incurred by G^L and G^C takes up 0.04% and 12.5% of the computational cost of a residual block in ResNet-34, respectively. Note that although CNN gates seem to have caused more overhead, applying it to channel skipping still brings overall resource savings as we will show in Sec. 4. We leave the more efficient design of CNN gates for future work.

3.1.3 Extending MGL2S to DenseNet

DenseNet [14] shows superior performance to ResNet, thanks to its much heavily connected intermediate layers. Meanwhile, extending MGL2S to DenseNet has not been explored in previous dynamic inference works. Compared to chain-like ResNets, the output of each layer in DenseNet is concatenated with the outputs of all preceding layers through shortcuts. That leads to an even more enlarged routing space to decide on. Moreover, the layer-wise input dimension changes throughout DenseNet also turn the (implicit) underlying assumption in ResNet layer skipping invalid, i.e., F_{i-1} and F_i always having the same dimension.



(a) Branch position.

(b) Branch classifier design

Figure 4: Illustration of: (a) branch position within one stage of ResNet: two branch classifiers are added at positions approximately 1/4 and 3/4 depth of the stage, thus resulting in one at the 2nd layer and the other at the 6th layer given a total of 8 layers in this example, and (b) branch classifier design, in RADI.

To alleviate the above challenges, we propose a modified layer skipping strategy as illustrated in Fig. 3. If the i-th layer is skipped, then the outputs of all layers preceding the i-th layer will directly bypass it and form the new input for the $(i+1)$-th layer. For channel skipping, the skipped channels will directly output all-zero feature maps of the same original dimension. The layer and channel skipping are then combined similarly as in ResNet.

Design of G^L and G^C in DenseNet: We apply the same LSTM gating network used in ResNet for implementing G^L in DenseNet. For channel skipping, we also use a CNN gating function to implement G^C. However, the channel dimension of the input for the gate will gradually increase due to feature concatenation in DenseNet. Therefore, directly applying the CNN gate in ResNet will cause unacceptable heavy computation overhead. For example, directly adopting the same G^C in Fig. 2(b) for DenseNet-100 will lead to the gating computation overhead 4 times higher than the base network. To this end, we design a light-weight CNN gate for DenseNet, consisting of a 1×1 bottleneck layer followed by a 3×3 convolution layer. This new gating design has around 11 % overhead of the original DenseNet model.

3.2. Resource-Adaptive Dynamic Inference

Resource-Adaptive Dynamic Inference (RADI) performs anytime prediction in order to meet various hard resource constraints. Specifically, following similar ideas from [37], RADI adds multiple branch classifiers to the network, and make inference decisions whenever a branch classifier gets confident prediction and/or a resource constraint is reached. Fig. 4 shows the position principle and design of branch classifiers.

Training for RADI. During training, each branch classifier L_i has the same softmax loss L as the final classifier L_o, $i = 1, ..., N$. The branches and main output are trained

jointly. The overall loss is a weighted combination:

$$L_{final} = L_o + \sum_{i=1}^{N} w_i L_i \quad (4)$$

The weights w_i, $i = 1, ..., N$ are introduced to balance between the early predictions and the main inference output. We adopt an adaptive heuristic to adjust $\{w_n\}_{n=1}^{N}$: for each mini-batch, we first compute the current losses of all L_is and L_o (averaged over the mini-batch), denoted as C_i and C_o for simplicity. We then set $w_i = C_o/C_i$ for this mini-batch to back-propagate and update weights. The intuition behind this heuristic method is that we prefer all classifiers to have "comparable" impacts on parameter updating.

Testing for RADI. During testing, we associate each branch classifier with an uncertainty threshold T measured in entropy [37]. If the prediction made by this intermediate classifier has an entropy that is below T, the current data sample will exit at this classifier, and this intermediate prediction will be used as the final result. Under a hard resource constraint, we search for optimal uncertainty thresholds for the branch classifiers, such that the model can perform within the resource limit, while maximizing the expressive power. The strategy of choosing Ts for different branch classifiers will be discussed in Sec. 4.

3.3. Training Strategy for DDI

Training DDI takes two phases. We first train MGL2S on the base network. We then add RADI to the pre-trained IADI network, and tune from end to end. The hyperparameters for training DDI will be found at supplementary. 4.1.

Training MGL2S. we use supervised learning to train layer and channel skipping policies. The dynamic skipping policies are learned by minimizing a hybrid loss consisting of prediction accuracy loss L and the resource-aware loss E. The learning goal is defined as:

$$\min_{W,G} \; L(W, G) \; + \alpha E(W, G) \quad (5)$$

where α is a weighted coefficient, and W and G denote the parameters of the base model and the gating networks, respectively. Possible options for E include FLOPs as an indicator of the run-time latency, or the hardware-aware energy loss. To stabilize training, we always start from a pre-trained base CNN with weights W, and initialize the gating network parameters G randomly using Gaussian. We first freeze W, and only train G until the skipping ratio reaches a plateau. We then unfreeze W to be jointly trained with G further, usually observing the channel/layer skipping ratios to climb higher until reaching another plateau.

4. Experiments

In this section, we present extensive evaluation results of the proposed techniques. Sec. 4.1 describes the experiment setup including the employed CNN models and datasets, and model design/training details. In Sec. 4.2, we 1) evaluate our IADI technique against state-of-the-art designs of both dynamic skipping and static compression, 2) compare IADI over the base models with various skipping strategies, and 3) study IADI when using different resource-aware losses. Sec. 4.3 summarizes DDI's performance. Finally, we discuss visualization for inputs and their corresponding skipping ratios in Sec. 4.4.

4.1. Experimental Setup

Evaluation Models, Datasets and Metrics. [1] We evaluate our proposed techniques using state-of-the-art CNN architectures including ResNet [32] and DenseNet [14] on two image classification benchmarks: CIFAR-10 and ImageNet. *CNN Models:* For CIFAR-10, we use ResNet38, ResNet74, and DenseNet100. In particular, ResNet38 and ResNet74 start with a convolutional layer followed by 18 and 36 residual blocks with each having two convolutional layers. The 18 and 36 residual blocks are divided into 3 stages uniformly. For ImageNet, we employ a standard DenseNet model DenseNet121 [14]. *Metrics:* Performance is evaluated in terms of both classification accuracy and computational/energy savings. More details on the models and datasets are provided in the Supplementary Materials.

Gating Network Design for IADI. As shown in Fig. 2(a), for *layer skipping*, we utilize an LSTM to implement the gating network [39]. For reducing the associated computation overhead, the gating network pipeline consists of 1) an average pooling layer that is designed to compress the input feature map into a $1 \times 1 \times c$ vector with c denoting the number of input channels, 2) a 1×1 convolutional layer for further feature extraction, and 3) a single layer LSTM with a hidden unit size of 10. For *channel skipping*, we employ a light-weight CNN gate, which is made of 1) a 3×3 convolutional layer with a stride of 2, 2) a global average pooling layer for compressing the feature map into a $1 \times 1 \times c$ feature, and 3) a fully connected layer of size $c \times k$ (k is the number of output channels).

Branch Position and Design for RADI. *Branch Position:* we add branch classifiers at all 3 stages of ResNet74 and DenseNet100 in order to exploit the early prediction power of the whole span of layers in the network. Specifically, within each stage, we add two branch classifiers at positions approximately 1/4 and 3/4 depth of the stage (see Fig.4), leading to a total of 6 branch classifiers. *Branch Classifier:* CNNs learn features hierarchically, i.e., extracting low-level detailed features in early layers and high-level abstract features at later layers, the latter of which is critical for correctly classifying the input. Therefore, our branch classifier design strives to compensate for the lack of high-level features for making confident classification, resulting in a gradual increase in the classifiers' complexity from the first to the last stage.

[1] The stride is set to be 1 unless otherwise specified in this subsection.

Training Details. The training parameters of CNNs on CIFAR-10 and ImageNet could be found in the supplementary. Training DDI involves two phases: train IADI and then train RADI. Train IADI: given a pre-trained CNN model, if we directly train both the gating networks and the pre-trained model together by randomly initializing the former using the Gaussian distribution, e.g., consider 50% skipping ratio, the resulting accuracy is observed to decrease drastically as compared to that of the pre-trained model. We conjecture that this is probably because the batch normalization parameters trained for the original model cannot capture the statistics of the updated feature maps due to layer/channel skipping. To resolve this issue, we propose to start with a warm-up process, during which the base network is fixed and only the gating network is trained to reach a skipping ratio of 0. After that, the base and gating networks are jointly trained using the standard stochastic gradient descent algorithm.

For handling the hybrid loss (see eq. 5) in the learning goal, we train the model iteratively using the following two steps: 1) given a specified resource-aware budget, e.g., E_{IADI}, the IADI model is trained to skip layers/channels in order to reach E_{IADI}; and 2) α is then set to 0 to ensure stable training and to fine-tune the network for restoring accuracy while minimizing the resource-aware loss. Train RADI: once IADI is trained to reach the specified learning goal, we add the branch classifiers and then train the IADI model and branch classifiers jointly as described in Sec. 3.2.

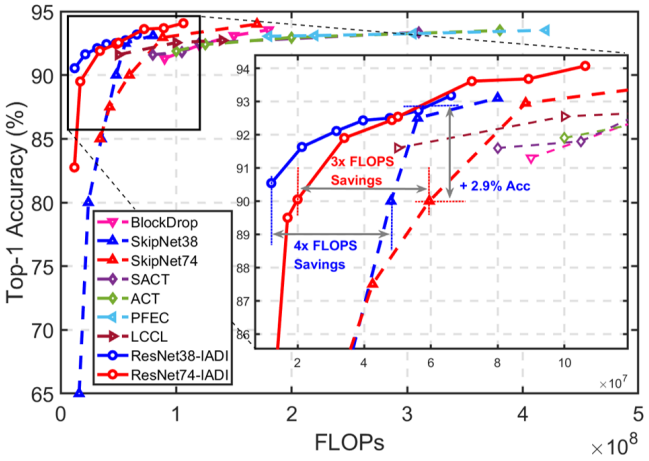


Figure 5: Comparing IADI with six state-of-the-art techniques in terms of Accuracy vs. FLOPs on CIFAR-10.

4.2. Performance of the Proposed IADI

IADI vs. State-of-the-art techniques. We compare IADI against six state-of-the-art techniques including four dynamic skipping techniques (SkipNet [9], BlockDrop [30], SACT and ACT [7]) and two static compression techniques (PFEC [40] and LCCL [41]). To be consistent with the baselines, we apply IADI on ResNet. Fig. 5 shows that the models resulted from IADI, i.e., ResNet38-IADI-CT and ResNet74-IADI-CT, outperform all state-of-the-art techniques by achieving a better accuracy given the same computational cost (i.e., FLOPS) or requiring less computational cost to achieve the same accuracy. Specifically, comparing to the most competitive baselines (SkipNet38 and SkipNet74), ResNet38-IADI-CT and ResNet74-IADI-CT can save up to **4**× and **3**× computational cost while achieving a slightly higher accuracy (90.55% vs. 90% and 90.01% vs. 90%), resepctively. Furthermore, ResNet38-IADI-CT can even achieve up to a 2.9% higher accuracy compared with SkipNet74 under the same computational cost (i.e., 6×10^8 FLOPs).

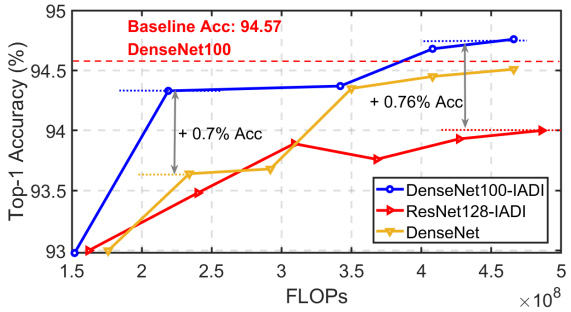
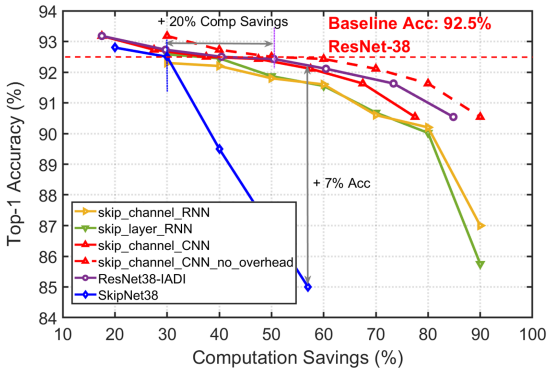


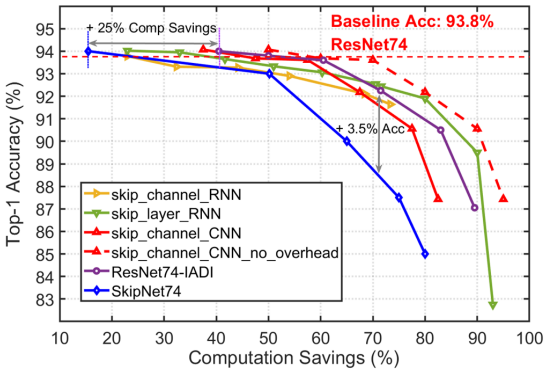
Figure 6: Comparing DenseNet100-IADI with ResNet128-IADI and original DenseNet models in terms of Accuracy vs. FLOPs on CIFAR-10.

Next, we evaluate IADI on DenseNet to show that IADI consistently achieves a better performance when being applied to a different CNN model, and a better network backbone can further boost the performance of IADI. Fig. 6 shows that when the backbone models DenseNet100 and ResNet128 have a similar computational cost (<1% difference), DenseNet100-IADI outperforms ResNet128-IADI in accuracy by a non-trivial margin (up to 0.76%) under a wide range of computational cost. To further demonstrate the superiority of IADI on DenseNet, we compare DenseNet100-IADI with the base DenseNets. We can see in Fig. 6 that DenseNet100-IADI consistently achieves a better accuracy (up to 0.7%) given the same computational cost.

Comparison with Merely Layer/Channel Skipping. We here compare IADI with various skipping strategies (including skipping layer with RNN gates, and skipping channel with RNN or CNN gates) on both ResNet and DenseNet. Fig. 7 shows the comparison on ResNet. We can see that IADI implemented using MGL2S outperforms all other skipping strategies by achieving a higher accuracy given the same FLOPS or requiring less computational cost to achieve the same accuracy. Specifically, ResNet38-IADI and ResNet74-IADI can boost the accuracy by up to 7% and 3.5%, respectively, compared with SkipNet38 and SkipNet74 under the same computational savings (57% and 71%, respectively). Furthermore, we can see that ResNet38-IADI and ResNet74-IADI will not incur an accuracy loss until up to 50% and 60%, respectively, whereas SkipNet38 and SkipNet74 start to have an obvious accuracy degradation at computational



(a) ResNet38



(b) ResNet74

Figure 7: Comparing IADI with various skipping strategies on ResNet and CIFAR-10.

savings of 20% and 15%, respectively.

Fig. 8 shows the comparison on DenseNet. Similarly, we can see that DenseNet100-IADI outperforms (up to 1.2%) both layer and channel skipping over a wide range of computational cost, showing the consistent superiority of IADI. In addition, we can observe in both Figs. 7 and 8 that channel skipping with CNN gates in general achieve a higher accuracy (up to 1%) compared with that of using RNN gates, justifying our reasoning in Sec. 3.1.2. The promising performance of channel skipping when excluding the gating overhead and the relatively large overhead of CNN gates (11 % vs. 0.04% when using RNN gates) suggests that there is a potential to further reduce the overhead of CNN gates using compression techniques such as quantization.

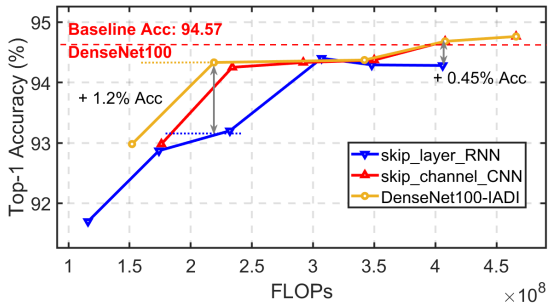


Figure 8: Comparing IADI with various skipping strategies on DenseNet and CIFAR-10.

IADI with Different Resource-aware Losses. IADI can adapt to the most critical resource constraint for various applications by employing different resource-aware losses (i.e., E in eq. 5). For example, it has been shown that computational cost might not align with energy consumption because CNNs' energy cost is mostly dominated by data movement and memory accesses [42]. As such, for energy-limited platforms such as battery-powered wearable devices, energy instead of computational cost, i.e., FLOPs, should be used as the resource-aware loss in eq. 5 for making use of IADI's flexibility in adapting to various resource constraints. Fig.9 shows that ResNet74-IADI-Energy consistently leads to a larger energy savings compared to Resnet74-IADI-CT, where the former adopts energy as the resource-aware loss and the latter uses computational cost.

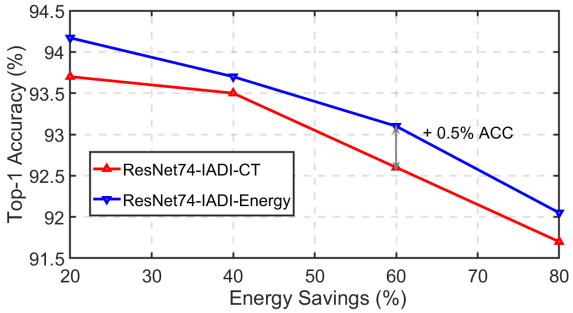


Figure 9: Accuracy vs. energy savings of IADI on CIFAR-10 and ResNet74 when using energy and computational costs as the resource-aware loss, respectively.

IADI *vs.* state-of-the-art methods on ImageNet. We evaluate IADI on the DenseNet121 trained with the ImageNet dataset. We evaluate the performance based on the top-1 accuracy v.s. computational savings on the validation set, compared to the DenseNet baseline whose top-1 accuracy is 75%. Our trained DenseNet121-IADI achieves 25% computational savings with a 0.3% accuracy degradation. More experiment results could be found in Appendix A of Supplementary.

4.3. Performance of the Proposed DDI

We obtain the DDI models by adding early exiting to well-trained IADI models, and here present the ones that are on top of ResNet74-IADI and DenseNet100-IADI.

Branch Test. Tbl. 1 shows the accuracy vs. FLOPs of the trained DDI models when all the images are forced to exit at a particular branch. While early exiting can save computational cost with a possibility of degrading the accuracy, an optimal balance between computational cost (measured in FLOPs) and accuracy can be obtained by adjusting the corresponding uncertainty threshold. To study this optimal balance, we evaluate DDI when activating early exiting only at one particular branch. Fig. 10 shows how the accuracy of ResNet74-DDI varies with the uncertainty threshold values when activating early exiting at the 5th branch. We can see

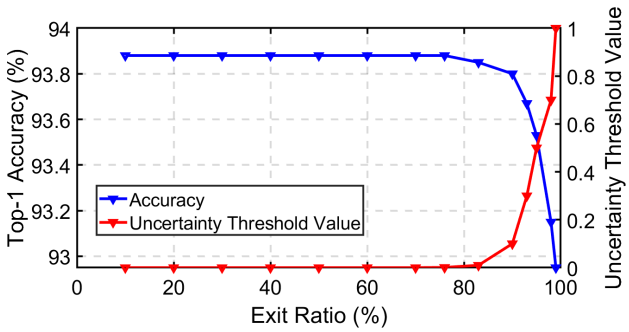


Figure 10: Accuracy of ResNet74-DDI on CIFAR-10 when increasing the exit ratio by varying the uncertainty threshold.

Branch	ResNet74-DDI		DenseNet100-DDI	
	Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)
1st	81.98	9	84.36	65
2nd	84.45	21	85.62	200
3rd	90.20	51	89.46	266
4th	91.00	58	92.05	343
5th	92.82	74	93.38	372
6th	93.81	117	94.44	392
Main	93.88	144.5	94.68	408

Table 1: Accuracy and corresponding computational cost when exiting at different branches of ResNet74-DDI and DenseNet100-DDI. 1M means one million FLOPs

that more computational cost can be saved as the uncertainty threshold increases to allow more data-driven decisions to be made at this branch without degrading the accuracy when the exit ratio is less than 80%. After that, an obvious accuracy degradation is observed, showing the optimal balance is at around 80% of exit ratio.

Model	Budget(M)	DDI Acc	Base Acc	Acc Δ
ResNet 74-DDI	59	91.00 %	90.70 %	**0.30 %**
	85	92.82 %	91.25 %	**1.57 %**
	110	93.50 %	92.30 %	**1.20 %**
	170	93.88 %	92.50 %	**1.33 %**
DenseNet 100-DDI	180	93.51 %	93.00 %	**0.51 %**
	250	93.83 %	93.64 %	**0.19 %**
	290	94.05 %	93.68 %	**0.37 %**
	420	94.68 %	94.45 %	**0.23 %**

Table 2: Comparing the accuracy between DDI models (ResNet74-DDI and DenseNet100-DDI) and the corresponding base models given the same computational cost budget. 1M means one million FLOPs

Trade-off Resource-usage and Accuracy. One unique advantage of DDI is that it can ensure boosting and controlling the computational efficiency, by both removing unnecessary operations and halting inference when hard constraints, such as computational budget, are reached. To demonstrate DDI's capability to achieve an optimal trade-off between resource usage and accuracy, we compare DDI with those of the base models in terms of computational cost and accuracy when given the same computational budget. Specifically, as shown in Tbl. 2, for each of the DDI model, we choose 4 hard computation budgets (measured in FLOPs) that corresponds to 4 smaller base models: for ResNet74-DDI, the budgets are: 59M (ResNet14), 84M (ResNet20), 110M (ResNet26), 170M (ResNet38); For DenseNet100-DDI, the budgets are: 180M (DenseNet46), 250M (DenseNet58), 290M (DenseNet64), 420M (DenseNet82). We can see that DDI models consistently achieve higher accuracies than their base models given the same computational budgets which range over 59-170 and 180-420 M FLOPs, respectively. Specifically, DDI can offer at least 0.15% and up to 1.57% accuracy improvement.

4.4. Skipping Behavior Visualization and Analysis

Easy/Hard Inputs vs. Skipping Ratio. To visualize and analyze IADI's effectiveness in adapting its complexity to the classification difficulty of the input images, we select two groups of input images with corresponding skipping ratio being larger than 60% ("Easy") and smaller than 40% ("Hard"), respectively. Fig. 11 shows a subset of these two groups. It is interesting to see that the "Easy/Hard" images identified by IADI is consistent with our human eyes. In particular, we can see that the "Easy" images have a clearer boundary while the "Hard" images tend to have a blurry one.

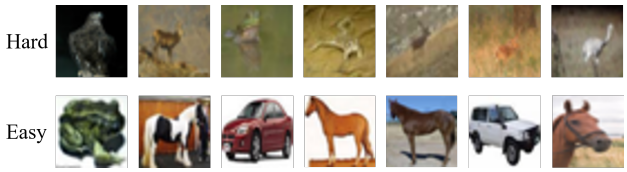


Figure 11: Visualization of input images with a larger than 60% ("Easy") and smaller than 40% ("Hard") skipping ratio, respectively, the former of which indeed looks easier to be classified correctly than the latter.

5. Conclusion and Discussions

We have proposed DDI, a novel framework that unifies input-dependent (IADI) and resource-dependent (RADI) dynamic inference. For IADI, we develop a MGL2S approach that allows simultaneous coarse-grained layer and fine-grained channel skipping. Applied on ResNet trained with CIFAR10, our IADI model achieves up to 4 times computational savings with the same or higher accuracy compared to the most competitive baseline SkipNet. We also applied MGL2S to DenseNet with novel gating and skipping design, achieving consistently better accuracy-resource balance than ResNet. We further combine the IADI framework with early exiting and demonstrate the that DDI model have the flexibility in achieving higher accuracy over a series of base models when performing inference different hard resource constraints.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[2] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume abs/1311.2524, pages 580–587, 2014. 1

[3] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014. 1

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. In *IEEE Internet of Things Journal*, volume 3, pages 637–646, Oct 2016. 1

[5] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, Jan 2017. 1

[6] Lukasz Kaiser, Aidan N. Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *CoRR*, abs/1706.05137, 2017. 1

[7] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 2, 2017. 1, 2, 6

[8] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2017. 1

[9] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 1, 2, 3, 6

[10] Yingyan Lin, Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. Predictivenet: An energy-efficient convolutional neural network via zero prediction. In *Proceedings of ISCAS*, 2017. 2

[11] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015. 2

[12] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2129–2137, 2016. 2

[13] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, 2017. 2

[14] 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. 2, 4, 5

[15] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2403–2412, 2018. 2

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016. 2

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 2

[18] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *ACM SIGARCH Computer Architecture News*, volume 45, pages 548–560. ACM, 2017. 2

[19] Yu Ji, Ling Liang, Lei Deng, Youyang Zhang, Youhui Zhang, and Yuan Xie. Tetris: Tile-matching the tremendous irregular sparsity. In *Advances in Neural Information Processing Systems*, pages 4119–4129, 2018. 2

[20] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations*, 2017. 2

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2018. 2

[22] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2755–2763. IEEE, 2017. 2

[23] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406. IEEE, 2017. 2

[24] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076. IEEE, 2017. 2

[25] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. 2

[26] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016. 2

[27] Xiaofan Xu, Mi Sun Park, and Cormac Brick. Hybrid pruning: Thinner sparse networks for fast inference on edge devices. In *International Conference on Learning Representations*, 2018. 2

[28] Eunwoo Kim, Chanho Ahn, and Songhwai Oh. Nestednet: Learning nested sparse structures in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8669–8678, 2018. 2

[29] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017. 2

[30] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. 2, 3, 6

[31] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. *Lecture Notes in Computer Science*, page 318, 2018. 2

[32] 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. 2, 5

[33] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2181–2191. Curran Associates, Inc., 2017. 2

[34] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network, 2018. 2

[35] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. 2

[36] Ravi Teja Mullapudi. Hydranets: Specialized dynamic architectures for efficient inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8080–8089, 2018. 2

[37] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, Dec 2016. 2, 4, 5

[38] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):62, 2016. 2

[39] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 5

[40] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2018. 6

[41] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. 6

[42] Yue Wang, Tan Nguyen, Yang Zhao, Zhangyang Wang, Yingyan Lin, and Richard Baraniuk. Energynet: Energy-efficient dynamic inference. 2018. 7