Transfer Learning for Convolutional Neural Networks in Tiny Deep Learning Environments

Evangelia Fragkou*
efragkou@uth.gr
University of Thessaly
Department of Electrical & Computer Engineering
Greece

Vasileios Lygnos
vlygnos@uth.gr
University of Thessaly
Department of Electrical & Computer Engineering
Greece

Dimitrios Katsaros
dkatsar@uth.gr
University of Thessaly
Department of Electrical & Computer Engineering
Greece

ABSTRACT

Tiny Machine Learning (TinyML) and Transfer Learning (TL) are two widespread methods of successfully deploying ML models to resource-starving devices. Tiny ML provides compact models, that can run on resource-constrained environments, while TL contributes to the performance of the model by using pre-existing knowledge. So, in this work we propose a simple but efficient TL method, applied to three types of Convolutional Neural Networks (CNN), by retraining more than the last fully connected layer of a CNN in the target device, and specifically one or more of the last convolutional layers. Our results shown that our proposed method (F_xC1) achieves about 19% increase in accuracy and 61% increase in convergence speed, while it incurs a bit larger energy consumption overhead, compared to two baseline techniques, namely one that retrains the last fully connected layer, and another that retrains the whole network.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Machine learning**; **Machine learning approaches**;

KEYWORDS

Resource-constrained devices, Convolutional neural networks, Deep learning, Transfer learning, TinyML

ACM Reference Format:
Evangelia Fragkou, Vasileios Lygnos, and Dimitrios Katsaros. 2022. Transfer Learning for Convolutional Neural Networks in Tiny Deep Learning Environments . In *26th Pan-Hellenic Conference on Informatics (PCI 2022), November 25–27, 2022, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3575879.3575984

*The research work is supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 5631)

PCI 2022, November 25–27, 2022, Athens, Greece

ACM ISBN 978-1-4503-9854-1/22/11…$15.00
https://doi.org/10.1145/3575879.3575984

1 INTRODUCTION

Tiny Machine/Deep Learning *(Tiny ML/DL)* is a fast-growing field, which boosts machine learning tasks deployment in devices that lack of computational resources, like sensors, IoT devices etc. It is a useful tool since it combines both software and hardware optimization and hence it enables the running of ML inference tasks in those devices [26], etc. Moreover, TinyML, by encouraging data processing on the edge device, contributes to the maintenance of the privacy of data, since information that is collected, stays to the device, while in parallel, it lessens the energy cost of the possible and now not necessary communication between the edge device and the server[1]. Furthermore, Transfer Learning *(TL)*, is a method in which pre-knowledge of a source model is used, in order to enhance the learning ability of the device in which this model is further applied. TL means either retraining the whole model or the last fully connected layer to the resource-constrained (target) device and hence boosting its performance with both reducing the training time needed for the model to converge, as it inherits the already learned features and by giving the opportunity for the model to be trained with sufficient already-existed data. In this work, we combine the two aforementioned techniques, by proposing algorithms that go the common TL and the already known TinyML optimization techniques a step further, in order to maximize the performance of low-powered devices. Specifically, our endeavor is inspired by the work [12], in which authors propose the retraining of the fully connected layers to low-powered device. We present methods of retraining, which include also one or more convolutional layers, with the one that uses only one convolutional layer to be our best method, due to the fact that we achieve increase in accuracy levels, with the less energy consumption, regarding to the other two methods, proposed in this work. Specifically, F_xC1 method gives approximately 19% increase in classification accuracy of the model.

1.1 Motivations and contributions

TinyML contributes to the efficiency of models performance, by running mainly inference tasks on ultra-low power devices. Moreover, TL offers pre-knowledge to the target device, maximizing the levels of its performance. Most of the works, presented in existing literature, are referred to inference tasks. By combining the two aforementioned methods in this work, we try to enable step to step not only the inference but also the training procedure in resource-constrained devices, by retraining not only the last fully connected layer, but also both the fully connected and the last convolutional layer. The main prior work is the one presented in [12]).

The basic motivation is extracted by the way a CNN network performs its tasks. Specifically, in an image classification task, the first layers of a CNN network learn the basic concepts of an image, then this knowledge is transferred to the next layers in order for the final ones to be able to learn the high level concepts of the input-image. This means that the final convolutional layer/layers is/are responsible for the classification of the data, being processed by the network. So, when the network is exposed to new data, it is of high importance to update not only the categorization of data, but also the categories themselves, so as to achieve high Accuracy levels.

By this way, we start to enter the core of the network, without needing the resources, that the whole network requires, in order to operate. Overall, our work contributes in the following:

- We develop a technique where not only the final fully connected layers are retrained, but also some of the immediately previous convolutional layer are retrained. This retraining can potentially take place in resource-starving devices, thus improving data privacy and being ideal for cases of federated learning [19] or Internet of Things implementations [22].
- We investigate the tradeoff between training more layers versus accuracy versus energy consumption which has not been explored so far. Evidently, retraining more of the last layers (both convolutional and fully-connected) will result in improved accuracy, but it will cost in energy consumption. Is there any optimal point with respect to how many retrained layers are enough before consuming too much energy without gaining in accuracy?
- We use homogeneous data to experiment with, but by affecting a convolution layer through retraining, we can more easily deal with heterogeneous data in the future, since the convolutional layers are the ones responsible for the better understanding and categorization of the hierarchical features of image data; we compare the examined methods against two baseline methods: a) the methods reported in [12] (*Baseline-1*) and b) when the whole CNN is retrained (*Baseline-2*).

The rest of the paper is structured as follows: section 2 presents the related work, section 3 describes the proposed CNN architectures used. In section 4 we evaluate the neural network's accuracy, loss and energy consumption, and in section 5 we summarize the described results. Finally, section 6 concludes this article.

2 RELATED WORK

Tiny ML/DL is an emerging field of machine learning technologies and applications, including both hardware [15] and software development, with the aim of performing on-device inference tasks on ultra-low power devices, e.g. sensors, micro-controllers, IoT devices etc. Software-based Tiny ML methods are currently focused on combining optimization techniques like neural networks model compression (Network Architecture Search - NAS [18], Pruning [16], [3], [7], Weights Quantization [10], [24]), [11], or on model pre-training into the cloud etc. [14], [17] and then fine-tuning in an aforementioned device, so as for the model to achieve better performance, without sacrificing the levels of accuracy. However, the existing literature, is mainly referred to methods that accelerate inference tasks, while only few papers introduces ways that deal with the on-line training procedure of a model [23], [12], [6], which is a prospect that have to be further researched and discussed.

On the other hand, *TL* is a widely-used method, used for leveraging knowledge of a domain (source), trained on large-scale data to the target domain, which is used to be a smaller and less data-processing-capable one, with the aim of boosting the learning ability of the latter one [21], [27], [4]. A neural network needs plenty of auxiliary data so that it can be trained sufficiently, which is a very often problem taking into account that not all systems do have available data to do so or the appropriate technical specifications, like many gigabytes of RAM (resource-starving devices), in order to process all these data. So, TL came to solve not only the lack of sufficient data for processing, but also the tremendous training time and the massive storage capacity needed from a device in order to perform training, which are two bottlenecks in training procedure. As a consequence, TL mechanisms achieve high performance in real – world applications, such as cross-language text categorization, text-to-image classification, etc. Some already known deep-learning-based approaches are DAN [20] , DCORAL [25], and DANN [8], [9] which are applied for solving image classification problems.

The combination of Tiny ML and TL can contribute significantly in both the reduce of training time by providing more compact and hence faster models regarding their convergence and the maintenance of their high classification accuracy, due to the knowledge, being inherited by the pre-training of the model with plenty of data. Concurrently, by pre-training the model to the cloud and then fine-tuning it to the device contributes to the security of data, too, as every device knows only the weights of the model, being trained to specific data, and not the data themeselves. The most relative-to-our-work paper is the [12], in which a model is trained to a processing capable machine, e.g. a server with hundred of instances of data and then the pre-trained network is applied to the target domain. After that, only the last (Fully-Connected) layer of a Convolutional Neural Network (CNN) is recovered by the exposition to the new data in the target device, taking into consideration that it inherits the knowledge, or else the "frozen" weights of the pre-trained network. This procedure is more cost-friendly since we retrain one layer than the whole network. So, we gain time by training only the last layer but is it the most optimal solution regarding accuracy levels? In this work we propose a new TL implementation method, in which we re-train more than one layer of the pre-trained network to the target device and then we compare the results with the existing implementations. Our method is not only model but also data aware, which means it can be applied in conjunction with any other method or in any case, being described in this section.

3 THE FAMILY OF F_xC_yTECHNIQUES

The most widespread technique in Transfer Learning is to train the model to a dataset, then 'freeze' the weights and after that either retrain the last fully connected layer, or retrain the whole model to the new data. In our research, we conducted experiments in order to observe how different the model will perform when we retrain more layers than just the fully connected ones and compare our

results with the already existed TL methods. So, we created three different experimental cases:

F_xC1. In this case, we train not only the last fully connected layers, but also the last convolution layer, while the remaining model is frozen.

F_xC2. In this case, we train the fully connected layers and the last two convolution layers, while the remaining model is frozen.

F_xC3+. In the last case, we train the fully connected layers and up to three convolution layers, while the remaining model is frozen.

In our experiments, the fully- connected layers of the networks, used, are more than one, so in order to elaborately present our Figures, we display exactly the number of both fully-connected and convolutional layers, taking place in training, by using the abbreviation F_xC_y, while x is the number of final fully- connected layers and y the number of Convolution layers, re-trained in every experiment, as it is illustrated in Table 2.

Algorithm 1 Pseudocode of the family of F_xC_y techniques

```
Procedure Pre-training() on a large dataset;
for i in range(1, epochs) do
    if method == F_xC1 then
        Re-train(FC, Conv.Layer-1);        //1 conv. Layer
    else if method == F_xC2 then
        Re-train(FC, Conv.Layer-1, Conv.Layer-2);        //2 conv. Layers
    else if method == F_xC3+ then
        Re-train(FC, Conv.Layer-1, Conv.Layer-2 Conv.Layer-3, ...);
        //up to 3 conv. Layers
    end if
end for
```

The experiments emphasize on the large models, in most of the cases, since it is more difficult to optimize them, without sacrificing performance. For each case, we have computed the flops needed for them to run for one epoch so we have made an estimation of how much energy they will consume, taking into account a CMOS 45nm processor.

4 EXPERIMENTAL EVALUATION

This section introduces details about the competitors, the datasets used, the size of the neural networks we experiment with and then it presents the results of the experimental evaluation of the proposed algorithms.

4.1 Evaluation settings

Competitors. In order to evaluate our algorithms, we compare the performance of our method with the two following baseline methods.

Baseline-1. In this method, only the last fully connected layer is retrained, while the remaining model is frozen [12].

Baseline-2. In this method, the whole model is retrained.

Architectures of neural networks. We will use 3 CNN models, a small one which has an input of 32x32x1 and 550 thousand parameters with 6 convolution layers with relu as the activation function except for the last dense layer, in which we used softmax activation function, called Small CNN. The other two models are from the family of EfficientNet, which is a group of convolutional network models that acheive high accuracy, with very few parameters and hence is a viable solution for resource-constrained devices. (specifically, EfficientNetB0 and EfficientNetB2). We picked EfficientNetB0 and EfficientNetB2 as our large models with 4.05 million and 7.76 million parameters, respectively, as we see in Table 2. In the fully connected layers, we used softmax as the activation function.

Measures. For the small CNN model, we did 50 epochs for every case we tested, while for the EfficientNet models we did 10 epochs, respectively. Considering we have a classification problem to tackle, we used the 'Accuracy' measure (implemeted in Keras) in order to evaluate and compare our methods. Although we address a multi-class problem, we used *accuracy* metric instead of *F1-measure/Score*, taking into consideration that our first experiments were conducted, using the family of CIFAR datasets (see the following paragraph), whose classes are equally important, or else every class constitues by the same number of instances (balanced classification dataset), and consequently, these two metrics can equally produce safe results, in this case. We continued our experiments, using other two datasets, ImageNet and Intel classification, which don't have exactly the same number of instances in every class, but the difference between the number of data tends to be very small. The proportion of data in the first dataset is 1:7 and it is mainly used for training the model, since it contains 1,000 classes of images, which means plenty of images and therefore, better learning of the model. On the contrary, the second one (whose difference among class instances is about 2% - almost balanced) is used only for testing purposes, so in these cases, we can safely use *accuracy* metric, too. Moreover, we used both categorical cross-entropy (especially used for one-hot encoded vectors) and sparse categorical cross-entropy function (class vectors contain only the integer that stipulates their input data category - we gain processing time during training), implemented in Keras API, as the loss function for small CNN model and the EfficientNet models, respectively. The last measure we used, is the one referred to the energy, required by our methods. The most reliable way to estimate it, is through the FLOPS of every model needed for one Epoch. Therefore, we computed how many FLOPS each test we did, needs for one epoch and how much energy it would consume, being implemented to a certain processor (45nm CMOS technology processor) with the admission that a MAC operation is roughly equal to two flops. Even if the admission is far from reality, the percentage difference between the two cases would be the same, if FLOPS remain the main comparing factor. Nano Joule (nJ) is the unit of measurement in this case. The type we used is the following:

$$ENERGY = \frac{FLOPS}{2} * Emac$$

.

Datasets. In this work, we deal with an image classification task. In order to test our methods, we used datasets that include both colour and greyscale images. Furthermore, datasets have a lot of instances and the same feature space (homogeneous data) to a great extend, so as to enhance the adaptation of data from the source to the target domain. The test samples we used, are 20% smaller in size compared to the training samples ragarding the CIFAR-10, CIFAR-100 and Intel classification datasets and approximately 8% smaller, compared to Imagenet dataset (due to the tremendous training time, needed in order to run our experiments).Besides, the most important thing is for the test dataset to be smaller than the

Table 1: Dataset characteristics.

Dataset	Size of Images	Train Samples	Test Samples	Classes
ImageNet	224x224	1, 281, 167	100, 000	1, 000
CIFAR-100	32x32	50, 000	10, 000	100
CIFAR-10	32x32	50, 000	10, 000	10
Intel Image Classification	150x150	14, 000	3, 000	6

Table 2: Trainable Layers, used in every model architecture.

Model	**Small CNN**		**EfficientNetB0**		**EfficientB2**	
Trainable Parameters	550, 000		4, 049, 571		7, 768, 569	
Method/ Trainable Layers	Total	#of Conv.	Total	#of Conv.	Total	#of Conv.
F_xC1	9	1	3	1	3	1
F_xC2	11	2	5	2	7	2
F_xC3+	15	3	8	4	10	4
F_xC3+	-	-	16	5	15	5
F_xC3+	-	-	20	6	23	8
F_xC3+	-	-	23	8	31	9
F_xC3+	-	-	31	10	38	12

training one, or else it will cause accuracy drops. Information about the datasets is illustrated in Table 1. Moreover, we conducted four experiments, in which we use different combination between the datasets for both pre-trained and retrained networks.

Hardware used. Our methods were tested on a desktop computer with 16 GB of RAM and an Intel Xeon W 2123 processor, clocked at 3.60 GHz.

4.2 Evaluation results

In the following experiments, we compare the results with the two baseline methods, mentioned in section 4.1. In plots, every method is represented by the number of Layers that are unfreezed during the retraining phase of the model, so as to know exactly which architecture we use. In Table 2, we see the total number of layers, taking place in retraining and how many of them are convolutional ones.

4.2.1 Experiment-1 (Evaluation of Small CNN with CIFAR-100 and CIFAR-10). In our first experiment, we used CIFAR-100 to train the model and CIFAR-10 [13] to evaluate it. As the Figure 1 (left) illustrates, training both the fully connected and the last convolution layer, gets almost a 10% boost in accuracy, regarding *Baseline-1* method. Hence, adding more layers to the training procedure, finally contributes to the accuracy improvement, but it is expensive regarding the energy requirements.

Regarding the energy consumption, depicted in Figure 1 (right), we verify our intuition, regarding low power consumption of all models being presented against the *Baseline-2* method. This does happen due to the smaller amount of trainable parameters of both the *Baseline-1* method and our proposed techniques, which makes the training procedure faster and therefore, less energy-expensive. As we can understand, the energy consumption increases proportionately with the number of layers, being unfreezed in every experiment.

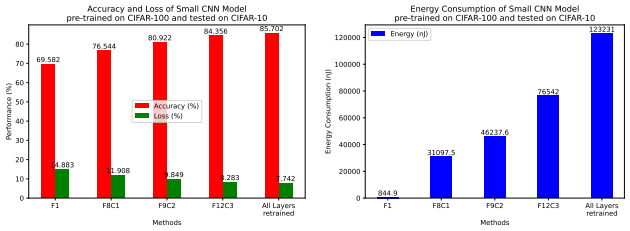

Figure 1: Experimental evaluation of small CNN model, pre-trained on CIFAR-100 and tested on CIFAR-10.

4.2.2 Experiment-2 (Evaluation of EfficientNetB0 with Imagenet/ CIFAR-100. In this test, EfficientNetB0 is used and trained with ImageNet dataset [5]. We will evaluate it with CIFAR-100 and the results are depicted in Figure 2. The difference in both accuracy and loss regarding the two baseline methods is significant. We can easily observe that our method, namely F_xC1, continues to stands out against *Baseline-1* method, while having a bigger and more structurally-complicated network, compared to Experiment-1. Training the fully connected layer and the last convolution layer we get an increase of 20% compared to the training of only the fully connected layers. Also, the accuracy of this case is slightly less than the accuracy, retained by training all layers of the model, which means that we don't have important information loss.

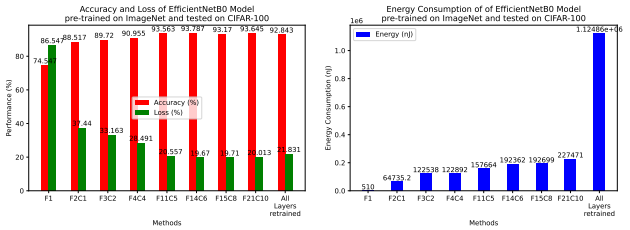

Figure 2: Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on CIFAR-100.

4.2.3 Experiment-3 (Evaluation of EfficientNetB2 with Imagenet/ CIFAR-100. In this experiment, we use the same order of datasets of the previous experiment (ImageNet for pre-training and CIFAR-100 for fine-tuning of the network respectively), whereas a quite larger network, the EfficientNetB2 one is used. It is of a great importance that the results are encouraging in this experiment, too. Specifically, we notice a small increase of 18.87% in accuracy, as it is shown in Figure 3 (left), while the amount of the trainable parameters of the network is extremely large, compared to both the small CNN and EfficientNetB0. Moreover, it is worth highlighting that our method F_xC1 generalizes the given data faster (about 60, 81% faster convergence), compared to *Baseline-1*, in the same number of epochs. Furthermore, the more the unfreezed layers are, the greater the improvement of accuracy measure is, bearing in mind the results in Figure 3. Regarding the energy consumption, we can conclude that the less the unfreezed layers are, the less the energy consumption of every model, being tested, is but prioritizing the measure of accuracy, we can conclude that F_xC1 method outperforms any other method mentioned, since it offers better accuracy, regarding the common baseline-1 method, while it increases the energy requirements of the model, to a small extend, comparing with the other proposed methods.

4.2.4 Experiment-4 (Evaluation of EfficientNetB0 with Imagenet/Intel Classification Dataset). In the last experiment, we use EfficientNetB0 network,

Table 3: Summary of experiments.

Model Datasets Sets (pre-trained/tested)	Accuracy (%) FC (*Baseline-1*)	Accuracy (%) $\mathbf{F_xC1}$	Energy (nJ) FC (*Baseline-1*)	Energy (nJ) $\mathbf{F_xC1}$	Percentage Increase in Accuracy, compared to *Baseline-1*(%)	Percentage of faster convergence, compared to *Baseline-1*(%)
Small CNN (CIFAR-100/CIFAR-10)	69, 5	76, 5	844, 9	31097, 5	10, 01	19, 98
EfficientNetB0 (ImageNet/CIFAR-100)	74, 5	88, 5	510, 0	64735, 2	18, 79	56, 74
EfficientNetB0 (ImageNet/Intel Classification)	86, 9	92, 2	137, 4	64350, 2	6, 09	39, 38
EfficientNetB2 (ImageNet/CIFAR-100)	78, 0	91, 1	560, 9	78273, 5	16, 79	60, 81

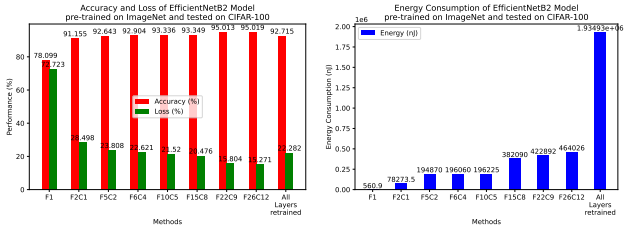

Figure 3: Experimental evaluation of EfficientNetB2 model, pre-trained on ImageNet and tested on CIFAR-100.

being pre-trained on ImageNet dataset. In this test, we use a smaller dataset to evaluate it, which is the Intel classification dataset [2]. As we expected, the pattern in the bar graph, being illustrated in Figure 4 is similar to all the experiments, presented so far(experiment-1-2-3). Furthermore, because of the smaller size of the dataset, used, we achieve high accuracy, faster than the *Baseline-1* method, despite the handred of parameters used in this network. Specifically, in this experiment, F_xC_1 method achieves about 6, 09% boost in accuracy and approximately, 39, 38% faster model convergence.

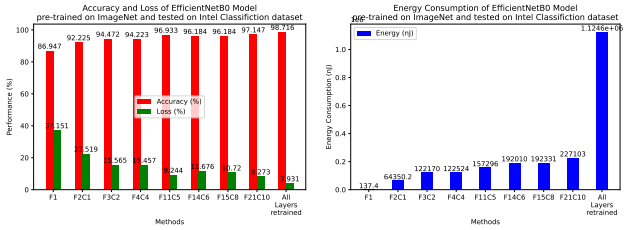

Figure 4: Experimental evaluation of EfficientNetB0 model, pre-trained on ImageNet and tested on Intel classification dataset.

5 SUMMARY OF EXPERIMENTS

To sum up, in Table 3, we compare the performance of the classic TL method (*Baseline-1*) and the one (F_xC1) we end up proposing, after conducting our experiments. We compare only the *Baseline-1* model since the *Baseline-2* consists of the whole model and therefore it is not a viable solution for a TinyML device, due to extravagant energy consumption (up to 143x larger than the one needed by the *Baseline-1* method). We conducted several experiments, using three types of neural networks (see Table 2) and four large-scale datasets (see Table 1) in a pair of two (for pre-training or re-training). In every test F_xC1 method outperforms (regarding accuracy) the *Baseline-1* method in a range from 10, 7% to 18, 79%, proportionately to both the datasets and the model, used, while it also achieves faster convergence than *Baseline-1* method, in a range from 19, 98%to60, 81%, as we can see in Figures 1, 2, 3, 4. However, the energy consumption required is a tradeoff since it is a bit larger than the one, needed for the *Baseline-1* method (see Table 3). The rest of the methods proposed, even if they had even better results regarding accuracy, they can't run on resource-starving devices due to the large energy consumption they require, which increases proportionately to the layers that are retrained in each case.

6 CONCLUSIONS

Overall, we presented new Transfer Learning methods in order to enhance the performance of a network, being deployed to resource-constrained environments. We present a simple but efficient method, in which we don't fine-tune only the last fully connected layer of a pre-trained network, but we also retrain one or more of the convolutional layers of three kinds of a CNN network (a Small CNN, EficientNetB0, EficientNetB2). The results shown that F_xC1 proposed method acheives approximately 19% increase in network accuracy and about 61% faster convergence, whereas it is a model-aware method since it outperforms the classic Transfer Learning techniques in all cases, tested, with almost a small increase in energy consumption, compared to the *Baseline-1* method (retraining the last fully connected layer) and incomparably better performance, regarding the energy consumption, needed by *Baseline-2* (retraining the whole network) method.

REFERENCES

[1] Colby R. Banbury, Vijay Janapa Reddi, Maximilian Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David A. Patterson, Danilo Pau, Jae sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. 2020. Benchmarking TinyML Systems: Challenges and Direction. *ArXiv* abs/2003.04821 (2020).

[2] PUNEET BANSAL. [n. d.]. *Intel Image Classification.* https://www.kaggle.com/datasets/puneet6060/intel-image-classification/discussion

[3] Andreas Chouliaras, Evangelia Fragkou, and Dimitrios Katsaros. 2021. Feed Forward Neural Network Sparsificationwith Dynamic Pruning. In *25th Pan-Hellenic Conference on Informatics* (Volos, Greece) *(PCI 2021).* Association for Computing Machinery, New York, NY, USA, 12–17. https://doi.org/10.1145/3503823.3503826

[4] Oscar Day and Taghi M. Khoshgoftaar. 2017. A survey on heterogeneous transfer learning. *Journal of Big Data* 4 (2017), 1–42.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 248–255.

[6] Simone Disabato and Manuel Roveri. 2020. Incremental On-Device Tiny Machine Learning. *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (2020).

[7] Evangelia Fragkou, Marianna Koultouki, and Dimitrios Katsaros. 2022. Model reduction of feed forward neural networks for resource-constrained devices. *Applied Intelligence* (2022).

[8] Yaroslav Ganin and Victor S. Lempitsky. 2015. Unsupervised Domain Adaptation by Backpropagation. *ArXiv* abs/1409.7495 (2015).

[9] Yaroslav Ganin, E. Ustinova, Hana Ajakan, Pascal Germain, H. Larochelle, François Laviolette, Mario Marchand, and Victor S. Lempitsky. 2016. Domain-Adversarial Training of Neural Networks. In *J. Mach. Learn. Res.*

[10] Sedigh Ghamari, Koray Ozcan, Thu Dinh, Andrey Melnikov, Juan Carvajal, Jan Ernst, and Sek M. Chai. 2021. Quantization-Guided Training for Compact TinyML Models. *ArXiv* abs/2103.06231 (2021).

[11] Lennart Heim, Andreas Biri, Zhongnan Qu, and Lothar Thiele. 2021. Measuring what Really Matters: Optimizing Neural Networks for TinyML. *ArXiv* abs/2104.10645 (2021).

[12] Kavya Kopparapu and Eric Lin. 2021. TinyFedTL: Federated Transfer Learning on Tiny Devices. *ArXiv* abs/2110.01107 (2021).

[13] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n. d.]. *CIFAR10 and CIFAR100.* https://www.cs.toronto.edu/~kriz/cifar.html

[14] Jisu Kwon and Daejin Park. 2021. Toward Data-Adaptable TinyML using Model Partial Replacement for Resource Frugal Edge Device. *The International Conference on High Performance Computing in Asia-Pacific Region* (2021).

[15] Liangzhen Lai, Naveen Suda, and Vikas Chandra. 2018. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *ArXiv* abs/1801.06601 (2018).

[16] Josen Daniel De Leon and Rowel Atienza. 2022. Depth Pruning with Auxiliary Networks for Tinyml. *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2022), 3963–3967.

[17] Yingling Li, Zhipeng Li, Tianxing Zhang, Peng Zhou, Siyin Feng, and Kunqin Yin. 2021. Design of a Novel Neural Network Compression Method for Tiny Machine Learning. *Proceedings of the 2021 5th International Conference on Electronic Information Technology and Computer Engineering* (2021).

[18] Edgar Liberis, Lukasz Dudziak, and Nicholas D. Lane. 2021. μNAS: Constrained Neural Architecture Search for Microcontrollers. *Proceedings of the 1st Workshop on Machine Learning and Systems* (2021).

[19] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Tao Niyato, and Chunyan Miao. 2020. Federated Learning in Mobile Edge Networks: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 22 (2020), 2031–2063.

[20] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. 2015. Learning Transferable Features with Deep Adaptation Networks. *ArXiv* abs/1502.02791 (2015).

[21] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[22] Dimitrios Papakostas, Theodoros Kasidakis, Evangelia Fragkou, and Dimitrios Katsaros. 2021. Backbones for Internet of Battlefield Things. In *2021 16th Annual Conference on Wireless On-demand Network Systems and Services Conference (WONS)*. 1–8. https://doi.org/10.23919/WONS51326.2021.9415560

[23] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), 1–8.

[24] Muhammad Akmal Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann. 2021. TinyML: Current Progress, Research Challenges, and Future Roadmap. *2021 58th ACM/IEEE Design Automation Conference (DAC)* (2021), 1303–1306.

[25] Baochen Sun and Kate Saenko. 2016. Deep CORAL: Correlation Alignment for Deep Domain Adaptation. In *ECCV Workshops*.

[26] Maxim Zemlyanikin, Alexander Smorkalov, Tatiana Khanova, Anna Petrovicheva, and Grigory Serebryakov. 2019. 512KiB RAM Is Enough! Live Camera Face Recognition DNN on MCU. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* (2019), 2493–2500.

[27] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2021. A Comprehensive Survey on Transfer Learning. *Proc. IEEE* 109, 1 (2021), 43–76. https://doi.org/10.1109/JPROC.2020.3004555