A Two-Level Architecture for Deep Learning Applications in Mobile Edge Computing

Zao Zhang, Yuning Zhang, Dong Yuan, Wei Bao
{zao.zhang,yuning.zhang1,dong.yuan,wei.bao}@sydney.edu.au
Faculty of Engineering, The University of Sydney
Sydney, NSW, AU

ABSTRACT

As the deep learning models keep growing larger, it becomes more difficult to carry out highly accurate inference on mobile and Internet of things (IoT) devices with acceptable latency. By using edge computing, a popular approach to reducing inference latency is to partition the deep neural network (DNN) model and off-load part of the model inference to the edge server for processing. However, the partition approach still has some limitations, such as data transmission latency and inflexible model deployment. In this research, we propose a novel two-level inference architecture for deep learning applications in mobile edge computing. Instead of partitioning a large DNN model, we use small models to do two levels of inferences on edge devices and edge servers respectively. With our architecture, the on-device first-level inference is conducted in parallel with the feature uploading to the edge server and different types of second-level models can be flexibly deployed in edge servers according to their environments. Experiments demonstrate that our architecture can decrease computing costs and inference latency in complicated edge environments.

CCS CONCEPTS

• **Computing methodologies → Mobile agents**.

KEYWORDS

Mobile Edge Computing, Deep Learning, Computing Off-Loading, Accelerate Inference

1 INTRODUCTION

As a result of the rapid growth of deep neural networks (DNN), an increasing number of deep learning-based computer vision applications have emerged, such as mobile-based real-time video identification and online image classification. However, for mobile devices, computing resources and capacity are always insufficient for the majority of applications. Typically, there are two solutions for this type of issue: model compression or offloading. Methods for model compression can be subdivided into the following categories: pruning [19], quantization [18], and knowledge distillation[4]. Despite the success of many model compression techniques in reducing the DNN model size, it remains challenging to fulfil the requirements of edge devices and may result in a loss of accuracy. Numerous offloading strategies[7] [14]have also been developed to address the insufficient computational resources and capacity of edge devices [6]. However, offloading results in an additional issue. Transmission latency is very high, and result in unacceptable delays in real-time tasks[8]. Currently, researchers are conducting studies to offload inference duties to several edge servers [17]. Although the system's robustness will be greatly enhanced, transmission costs will increase as a result of offloading to several edge servers.

In addition, when multiple edge servers are present in the same edge environment, the distribution of inputs within the range covered by each edge server frequently varies. If there are numerous cameras on a street, for instance, the camera aimed at the main road will normally identify cars, whilst the camera aimed at the sidewalk will typically detect pedestrians. If the servers connected to the cameras in these different scenes are loaded with the same deep learning model, training the model will inevitably be more difficult, resulting in a more complex and computationally intensive model; if different servers are loaded with separate models for cars and for people, it is easy to miss anomalies such as pedestrians entering main roads and cars entering sidewalks, resulting in lower accuracy.

Another crucial fact is that the majority of models only consider a single output. Certain applications require two-level results to better accommodate usage scenarios[10]. In autonomous driving[9], for instance, the first-level result can return the deceleration request when it detects an item in front of the vehicle, and the second-level result can return the object's speed and instruct the car when to cease decelerating. The second-level model's inference and the car's deceleration can simultaneously reduce the total time cost. Another example is a shipping company that must organise various computer peripherals, such as the keyboard, mouse, and headset. First-level results can aid in categorising product categories, while second-level results can aid in categorising product brands. The inference of the second-level model and the subsequent operations of the first-level inference can operate concurrently to expedite the sorting of goods. In the context of mobile edge computing, a two-level result is more applicable to prospective applications. The result of the initial phase can be used to determine which mid-layer features should be offloaded to a certain edge server.

We presented a novel two-level inference architecture for deep learning applications to satisfy the aforementioned issues and requirements. Our two-level architecture is ideally suited for situations in which there are numerous distinct clusters in the same edge environment and the input distribution obtained by each cluster differs significantly. Fig. 1 depicts an overview of our two-level inference architecture. The input is initially exposed to first-level inference on edge devices, where the high-dimensional features of the input and the superclass confidence are obtained. For instance, an assignment involving the classification of vehicle models could involve the Ford Focus ST, Ford Fiesta ST, Ford Mustang, Toyota Corolla, Toyota Camry, etc. Ford and Toyota are superclass labels

MobiArch'22, October 21, 2022, Sydney, NSW, Australia
2022. ACM ISBN 978-1-4503-9518-2/22/10…$15.00
https://doi.org/10.1145/3556548.3559630

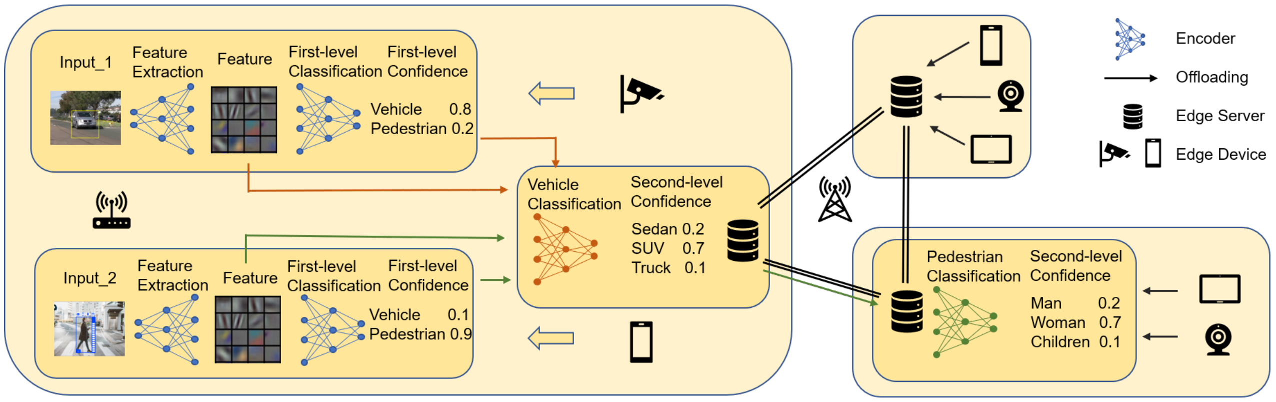

Figure 1: The Overview of Two-Level Inference Architecture for Mobile Edge Computing

in this classification task. The extracted high-dimensional features and superclass confidence will be offloaded to the closest edge server first; if no deep learning model that performs second-level classification for this superclass is loaded in this edge server, the feature will be transmitted to an edge server containing a specific deep learning model. After edge servers conduct second-level inference, the inference result is returned to edge devices. In Sec. 3, the specifics of our two-level inference architecture will be presented.

The main contribution of our paper are as follows.

(1) We propose a two-level architecture for off-loading DNN inference tasks of deep learning applications in the edge edge computing environment. We use small models to do two levels of inferences on edge devices and edge servers respectively, which reduces the inference latency by up to 42% compared to the model partition based approach.
(2) Our two-level architecture accelerates the DNN inference tasks by reducing up to 51% floating point operations(FLOPs) of the DNN model while maintaining the same level of inference accuracy. The on-device first level inference and the feature uploading are conducted in parallel, which further reduces the off-loading data transmission time.
(3) Our two-level architecture enables flexible models deployment to optimise the resource utilisation in mobile edge computing. By taking into account the various input distributions of the different scenarios, we can deploy the second level models flexibly in the edge servers according to the nature of their inference tasks.

2 CHALLENGE AND METHOD

To retrain the deep learning model for our two-level architecture and construct the inference system, several obstacles must be overcome.

(1) Not all data has an evident superclass label. If there are no existing superclass labels, we should build them using a representational learning strategy. These methods for learning representations can be clustering methods [12] or self-supervised learning methods [1]. We can produce superclass labels for individual input by grouping classes close to the same clustering centre into the same superclass. In Fig. 2, we display the two-dimensional representations of the CIFAR-10 and CIFAR-100 datasets. Points with different colours belong to different classes, the majority of points with the same colour can be grouped together, and these representations can be clustered into superclasses. Due to the existence of non-convergent points, we employ the median of each class's high-dimensional values for further clustering.

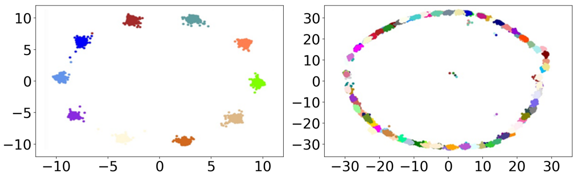

Figure 2: Two-dimensional representations of CIFAR dataset.

(2) To provide adequate accuracy, the extracted features must function well in two-level inference. To ensure the extracted feature can be used effectively in two-level inference, we must ensure the first-level deep learning encoder and second-level deep learning encoder have high consistency. Specifically, the second-level inference encoders of each super-class must have a large intra-class distance while maintaining a high degree of consistency with the first-level encoder. To solve this issue, we can retrain the pretrain model to acquire the first-level encoder and use distillation to decouple the same pretrained deep learning model to obtain several second-level inference encoders[15].
(3) A first-level inference error will cause a second-level inference to produce an incorrect conclusion[13]. In order to improve accuracy, we designed a recall mechanism for the inference system. If the confidence level of second-level inference is below a certain threshold, the result of this inference will not be accepted, and more second-level inference will be performed on superclasses with high confidence. Comparing the confidence levels of each second-level inference yields the final inference result.

The design and implementation of our two-level architecture will be introduced in multiple parts: the deep learning model, the deep learning model load strategy, and the inference algorithm.

3 DESIGN AND IMPLEMENTATION

The design and implementation of our two-level architecture will be introduced in several parts: deep learning model, second-level encoders load algorithm, and inference algorithm.

3.1 Deep Learning Model

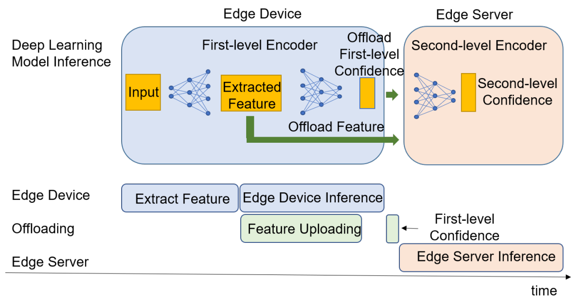

Figure 3: The Overview of Deep Learning Model Inference

The overview of deep learning model is shown in Fig. 3. The first-level encoder will extract the mid-layer embedding and offload it to an intra-cluster edge server. As described in Sec. 2, the high consistency between first-level encoders and second-level encoders ensures that extracted features function properly, hence reducing the computation cost through parameter sharing. The second-level encoder is loaded on the edge server based on the server's capacity and the empirical distribution of inputs. If the deep learning model for the predicted superclass is not available on the server, the extracted feature and first-level prediction will be transmitted to an edge server with the appropriate deep learning model loaded. Note that the extracted feature and the first-level prediction are independently offloaded. When the extracted feature is offloading, the first-level encoder inference continues to operate. This drastically decreased the entire time cost.

3.2 Model Load Strategy

Since we have many second-level encoders with the same number of superclasses, each edge server may not be able to load all second-level encoders simultaneously owing to memory constraints. Even if there is sufficient memory, the overhead of context switching in GPU is unacceptable[11], thus we will load the commonly used model in each cluster according to the empirically-based input distribution and capacity of the edge servers. Algorithm for model load is depicted in Alg.1. Due to the fact that all second-level encoders share the same network with the exception of the final prediction head, the memory cost and inference cost for all second-level encoders can be considered equivalent. The limitation of this technique is that the number of second-level encoders that can be loaded by all edge servers must be larger than or equal to the number of types of second-level encoders in order to demonstrate that every second-level encoder has been loaded at least once.

Algorithm 1 Model Load Algorithm

```
Input: Number of Clusters: $C$, Distribution in $c^{th}$ cluster: $D^c$,
Distribution in whole edge environment: $D$, Left GPU memory
capacity in $c^{th}$ cluster: $M^c$, Memory cost of each second-level
encoder: m, Number of second-level encoders: $N$
Output: Model Load Strategy.
1: while $\min(M^0, M^1, ..., M^C) > m$ do
2:   for each $n$ in $[1, N]$ do
3:     $I_n = \arg\max(D_n^0, D_n^1, ...D_n^C)$
4:     if $M^{I_n} > m$: then
5:       Load $n^{th}$ second-level encoder to $I_n$ cluster's edge server
6:       $M^{I_n} = M^{I_n} - m$
7:       $D_n^{I_n} = 0$
8:     else
9:       $D_n^{I_n} = 0$
10:      $I_n = \arg\max(D_n^0, D_n^1, ...D_n^C)$
11:      repeat line 4-7 until find available edge server
12:    end if
13:  end for
14: end while
```

3.3 Inference Algorithm

The inference method of our two-level design is depicted in Alg.2. All thresholds must be adjusted based on different tasks in order to determine the optimal hyperparameter. If the required second-level encoder is kept on many edge servers other than the edge server in the current cluster, a request to check the estimated waiting time should be sent to all qualified edge servers in order to balance computing resources.

4 EXPERIMENTS AND EVALUATION

In this section, we will describe the experimental setup, evaluation metrics, and the comparison between our two-level architecture and the standard offloading technique.

4.1 Experiment Setup

4.1.1 Deep Learning Model. Resnet[2] and Densenet [3] are utilised as pretrained backbone networks to demonstrate that our two-level architecture works well with widely-used deep learning models. Vision transformer has significant potential in the field of computer vision, but its model size is too large for edge devices, so it is not considered in our research.

4.1.2 Datasets. On CIFAR-10 and CIFAR-100 datasets[5], the two-level design is evaluated. The CIFAR100 dataset includes two levels of labels, but we will not utilize them. Because the superclass labels generated by our approach presented in Sec. 2 perform better.

4.1.3 Edge Devices. We employ Raspberry Pi 4 Tech Specs with Broadcom BCM2711, Quad core Cortex- A72 (ARM v8) 64-bit SoC @ 1.5GHz, 8GB LPDDR4-3200 SDRAM, and Gigabit Ethernet-equipped as edge devices.

4.1.4 Edge Servers. We utilize three edge servers: the first has two GeForce RTX 3090 GPU, an AMD Ryzen Threadripper 3970X

Table 1: Accuracy, FLOPs and parameter number of Backbone Models.

Model	CIFAR-10 Acc[%]	CIFAR-100 Acc[%]	FLOPs[G]	♯Parameters[M]
Resnet18	94.93	75.95	0.558	11.17
Resnet34	95.19	76.22	1.165	21.28
Resnet50	95.23	77.28	1.315	23.52
Resnet101	95.35	77.82	2.536	42.51
Resnet152	95.60	78.02	3.758	58.16
Densenet121	94.83	77.69	0.913	6.956
Densenet169	95.36	78.15	1.090	12.49
Densenet201	95.59	78.22	1.403	18.10

Table 2: Accuracy, FLOPs and parameter number of our two-level architecture.

Two-level Architecture	CIFAR-10 Acc[%]	CIFAR-100 Acc[%]	FLOPs[G]	♯Parameters[M]
Pretrained with Resnet18	95.25	76.69	0.827	27.96
Pretrained with Resnet34	95.43	77.63	1.585	47.51
Pretrained with Resnet50	95.61	78.18	1.846	53.44
Pretrained with Densenet121	95.56	78.32	0.983	11.27
Pretrained with Densenet169	95.67	78.73	1.283	24.32

Table 3: Inference time cost per input for single inference.

Architecture	Time Cost per Input[ms]
Partitioned Resnet50	5.2636
Two-level with Resnet18	3.0528
Partitioned Resnet101	8.7359
Two-level with Resnet34	5.1681
Partitioned Densenet169	15.7521
Two-level with Densenet101	14.2486

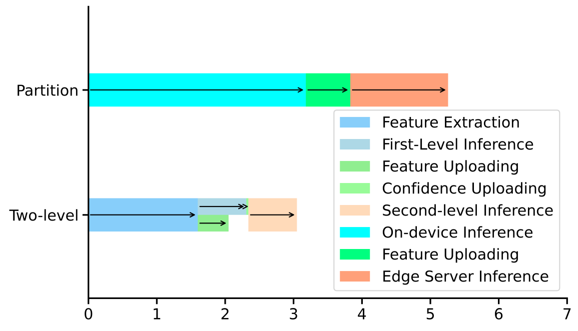

Figure 4: Inference time cost (ms).

32-Core Processor, 3.7GHZ CPU-F, and 100M/S bandwidth; the second has two RTX A5000 GPU, an AMD Ryzen Threadripper PRO 3945WX 12-Cores CPU, 4GHZ CPU-F, and 100M/S bandwidth; and the third has four GeForce RTX 2080 Ti GPU, two Intel(R) Xeon(R) Bronze 3104 CPU, 1.7GHZ CPU-F and 100M/S bandwidth. All servers operate on Ubuntu 18.04 with Linux kernel v4.15.0, PyTorch v1.7.0, and CUDA v10.2.

4.2 Results Evaluation

4.2.1 Model Level Performance. To eliminate the performance variance induced by hardware, we begin with a model comparison. Tab. 2 displays the accuracy, FLOPs, and overall parameter number of our two-level architecture. Comparing our two-level inference architecture to the benchmarks in Tab. 1, it is evident that it achieves superior performance. The two-level architecture trained with Resnet18 demonstrated greater accuracy on both datasets than Resnet50 with 63 % FLOPs. Similarly, the performance of our two-level architecture pretrained with Resnet34, Resnet 50, Densenet121, and Densenet169 was superior to Resnet101, Resnet152, Densenet169, and Densenet201. This demonstrated that our superclass label generation and decoupling strategy was successful in ensuring consistency between first-level and second-level encoders.

4.2.2 System Level Performance. Following the comparison at the model level, we can now display the comparison at the system level. All outcomes are evaluated using CIFAR-100 datasets. Our two-level architecture is implemented using the technique outlined in Sec. 3. There are three clusters, and each cluster has one edge server. There are five superclasses, which is the optimal number for CIFAR-100 datasets. The input distribution within each cluster is produced at random, but we presume we are familiar with the distribution based on our prior experience. Moreover, standard partitioning and offloading methods are used to evaluate pretrain models[16]. In the standard offloading method, the edge device sends requests to edge servers to determine the estimated waiting time of each server and then offloads the mid-layer feature map to the server with the shortest estimated wait time. All Resnet-based models are partitioned after the *conv*3*x* block and all Densenet-based models are partitioned after the *transition*2 block in order to ensure a fair comparison. From Fig. 4, we can find we successfully reduce the time cost by two-level design. Tab. 3 and Tab. 4 reveal that our two-level architecture greatly reduces the inference cost by up to 42% for single inference and up to 60% when edge servers are full-loaded.

Algorithm 2 Inference Algorithm

```
Input:Input data:$I$, Upper threshold of first-level encoder confidence: $T^L_{max}$, Lower threshold of first-level encoder confidence: $T^L_{min}$, Lower threshold of second-level encoder confidence$T^H$
Output:Classification result: $O$
 1: Feed $I$ into the first-level encoder to extract mid-layer feature map $FM$
 2: Offload $FM$ to edge server in cluster
 3: Wait first-level inference finished and offload first-level prediction confidence sequence $C^L$ to edge server in cluster
 4: if $\max(C^L) >= T^L_{max}$ then
 5:     $n = \arg\max(C^L)$
 6:     if $n^{th}$ second-level encoder is loaded in this edge server then
 7:         Feed $FM$ into $n^{th}$ second-level encoder to obtain $C^H_n$
 8:         $O = \arg\max(C^f_n)$
 9:     else
10:         Offload $FM$ and $C^L$ to the edge server which $n^{th}$ second-level encoder is loaded
11:         Feed $FM$ into $n^{th}$ second-level encoder to obtain $C^H_n$
12:         $O = \arg\max(C^f_n)$
13:     end if
14: else
15:     Declare empty list $L$
16:     while $\max(C^L) >= T^L_{min}$ do
17:         Append $\arg\max(C^c)$ to list $L$
18:         $C^L_{\arg\max(C^L)} = 0$
19:     end while
20:     $P = \min(P_{max}, length\ of L)]$
21:     $L = L[0 : P]$
22:     Offload $FM$ to edge servers which have loaded second-level encoders in list $L$ to obtain $C^H_{L_0}$ to $C^H_{L_{P-1}}$
23:     $O = \arg\max(C^H_{L_0}, C^H_{L_1}, ..., C^H_{L_{P-1}})$
24: end if
25: return $O$
```

Table 4: Average inference time cost when all edge servers are full-loaded(GPU-Util=99%).

Architecture	Time Cost per Input[ms]
Partitioned Resnet50	0.2582
Two-level with Resnet18	0.1027
Partitioned Resnet101	0.4129
Two-level with Resnet34	0.2031
Partitioned Densenet169	0.3751
Two-level with Densenet101	0.3468

5 CONCLUSION AND FUTURE WORK

In this research, we propose a two-level architecture for mobile edge computing deep learning applications. Complex mobile edge environments with several clusters and widely variable inputs per cluster are the most appropriate application environment for our two-level architecture. By decoupling the pretrained deep learning model into distinct superclasses, we were able to minimize the deep learning model's size and compute cost, hence accelerating inference. And we designed a two-level architecture to allow the first level of inference and feature offloading to work in parallel, hence further accelerating inference. Although the memory cost will increase slightly, this issue was effectively resolved by constructing a system with multiple edge servers.

In the future, we would like to concentrate on representation learning techniques that improve the embedding capability of decoupled deep learning models. In addition, we would like to try to find better model load technique to reduce the memory cost of our two-level architecture.

6 ACKNOWLEDGEMENT

The first author also would like to acknowledge the scholarship supported by China Scholarship Council.

REFERENCES

[1] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems* 33 (2020), 9912–9924.

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

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

[4] Deyi Ji, Haoran Wang, Mingyuan Tao, Jianqiang Huang, Xian-Sheng Hua, and Hongtao Lu. 2022. Structural and Statistical Texture Knowledge Distillation for Semantic Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16876–16885.

[5] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[6] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[7] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376.

[8] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th annual international conference on mobile computing and networking*. 1–16.

[9] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* 107, 8 (2019), 1697–1716.

[10] Dario Sabella, Alessandro Vaillant, Pekka Kuure, Uwe Rauschenbach, and Fabio Giust. 2016. Mobile-edge computing architecture: The role of MEC in the Internet of Things. *IEEE Consumer Electronics Magazine* 5, 4 (2016), 84–91.

[11] Kun Suo, Yong Shi, Chih-Cheng Hung, and Patrick Bobbie. 2021. Quantifying context switch overhead of artificial intelligence workloads on the cloud and edges. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 1182–1189.

[12] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. 2016. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision*. Springer, 499–515.

[13] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. 2015. HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition. In *Proceedings of the IEEE international conference on computer vision*. 2740–2748.

[14] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2021. Deep Compressive Offloading: Speeding Up Edge Offloading for AI Services. *GetMobile: Mobile Computing and Communications* 25, 1 (2021), 39–42.

[15] Fuxun Yu, Zhuwei Qin, and Xiang Chen. 2018. Distilling critical paths in convolutional neural networks. *arXiv preprint arXiv:1811.02643* (2018).

[16] Shuai Yu, Xin Wang, and Rami Langar. 2017. Computation offloading for mobile edge computing: A deep learning approach. In *2017 IEEE 28th Annual International*

Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC). IEEE, 1–6.

[17] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 201–214.

[18] Yunshan Zhong, Mingbao Lin, Gongrui Nan, Jianzhuang Liu, Baochang Zhang, Yonghong Tian, and Rongrong Ji. 2022. IntraQ: Learning Synthetic Images with Intra-Class Heterogeneity for Zero-Shot Network Quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12339–12348.

[19] Weiqi Zou, Yang Wang, Xueyang Fu, and Yang Cao. 2022. Dreaming To Prune Image Deraining Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6023–6032.