MTL-Split: Multi-Task Learning for Edge Devices using Split Computing

Luigi Capogrosso[1], Enrico Fraccaroli[1,2], Samarjit Chakraborty[2], Franco Fummi[1], Marco Cristani[1]
[1]name.surame@univr.it, [2]enrifrac@cs.unc.edu, samarjit@cs.unc.edu
[1]Department of Engineering for Innovation Medicine, University of Verona, Italy
[2]Department of Computer Science, The University of North Carolina at Chapel Hill, USA

ABSTRACT

Split Computing (SC), where a Deep Neural Network (DNN) is intelligently split with a part of it deployed on an edge device and the rest on a remote server is emerging as a promising approach. It allows the power of DNNs to be leveraged for latency-sensitive applications that do not allow the entire DNN to be deployed remotely, while not having sufficient computation bandwidth available locally. In many such embedded systems scenarios, such as those in the automotive domain, computational resource constraints also necessitate Multi-Task Learning (MTL), where the same DNN is used for multiple inference tasks instead of having dedicated DNNs for each task, which would need more computing bandwidth. However, how to partition such a multi-tasking DNN to be deployed within a SC framework has not been sufficiently studied. This paper studies this problem, and MTL-Split, our novel proposed architecture, shows encouraging results on both synthetic and real-world data. The source code is available at https://github.com/intelligolabs/MTL-Split.

KEYWORDS

Split Computing, Multi-Task Learning, Deep Neural Networks, Edge Devices

ACM Reference Format:
Luigi Capogrosso[1], Enrico Fraccaroli[1,2], Samarjit Chakraborty[2], Franco Fummi[1], Marco Cristani[1] . 2024. MTL-Split: Multi-Task Learning for Edge Devices using Split Computing. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655686

1 INTRODUCTION

In the last decade, Deep Neural Networks (DNNs) have achieved state-of-the-art performance in various problems. However, DNN models often present computational requirements that cannot be met by most of the resource-constraint edge devices available today [4]. This prohibits the full deployment of DNN-based applications on these systems, leading to what is commonly known as the Local-only Computing (LoC) approach. However, using simplified models negatively affects the overall accuracy. As such, the most common deployment approach of DNN-based applications on resource-constraint edge devices is the Remote-only Computing (RoC). With this, the network runs on the server side, and the input is directly transferred from the edge device to the server through a network connection. Then, the server computes the inferences and sends the output back to the device. However, such data transfer could lead to excessive latency times, especially in degraded channel conditions.

As a compromise between the LoC and the RoC approaches, recently suggested *Split Computing (SC)* frameworks [21] propose to split DNN models into a head and a tail, deployed on edge device and server, respectively. In particular, early implementations of SC, like [15], select a layer and divide the model to define the head and tail sub-models. Instead, more recent SC frameworks introduce the bottleneck concept to achieve in-model compression toward the global task [20].

At the same time, current state-of-the-art approaches in different Machine Learning (ML) applications rely on advanced learning procedures, such as the *Multi-Task Learning (MTL)* [6]. In particular, MTL is a paradigm in which multiple related tasks are jointly learned to improve the generalizability of a model by using shared knowledge across different aspects of the input. This is achieved by jointly optimizing the model's parameters across all tasks, allowing the model to learn both task-specific and shared representations simultaneously.

Innovations. In this paper, we present a new combination of SC and MTL to solve multiple inference tasks on edge devices. Solving multiple tasks with a common DNNs can lead to substantial resource savings. For example, consider the automotive domain: detecting a person with a camera requires solving both a classification task (identifying pedestrians, vehicles, buildings, etc.) and a regression task (determining bounding boxes corresponding to the classifications). Our proposed approach aims to solve multiple tasks simultaneously, i.e., $T_1 \dots T_N$, where N represents the number of tasks, using only a single neural network, in contrast to current methods, where the emphasis is on Single-Task Learning (STL), which would need N neural networks to solve the tasks.

As a result, by employing MTL, we enhance performance across multiple tasks, elevating the design challenge beyond that of preserving a single task's performance, as in regular SC. Further, this allows systems to operate effectively even in scenarios where data is scarce for specific tasks but abundant for others, as explained and theoretically demonstrated in [2]. Lastly, the shared feature space output is remarkably lightweight, significantly reducing network latency encountered in SC scenarios.

In summary, the main contributions of this paper are:

- A new SC design merged with MTL. Our design handles multiple tasks concurrently, instead of the current focus on STL in SC.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

- Through MTL, we increase task performance, overcoming the challenge of preserving only the performance of the main task.
- Finally, the output from the shared feature space is remarkably lightweight, significantly mitigating the impact of network latency in a SC scenario.

2 RELATED WORK

This section provides an overview of distributed deep learning applications, specifically focusing on SC and MTL.

2.1 Distributed deep learning

We focus on architectures operating through a DNN model $M(\cdot)$, whose task is to produce the inference output y from an input x. Three types of paradigms used for distributed deep learning can be identified in the literature, *viz.*, LoC, RoC, and SC.

Local-only Computing (LoC). Under this policy, the entire computation is performed on the sensing devices. Therefore, the edge device entirely executes the function $M(x)$. Its advantage lies in offering low latency due to the proximity of the computing element to the sensor. However, it may not be compatible with DNN-based architectures that demand robust hardware capabilities. Usually, simpler DNN models $\bar{M}(x)$ that use specific architectures (e.g., depth-wise separable convolutions) are used to build lightweight networks, such as MobileNetV3 [14].

Besides designing lightweight neural models, in the last few years, progress has been made in DNN compression. Techniques such as network pruning and quantization [18], or knowledge distillation [13], achieve an efficient representation of the neural network but with some quality degradation.

Remote-only Computing (RoC). The input x is transferred through the communication network and processed at the remote system through the function $M(x)$.

This paradigm preserves full accuracy considering the higher computation bandwidth of the remote system but leads to high latency and communication bandwidth due to the input transfer, especially when the cloud is located far from the edge device.

Split Computing (SC). A typical SC scenario is discussed in [10], where neither LoC nor RoC approaches are optimal, and a split configuration is an ideal solution. The SC paradigm divides the DNN model into a head, executed by the local sensing device, and a tail, executed by the remote system. It combines the advantages of both LoC and RoC, thanks to the lower latency and, more importantly, drastically reduces the required transmission bandwidth by compressing the input x to be sent through the use of an autoencoder [20]. We define the encoder and decoder models as $z_l = F(x)$ and $\bar{x} = G(z_l)$, which are executed at the edge and remotely, respectively. The distance $d(x, \bar{x})$ defines the performance of the encoding-decoding process.

One of the earliest works on SC is the study by Kang *et al.* [15], in which the initial layers of a DNN are the most suitable candidates for partitioning, as they optimize both latency and energy consumption. Additionally, latency reduction can be achieved through two methods: quantization, as explored in [17], and the utilization of lossy compression techniques prior to data transmission, as investigated in [7]. The concept of employing autoencoders to further compress the data to be transferred is discussed in various studies, such as [11].

Prevalent methods for identifying potential splitting points have evolved from architecture-based, to more refined neuron-based ones. Within the domain of architecture-based approaches, in [24], candidate split locations are where the size of the DNN layers decreases. The rationale is that compressing information by autoencoders, where compression would still occur due to the shrinking of the architecture, seems reasonable. On the other hand, in [8] and [5], it was shown that not only the type of the layers but also the saliency of individual layers is a crucial factor. A neuron's saliency is determined by its gradient in relation to the accurate decision. Thus, optimal splitting points should be positioned following layers housing impactful neurons, to preserve the information flowing until then.

Notably, while existing approaches target STL problems, we propose the first SC solution for multi-task learning challenges. Furthermore, while state-of-the-art methods strive to minimize the drop in accuracy, our approach aims at enhancing the accuracy. To the best of our knowledge, the only work that combines concepts similar to ours is [31]. In this, the authors introduce task-oriented edge computing to reduce bandwidth consumption, which is different from our proposal.

2.2 Multi-Task Learning (MTL)

MTL to solve multiple learning problems at the same time [6], can help us reduce inference time, improve accuracy, and increase data efficiency [27]. In its basic formulation, MTL uses a common representation to predict several outputs from a single input. An important aspect of this procedure is the relationship between tasks and how much a shared representation can be transferred across tasks [30], or how to weight the losses of different tasks [16] to create a better joint optimization objective.

In recent years, numerous methods have emerged to address the simultaneous solving of multiple tasks, from approaches that learn how to weigh automatically the different tasks [12], to more sophisticated transformer-based architectures [29]. Parallel work also has explored different theoretical aspects of MTL, such as treating it as a multi-objective optimization [25] or using game-theoretic optimizations [22]. In particular, MTL approaches have remained theoretical without practical implementation at the edge. Thus, our work addresses this gap.

The DNNs solving the different tasks in MTL are commonly known as task-solving heads, which, we understand, might be confused with the head/tail terminology of SC. For the remainder of the paper, when we use the term head, we refer to the MTL terminology (i.e., task-solving heads) and not the SC one (i.e., head/tail).

3 METHODOLOGY

This paper proposes to combine SC and MTL to execute complex inference tasks on edge devices. After outlining our notation, this section delineates the formal components of our proposal, shown in Fig. 1. This architecture consists of two components: *i)* a shared *backbone* deployed on the edge device, and *ii)* a series of *task-solving*

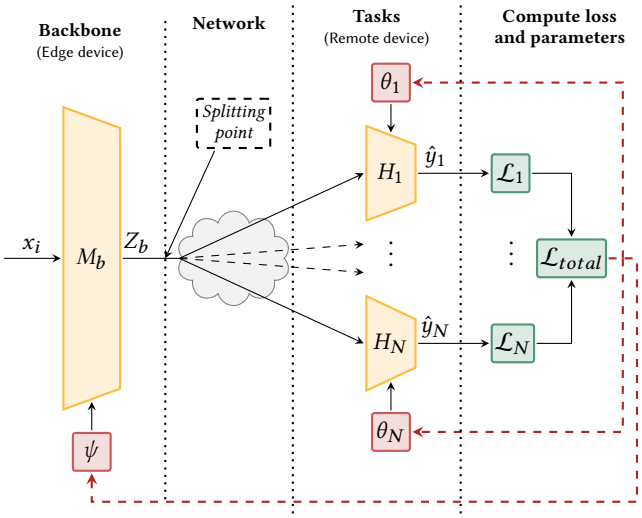

Figure 1: The proposed architecture for handling complex inference tasks on edge devices by integrating SC and MTL.

heads on a single or multiple remote devices. Orange trapezoids are DNN models, while their parameters are enclosed in red boxes. The green components on the right-hand are the loss functions used to update the learnable parameters. A communication network separates edge and remote devices.

Setting and notation. We assume the existence of a labeled image dataset defined as follows:

$$D = \left\{(x_i, y_i) \,|\, \forall i \in \{1 \dots K\},\, x_i \in \mathbb{R}^{w \times h \times c},\, y_i \in \mathbb{N}^N\right\} , \quad (1)$$

where K is the number of (image, labels) tuples, x_i is the input representing the image, and y_i a set of N labels associated with the i-th image, namely ground truth. The input x_i is a tensor with dimensions $w \times h \times c$, where w is the width, h is the height, and c is the number of channels (e.g., red, green, blue). In this work, we consider the *classification task* that tries to learn a mapping from the image space $\{x_i | \forall i \in \{1 \dots K\}\}$ to the corresponding set of labels $\{y_i | \forall i \in \{1 \dots K\}\}$.

3.1 Proposed architecture

Unlike a classic SC scenario, here we focus on architectures operating through a DNN model, whose task is to produce the inference outputs $\{\hat{y}_j | \forall j \in \{1 \dots N\}\}$ from an input x_i, where N is the number of tasks to be solved. In this way, one can build a single model that learns multiple tasks across the same input.

As shown in Fig. 1, the first module is the *backbone*, a DNN model $M_b(\cdot)$ sharing hidden layers among all tasks. In this way, we greatly reduce the risk of overfitting: the more tasks we learn concurrently, the more our model has to find a representation that captures all tasks, and the less likely we are to overfit the original task. We describe the backbone operations on the i-th input as follows:

$$Z_b = M_b(x_i; \psi) , \quad (2)$$

where ψ are the set of shared learnable parameters, and Z_b is the backone's output. The output Z_b is typically a tensor, which, in our approach, is flattened before being sent through the network. This output represents the point of the networks where the shared feature representation Z_b is extracted from the backbone and transferred to the task-solving heads.

Each task is implemented by its own task-solving head, or head hereafter, a DNN model H_j located outside the edge devices, e.g., on a remote server. We describe the operations of the j-th head H_j, as follows:

$$\hat{y}_j = H_j(Z_b; \theta_j) , \quad (3)$$

where θ_j is its set of learnable parameters, and $\hat{y}_j$ its output.

Putting it all together, the overall system output is a collection of outputs from all heads, organized either as a list or a single tensor.

3.2 Training strategy

The proposed methodology is architecture-independent. Any neural network architecture can implement the backbone network and heads, such as a Convolutional Neural Network (ConvNet) or a Recurrent Neural Network (RNN), designed to capture useful features from the input data x_i. Regardless of the desired architecture, the objective of the MTL system is to encourage the model to perform well on all tasks simultaneously. Let us denote the *task-specific loss function* for the i-th input and j-th task as $\mathcal{L}_j(y_i, \hat{y}_j)$, which measures the differences between corresponding label inside the ground truth y_i and the predicted output $\hat{y}_j$. The *overall loss function* for the MTL system with the i-th input can be defined as the sum of losses from each task, as follows:

$$\mathcal{L}_{total} = \sum_{j=1}^{N} \mathcal{L}_j(y_i, \hat{y}_j) . \quad (4)$$

The training process updates the shared backbone parameters ψ and the heads' parameters θ_j by backpropagating the gradient of the total loss with respect to these parameters and using an optimization algorithm like Stochastic Gradient Descent (SGD). The specific DNN architecture, activation functions, and optimization methods can vary based on the problem and input data.

3.3 Fine-tuning the model

A key aspect of the proposed methodology, besides exploiting SC to enable MTL on edge devices, is the fine-tuning process explained in this section. There are several reasons for performing fine-tuning, such as if we aim to enhance task-specific performance or if we want to introduce new tasks to the system. During the fine-tuning phase, we update the heads' parameters θ_j while keeping the shared backbone parameters ψ relatively fixed.

During the fine-tuning process, heads' parameters are updated using gradients with respect to the task-specific loss:

$$\theta_j := \theta_j - \alpha \cdot \nabla_{\theta_j} \mathcal{L}_j(y_i, \hat{y}_j) , \quad (5)$$

where α is the learning rate for updating heads' parameters. The shared backbone's parameters are often kept fixed or updated conservatively during fine-tuning. As such, we need to define a separate update process as follows:

$$\psi := \psi - \eta \cdot \nabla_{\psi} \mathcal{L}_{total} , \quad (6)$$

where η is the learning rate for updating the shared parameters, a small value compared to the one used to update the heads' parameters shown in Eq. (5).

Given the parameters update functions, we can now define the fine-tuning process as an optimization problem, which involves minimizing the sum of the total loss as follows:

$$\underset{\theta,\psi}{\text{minimize}} \ \mathcal{L}_{total} \cdot \tag{7}$$

It is worth keeping in mind that fine-tuning should be approached with care. It is important to find a balance between adapting the model to task-specific characteristics and retaining the general knowledge from the shared backbone. Too much fine-tuning can lead to overfitting on limited task-specific data, while too little fine-tuning might not fully harness the benefits of the MTL setup.

4 EXPERIMENTS

This section describes the experimental trials that have been performed to validate our claims, along with their implementation details and results.

Models details. In our experimental setup, we used three well-known DNNs, i.e., VGG16 [26], MobileNetV3 [14], and EfficientNet [28], as a shared backbone. We chose the first one because it is a well-established and widely used architecture in many image-processing tasks. While the others represent cutting-edge DNNs for embedded systems applications. The task-solving heads are custom MultiLayer Perceptron (MLP) composed of two linear layers activated by the Rectified Linear Activation Unit (ReLU) function. We want to point out that in this design, the task-solving heads are smaller than the backbone. However, even though the task-solving heads are smaller than the backbone individually, if we consider a large number of tasks N, their combined size becomes larger than that of the backbone. Due to this rationale, our architecture provides the deployment of task-solving heads on the remote server.

Datasets. To effectively showcase the capabilities of our proposal, we begin our experiments with the 3D Shapes [3] dataset, a widely used toy benchmark in the ML literature. We further demonstrate the effectiveness of our proposed method by exploring its performance on the MEDIC [1] and FACES [9] datasets, one of the well-known, and the newest MTL benchmarks, respectively.

3D Shapes is a dataset of 3D shapes generated from 6 independent factors. All possible combinations of these factors are present exactly once, resulting in 480,000 total images. These are the floor hue, wall hue, object hue, scale, shape, and orientation. Therefore, it is possible to treat the classification of each factor as a different task to solve, i.e., $T = [T_1 \dots T_6]$. Due to the straightforward nature of the synthetic images in 3D Shapes, solving the classification tasks with a DNN can be easy, leaving a limited possibility for improvement through MTL. Thus, to render this setting more realistic, we add salt-and-pepper noise of 15% of the image pixels, making the classification more difficult. In particular, with the presence of noise, the classification of object size (8 classes) and object type (4 classes) becomes challenging.

MEDIC is the largest social media image classification dataset for humanitarian response, consisting of 71,198 samples to address four different tasks. Specifically, we decided to address only the damage severity (3 classes) and disaster type (4 classes) tasks since informativeness and humanitarian are somewhat trivial.

FACES, is a set of 2,052 images of naturalistic faces. Here, the task corresponds to the classification of the perceived ages (3 classes), genders (2 classes), and facial expressions (3 classes).

Table 1: Classification accuracy on the test partition of the 3D Shapes dataset considering the object size (T_1) and the object type (T_2). Values are reported as a percentage.

Model	Single-Task Learning		Multi-Task Learning (T_1 + T_2)	
	$T_1 \uparrow$	$T_2 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$
VGG16	12.50	25.50	**51.10** (+38.60)	**81.74** (+56.24)
MobileNetV3	74.85	93.95	**77.23** (+2.38)	**94.00** (+0.05)
EfficientNet	95.49	99.07	**96.66** (+1.17)	**99.48** (+2.28)

Training and inference details. All the code is implemented in PyTorch Lightning, and the used pre-trained network corresponds to the implementations in PyTorch [23]. On the 3D Shapes dataset, we train our models for 10 epochs, with a learning rate of 1×10^{-5}, using AdamW [19] as an optimizer, on a NVIDIA RTX 3090. On the MEDIC and FACES dataset, we train our models for 50 epochs, with a learning rate of 1×10^{-4}, always using AdamW as an optimizer, on an NVIDIA RTX 3090.

We run all the experiments on an NVIDIA Jetson Nano with 4 GB of memory.

4.1 Multi-Task Learning (MTL) results

In this section, we validate our claims within the MTL context, which evidences the effectiveness of our proposal. Given the distinct nature of our methodology (see Section 2), a direct comparison with state-of-the-art MTL methods would be inadequate since these approaches are based on advanced loss functions [16] rather than models [29]. As a result, based on [6], our experimental protocol involves benchmarking our models against their respective single-task performance.

We begin our analysis on the 3D Shapes dataset. The results, in terms of accuracy, are shown in Table 1 and demonstrate that MTL-Split improves the performance on all the tasks with respect to the STL design choice. This confirms the first two claims of our proposal, i.e., our architecture handles multiple tasks simultaneously and improves accuracy across the entire task set by collectively optimizing the model's parameters for all tasks. Hence, in the context of SC (which encompasses multiple tasks to be solved), our approach guarantees performance improvement rather than merely aiming to minimize performance degradation, which is the SC trend observed in all previous state-of-the-art methods. Furthermore, the ability to address multiple tasks within the same network simultaneously has resulted in space and computational savings during inference because it only requires the evaluation of a single network instead of N neural networks to solve each task.

Table 2 further demonstrates the efficacy of our proposal exploring its performance on the MEDIC dataset. This experiment serves as a compelling validation of our previous claims, showcasing the robustness of our architecture even when applied to complex

Table 2: Classification accuracy on the test set of the MEDIC dataset considering the damage severity (T_1), and disaster type (T_2). Values are reported as a percentage.

Model	Single Task Learning		Multi Task Learning (T_1 + T_2)	
	$T_1 \uparrow$	$T_2 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$
VGG16	61.78	59.14	**62.65** (+0.87)	**60.54** (+1.40)
MobileNetV3	61.73	52.66	**61.90** (+0.17)	52.29 (-0.37)
EfficientNet	61.00	53.94	**62.42** (+1.42)	**55.74** (+1.80)

datasets. In this case, it is important to highlight the *inductive transfer* between tasks, as even a small increase in decimal points in this challenging context represents a significant achievement.

The minor decrease of 0.37% of T_2's performance in the MTL setting does not represent a problem. Specifically, what is known in the MTL literature as *negative transfer* comes up when there is a significant deterioration in performance across all tasks, typically resulting from conflicting or unrelated task objectives. Since the performance improves in all the other cases, we can confidently say that negative transfer doesn't occur here. We attribute this outcome to gradient fluctuations. These results demonstrate the effectiveness of our approach, as it consistently yields significant improvements even in difficult scenarios.

Finally, Table 3 shows the results achieved on the FACES dataset employing the fine-tuning strategy starting from pre-trained networks on ImageNet. The overall accuracies obtained are quite high, which was expected given the utilization of a pre-trained network as a starting point. However, once again, our approach demonstrated its efficacy in enhancing performance across all tasks. This is significant since it increases accuracies approaching near-maximum values. Usually, such improvements necessitate the network's ability to correctly classify the intricate corner cases within the datasets. In all instances where we do not achieve performance improvements, our results consistently align with the single-task performance (also in this case, ruling out the possibility that the non-performance improvements are due to negative transfer).

4.2 Split Computing (SC) analysis

In this section, we examine the advantages of our approach in comparison to the other types of distributed deep learning paradigms while also presenting deployment considerations.

Local-only Computing (LoC). Under this paradigm, for the 3D Shapes and the MEDIC, two distinct DNNs are required since we address two different tasks. Hence, the estimated memory size utilizing MobileNetV3 as a backbone is $\approx$ 1.5 GB, while it is $\approx$ 6.9 GB for the EfficientNet. Instread, for FACES, which involves three different tasks (i.e., three distinct DNNs are required), the estimated memory size using MobileNetV3 is $\approx$ 2.1 GB, and for the EfficientNet is $\approx$ 10.3 GB.

As a result, due to memory constraints, the only feasible implementation on the Jetson Nano is restricted to MobileNetV3 on the 3D Shapes dataset. However, as indicated in Table 4, our approach, utilizing a single shared backbone on the edge device, enables the execution of all implementations on the same board. Specifically, using EfficientNet, we achieve memory size improvements of $\approx$ 38% for the 3D Shapes and MEDIC datasets, and $\approx$ 57% for the FACES dataset. As aforementioned, VGG16 is not optimal for embedded system applications, so we do not report data on that model.

Remote-only Computing (RoC). Under this policy, the goal is to minimize the data sent from the backbone to task-solving heads.

In the FACES dataset, the images are RGB with 2835×3543 pixels. Consequently, transmitting each input from the edge to the cloud involves transefing a tensor of size $2835 \times 3543 \times 3$, equivalent to $\approx$ 115 MB over the network channel.

Whereas, Table 4 also highlights the minimal burden placed on the network channel when employing our methodology, thanks to the shared backbone's neural processing. For example, assuming a gigabit channel: the time required to transfer 100 inputs of size $\approx$ 115 MB each is $\approx$ 98 s, whereas for our inputs of the size of 1.5 MB, it is $\approx$ 12 s, i.e., we obtain an improvement of $\approx$ 87% in the overall latency time. This is important as Internet congestion will increasingly be driven by machine learning workloads.

This claim holds significant importance in a world that increasingly relies on efficient data transmission and reduced network congestion.

Discussion. The above analyses showcase the advantages of our proposal compared to LoC. Our approach also excels in terms of data transmission, resulting in reduced total latency compared to RoC. Furthermore, our design handles multiple tasks concurrently, thereby enhancing overall accuracy across all tasks.

5 CONCLUDING REMARKS

In this paper, we propose a new combination of SC and MTL to execute complex inference tasks on embedded devices. The proposed architecture consists of a unified backbone that serves all tasks at the edge, and multiple task-solving heads situated outside the edge device. This design enables the architecture to tackle multiple tasks simultaneously, in contrast to SC methodologies that focus solely on a single task. Furthermore, by incorporating MTL, our approach enhances the accuracy performance across all tasks. Additionally, the output from the shared backbone is notably lightweight, significantly mitigating network latency's impact in the context of SC. Extensive experimental validation confirms our claims. Our approach succeeds at delivering superior performance across all tasks, whether they are simple or complex. Further, comparison with other types of distributed deep learning paradigms and the deployment considerations confirm the efficacy of our design.

ACKNOWLEDGMENTS

This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043), and by the European Union's Horizon Europe research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 101109243. This manuscript reflects only the Authors' views

Table 3: Classification accuracy on the test set of the FACES dataset considering the the perceived ages (T_1), genders (T_2), and facial expressions (T_3). Values are reported as a percentage.

Model	Single Task Learning			Multi Task Learning (T_1 + T_3)		Multi Task Learning (T_2 + T_3)		Multi Task Learning (T_1 + T_2 + T_3)		
	$T_1 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$	$T_1 \uparrow$	$T_3 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$	$T_1 \uparrow$	$T_2 \uparrow$	$T_3 \uparrow$
VGG16	96.83	95.61	19.02	**97.80** (+0.87)	**91.46** (+72.44)	**99.02** (+3.41)	**90.24** (+80.22)	**98.54** (+1.71)	**99.51** (+3.90)	**89.27** (+70.25)
MobileNetV3	97.07	99.51	95.12	**99.51** (+2.44)	95.12 (+0.00)	99.51 (+0.00)	**95.61** (+0.49)	**99.27** (+2.20)	99.51 (+0.00)	**95.85** (+0.73)
EfficientNet	99.76	99.76	94.63	**100** (+0.24)	**95.61** (+0.98)	99.76 (+0.00)	**97.32** (+2.96)	**100** (+0.24)	**100** (+0.24)	**95.61** (+0.98)

Table 4: Computing the size of the backbone M_b, and of its output Z_b. The reader should pay particular attention to the green columns in the table, as these are the columns displaying the results, which show that our proposal is really efficient for SC.

Model	M_b #params (M)	M_b #params size (MB)	Forward/backward pass size (MB)	M_b estimated size (MB)	Z_b #params (M)	Z_b size (MB)
MobileNetV3	0.9	3.58	724.08	727.66	55.3	0.21
EfficientNet	4	15.45	3452.09	3467.54	406.06	1.56

and opinions. Neither the European Union nor the European Commission can be considered responsible for them. This work was also partially supported by the US NSF grant 2038960.

REFERENCES

[1] Firoj Alam, Tanvirul Alam, Md Arid Hasan, Abul Hasnat, Muhammad Imran, and Ferda Ofli. 2023. MEDIC: a multi-task learning dataset for disaster image classification. *Neural Computing and Applications* 35, 3 (2023), 2609–2632.

[2] Etienne Boursier, Mikhail Konobeev, and Nicolas Flammarion. 2022. Trace norm regularization for multi-task learning with scarce data. In *Conference on Learning Theory*. PMLR, 1303–1327.

[3] Chris Burgess and Hyunjik Kim. 2018. 3D Shapes Dataset. https://github.com/deepmind/3dshapes-dataset/.

[4] Luigi Capogrosso, Federico Cunico, Dong Seon Cheng, Franco Fummi, and Marco Cristani. 2024. A Machine Learning-oriented Survey on Tiny Machine Learning. *IEEE Access* (2024).

[5] Luigi Capogrosso, Federico Cunico, Michele Lora, Marco Cristani, Franco Fummi, and Davide Quaglia. 2023. Split-Et-Impera: A Framework for the Design of Distributed Deep Learning Applications. In *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 39–44.

[6] Rich Caruana. 1997. Multitask learning. *Machine learning* 28 (1997), 41–75.

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

[8] Federico Cunico, Luigi Capogrosso, Francesco Setti, Damiano Carra, Franco Fummi, and Marco Cristani. 2022. I-split: Deep network interpretability for split computing. In *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 2575–2581.

[9] Natalie C Ebner, Michaela Riediger, and Ulman Lindenberger. 2010. FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation. *Behavior research methods* 42 (2010), 351–362.

[10] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 565–576.

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

[12] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L Yuille. 2019. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3205–3214.

[13] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129 (2021), 1789–1819.

[14] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1314–1324.

[15] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[16] Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7482–7491.

[17] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. 2018. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 402–411.

[18] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.

[19] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

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

[21] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2022. Split computing and early exiting for deep learning applications: Survey and research challenges. *Comput. Surveys* 55, 5 (2022), 1–30.

[22] Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. 2022. Multi-Task Learning as a Bargaining Game. In *International Conference on Machine Learning*. PMLR, 16428–16446.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

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

[25] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems* 31 (2018).

[26] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[27] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. 2020. Which tasks should be learned together in multi-task learning?. In *International Conference on Machine Learning*. PMLR, 9120–9132.

[28] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[29] Xiaogang Xu, Hengshuang Zhao, Vibhav Vineet, Ser-Nam Lim, and Antonio Torralba. 2022. Mtformer: Multi-task learning via transformer and cross-task reasoning. In *European Conference on Computer Vision*. Springer, 304–321.

[30] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. 2018. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3712–3722.

[31] Jianfeng Zhang, Wensheng Zhang, and Jingdong Xu. 2022. Bandwidth-efficient multi-task AI inference with dynamic task importance for the Internet of Things in edge computing. *Computer Networks* 216 (2022), 109262.