2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON) | 978-1-6654-4108-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/SECON52354.2021.9491600

Cut, Distil and Encode (CDE): Split Cloud-Edge Deep Inference

Marion Sbai, Muhamad Risqi U. Saputra, Niki Trigoni and Andrew Markham
Department of Computer Science
University of Oxford
United Kingdom
{marion.sbai,muhamad.saputra,niki.trigoni,andrew.markham}@cs.ox.ac.uk

***Abstract*—In cloud-edge environments, running all Deep Neural Network (DNN) models on the cloud causes significant network congestion and high latency, whereas the exclusive use of the edge device for execution limits the size and structure of the DNN, impacting accuracy. This paper introduces a novel partitioning approach for DNN inference between the edge and the cloud. This is the first work to consider simultaneous optimization of both the memory usage at the edge and the size of the data to be transferred over the wireless link. The experiments were performed on two different network architectures, MobileNetV1 and VGG16. The proposed approach makes it possible to execute part of the network on very constrained devices (e.g., microcontrollers), and under poor network conditions (e.g., LoRa) whilst retaining reasonable accuracies. Moreover, the results show that the choice of the optimal layer to split the network depends on the bandwidth and memory constraints, whereas prior work suggests that the best choice is always to split the network at higher layers. We demonstrate superior performance compared to existing techniques.**

***Index Terms*—Edge computing, deep learning, tasks partitioning, CNN.**

I. Introduction

In the past decade, we have witnessed the increasing development of edge computing operating on a wide range of devices including mobile phones, home intelligence devices, robots, drones, and sensor networks. These devices, referred to as *edge devices*, are physical devices equipped with sensing, computing, and communication capabilities. They typically operate continuously, collecting a large amount of data including images, videos, audio, and sensor data with the ultimate goal of providing actionable information to the user or system. Concomitantly to the emergence of the edge computing paradigm, deep learning has become one of the most dominant data analytic approaches due to its capability to achieve impressively high accuracies on a variety of key computing tasks such as classification [1], scene understanding [2], speech recognition [3], and face detection [4]. However, neural networks are known to be very expensive in terms of memory, computation and energy, making their execution by edge devices challenging [5].

The traditional approach to alleviate the distributed deep learning problem is to offload all computing tasks to distant and powerful cloud servers by sending them the raw sensor data. This solution has limitations, most importantly high latency and network load due to limited bandwidth, impacting the performance of real-time applications. Moreover, sending large amounts of sensor data is very energy-expensive for limited battery powered end devices, making this solution impractical for long period operations. Other issues including leaking user privacy (e.g., raw data collected at edge devices may contain sensitive and private information about individuals), and suffering from unpredictable delays.

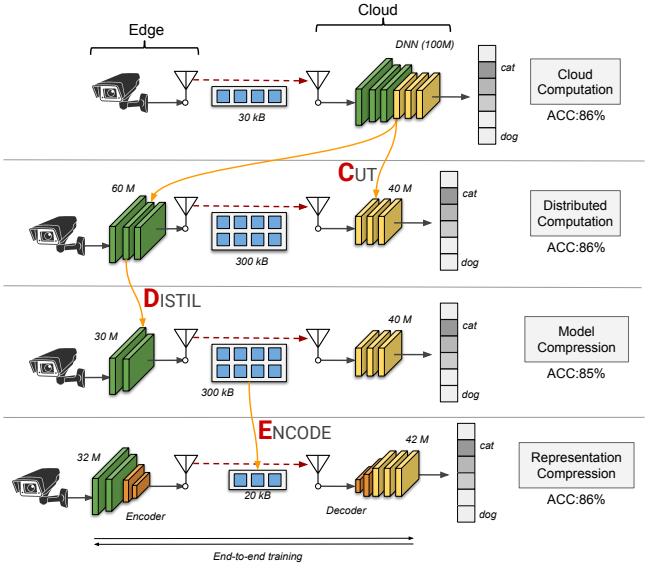


Fig. 1: The overall process of Cut, Distil and Encode (CDE). In the **Cut** phase, the network is partitioned at a particular layer. In the **Distil** step, the edge-based sub-network is optimized using knowledge distillation, reducing memory and computational load on the edge. Lastly, in the **Encode** step, the intermediate network data is compressed, reducing bandwidth.

As a result, the design of compact models is an active topic of deep learning research, which is mainly centered around mobile phones as the target platform. Such solutions certainly eliminate the issue of congestion in the network, but they are greatly limited by the amount of computation and storage resources that can be included on the mobile device, mainly for power and cost reasons, as discussed in Section

978-1-6654-4108-7/21/$31.00 ©2021 IEEE

TABLE I: Some off-the-shelf microcontroller development platforms with their SRAM and flash storage constraints [7]. Note that we list the microcontrollers with at least 250 KB SRAM, though other examples such as Cortex M0 only have 20–30 KB SRAM.

MCU platform	Processor	Frequency	SRAM	Flash
ATSAM4E16E [8]	Cortex-M4	120 MHz	128 KB	1 MB
FRDM-K64F [9]	Cortex-M4	120 MHz	256 KB	1 MB
STM32 F723 [10]	Cortex-M7	216 MHz	256 KB	512 KB
Nucleo F746ZG [11]	Cortex-M7	216 MHz	320 KB	1 MB
NXP i.MX1060 [12]	Cortex-M7	600 MHz	1 MB	External

2. These difficulties drastically increase for edge devices that have extremely limited resources such as low-end IoT devices, which may not be capable of executing an entire Deep Neural Network (DNN) model locally. These devices are often equipped with microcontrollers (MCUs), which have the advantages of being cheap, widespread and geared towards energy-efficient workloads. However, an MCU typically consists of a low-frequency processor and only several hundred kilobytes of on-chip memory [6], as shown in Table I.

Typical microcontrollers have extremely limited on-chip memory (100–320 KB SRAM) and flash storage (256 KB–2 MB). The entire DNN model with its weight parameters and code has to fit within the small memory budget of flash storage. As a comparison, for a classification task, VGG16 [13] has a size of 553.16 MB and even a model like MobileNetV1 [14] which is specifically targeted to mobile platforms has a size of 17.3 MB, which clearly exceeds the memory capacity of MCUs. Therefore, neural networks targeting microcontrollers are specific to limited applications such as binary classification (e.g. Visual Wake Word [7]). Aside from these resource constraints, there may be other restrictions preventing full execution of neural networks at the edge, such as trade secrets: an analytics service or an app provider might not be keen on sharing its valuable and highly tuned model. Therefore, it is not always possible to assume local processing is a viable solution even if the task duration, memory, and processing requirements are not critical for the user, or the task can be performed when users are not actively using their devices (e.g., when the device is being charged overnight) [15].

Considering these drawbacks, an alternative option is instead to distribute the inference of machine learning algorithms between the edge and the cloud. In an ideal world, this has the advantage of reducing the communication load since the raw data are directly processed on the edge by a subnetwork whilst at the same time maintaining the model accuracy by executing the remaining, resource-intensive parts of the network (e.g. fully connected layers), on the cloud. This distributed computing scenario has resulted in a research problem that is both complex and of great practical interest. In the context of distributed DNN for example, it consists of determining a 'good' partitioning of neural network layers between the edge and the cloud.

Towards this goal, several key aspects must be taken into account. The first one is the memory capacity of the edge device because, as said earlier, the model must fit the memory of the edge device. A second important aspect is the size of the intermediate data. The sizes of intermediate results generated out of each layer decrease from lower layers to higher layers. As a result, partitioning at lower layers would generate larger sizes of intermediate results, which could increase the transmission latency. Moreover, there is another important difficulty: the *data amplification* problem. This problem refers to the fact that the output data of the intermediate layers of a DNN are typically much larger than the original image. A possible approach to alleviate this problem is by compressing the data produced by the last layer of the network, but this will cause a loss of precision. To limit this effect, previous works proposed to split the network in the last layers, just before the computationally intensive fully connected layers [16]. This solution, where most of the network execution is left to the edge device, is not feasible for devices such as MCUs. Table II shows that the memory sizes of VGG16 and MobileNetV1 when splitting at different layers are generally bigger than the capacity of MCUs.

TABLE II: Memory size required at the edge when splitting MobilenetV1 and VGG16 at different layers.

Model	Layer	Mem. size
MobileNetV1	28	21KB
MobileNetV1	41	662KB
VGG16	7	1MB
VGG16	11	7MB
VGG16	15	29MB

Hence, there is a need to introduce into these systems compression techniques offering better performance at reducing the size of the transmitted data while minimising the loss of precision and also taking into account the constraints of the edge device. This paper proposes a novel distributed DNN which distributes the network inference between the edge and the cloud. The key to our approach is a fusion between auto-encoder and knowledge distillation which enables end-to-end training of the network while respecting the computational resources available on the edge and the bandwidth provided by the networking protocols (as described in Fig. 1). In particular, knowledge distillation is utilised to compress the subset of the networks that run on the edge such that real time inference can be achieved while at the same time maintaining the overall neural networks accuracy by transferring the knowledge from the original network (teacher) to the distilled network (student). More precisely, the novel contributions are the following:

- We propose the first neural network partitioning approach considering both resource constraints at the device and bandwidth constraints.
- Towards this goal, we introduce a novel model by jointly training knowledge distillation at the edge and auto-encoder. The proposed method enables to partition the neural network execution with no accuracy drop while respecting important memory and bandwidth constraints.
- We provide an unexpected insight regarding the behaviour

of the network: our results show that it is not always beneficial to push as much computation as possible on the device. Under some conditions, it may be better to split the network at lower layers for a better accuracy with an identical communication cost, even if the device can afford more computation.

II. RELATED WORK

A. Distributed Computing Hierarchy

It is widely known that processing image and video data by DNNs requires intensive computation [17]. In this section, we provide an overview of distributed computing hierarchy, particularly seen from the perspective of DNN deployment.

1) Cloud Computing: Traditional cameras are typically not equipped with enough computational resources for processing large video data, especially when latency and energy have to be respected. A common solution to this problem is to transmit and offload all the data to be processed by the cloud as seen in [18] and [19]. This data must be compressed locally on the camera beforehand and then sent over a channel with limited bandwidth. Unfortunately, the offloading of this massive video data results in very high latency and energy costs. In addition to this disadvantage, the appearance of artefacts on compressed images (videos) involves risks of losing their semantic content, and in any case considerably reducing the performance of the DNNs [20].

2) Edge Computing: The other solution is to perform the entire computation on the edge devices (e.g. mobile phone, camera equipped with micro-controller). This technique is motivated by the very high cost generated by the transfer of a mass of data from the edge to the cloud. Nevertheless, executing DNNs on the edge devices remains a challenge. Conventional DNN architecture (e.g. AlexNet, VGGNet, ResNet, etc.) typically require specialised hardware (such as GPU) to speed up calculations and operate in real time on high definition videos [21] [41]. This is highly power consuming and not feasible for applications that continuously need to perform computations. SSDLite [22], a mobile version of Single Shot Detector (SSD) [23], requires 15 times more computation time to process a high resolution video even when it is optimised for edge devices [20]. MCUNet [24], despite a remarkable compression rate which enables ImageNet inference on microcontrollers, can only reaches around 70% top1 accuracy.

3) Distributed Computing: A compromise between performing computation entirely on the edge or only on the cloud has just emerged in the last years [16], [20], [25], [26]. In the literature, several works such as [27] and [28] have attempted to distribute the computational load between the edge and the cloud. However, most of them suffer from the phenomenon of data amplification even after reducing the size of the intermediate data [29], [26]. [16] presents a solution for offloading the computation of the conventional DNNs between the edge and the cloud by providing a natural sharing point at their fully connected (FC) layers. In that sense, the distributed model will not suffer from data amplification with the same degree as conventional DNNs. Nevertheless, the need for data compression before transmission has not been addressed. The authors of [30] propose a collaborative cloud-edge computing framework to optimally deploy neural network model into the edge nodes. An adaptive task scheduling algorithm is designed to adaptively optimize the task assignment by using the improved ant colony algorithm. The simulation results show that the framework can reduce time delay and energy consumption, and improve task accuracy in comparison to the cloud-only and to the edge-only scenarios. However, the neural network is processed blindly and no optimisation is done to remedy the data-amplification issue and the memory usage at the edge. [26] introduced lossless compression (run-length encoding combined with Huffman encoding) on the data sent to the cloud, which allows a compression rate ranging from 3 to 10. However, this data is still bigger than the original video stream. In order to partition the DNN without amplification of data, the authors of [20] present a compression technique composed of three steps: quantization, factorization and coding. This compression technique allows one to obtain intermediate data which is up to 7x smaller than the original video stream. Despite the interesting results obtained via this approach, the authors point out their use of a traditional compression technique. It would be interesting to think of using a more advanced compression technique such as techniques based on auto-encoders (AE). Moreover,the memory capacity at the edge is not considered to find the optimal partitioning point.

B. DNN compression

In the last few years, great progress has been made in the area of DNN compression. Recent advanced techniques for squeezing and accelerating DNNs are generally classified into four groups: network pruning, quantization, decomposition, and knowledge distillation. Network pruning removes redundant and uninformative weights in the network [31]–[34]. Quantization compresses the network parameters in order to reduce the size of the weight representation [35], [36]. By restricting weights to only two values, binary networks can drastically reduce computing time and memory consumption, but the accuracy is also significantly reduced [37]. The decomposition technique reduces the complexity of the network by introducing low-level constraints on the network weights (low-rank decomposition) [38] and [39]. By using this factorization, the number of multiplications is considerably reduced. However, the obtained compression ratio is generally lower than that obtained by the pruning-based approach [40]. Moreover, the low-level constraint imposed on the network may have a negative impact on the network performances. Knowledge Distillation (KD) technique is an approach that transfers knowledge from a large network of neurons, called teacher, to a small network called student. We find encouraging results in the literature with networks of students having finer and deeper architectures (e.g. Fitnet) [40], [41]. This approach is based on step wise training by training the first part of the student before training the entire network. This can be adapted to optimize memory usage at the edge device.

TABLE III: The size of intermediate data when splitting MobileNetV1 and VGG16 at different layers.

Model	Layer	Size of intermediate data
MobileNetV1	28	392KB
MobileNetV1	41	191.5KB
VGG16	7	2MB
VGG16	11	784KB
VGG16	15	392KB

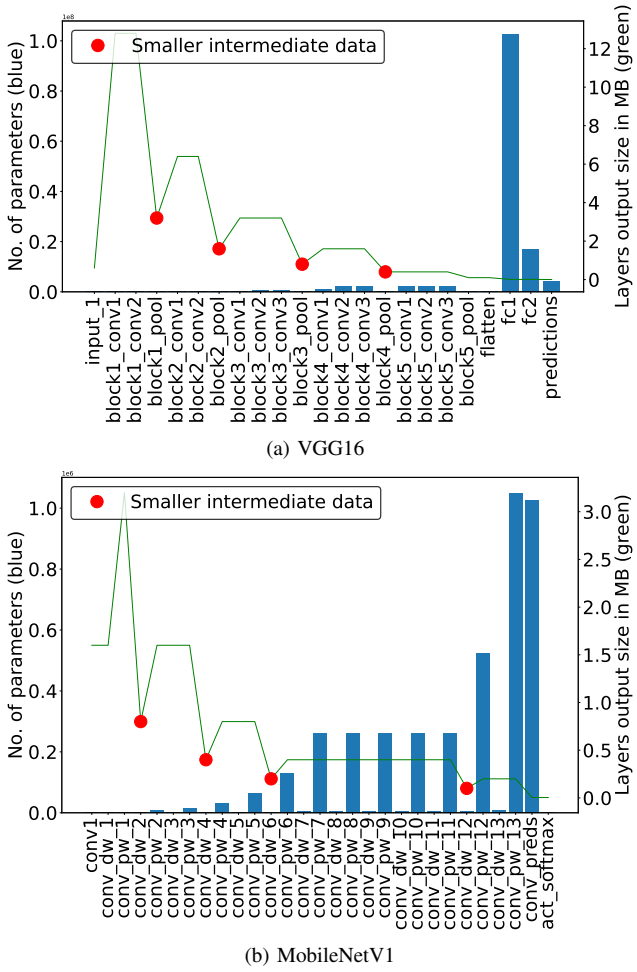
III. METHODS

A. Motivation

Dividing the inference of the DNN between the node and the cloud at the most fundamental requires careful choice of the layer at which to split, which must satisfy certain constraints. It must represent the best compromise regarding memory, bandwidth and precision. As shown in Fig. 2 (a) and Fig. 2 (b), the curves representing the output data sizes of each layer, whether for MobileNetV1 or VGG16 networks, have easily recognizable inflection points. These points correspond to the layers for which the output data sizes are local minima of the curves. This observation restricts the optimization of search space and helps choosing the suitable layers for network splitting. However, as illustrated in Table III, even when choosing these layers to cut the networks, the output size of the intermediate data is too high for real-time applications with limited bandwidth such as LoRa (4.68KB/s) or even 3G (230KB/s). Moreover, as explained in Section 1 in Table II, the size of the first part of network is still big for edge devices with memory capacity limited to few KB or MB. Therefore, we designed a framework aiming at optimising these different constraints: memory size at the edge, size of the transmitted intermediate data, and the model accuracy. We focus this study on the challenging trade off between memory and bandwidth. A further study into a trade off that incorporate computation time and energy is an interesting direction for future work.

B. Proposed architecture

The proposed architecture is composed of three main parts (as seen in Fig. 3) which are motivated by the constraints presented in the previous section, and which are trained end-to-end as explained in the next section. The first part is the part which is executed on the edge device. This part is compressed using the knowledge distillation technique (KD) in order to fit the memory constraints of the device. The second part, at the center of the architecture, is an auto-encoder (AE) aiming at reducing the size of the intermediate data, like a bottleneck. This is an under complete convolutional auto-encoder using ReLU as activation function after each layer. The first block of the auto-encoder consists of layers whose size gradually decreases (encoding). It is followed by a second block of layers, which are increasing the size of the data to match the size of the second part of the network (decoding). Therefore, the size of the intermediate data which is sent across the network is determined by the size of the auto-encoder's code. The execution of the auto-encoder is split between the edge



(a) VGG16

(b) MobileNetV1

Fig. 2: The size of output intermediate data and number of parameters of each layer for (a) VGG16 and (b) MobileNetV1. Inflection points (potential splitting points) are marked in red.

and the cloud. The third part simply is the remainder of the original network, which is executed on the cloud.

Therefore, we proposed an architecture based on the distillation of the part executed on the edge only, and preserving the cloud part without distillation. Indeed, as resource constraints (memory, computation, energy) being much more important at the edge than in the cloud, it is not useful to distill the whole network and incur a decrease in accuracy. We can expect several advantages from this approach. Compressing part of the network reduces the local memory consumption considerably while the auto-encoder reduces the amount of data transmitted to the cloud. The end-to-end training described in the previous section aims at maintaining an acceptable accuracy rate (which might even improve in some cases) by jointly training these two optimisations. Furthermore, the distillation technique is generalisable and applicable on several types of networks: MobileNet, VGGNet, ResNet, etc., without requiring bespoke edge architectures to be created. Finally, the distillation technique and the auto-encoder are flexible, on the same network, in depth (number of layers) and in width (number of filters), which allows adaptation with respect to

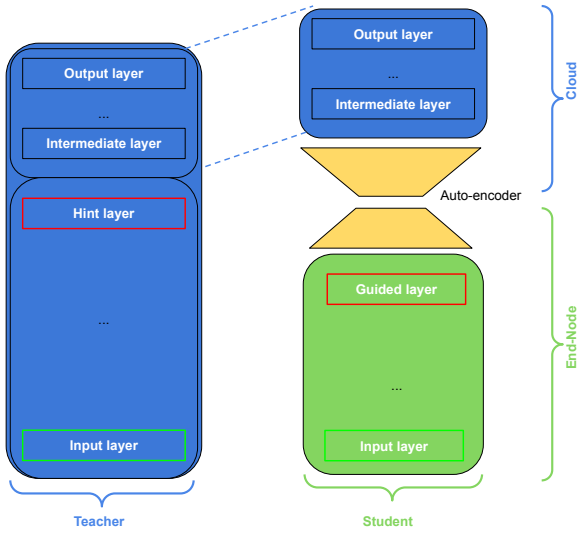


Fig. 3: The distributed DNN architecture.

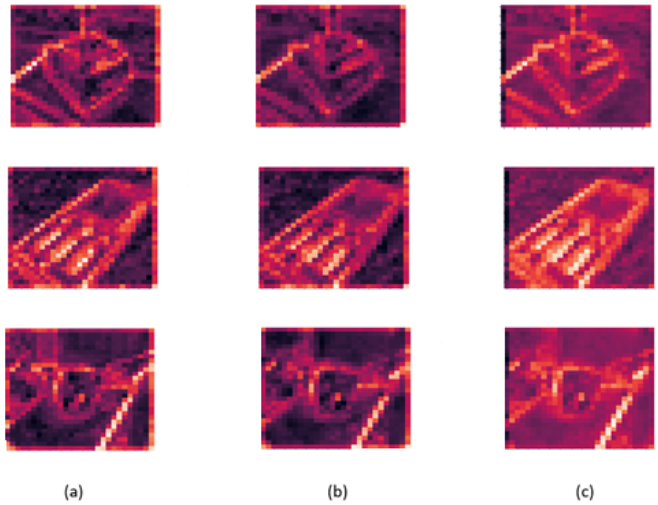


Fig. 4: Visualisation of the output latent representation from: (a) the hint layer, (b) the guided layer, and (c) the auto-encoder's output. Note that our auto-encoder can reconstruct the hint/guided layer's representation accurately.

the bandwidth and memory capacity at the edge.

C. Learning Methods

In order to train the whole model, we firstly need to train the small student network S such that the output of the intermediate layer of S is similar to the output of the intermediate layer of the teacher network T. We achieve this by employing a Hint Training (HT) procedure [40], [41] such that S can mimic the T's capability to generate useful latent representation for the task. In HT, a *hint* is defined as a layer in T that is utilized to guide a *guided* layer in S. In other words, the guided layer is trained to reproduce the same output as the hint layer. The objective function is the mean-squared error between the output of these two layers. More specifically, let $\mathbf{W}_{hint}$ and $\mathbf{W}_{guided}$ be the weights of T and S up to their hint and guided layers respectively. Then, given $\{\mathbf{I}_i, i = 1, 2, ..., n\}$ as the n number of training data, the network will be trained by using the following objective function

$$\mathcal{L}_{KD} = \frac{1}{n}\sum_{i=1}^{n} \left\| \Phi_T(\mathbf{I}_i; \mathbf{W}_{hint}) - \Phi_S(\mathbf{I}_i; \mathbf{W}_{guided}) \right\|^2, \quad (1)$$

where Φ_T and Φ_S are T's and S's DNN functions up to their respective hint or guided layers. Note that the splitting points of Φ will be different between T and S.

After training the student network in the edge device, we compressed S's latent representation by using Auto-Encoder (AE) in order to respect the desired communication requirements (e.g., 3G/LoRa). In that sense, AE will be trained to generate a compressed S's representation while at the same time should be able to reproduce the original S's representation to be consumed by the part of the network in the cloud device. Let $\mathbf{W}_{AE}$ be the weight of the AE, then in the second stage, the network is trained by using this loss function

$$\mathcal{L}_{AE} = \frac{1}{n}\sum_{i=1}^{n} \left\| \Phi_S(\mathbf{I}_i) - \Psi_S(\Phi_S(\mathbf{I}_i); \mathbf{W}_{AE}) \right\|^2, \quad (2)$$

where Ψ_S is the AE part of S. The example of the student's and the auto encoder's output latent representations after training can be seen in Fig. 4.

Finally, to improve the model performance, we can optionally fine tune the whole network in an end-to-end manner by minimizing the task specific objective function (e.g., classification, regression) using the following objective function

$$\mathcal{L}_{task} = \frac{1}{n}\sum_{i=1}^{n} \left\| \Omega_S(\mathbf{I}_i; \mathbf{W}_S) - \hat{y}_i \right\|^2, \quad (3)$$

where Ω_S, $\mathbf{W}_S$, and $\hat{y}$ are the full S's DNN layers, the full weight of S, and the true label respectively.

Regarding the implementation, we performed the training process on the cloud and copied the student and the encoder models to the edge. About the different settings we used during training (e.g., splitting layer), we chose as hint layers the layers which are optimal in terms of size of intermediate data as illustrated in Fig. 2 (e.g., layers 28 and 41 for MobileNetV1, and layers 7, 11 and 15 for VGG16). The training details are described in Section IV-A.

IV. Experiments

A. Implementation Details

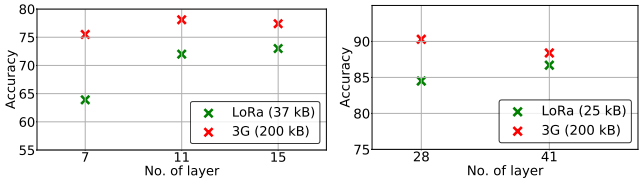
We implemented our model using Keras framework and the model was trained and tested on GPU from Google Colab. For all experiments, we used a subset of ImageNet dataset (700 classes) [42] with medium image definition (224x224).

To demonstrate the effectiveness of our methods, we employed VGG16 [43] and MobileNetV1 [14] with the baseline accuracy 70% and 86.5% respectively in the used dataset. Although MobileNet can run locally on a smartphone, it is still important to optmise it in the context of multiple applications running simultaneously. Moreover, optimising MobileNet makes it possible to use it at a device with more stringent constraints, such as MCUs.

In the first stage of training, we trained the student network (the lower part of the network deployed in the edge) with Adam optimizer [44] for up to 500 epochs. We used $1e-4$ as learning rate. In the second stage of training, we trained the auto-encoder to compress the output of the student network by using Adam optimizer with learning rate 0.005 for up to 500 epochs. Finally, we bring together the student, the auto-encoder, and the upper-part of the teacher, and jointly trained them using Adam optimizer with $1e-6$ learning rate for up to 100 epochs. For all stages, we divided 20% of the training data as the validation set and used 32 as the batch size. We performed more than 270 end-to-end trainings with different architectures and training parameters. We present the more significant results in the next section.

B. Sensitivity Analysis

1) The Impact of Cutting at Different Layers: In this experiment, we study the impact of accuracy when cutting the network at different layers. Given the communication constraint for 3G (200KB) and LoRa (25KB), we want to see at which layer the best performance are obtained. Note that we only cut a layer that has a better chance to respect the communication constraints as we described in Fig. 2. The results for this experiment are illustrated in Fig. 5. While previous works [20] suggest that it is better to split the network as far as possible from the input (because the intermediate data is smaller), especially when we do not have any constraint on the edge capacity, the result from Fig. 5 indicates that this depends on the bandwidth constraint. For a LoRa constraint (25KB), VGG16 and MobileNetV1 produces the best performance when we cut the network at layer 15 and 41 respectively, whereas for a 3G constraint (200KB), better results are obtained when splitting these networks at layers 11 and 28 respectively, not the upper ones. This results are quite unexpected, because we could think that as the size of the intermediate data decreases, we can always obtain a better accuracy when cutting the network at higher layers. These results could be explained by the fact that knowledge distillation produces optimal results when the hint layer is chosen in the middle layers. Another interesting remark from this experiment is that even if we use LoRa bandwidth (25KB) as the communication constraint, MobileNetV1 still produces accurate classification, very close to those obtained with the 3G communication constraint. This is because the MobileNetV1 model contains many normalisation layers which could potentially improve the generalization capability even when we introduce a bottleneck layer.

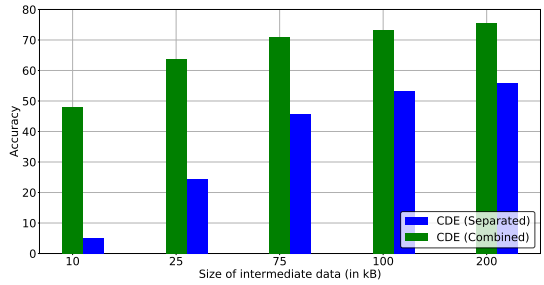


(a) VGG16 (b) MobileNet

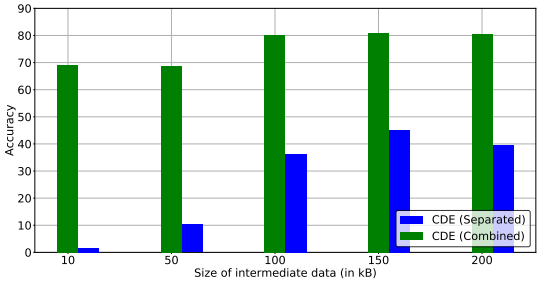
Fig. 5: Accuracy of (a) VGG16 when splitting the network at layers 7, 11 and 15, and (b) MobileNetV1 when splitting the network at layers 28 and 41 for both LoRa and 3G communication constraint.

2) Trade-off Between Communication Cost and Accuracy: In this section, we measure the impact of communication cost on accuracy when we split the network at a given layer. We use 3G bandwidth capacity, i.e. 200KB, as the limit. We also perform an ablation study by comparing joint training CDE (combined) and separated training CDE (separated). Fig. 6 show the result. As you can see, in all cases, CDE (combined) yields more accurate classification compared to CDE (separated). In general, the model and data compression performed by distillation (D) and auto-encoder (E) might cause a loss of information. By training the network jointly via CDE (combined) approach, the upper part of the network (which resides on the cloud) can be refined, adapting with the new (encoded) feature information. Another observation that we can see from Fig. 6 is, in general the smaller the encoded data is, the lower is the accuracy. As we have hypothesised, with the smaller size of encoded data, we lose more fine-grained feature information such that accurate classification is hard to achieve. In the low bandwidth scenario, the difficulty to generate accurate classification is even more pronounced in VGG16 than in MobileNetV1. For VGG16, in order to respect the constraint for 3G communication (i.e. 200KB), we have to compress the intermediate data from 2MB to 200KB (10% of than the original data). On the other hand, we only require to perform 50% compression for MobileNetV1 to satisfy the 3G communication constraint (as can be seen from Table III). In that case, a much higher reduction in accuracy is expected for VGG16. Nevertheless, our approach is not only able to retain the accuracy of VGG16 with 200KB encoded data, but even surpasses the performance of the original network with 75.5% classification accuracy.

3) Trade-off Between Memory Size and Accuracy: In this section, we employed both the bandwidth and the memory constraint on the edge device, and measured the impact on accuracy. Note that the memory size of the edge device was ignored in the previous experiments. Given the bandwidth constraint, we constructed several student networks (part of the whole network that resides on the edge device) to generate models with different memory sizes. The same layer cutting point can generate different network sizes because the students have different architectures depending on the constraint on memory size: a bigger student has more convolution layers and/or more neurons in each layer. The results are presented



(a) VGG16



(b) MobileNetV1

Fig. 6: The impact of communication cost on accuracy when we split (a) VGG16 at layer 7 and (b) MobileNetV1 at layer 28, with separated and combined training of KD + AE.

TABLE IV: The accuracy of VGG16 and MobileNetV1 for different memory sizes at the edge with 3G (200 KB) communication constraint. The white rows indicate that the model size can meet the MCUs capacity. We also indicate the accuracy and memory of the unmodified approach (Teacher T). This is the original accuracy without compression of the raw data.

	Splitting layer	Memory size at the edge	Accuracy
VGG16	No Splitting (T)	553.16 MB	70%
	Layer 7	320.3 KB	70.9%
	Layer 7	324.4 KB	73.4%
	Layer 7	340.8 KB	75.5%
	Layer 11	1.0 MB	72.2%
	Layer 11	1.1 MB	78.1%
	Layer 15	3.9 MB	70.4%
	Layer 15	4.2 MB	77.4%
MobileNetV1	No splitting (T)	17.3 MB	86.5%
	Layer 28	161.8 KB	80.8%
	Layer 28	244.3 KB	87.2%
	Layer 28	252.5 KB	89.6%
	Layer 41	590.1 KB	85.9%
	Layer 41	680.5 KB	87.1%
	Layer 41	688.7 KB	87.6%

in Table IV (3G) and Table V (LoRa).

First, we can notice that in several cases, the proposed model implies an improvement of the accuracy despite the constraints on memory size and intermediate data. This is more observable in the 3G case, when the communication constraint is less important. This improvement is due to knowledge distillation, which can reduce the size of the neural network while improving the accuracy thanks to transfer learning. Moreover, as previously observed in Fig. 6, the joint end-to-end training of knowledge distillation and auto-encoder proposed in this paper allows better results.

TABLE V: The accuracy of VGG16 and MobileNetV1 for different memory sizes at the edge with LoRa (25 KB) communication constraint. The white rows indicate that the model size can meet the MCUs capacity. We also indicate the accuracy and memory of the unmodified approach (Teacher T). This is the original accuracy without compression of the raw data.

	Splitting layer	Memory size at the edge	Accuracy
VGG16	No splitting (T)	553.16 MB	70%
	Layer 7	377 KB	63.9%
	Layer 11	1.0 MB	62.2%
	Layer 11	1.0 MB	67.2%
	Layer 11	1.1 MB	72%
	Layer 15	3.8 MB	67.2%
	Layer 15	4.3 MB	73%
MobileNetV1	No splitting (T)	17.3 MB	86.5%
	Layer 28	198.4 KB	76.2%
	Layer 28	282.5 KB	84.5%
	Layer 41	477.4 KB	85.5%
	Layer 41	571.6 KB	85.5%
	Layer 41	568.6 KB	86.7%

Furthermore, we can observe that generally, the smaller the student capacity (which in turn produces smaller memory size), the lower the accuracy, but this is not always the case. We find the same unexpected results as previously: under 3G constraint, splitting MobilenetV1 at layer 11 gives a better accuracy than splitting it layer 15, even if the memory size of the network at the edge is smaller (78.1% accuracy with memory 1.1 MB when splitting at layer 11 / 77.4% with memory 4.2 MB when splitting at layer 15). This implies that even if we have an edge device with enough memory to run the 15 first layers, it's more optimal to run only the 11 first layers. As explained in previous sections, this is probably due to the fact that knowledge distillation is optimal when splitting the network in the middle layers. However, we can observe that the optimal splitting layer is not same under different network constraints : with a higher bandwidth constraint (LoRa), a better accuracy is obtained when splitting the network at layer 15. Indeed, in a very constrained network, the effect of intermediate data compression is much higher. Therefore, cutting the network at higher layers, generating a smaller output, gives a better accuracy at the cost of higher memory requirements at the edge.

Finally, we can notice that, thanks to knowledge distillation, splitting the neural networks at low layers enables to meet memory requirements of very constrained devices such as MCUs (as depicted in Table I) with flash memory size ranging from 256 KB to 2 MB, while maintaining a good accuracy, or even improve it. Hence, thanks to the model proposed in this paper, it is possible to choose where to split the network in an optimal way given memory and/or bandwidth constraints. The results show that the optimal splitting may vary according to the different constraints. For example, when running MobilenetV1 in a scenario where the network condition changes from high available bandwidth to saturated network, it is necessary to switch the splitting from layer 11 to layer 15 in

TABLE VI: Comparison of CDE with E-JPEG and E-AE.

Model	Network	CDE	E-JPEG [45]	E-AE [46]
MobileNetV1	LoRa	**86.7%**	84.5%	45.9%
VGG16	LoRa	73%	**76.5%**	27%
MobileNetV1	3G	**90.3%**	86.3%	75.9%
VGG16	3G	**78.1%**	77.1%	46.4%

TABLE VII: The trade-off between accuracy and memory size at the edge between our CDE and CE-AE [26].

Model	Network	CDE (Acc.)	CDE (Memory)	CE-AE (Acc.)	CE-AE (Memory)
MobileNetV1	LoRa	**85.5%**	**477.4KB**	83.3%	870.3KB
VGG16	LoRa	**73%**	**4.3MB**	67.6%	31.2MB
MobileNetV1	3G	85%	**438.8KB**	**85.1%**	832.4KB
VGG16	3G	**77.4%**	**4.2MB**	69.8%	31.1MB

order to keep an optimal accuracy, if the edge device possesses enough memory. Another example can be given in the case of a smartphone running several computationally intensive applications at the same time: depending on the applications which are running, more or less memory will be available so the choice of the splitting layer may differ over time.

C. Comparison with Related Works

In order to compare our approach with the related works, we performed comparison in these two categories:

Encode (E). This category encodes the images directly before sending it to the cloud which is the classical approach (first approach in Fig. 1). For this comparison, we compress the image by using JPEG compression (**E-JPEG**) as described in [45] and auto-encoder (**E-AE**) as seen in [46]. The raw data is compressed to achieve a size of 25KB (LoRa) or 200KB (3G). The results are presented in Table VI. We observe that when we directly apply the auto-encoder at layer 0 to compress the image instead of compressing intermediate data, we obtained a lower accuracy. This could be explained by the fact that if we split the network, the output features are already pre-trained and suitable for the task. When we just compress directly the image with an auto-encoder, the auto-encoder is not exposed with the final objective (e.g. classification), so it is trained blindly without knowing what the actual objective is. On the other hand, when we compress the features from the split network using an auto-encoder, the features are already suitable for the task. This is also the case when the image is compressed with JPEG, even if the difference of accuracy is less visible. This reinforces the importance of the proposed approach: splitting the network, then compressing with an auto-encoder and training with distillation.

Cut-Encode (CE). In this category, the execution of DNN is split between the edge and the cloud while the intermediate output is encoded with the auto-encoder (**CE-AE**) to achieve a size of 25KB (LoRa) or 200KB (3G). Previous work that follows **CE-AE** paradigm includes [26]. [20] also encodes intermediate data with lossy compression (quantization, huffman coding), and does not distil the edge part. The results are presented in Table VII. We observe that, while for MobileNetV1 the accuracies of CDE and CE-AE are relatively the same, for VGG16 the knowledge distillation of the edge part allows an improvement of 6 to 8%. Moreover, this enables us to significantly reduce the memory size of the model at the edge, which is divided by 2 for MobileNetV1 and by 7 for VGG16.

V. Conclusion

This is the first work proposing an approach to split neural network inference between the edge and the cloud while optimising both bandwidth and memory constraints. For this, we proposed a novel method which jointly trains knowledge distillation at the edge and intermediate data compression with an auto-encoder. This approach allows to find the optimal partition under specific constraints. It makes it possible to execute part of the network on very constrained devices such as MCUs, and under poor network conditions (LoRa) while limiting the accuracy drop. Moreover, we showed that the optimal splitting layers are not necessarily the upper ones, as suggested by previous work, but must be chosen carefully according to the constraints. This is a promising technique for applications with limited bandwidth and memory size at the edge devices. It enables to have guidelines to choose the splitting layer depending on these constraints. This optimal splitting may vary over time according to the evolution of the constraints. In future work, we plan to design a framework to dynamically split the neural network inference between the edge and the cloud by taking several constraints into consideration (e.g., available bandwidth, battery, memory, etc.). One challenge would be to make this approach adaptive to changing bandwidth availability, or changes in the scene (context) as a result of the camera moving, or dynamic changes in the scene in the case of a static camera. It would also be interesting to extend it to multi-camera networks in order to find the optimal partitioning between the different edge devices.

References

[1] I. S. Alex Krizhevsky and G. E. Hinton, "Classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. X. A. T. Bolei Zhou, Agata Lapedriza and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, 2014, pp. 487–495.

[3] D. Y. G. E. D. A.-r. M. N. J. A. S. V. V. P. N. T. N. S. Geoffrey Hinton, Li Deng and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] M. R. Yaniv Taigman, Ming Yang and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[5] M. Zhang, F. Zhang, N. D. Lane, Y. Shu, X. Zeng, B. Fang, S. Yan, and H. Xu, "Deep learning in the era of edge computing: Challenges and opportunities," 2020.

[6] STMicroelectronics. Stm32 high performance mcus. [Online]. Available: https://www.st.com/en/ microcontrollers-microprocessors/stm32-high-performance-mcus.html

[7] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," 2019.

[8] Microchip. Atsam4e16e. [Online]. Available: https://www.microchip.com/wwwproducts/en/ATSAM4E16E

[9] Nxp mcus. [Online]. Available: https://www.nxp.com/support/developerresources/evaluation-and-developmentboards/freedom-development-boards/mcuboards/freedom-development-platform-for-kinetisk64-k63-and-k24-mcus

[10] Stm32f7 series of mcus. [Online]. Available: https://www.st.com/en/microcontrollersmicroprocessors/stm32f7-series.html

[11] Stm32 nucleo boards. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32- nucleo-boards.htmll

[12] Nxp i.mx rt series: Crossover processor. [Online]. Available: https://www.nxp.com/products/processors-andmicrocontrollers/arm-based-processors-andmcus/i.mx-applications-processors/i.mx-rt-series.

[13] H. Qassim, D. Feinzimer, and A. Verma, "Residual squeeze vgg16," 2017.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] S. A. Osia, A. Shahin Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, vol. 7, no. 5, p. 4505–4518, May 2020. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2020.2967734

[16] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," vol. 45, no. 1. New York, NY, USA: Association for Computing Machinery, Apr. 2017, p. 615–629. [Online]. Available: https://doi.org/10.1145/3093337.3037698

[17] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2015, pp. 27–40.

[18] B. Lovejoy, " Apple moves to third-generation Siri back-end, built on opensource Mesos platform." http ://9to5mac.com/2015/04/27/siribackend-mesos/.

[19] Google, " The Google Brain is a real thing but very few people have seen it." http ://www.businessinsider.com/what-isgoogle- brain-2016-9.

[20] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, ser. HotEdgeVideo'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 27–32. [Online]. Available: https://doi.org/10.1145/3349614.3356023

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[24] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[25] V. Ashish, C. Carlo, G. Philip, J. Thomas, K. Konstantinos, P. Jitendra, and V. George, "Wanalytics : Geo-distributed analytics for a data intensive world," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1087–1092.

[26] K. Jong Hwan, N. Taesik, A. Mohammad Faisal, and M. Saibal, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *Proceedings of the 15th IEEE International Conference on Advanced Video and Signal Based Surveillance*. IEEE, 2018, pp. 1–6.

[27] C. Eduardo, B. Aruna, C. Daeki, W. Alec, S. Stefan, C. Ranveer, and B. Paramvir, "Maui : Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*. ACM, 2010, pp. 49–62.

[28] H. Chien-Chun, A. Ganesh, G. Leana, Y. Minlan, and Z. Mingyang, "Wide-area analytics with multiple resources," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, pp. 1–16.

[29] W. Peisong, H. Qinghao, Z. Yifan, Z. Chunjie, L. Yang, and C. Jian, "Two-step quantization for low-bit neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 4376–4384.

[30] Z. Z. K. M. e. a. Xu, S., "A collaborative cloud-edge computing framework in distributed neural network," *J Wireless Com Network 2020*, no. 211, 2020. [Online]. Available: https://doi.org/10.1186/s13638-020-01794-2

[31] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017, pp. 1–14.

[32] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *arXiv preprint arXiv:1701.05369*, 2017.

[33] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015, pp. 1135–1143.

[34] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016, pp. 1379–1387.

[35] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1737–1746.

[36] V. Vincent, S. Andrew, and M. Mark, "Improving the speed of neural networks on cpus," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, vol. 1, 2011.

[37] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[38] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, "Learning separable filters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[39] S. Bhattacharya and N. D. Lane, "Sparsification and separation of deep learning layers for constrained resource inference on wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, ser. SenSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 176–189. [Online]. Available: https://doi.org/10.1145/2994551.2994564

[40] M. R. U. Saputra, P. P. B. d. Gusmao, Y. Almalioglu, A. Markham, and N. Trigoni, "Distilling knowledge from a deep pose regressor network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[41] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6550

[42] S. Gugger and J. Howard. Fastai - external data. [Online]. Available: https://docs.fast.ai/data.external.html

[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2046–2049, 2018.

[46] Y. Dong, P. Zhao, H. Yu, C. Zhao, and S. Yang, "Cdc: Classification driven compression for bandwidth efficient edge-cloud collaborative deep learning," *arXiv preprint arXiv:2005.02177*, 2020.