OPA: One-Predict-All For Efficient Deployment

Junpeng Guo* Shengqing Xia* Chunyi Peng
Department of Computer Science, Purdue University, West Lafayette, IN, USA
{guo567, xia170, chunyi}@purdue.edu

***Abstract*—Deep neural network (DNN) is the de facto standard for running a variety of computer vision applications over mobile and embedded systems. Prior to deployment, a DNN is specialized by training to fit the target use scenario (depending on computing power and visual data input). To handle its costly training and meet diverse deployment needs, a "*Train Once, Deploy Everywhere*" paradigm has been recently proposed by training one super-network and selecting one out of many sub-networks (part of the super-network) for the target scenario; This empowers efficient DNN deployment at low training cost (training once). However, the existing studies tackle some deployment factors like computing power and source data but largely overlook the impact of their runtime dynamics (say, time-varying visual contents and GPU/CPU workloads). In this work, we propose OPA to cover all these deployment factors, particularly those along with runtime dynamics in visual data contents and computing resources. To quickly and accurately learn which sub-network runs *"best"* in the dynamic deployment scenario, we devise a *"One-Predict-All"* approach with no need to run all the candidate sub-networks. Instead, we first develop a shallow sub-network to test the water and then use its test results to predict the performance of all other deeper sub-networks. We have implemented and evaluated OPA. Compared to the state-of-the-art, OPA has achieved up to 26% higher Top-1 accuracy for a given latency requirement.**

I. Introduction

Deep neural network (DNN) has been widely applied in various computer vision tasks over heterogenous mobile and embedded systems, transforming our daily life with thrilling applications such as autonomous driving, searching what you see (e.g., Google Lens), smart home, security surveillance, drone delivery, remote healthcare, and to name many [1].

A DNN is a powerful tool for computer vision inference because it uses a huge amount of trainable parameters at multiple processing layers (see Fig. 2). Prior to deployment, each DNN is specialized through training over pre-selected data, in order to learn a computational model that mimics how the brain perceives and understands visual data in the target use scenario. While effective, such a *train-then-deploy* paradigm is extremely costly because one DNN can not fit all. New training is required for each new use scenario, which is impacted by a wide variety of deployment factors including but not limited to source camera capabilities (e.g., capturing resolution, lens type, the field of view), visual data contents (e.g., traffic signs at daytime or at night), computing hardware (e.g., GPU cluster/GPU/CPU), available resource dynamics (e.g., stable or unstable GPU usage), as well as inference quality requirements (say, acceptable accuracy and/or latency). Obviously, a DNN for real-time (say, within 50 ms) road sign classification that runs on a self-driving car should be different from the one for searching what you see on smartphones. Considering rich real-world diversity, the conventional *train-then-deploy* paradigm hardly scales up with efficient deployment.

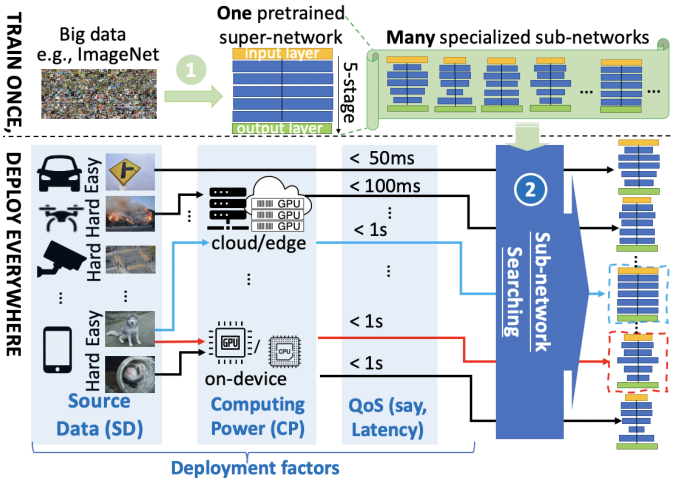


Fig. 1: The "*Train Once, Deploy Everywhere*" paradigm is realized by Once-For-All (OFA) which performs generic training at the start and searches for a specialized subnet for the deployment.

To tackle this problem, a new paradigm of "*Train Once, Deploy Everywhere*" has been recently proposed [2]. The core idea is a *Once-For-All (OFA)* framework, as illustrated in Fig. 1. It typically involves two steps: generic training and specialization (DNN search). First, OFA only needs to train *one* super-network, which can cover a bunch of *many* specialized sub-networks (subnets, explained in §II). Then, the deployment is to search for a specialized subnet that "*best*" fits the target scenario. The main advantage is that OFA explicitly searches in the subnet space rather than re-training a DNN for the new target scenario. DNN search is much more efficient compared to compute-intensive training.

Following [2], a number of recent studies have expanded the "Train Once, Deploy Everywhere" family [3]–[15]. They all are centered on DNN search that looks for a specialized subnet by taking into account various deployment factors. These factors, except inference requirements, are roughly classified into (1) source data and (2) computing power, along at different levels of runtime dynamics. The first OFA work [2] considered both but without any runtime dynamics. Dynamic-OFA [15] extended OFA by adding dynamic workload and computing resources into consideration. Several follow-up studies focused on computing power only, either satisfying the constraints on hardware capabilities [3], [4] or adapting to runtime computing

*Co-primary authors.

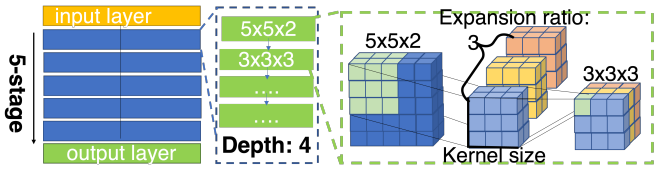


Fig. 2: Illustration of different dimensions in one super network that are used to compose all the subnets.

Representative Studies	Computing Power		Source Data		QoS Req.
	HW (offline)	Resource (runtime)	CC (offline)	Content (runtime)	
OFA [2]	√	⊗	√	⊗	Latency
Dynamic-OFA [15]	√	√	√	⊗	Latency
Chamnet [3]	√	⊗	⊗	⊗	Latency
LegoDNN [7]	√	√	⊗	⊗	Latency
BranchyNet [11]	⊗	⊗	√	√	None
OPA (our work)	√	√	√	√	**Latency**

TABLE I: Comparison of representative studies and our work. CC: camera capabilities. HW: hardware.

resources [5]–[7]. On the other hand, other groups consider source data only and use an early exit mechanism to handle various data contents [8]–[14]. They all need to meet a latency requirement where inference completes within a given time window from several tens of milliseconds to a few seconds. Table I compares the representative studies.

However, none of the existing studies have supported all the factors. Previous studies show that the best-fit subnet should change when computing resources vary (e.g., GPU becomes overloaded without sufficient resources to run the pre-selected subnet) or visual contents change (e.g., detecting traffic signs at backlight is much harder than at full light). There is no surprise that the best-fit subnet should change when both vary at runtime. We assess the benefits of putting more (if not all) factors together in our motivation study (§III). It brings a considerable accuracy gain ($> 6\%$) for the target deployment.

There is no surprise that it is a very challenging task. Essentially, the task requires learning the performance (accuracy) of every candidate subnet for any given deployment scenario, and then picking the best fit. Exhaustive search is computationally prohibitive by running every candidate subnet in the given deployment scenario. All previous approaches that predict the performance of candidate subnets can not work well because they are conditioned on some, but not all factors.

To tackle this challenge, we propose OPA (*One-Predict-All*), which only needs to run one (special) subnet and use its results to predict how other subnets perform instead of running all these subnets under the target deployment. We find it feasible because we leverage a *"shallow"* subnet as a *pioneer* to test the water. Compared to those deeper subnets, this shallow subnet uses fewer layers (here, 2 stages instead of 5 stages). Testing the water is thus done quickly, which is acceptable (given the latency requirement). Moreover, we will show that the testing time is not wasted; The deeper subnet selected later can continue where the shallow subnet stops, completely inheriting the intermediate results obtained by this shallow subnet. We find that this shallow subnet is able to predict the accuracy of other deeper subnets because their inference accuracies are highly correlated. It is not hard to understand because the inference accuracy is inherently impacted by these deployment factors (e.g., it drops when visual contents are hard to recognize). This is what we mean by *"test the water"*.

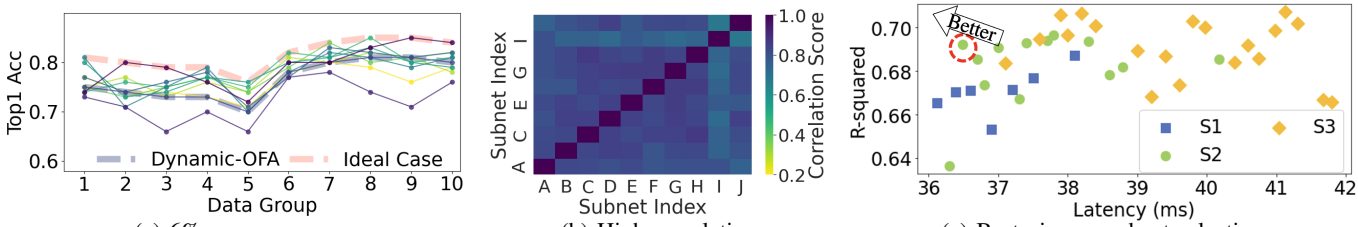
Turning this idea into a practical solution, OPA has to address several technical challenges. First, it is hard to know the accuracy results when testing the water because there is no ground truth available in real time. OPA comes up with an accuracy estimator to infer the accuracy of the pioneer subnet over only the intermediate results (more precisely, the confidence values). Second, the search space is huge. To reduce the extra cost of running the pioneer subnet, we narrow down the search space and only search for the subnets that are supersets of the pioneer subnet. We devise a quick online search algorithm to start the search near the best candidate subnet, which is profiled offline. Third, testing the water using a shallow subnet still takes time, though it is faster than using a deeper subnet. We develop a zero-waste strategy where the deeper subnet directly and completely inherits the intermediate results obtained by the shallow subnet and continues to run the rest unshared layers. More details are elaborated in §IV.

We have implemented OPA on top of PyTorch and evaluated it under various scenarios, using two computing hardware (GPU and CPU), dynamic computing resources, and different data groups associated with multiple latency requirements. We choose image classification as our showcase application. Compared with state-of-the-arts, OPA improves Top-1 classification accuracy by 6.6% – 9.0% when working independently on the IMAGENET dataset and achieves up to 26% improvement when computing resources are saturated (§V).

II. BACKGROUND AND RELATED WORK

One super-network and many subnets. In a full DNN architecture (denoted as a super-network in OFA [2]), there are many trainable parameters. Different combinations of these parameters customize a batch of subnets, each specialized for its target deployment scenario. Fig. 2 illustrates one super-network and its many subnets using MOBILENET V3, one of the most popular DNN models [16]. It consists of multiple stages (here, 5 stages). Each stage is composed of several stacked layers, and the number of layers is defined as its depth. Each layer applies a few kernel filters to the input. Among all the filters, convolution kernels are the most popular ones, which are also referred to as convolutional neural networks (CNNs). Various kernel sizes and expansion ratios will construct different subnets. In this work, the depth is ranged in $\{2, 3, 4\}$, the kernel size in $\{3, 5, 7\}$, and the expansion ratio in $\{3, 4, 6\}$; The input image resolution ranges from 128 to 224 with a stride of 20. As a result, there are up to $3^{20} * 3^{20} * 3^5 * 5 \approx 6 * 10^{21}$ subnets.

Related work and limitations. Existing studies search for a specialized subnet to meet different deployment requirements. The deployment factors can be roughly grouped into two categories: *computing power* and *source data*. Early studies focus on selecting the optimal subnets based on the computing power provided by different hardware, such as GPU clusters at the edge server, GPU at mobile systems,



(a) 6% accuracy gap (b) High correlation (c) Best pioneer subnet selection

Fig. 3: A pilot study to motivate *One-Predict-All*.

Subnet	A	B	C	D	E	F	G	H	I	J
R-squared	0.62	0.61	0.67	0.69	0.67	0.68	0.71	0.70	0.65	0.61
Latency (ms)	69	70	71	72	73	74	75	76	77	78

TABLE II: Comparison on R-squared and latency.

and CPU at embedded systems [2]–[4]. To keep up with the dynamics of computing power, several studies further monitor resource availability at runtime and then update the subnet search accordingly [5]–[7], [15]. Among them, Dynamic-OFA is one remarkable work that proposes a predictor-guided search process [15]; specifically, an accuracy predictor takes the subnet's architecture as the input to predict its inference accuracy without actually running it, which thus accelerates the search process. However, this approach suffers from less accurate results when data contents considerably change over time, likely resulting in an improper or even poorly-performed subnet. In front of the other factor group on source data, some studies consider the impact of only source data and select different subnets per frame or a few frames (over a short time) [8]–[14]. They all adopt an early exit mechanism that stops running the model as long as the requirement is satisfied and thus skips the rest of the model search. However, these studies require enumerating all supported subnets during the training with their specific exit branches. Consequently, it is computationally prohibitive to support a broad range of many subnets. Moreover, it is costly to support varying latency requirements because their exit policies must be pre-trained over the given latency requirements.

Table I compares the most related work. In a nutshell, these prior studies consider some deployment factors and largely ignore their runtime dynamics, which impedes the DNN search for efficient deployment in practice.

III. Motivation

In this section, we use a pilot study to motivate the need for taking more deployment factors into account. To handle all deployment factors, we propose an idea of *One-Predict-All*: we run a shallow subnet (called a pioneer subnet) to test the water and predict the performance of other deeper subnets under the given use scenario. We not only demonstrate the potentials of this approach but also present the challenges of integrating this idea into the practice.

Ideal subnet selection. Ideally, the subnet selection should take all deployment factors into account and be done in a fine-grained manner to accommodate both static heterogeneous and runtime dynamics. To demonstrate its potential gain over prior work, we use image classification as our showcase application to conduct the following study. We randomly select 5K images from the ImageNet dataset [17] and divide them into groups. Each group contains 100 images; the group size impacts the accuracy and overhead of the subnet search; here, we choose 100 because it reaches a good trade-off between the accuracy and the overhead in this study; more group sizes will be evaluated in §V. Constrained by a huge amount of possible subnets, we cannot enumerate them to find the optimal one. Without loss of generality, we test with a subset of five-stage subnets to demonstrate the substantial gain of the optimal selection over the one selected by existing approaches. Specifically, we run these subnets over the first ten image groups (labeled from 1 to 10) over GPU. No other background applications are running over GPU. We sort these subnets in the descending order of inference latency and choose ten representative ones (labeled from A to J) with their latency roughly evenly distributed between the smallest and largest ones (under the 78 ms latency requirement, Table II).

Fig. 3a shows Top-1 accuracy of these ten selected subnets per each image group. Note that Top-1 accuracy is the most popular metric for evaluating image classification. We also plot the results using the optimal subnet (marked in the pink dashed line) and the one selected by Dynamic-OFA (marked in the gray line). We choose to compare with Dynamic-OFA because it is one of the most recent studies which adapts to time-varying computing power [15]. The ideal selection chooses the subnet with the highest accuracy per group out of 10 candidate subnets, while Dynamic-OFA chooses an identical subnet across distinct image groups (here, subnet G). Evidently, Dynamic-OFA performs worse than the ideal one, with an accuracy gap of about 6%; It is because Dynamic-OFA selects one fixed subnet no matter how source data changes; It fails to handle diversity and dynamics in source data, which sacrifices the accuracy when image contents vary significantly. We notice that the actual gain is likely higher because this study considers only ten representative subnets; There are many more candidate subnets and the optimal one should be better, at least no worse than the ideal one used in this study.

To pursue higher accuracy, DNN search should consider all static and dynamic deployment factors on computing power and source data. However, it is not easy in practice. A straightforward approach requires an exhaustive search of running all candidate subnets under the target deployment scenario, which is computationally prohibitive. We next explore three opportunities to make it practical and efficient at runtime.

Opportunity 1: A pioneer subnet can test the water and predict the performance of other subnets under the deployment scenario. The good news is that there is no need to run all candidate subnets under the target deployment; Instead, running a pioneer subnet suffices. We observe that there exists a strong correlation among the accuracy results of distinct subnets, as illustrated in Fig. 3a. We further measure pairwise correlation coefficients between all ten subnets in Fig. 3b. All the coefficients are above 0.7, which implies that it is technically feasible to run one subnet to predict how others perform. Such a high correlation is attributed to the progressive shrinking training algorithm used by OFA [2]. It first trains the super-network and then progressively fine-tunes the network to support smaller subnets that share weights with the larger ones. As a result, most subnets are consistently impacted by varying source contents (e.g., the accuracy goes down when the group is harder). However, there is no single winner; The best subnet varies per group.

Inspired by this, we run one subnet and use its accuracy result to predict the accuracy of other subnets. The accuracy prediction uses a simple linear regression. We demonstrate its feasibility as follows. We enumerate all ten subnets. In each test, we pick one as the pioneer subnet to predict how the other nine subnets perform. We randomly select 75% of 5K images for model fitting (training) and use the remaining 25% for testing. Table II shows the prediction performance (accuracy) in terms of the R-squared value. We have two observations. First, all the R-squared values are relatively high (>0.6). It indicates that *One-Predict-All* is technically feasible. Second, the prediction performance does vary with different subnets. It implies that a careful search for the pioneer subnet is required.

Opportunity 2: A shallow subnet, instead of a deep subnet, can act as the pioneer one. The better news is that the above prediction does not need a deep (five-stage) subnet. We find that a shallow subnet can also act as the pioneer to test the water. We test with various shallow subnets with their stage sizes from one to three and present the results in Fig. 3c. Since the subnet complexity grows with more stages, we randomly choose more subnets as the stage size increases. Specifically, we choose 7 one-stage subnets (S1), 13 two-stage subnets (S2) and 20 three-stage subnets (S3). Given one subnet, we use two measures – R-squared value and latency – to assess its prediction performance for five-stage subnets which are candidates of the DNN search. We find that, in general, the prediction accuracy (namely, the R-squared value) grows with more stages. However, with a larger stage size, a pioneer subnet pays a higher latency cost. As the prediction is often constrained by the latency requirement, an optimal choice should pursue a good trade-off between the R-squared value and latency. In this study, the best one out of 40 candidates is a two-stage subnet (circled in Fig. 3c). Its R-squared value is 0.69 and the latency takes 36.5 ms. We notice that a shallower subnet does not necessarily runs faster. For instance, running a five-stage subnet with only 20 neurons per stage is still faster than a two-stage subnet with 400 neurons per stage. The latency depends on the complexity of the subnet (the number of trainable parameters). The more parameters, the more powerful for higher inference accuracy, but it runs slower (at the cost of higher latency).

Opportunity 3: No extra latency occurs for running the shallow pioneer subnet. Its intermediate results can be fully reused by the deep (five-stage) subnet which is finally used. Running a shallow pioneer subnet to test the water is faster but still takes time. Seemingly, it incurs extra overhead and latency as we have to run one more pioneer subnet in addition to the final subnet used for the visual inference task. The good news is subnet inheritance, where the results obtained by a shallow pioneer subnet can be fully inherited by a deeper subnet as long as both share the same shallow stages. The deeper subnet continues where the shallow one stops and runs the remaining stages. By this means, testing the water becomes part of running the final deep subnet; There is no extra latency. More precisely, we find that the extra latency is used to load one layer's results, which takes several milliseconds. We note that pioneer subnets with fewer stages allow for a larger search space, which more likely contains the best-fit subnet.

Our idea: *One-Predict-All*. We propose *One-Predict-All*, which uses a shallow subnet to predict "all", thereby accelerating the search for the best-fit deep subnet. However, there are two technical challenges to turn this idea into reality.

Challenge 1: The accuracy of the pioneer subnet is hard to get in real-time because the ground truth is not provided. In principle, the accuracy of the pioneer subnet requires the ground truth, which is manually labeled and not available in real time. Recent studies propose to replace the ground truth labels with the results from a golden teacher's model for video analytics [18], [19]. However, the golden teacher's model uses a complex architecture (a large number of parameters) to ensure high accuracy (say, > 0.9) and can not work well to label a small fraction of data in real-time; Moreover, the approach is effective over stable source data (e.g., similar background and lighting) and cannot work well with varying source contents, which is our target scenario. Even worse, it slows down the processing as it has to execute two computing tasks in parallel (one for the golden teacher's model, the other for the target inference task). In order to learn the ground truth, we find that the latency almost doubles, increasing from 36.5 ms to 65 ms). As a result, we need an alternative solution to obtain (estimate) the accuracy at a small cost.

Challenge 2: The vast subnet space makes it hard to search quickly. The optimal subnet varies with data contents and available computing resources that change over time. Such runtime dynamics require an online search as learning in advance is not possible. While the subnet performance is predictable, it still takes time to search for the optimal one due to the many subnet candidates. Hence, another problem is how to quickly find the subnet which can directly inherit the output of the pioneer subnet and retain high inference accuracy.

IV. Design of OPA

We propose OPA to integrate the idea of *One-Predict-All* on top of the existing OFA framework. The core idea is to

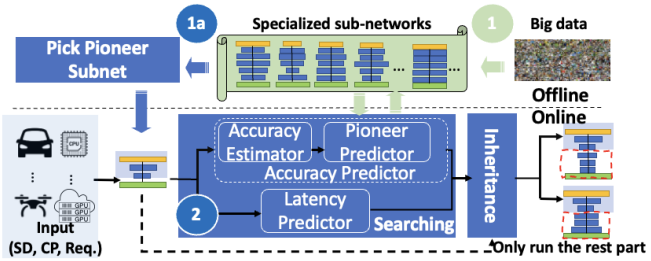


Fig. 4: OPA's architecture and work flow.

choose a shallow pioneer subnet that is able to well predict both the accuracy and latency of candidate subnets under the current use scenario and then apply it to accelerate the search for the best-fit subnet at runtime. To tackle the aforementioned two challenges, we propose an accuracy estimator and a novel search algorithm. Fig. 4 depicts the basic components and working flow. The modules added by OPA are marked in blue.

At a high level, it starts with offline training and then runs an online search for the specialized subnet at runtime, following the "Train Once, Deploy Everywhere" paradigm. OPA trains a super-network using the progressive shrinking algorithm proposed in [2]. Different from previous studies, OPA also profiles how each shallow subnet predicts the performance of deep subnets, which is later used to pick a pioneer subnet for runtime search. In this study, we find that a two-stage shallow subnet works well for a good trade-off between accuracy and latency. At runtime, we first run the pioneer subnet to test the water. To quickly determine an appropriate deep subnet for visual inference, we develop an accuracy predictor (consisting of an accuracy estimator and pioneer predictor) and a latency predictor to estimate how five-stage candidate subnets perform. We choose the subnet which yields the highest accuracy while meeting the latency requirement. Eventually, we run the selected subnet for visual inference. To save time, we adopt subnet inheritance to continue where the shallow subnet stops. Namely, we run the remaining layers (bounded by the red boxes in Fig. 4). We next elaborate on these techniques.

A. Accuracy Predictor and Latency Predictor

We develop two predictors to test the water quickly. Both predictors use the pioneer subnet's performance to predict how other deeper subnets perform without running them.

• **Leverage confidence values to estimate the accuracy of the pioneer subnet.** To predict the accuracy of all deep subnets, we first need to obtain the accuracy of the pioneer subnet, which is challenging due to the lack of ground truth. We propose an accuracy estimator to infer the accuracy by using its intermediate results only. In our prior study, we observed that the confidence values could be leveraged to learn how the inference accuracy changes [20]. For the task of image classification, we extend this idea by directly predicting the absolute value of Top-1 accuracy. We run a Multinomial Naive Bayes classifier, where the inputs are the confidence values, each corresponding to a candidate class, and the output is either 1 (classified as correct) or 0 (wrong). Its objective is

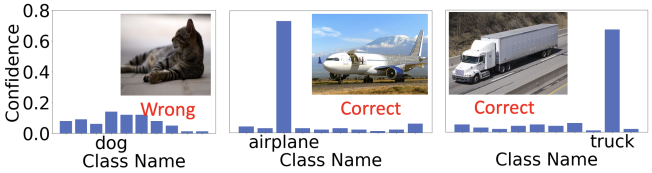


Fig. 5: Classification likely succeeds when one confidence value is significantly larger than the others. .

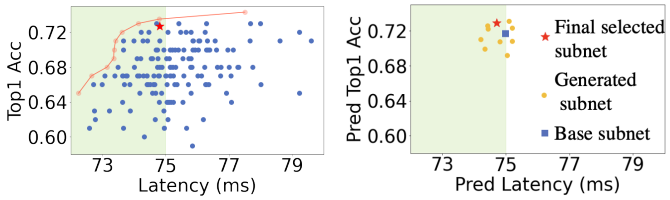
to minimize the zero-one loss for this correct/wrong classification. The accuracy of the pioneer subnet is estimated as

$$\hat{\mathrm{A}}_{pioneer} = \frac{\sum_{\mathrm{i}}^{n} Bayes(\text{Confidence values of image i})}{\text{Number of images}}. \quad (1)$$

However, it is not very accurate (45% in our pilot study) by directly using the confidence values via Eqn (1). We further closely examine how confidence values are associated with classification accuracy. Fig. 5 gives three illustrative examples. We see that a correct classification is often associated with one strong candidate; Its confidence value is relatively high (see the airplane and truck examples). A wrong classification often has several possible classes with comparable confidence levels. Here, it is hard to tell if the first animal is a dog or a cat. We see that a dominating class plays a decisive role, regardless of its absolute confidence value and class name. As a result, we take a two-step preprocessing to get rid of useless information while retaining their relative ranks. The first step is to apply MINMAXSCALER to all the confidence values in order to magnify their relative relationship after putting them into the same scale. The second step is to sort the scaled confidence values in descending order. The two-step preprocessing is effective, increasing the accuracy to 74%. We admit that the accuracy is not high enough, and we next show how to leverage another module (here, a pioneer predictor) to make up for unsatisfactory accuracy estimation.

• **Use a DNN-based pioneer predictor to infer the accuracy of deep subnets.** In the pilot study presented in §III, we use linear regression to estimate the accuracy of other subnets given the accuracy of the pioneer subnet. To enhance the estimation accuracy of candidate subnets and compensate for the accuracy estimation errors of the pioneer one, we use a DNN model. Specifically, we use a three-layer feedforward neural network with 124 hidden units at each layer. This pioneer predictor uses the estimated accuracy of the pioneer subnet (the output of the accuracy estimator) and the subnet architecture to estimate the accuracy of another subnet. To encode the architecture as the input, we concatenate all the parameters (including depth, width, kernel size, expand ratios, and resolution) into a large vector (46 dimensions). Putting the accuracy estimator and the pioneer predictor together, the accuracy prediction becomes acceptable. Specifically, the root-mean-square error (RMSE) between the predicted accuracy and the actual one of any given subnet is no larger than 0.024. OPA is able to precisely predict the accuracy of any subnet by leveraging the intermediate results of the pioneer subnet.

• **Measure floating-point operations (FLOPs) to predict latency.** Recent studies show that FLOPs can be used as



(a) Pareto curve (b) Illustration of search

Fig. 6: The heuristic idea behind the search algorithm.

features to predict latency [21], [22]. We get the count of FLOPs of the pioneer subnet and use a linear approximation to estimate the latency of a given subnet as

$$\hat{\tau}_{subnet} = \frac{\tau_{pioneer}}{\text{FLOP}_{pioneer}} \times F(subnet). \quad (2)$$

Here, $\tau_{pioneer}$ and $\text{FLOP}_{pioneer}$ are the latency and FLOPs used by the pioneer subnet, and $F(subnet)$ is a function to estimate the count of the FLOPs of a specific subnet. The good thing is that FLOPs are fixed for a given architecture and do not change with different computing power.

B. An Algorithm for Online Search

To quickly determine the *"best"* subnet with the highest accuracy while meeting the latency requirement, we develop a novel search algorithm at runtime. There are two key tasks: (1) reduce the search space and (2) reach the target subnet quickly (minimize the iterations). In this work, we consider the candidate subnets, which are the supersets of the two-stage pioneer subnet; Thus, we freeze the first two stages and adjust the remaining 3 stages with up to 12 stacked layers. We reduce the subnet space to around $3^{12} * 3^{12}$ (kernel sizes and expand ratios for 12 layers) $* 3^3$ (depths for the 3 stages) $* 5$ (resolutions) $\approx 3.8 * 10^{13}$.

It is not easy to locate the target subnet quickly. Existing studies [2], [3], [15] adopt a random search algorithm to find the subnets that meet the latency requirements and use evolutionary search to keep the one with the highest accuracy. Even if possible, searching over a huge number of subnets is inefficient, often taking several minutes or even hours. It is not acceptable when source contents change fast.

Our intuitive idea is to make the starting point near the target so that the search can converge quickly to the target. The starting point should be the subnet whose inference latency is closest to the requirement. Fig. 6a shows that the Pareto curve over the latency and accuracy trade-off increases monotonically (in the pink line). We also plot the actual accuracy and latency of 150 subnets for a single data group. Thus, once our starting point is near the QoS requirement, we can always get closer to the optimal subnet on the Pareto curve with several iterations.

We next show how to find the subnet whose latency is close (or the closest) to the requirement without running it. We leverage the observation that the latency of a subnet is approximately proportional to its FLOPs, which are fixed for a specific subnet as its intrinsic attributes. Therefore, we profile the mapping between different FLOPs and subnets offline and build a look-up table accordingly. Note that the FLOPs of a certain subnet are not impacted by the deployment scenario,

Algorithm 1 Online Search used by OPA

Input: Confidence values of the pioneer subnet ($conf$), latency requirement (τ_{req}), subnet lookup table (map_{subnet}, key: FLOPs, value: subnet), latency allowance (Δ_τ), FLOPs of the pioneer subnet ($\text{FLOP}_{pioneer}$), latency of the pioneer subnet under current deployment scenario ($\tau_{pioneer}$), the maximum iteration count (N_{max})
Output: Best subnet architecture ($subnet^*$)

1: **Initialization:** $subnet^* = NULL, \text{A}_{max} = 0, n = 0$
2: $\hat{\text{A}}_{pioneer}$ = AccuracyEstimator($conf$)
3: $\text{FLOP}_{allowed} = (\tau_{req} - \Delta_\tau) \cdot \frac{\text{FLOP}_{pioneer}}{\tau_{pioneer}}$
4: key = findnearest($\text{map}_{subnet}.keys(), \text{FLOP}_{allowed}$)
5: $subnet\_base$ = $\text{map}_{subnet}[key]$
6: **while** $n \leq N_{max}$ **do**
7: $subnet\_new$ = GenNewSubnet($subnet\_base, n$)
8: $\hat{\text{A}}_{subnet\_new}$=PioneerPredictor($subnet\_new$, $\hat{\text{A}}_{pioneer}$)
9: $\hat{\tau}_{subnet\_new}$ = LatencyPredictor($subnet\_new, \tau_{pioneer}$)
10: **if** $\hat{\tau}_{subnet\_new} + \Delta_\tau > \tau_{req}$ **then**
11: Continue
12: **end if**
13: **if** $\text{A}_{max} < \hat{\text{A}}_{subnet\_new}$ **then**
14: $\text{A}_{max} = \hat{\text{A}}_{subnet\_new}$
15: $subnet^* = subnet\_new$
16: **end if**
17: $n = n + 1$
18: **end while**
19: **return** $subnet^*$

so the look-up table profiled once can be used at runtime. We only need to convert the specific latency requirement (τ_{req}) to the maximum allowed FLOPs under the current deployment with the help of $\tau_{pioneer}$. Then look up the table to find the corresponding subnet. To build such a table, we randomly select 1k subnets and record their FLOPs. We emphasize that the maximum allowed FLOPs will always change according to the latency requirement as well as the varying latency of the pioneer subnet, which is the reflection of the current deployment scenario. Thus the selected subnets will be different. This is significantly different from [15], whose look-up table always gives the same output for a given latency requirement. To find the best-fit subnet, we slightly change the base subnet (starting point) by randomly tuning several dimensions and select the one with the highest predicted accuracy as the optimal subnet.

Algorithm 1 shows the pseudo-code for the online search. At runtime, the accuracy estimator leverages the intermediate results of our pioneer subnet, namely confidence values, to estimate its inference accuracy ($\hat{\text{A}}_{pioneer}$). Next, we generate the base subnet ($subnet\_base$) as the starting point for the search via the look-up table (lines 3-5). In detail, given the transformed latency requirement (75 ms in Fig. 6), we find the nearest FLOPs stored in the look-up table and get $subnet\_base$. Then, we randomly adjust a few layers (1-3 in our experiment) of the base subnet to generate a new subnet ($subnet\_new$) at each iteration (labeled as yellow dots in Fig. 6b). After multiple iterations, we choose the one with the highest accuracy (lines 5-18). Specifically, we alter either kernel size or expansion ratios for each layer through GenNewSubnet() with a random seed that is associated with n. Note that we do not change the depth, as it can

drastically change the architecture of the base model and hurt accuracy. Then for each subnet, the accuracy predictor and latency predictor are used to predict its performance. Note that if a subnet needs a longer inference time than required, we will not consider it anymore. Finally, we select the subnet marked as a red star in Fig. 6b. It is also marked in Fig. 6a for comparison. Its accuracy is quite close to the optimum one though it is slightly smaller. OPA dramatically shortens the search time compared to the existing studies, where candidate subnets are generated from scratch for every iteration.

C. Subnet Inheritance

Once the searching algorithm determines the final subnet, fast switching with little overhead is crucial. This is because running the pioneer subnets already takes time. It already occupies a significant portion of time for the given latency requirement and leaves less time for the final selected subnet to do inferences, which is unacceptable.

To alleviate this issue, we restrict the pioneer subnet to be the subset of the final selected one. Our solution is to exploit the intermediate processing results of the pioneer subnet fully. Instead of running the complete selected subnet, we inherit the output of the pioneer subnet's last convolution layer (the one before the fully connected layer) and continue running the remaining three stages. Fundamentally, it functions similarly to pausing running the selected final subnet in the middle. This idea only introduces a small overhead (measured in §V), namely loading the output of the pioneer subnet from memory into GPU. The only by-product is that the implementation of this method must satisfy that the final selected subnet is a superset of the pioneer subnet, which means that the number of possible subnets for the given input will be reduced. However, we still support around $3.8 * 10^{13}$, which is enough to ensure that a best-fit final subnet can be found, and we will verify our accuracy improvement in our evaluation.

V. Evaluation

We implement OPA on top of PyTorch and extensively evaluate its effectiveness and efficiency under various settings. Our key results are summarized as follows.

- OPA improves Top-1 accuracy over the state-of-art approaches – 6.6% over Dynamic-OFA [15] and 9% over BranchyNet [11] – while meeting the latency requirement. The accuracy gain is considerable, given the fact that the accuracy increases by about only 7% from 2017 to 2021 [23].
- Such accuracy improvement is widely observed in extensive evaluations with various combinations of deployment factors. OPA achieves an accuracy gain up to 26% for a single data group when computing resources are saturated.
- The overhead of OPA is small and tolerable (no more than 20% compared to Dynamic-OFA).

A. Experimental Setup

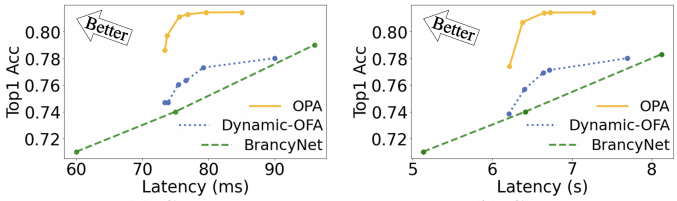
We deploy and evaluate OPA on the Nvidia TRX 3090, Intel Xeon CPU, with different latency constraints. We also consider several backend applications consuming different amounts of computing resources to showcase the hardware-adaptive nature of OPA. We use a 3-layer DNN to continually run the pioneer predictor as one backend application.

Models and dataset. We choose ImageNet 1K as the dataset for the showcase application of image classification. Note that ImageNet 1K contains image samples of 1K categories with different resolutions, reflecting different classification difficulty levels. We use the default training set for training. For testing, we randomly pick 40% of images for each category in the validation set, and in total, we consider 20K images. We use the pre-trained super-network (MobileNet V3, $width = 1$) by OFA [24] and randomly group 100 images for the evaluation. We use two measures – Top-1 accuracy and latency – to assess the performance of OPA.

Comparison with the state-of-the-art. We implement and compare with two state-of-the-art approaches in the literature: Dynamic-OFA [15] and BranchyNet [11]. Dynamic-OFA is the most recent work that considers the dynamics of computing power. It builds a look-up table storing the mapping between latency requirements and subnets. It leverages a real-time monitor (RTM) to monitor the computing power and update latency requirements. For a fair comparison, we build look-up tables on the training dataset based on our testbed. BranchyNet considers the adaptation of contents dynamics. It adds several early exit branches to the baseline network and will exit when the calculated score of the current exit branch's classification result exceeds a learned threshold. These exit thresholds are chosen based on the latency requirement. Thus, for different latency requirements, we need to learn different thresholds. We train BranchyNet with three exit branches whose latency are 70 ms, 80 ms, and 100 ms, respectively, on GPU.

B. System Performance

Overall. Fig. 7 reports overall comparisons between OPA and state-of-the-art. Each point represents the average accuracy and latency over all the data groups, and the selected subnets all satisfy the latency requirement. OPA consistently improves the trade-off between accuracy and latency by a significant margin. With similar latency, OPA achieves up to 6.6% accuracy gain over Dynamic-OFA and 9.0% accuracy gain over BranchyNet for both GPU and CPU. We would like to emphasize that the full model (namely, the super-network itself) may not achieve the highest accuracy for all data. OPA still outperforms the Dynamic-OFA by 4.4%, which selects the full model when the latency requirement is over 90 ms. This again confirms the need to consider the source data dynamics. When the latency requirements are between 73 - 80 ms for GPU and 6.3 - 6.7 s for CPU, OPA performs much better than Dynamic OFA, with at least a 5.6% accuracy improvement. When the latency requirement is small, there are fewer candidate subnets, leading to a smaller accuracy improvement (4.8%). The main reason for the outperformance is that OPA utilizes running a pioneer subnet (about 36.5 ms latency on GPU) to help find the best subnet under the real deployment condition, considering both computing power and source data. Instead, Dynamic-OFA only relies on the offline



(a) GPU (b) CPU

Fig. 7: Overall comparison of accuracy and latency trade-off.

Related Studies	Online		Offline
	Search time	Switch time	Training times
Dynamic-OFA	0.08 ms	17 ms	Once for all
BranchyNet	7.5 ms	—	Per latency requirment
OPA	18 ms	2.52 ms	Once for all

TABLE III: Comparison of the overhead.

profiled table and overlooks the dynamics of source data. Moreover, there is a gap between offline profiling and online condition; thus, it will lead to imperfect subnet selection with performance degradation.

In addition, compared to BranchyNet, OPA supports more subnets and conducts a finer-grained subnet search, while BranchyNet only supports tuning on the depth level. Correspondingly, there are more possibilities for OPA to find the optimal subnet to achieve high accuracy.

Overhead. We measure the average processing time of a batch on the GPU for different systems (Table III). At inference time, BranchyNet has the shortest processing time. It only supports a few subnets (3 in our settings), and its search cost is to calculate the entropy score for each exit branch to determine whether it needs to exit, which only takes about 7.5 ms for 100 images. Although its search cost is small, its accuracy performance is low. In terms of Dynamic-OFA, the switching time includes two parts: one is to load the batch norm for the selected subnet (2 ms), and another is to use RTM for monitoring and calculating (15 ms). It only needs to check the look-up table for searching, which takes little time. OPA introduces an overhead of about 18 ms for online search, and we apply inheritance to reduce the switching time. The only cost involved is to load the output of the last convolution layer of the pioneer subnet. Overall, the extra overhead is tolerable, which is no more than 20% compared to Dynamic-OFA.

Remarkably, OPA and Dynamic-OFA only need to train once, and they can produce entire trade-off curves with different points over the whole latency range, as shown in Fig. 7. However, for one latency requirement, BranchyNet has to learn new exit thresholds for all the exit branches. Thus, it needs to do experiments multiple times to find suitable thresholds for different requirements, which incurs a high training cost. We will next zoom in to see the performance results for each data group and show our benefits under different computing power. Note that BranchyNet has the lowest performance, and we will omit it from the following analysis.

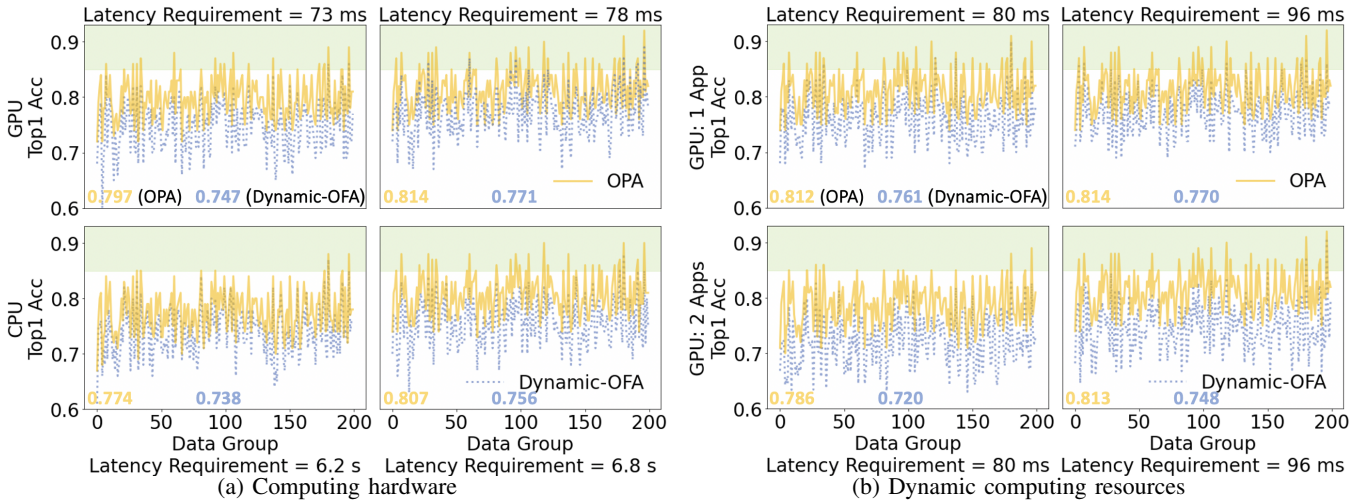
Performance under different hardware devices. Fig. 8a shows the Top-1 accuracy for two different hardware devices (GPU and CPU) with four different latency requirements. OPA outperforms Dynamic-OFA in all the deployment conditions (the yellow line is higher). We have two observations: first, a larger latency requirement gives more room to select a better subnet for each data group. Regardless of the hardware device, OPA achieves higher accuracy in all data with larger latency requirements. The improvements in the average Top-1 accuracy are about 2.1% and 4.3% for GPU and CPU, respectively. Second, GPU runs much faster than the CPU for comparable accuracy performance. And CPU may not be able to keep up with real-time video (or image) analytics. Specifically, when the device captures images at a high frame rate (e.g., 30 fps), the required processing speed should be at least 33 ms per image. However, each image requires at least 60 ms (6.2 s/100) for OPA. Not to mention running OPA directly on the capture device, the CPU performance of the embedded system would be even worse.

To better visualize the accuracy gain of OPA against Dynamic-OFA for a single data group, Fig. 9a shows the statistical results. OPA guarantees a positive accuracy gain against Dynamic-OFA. Specifically, given a sufficiently large latency requirement, the subnet selected by Dynamic-OFA via the offline table is consistent with OPA's (accuracy gain is 0) if the image is relatively easy to classify. However, when the image becomes difficult, our advantage will show up, and we achieve, at most, a 20% accuracy gain. In terms of average accuracy gain, OPA achieves around 6.8% improvement for a smaller latency and around 5.6% for a larger latency on GPU. Given such improvement, we confirm that OPA works well for all different hardware devices.

Performance under different available computing resources. To test OPA's adaptability to different available computing resources, we consider two workloads with different numbers of concurrent applications. Specifically, we run 1 or 2 applications at the backend. We only evaluate the GPU since the CPU has limited computing resources and is generally not chosen when multiple DNNs are running.

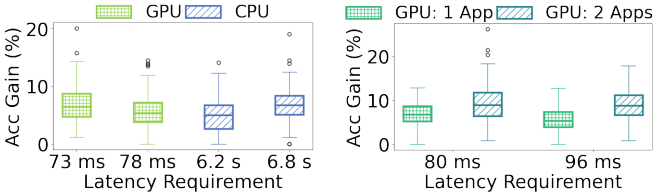
Fig. 8b displays the comparison results, from which we draw the conclusion: running multiple applications parallelly indeed affects subnet search performance by either scarifying the accuracy under the same latency requirement or requiring finding another subnet with higher latency to achieve the same accuracy. To be specific, running one backend application achieves 3.3% higher accuracy than two applications for a given latency requirement of 80 ms. Running one backend application with a 96 ms latency requirement has the same accuracy performance as running OPA alone with a 78 ms latency requirement. Both indicate that with more and more concurrent applications, the available computing power decreases, and the performance is degraded.

From Fig. 9a, we observe that OPA still achieves accuracy improvements against Dynamic-OFA even with different available computing resources. Particularly, when more applications are running, the accuracy gain will be greater. We obtain, at most, a 26% accuracy gain in the case of running two backend applications. Our approach performs better than state-of-the-art when computing resources are saturated. This is because Dynamic-OFA relies on RTM to



(a) Computing hardware (b) Dynamic computing resources

Fig. 8: Performance under different computing power. Green regions (accuracy $>$ 0.85) are for better comparison.



(a) Computing hardware (b) Dynamic computing resources

Fig. 9: Our accuracy gains under different computing power.

Batch size	2	50	100	150	200	500
Top-1 Accuracy	0.90	0.82	0.81	0.79	0.79	0.78
Overhead (ms)	10.5	0.42	0.21	0.14	0.11	0.042

TABLE IV: Accuracy and overhead results with varying batch sizes on GPU with a 76 ms latency requirement.

monitor the computing power and change the subnet with trial and error. Specifically, Dynamic-OFA measures the latency for the last data group to decide whether it violates the latency requirement and needs to switch subnets. However, it cannot guarantee switching to a subnet that could satisfy the current requirement and need to try multiple times. It may lead to an eventually unsatisfied latency or a switch to a subnet that is too small to get promising accuracy performance. Instead, OPA searches with a good starting point and guarantees quickly finding the best-fit subnet for a given deployment scenario.

Impact of the batch size. We empirically set 100 as the batch size for the above evaluation. Here, we show the impact of different batch sizes on performance (Table IV) and verify our selection. We calculate Top-1 accuracy for all the image data and the average overhead for each image. We observe that a smaller batch size yields better accuracy but higher overhead. This is because it allows finer-grained data adaptation. Specifically, when the batch size is 1, it indicates that for each image, we find its best-fit subnet. Obviously, it can achieve the highest accuracy. On the contrary, if all the data use the same model, it will lead to sub-optimal results (around 0.78). However, a smaller batch size means searching and switching to the best-fit subnet at a higher frequency. We have already shown that our cost for a one-time search and switch is about 21 ms per batch, and the high frequency will lead to considerable overhead in finishing all the tasks. Moreover, we find that when the batch size is 2, it will take 250x longer than when the batch size is 500. Thus, a trade-off is required, and we select 100, which has the minimum overhead with Top-1 accuracy over 0.80.

VI. Conclusion and Future Work

In this work, we present our attempt to efficiently deploy DNNs to run computer vision tasks under diverse and dynamic scenarios. We propose OPA, which uses the intermediate results of a shallow subnet to predict how other deeper subnets perform. We incorporate this idea into the "Train Once, Deploy Everywhere" paradigm. Extensive evaluations have validated its effectiveness and efficiency. OPA significantly outperforms the state-of-the-arts to improve accuracy while meeting the same latency requirement.

There are several remaining issues as our future work.

○ *Be more responsive.* All subnets supported by OPA use five stages. It takes at least 73 ms to run a batch of images on the GPU in our study. It implies that OPA does not fit a task that demands prompt responsiveness. We plan to support simpler subnets (say, with fewer stages/layers/neurons) to meet more stringent latency demands.

○ *Adoption with other DNN models and analytical tasks.* We admit that the selected super-network is not the one that yields the highest classification accuracy. There are new DNN models that exploit transformer (e.g., [25]) and graph neural networks (GNN) (e.g., [26]). In principle, OPA can be generalized to support more convolutions-based DNN models. We plan to extend our work to support these new models and more video analytics tasks in the near future.

Acknowledgment. We are grateful to anonymous reviewers for their constructive comments. This work was partially supported by NSF grant CNS-1750953.

REFERENCES

[1] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios," *Machine Learning with Applications*, vol. 6, pp. 100–134, 2021.

[2] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[3] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 398–11 407.

[4] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *IEEE/CVF international conference on computer vision (ICCV)*, 2019, pp. 1803–1811.

[5] P. Guo, B. Hu, and W. Hu, "Mistify: Automating dnn model porting for on-device inference at the edge," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2021, pp. 705–719.

[6] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[7] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, "Legodnn: block-grained scaling of deep neural networks for mobile vision," in *The 27th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2021, pp. 406–419.

[8] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," in *IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2020, pp. 2369–2378.

[9] Y. Wang, K. Lv, R. Huang, S. Song, L. Yang, and G. Huang, "Glance and focus: a dynamic approach to reducing spatial redundancy in image classification," in *Neural Information Processing Systems (NeurIPS)*, 2020, pp. 2432–2444.

[10] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "Bert loses patience: Fast and robust inference with early exit," in *Neural Information Processing Systems (NeurIPS)*, 2020, pp. 18 330–18 341.

[11] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 2464–2469.

[12] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1520–1530.

[13] J. Xin, R. Nogueira, Y. Yu, and J. Lin, "Early exiting bert for efficient document ranking," in *Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, 2020, pp. 83–88.

[14] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang, "Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 84–95.

[15] W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett, "Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 3110–3118.

[16] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[17] "Imagenet large scale visual recognition challenge 2012," https://image-net.org/challenges/LSVRC/2012/.

[18] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 119–135.

[19] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian, "Online model distillation for efficient video inference," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3573–3582.

[20] J. Guo, S. Xia, and C. Peng, "Vpplus: Exploring the potentials of video processing for live video analytics at the edge," in *IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, 2022, pp. 1–11.

[21] S. Reif, B. Herzog, J. Hemp, T. Hönig, and W. Schröder-Preikschat, "Precious: Resource-demand estimation for embedded neural network accelerators," in *First International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware (MLBench)*, 2020.

[22] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2021, pp. 81–93.

[23] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto *et al.*, "The ai index 2021 annual report," *arXiv preprint arXiv:2103.06312*, 2021.

[24] "Ofa pre-trained model zoo," https://github.com/mit-han-lab/once-for-all.

[25] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10 012–10 022.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.