MicroTL: Transfer Learning on Low-Power IoT Devices

Citation for the original published paper (version of record):

Profentzas, C., Almgren, M., Landsiedel, O. (2022). MicroTL: Transfer Learning on Low-Power IoT Devices. Proceedings - Conference on Local Computer Networks, LCN: 34-41.
http://dx.doi.org/10.1109/LCN53696.2022.9843735

MicroTL: Transfer Learning on Low-Power IoT Devices

Christos Profentzas
Chalmers University of Technology
Gothenburg, Sweden
chrpro@chalmers.se

Magnus Almgren
Chalmers University of Technology
Gothenburg, Sweden
magnus.almgren@chalmers.se

Olaf Landsiedel
Kiel University, Germany
Chalmers University of Technology, Sweden
ol@informatik.uni-kiel.de

***Abstract*—Deep Neural Networks (DNNs) on IoT devices are becoming readily available for classification tasks using sensor data like images and audio. However, DNNs are trained using extensive computational resources such as GPUs on cloud services, and once being quantized and deployed on the IoT device remain unchanged. We argue in this paper, that this approach leads to three disadvantages. First, IoT devices are deployed in real-world scenarios where the initial problem may shift over time (e.g., to new or similar classes), but without re-training, DNNs cannot adapt to such changes. Second, IoT devices need to use energy-preserving communication with limited reliability and network bandwidth, which can delay or restrict the transmission of essential training sensor data to the cloud. Third, collecting and storing training sensor data in the cloud poses privacy concerns. A promising technique to mitigate these concerns is to utilize on-device Transfer Learning (TL). However, bringing TL to resource-constrained devices faces challenges and trade-offs in computational, energy, and memory constraints, which this paper addresses. This paper introduces MicroTL, Transfer Learning (TL) on low-power IoT devices. MicroTL tailors TL to IoT devices without the communication requirement with the cloud. Notably, we found that the MicroTL takes 3x less energy and 2.8x less time than transmitting all data to train an entirely new model in the cloud, showing that it is more efficient to retrain parts of an existing neural network on the IoT device.**

***Index Terms*—IoT, Transfer Learning, Quantization**

I. Introduction

Compression methods like quantization and pruning [7], [14], [29] have enabled the deployment of Deep Neural Networks (DNNs) on resource-constrained devices [14], [18], [24]. However, the training, optimization, and compression of neural networks are commonly conducted before deployment using vast computational resources like GPUs or even cloud services. Once trained, compressed, and deployed, DNNs commonly remain static. We argue in this paper that this approach leads to three drawbacks. First, IoT devices are deployed and interact with real-world environments, and the initial domain may shift to a different distribution [9]. For example, a user might want to add or remove a class, such as adding a new exercise activity on her smartwatch. With current methods, deployed neural networks on IoT devices [14], [18], [24] cannot adapt to such changes. The typical approach is to (re)train a new DNN on the cloud, optimize it for the local hardware (e.g., post-quantization [7], [14], [18] or quantization-aware learning [12]), and re-download it from edge or cloud services. Due to the large amount of data needed for training [10] and the non-negligible size of the deep neural networks, an IoT device needs to spend significant energy sending training samples and downloading new models from cloud services each time the problem domain changes. Second, resource-constrained devices depend on energy-efficient communication with limited reliability and network bandwidth. In practice, this may delay or even prohibit the transmission of training samples from the device. Third, uploading sensor data to the cloud poses privacy concerns. As cloud services struggle to address privacy issues, they may aggregate data from different sources, where keeping independent and identically distributed (iid) samples along with efficient communication is an open challenge [30].

To avoid communication costs and address problem domain shifts, we can use Transfer Learning (TL) [25]. TL utilizes pre-trained deep neural networks and re-trains parts of the network using a smaller dataset [25] for a smaller but related problem. Today, TL is typically done on uncompressed floating-point DNNs (i.e., non-quantized), with high dimensional training data. Bringing transfer learning to quantized DNNs deployed on resource-constrained devices creates trade-offs in computational, energy, and memory constraints and leads to two main challenges. First, resource-constrained IoT devices have minimal memory, such as 64-256 KBs of RAM and 32-64 MHz CPU, often operate on batteries, and are unable to store and process high dimensional data. Second, typical quantize-aware training methods [12] retain the precision of gradients by duplicating all values of the neural networks, one with floating points and one with quantized values, respectively. This double-booking prohibitively increases the memory and computational for resource-constrained IoT devices.

We present MicroTL, which tackles the above challenges by tailoring Transfer Learning (TL) to resource-constrained IoT devices. MicroTL combines parts of the output of quantized hidden layer(s) of an existing deep neural network with new fully connected layer(s) specialized to the new classes of the problem domain. MicroTL enables (re)training parts of DNNs on the IoT device, and thus no communication with the cloud is required, thereby protecting sensitive information. Moreover, this enables the personalizing of deep neural networks to the end-user using local data while preserving the user's privacy at the same time.

In summary, this paper makes the following contributions:

- We enable transfer learning on resource-constrained devices, allowing neural networks to adapt to changes in the problem domain and removing the need to communicate with the cloud.
- We show that combining the outputs of the hidden layer(s) with fully connected layer(s) is sufficient for learning without duplicating the values of the network.
- We design and implement MicroTL, an open-source[1] transfer learning system for resource-constrained IoT devices. We provide a discussion of implementation challenges and trade-offs.
- We quantify the performance of MicroTL in terms of accuracy, computation, memory, and energy consumption. Notably, we find that for a particular dataset it takes 3x less energy and 2.8x less time than transmitting all the local data to the cloud to create a new model.

Paper outline. We organize the paper as follows. Section II provides the necessary background. Section III introduces the system design of MicroTL. Section IV presents the evaluation of MicroTL. Section V discusses related work, and Section VI concludes the paper.

II. Background

In this section, we provide the necessary background on transfer learning and quantization.

A. Transfer Learning

Transfer learning (TL) aims to reduce the amount of training data and speed up the training process. The motivation of TL [25] is to utilize existing neural networks pre-trained on large data sets for a generic problem domain referred to as the **source domain** and adapt for similar smaller domain referred to as the **target domain**. TL uses the hidden layers of a pre-trained network as feature extractors from the source domain to append and train end-layer(s) on the target domain. A typical approach of transfer learning consists of three steps. First, it removes the Fully Connected (FC) layers at the end of a pre-trained network. Second, it appends a new FC layer(s) with output matching the number of the target domain classes. Third, it freezes the weights (no backward calculations) of all layers prior to FC layer(s) and trains the network using the target domain data set by updating only the weights of FC layer(s).

B. Quantization

In order to deploy deep neural networks on IoT devices for inference, we need to train and optimize them using another library (e.g., PyTorch [21], Tensorflow [27]) on cloud services. Due to the resource constraints of IoT devices, it is common to compress the neural networks using quantization [12], [19]. Fixed-point quantization represents real numbers with integers using fixed-length fractional parts. Int-8 quantization [14], [18] is the most common approach to compress DNN using 7-bits for fixed-point representation of original values, plus 1-bit for the sign (8-bits in total). It reduces the size without changing the original architecture of the network and affecting the overall accuracy by a margin [14]. The quantized function requires both the scaling of the min-max range of real numbers to integers and a shift offset because the zero-point in the integer domain is different from the real number domain due to the quantization. There are two widely used methods to produce the final quantized version of the network: post-training and quantization-aware training.

Post-training quantization. In this method [7], [14], [18], the neural network is training as usual with floating-point. The network is statically quantized using the min-max range of the weights from floating-point to integers. However, this method needs a representative data-set from the device to calibrate the activation output of each layer. This happens only once, and the weights never change after deployment.

Quantization-Aware Training (QAT). This method [12] simulates the quantization effect during training by duplicating all the weights and bias, one in 8-bit integers for the forward pass, and one in 32-bit floating-points for the backward pass, together with meta-data regarding rounding effects. During the forward-pass, it utilizes the 8-bit integer weights of the network, but during the back-propagation, it calculates the gradient using the 32-bit floating-point weights. At the end of each iteration, it quantizes the floating-point weights to 8-bit integer weights. With this method, the quantization error is part of the learning process providing better accuracy, but the quantization parameters become part of the overall training.

III. MicroTL Design

MicroTL's main objective is to personalize deep neural networks to end-user preferences without the communication dependency on sending sensor data from IoT devices to the cloud and neural network updates from cloud to IoT devices. MicroTL allows training fully connected layers personalized to the end-user, and it keeps the data on the device, avoiding sending sensitive or private data to the cloud. MicroTL assumes a pre-initialization step of downloading or flashing a pre-trained neural network on IoT devices, as they are widely available today with CMSIS-NN [14] or Tensorflow-micro [7].

A. System Challenges

We identify two main system challenges when designing MicroTL. First, deep neural networks for low-power IoT are typically quantization-aware trained or post-quantized to address memory and computational constraints. Even though inference after quantization is straightforward, reversing the effects of quantization for additional learning is an open challenge [8]. Second, low-power IoT devices have memory and computational limitations for storing training samples and computing learning parameters.

1) Learning after quantization: Due to common gradient descent optimization methods in deep learning [10], any training algorithm needs to calculate and retain the precision of the gradients. Low-power IoT devices lack the resources to keep duplicate weights (like QAT, as explained in section II), and there are no standard methods to reverse quantized layers

[1] https://github.com/chrpro/MicroTL

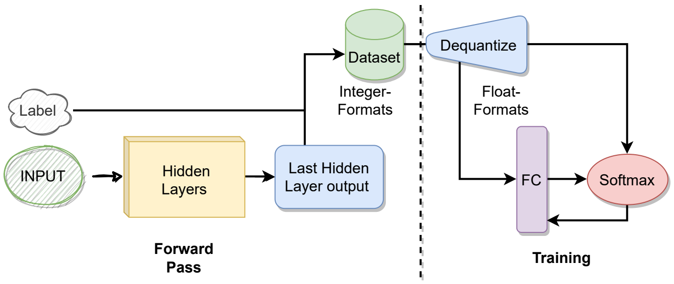


Fig. 1. The two-step MicroTL approach for transfer learning on IoT devices. First, it stores the output of the last hidden layer in int-8 format during the forward pass. Second, it de-quantizes the samples during training and trains a fully connected layer in floating-point format.

for additional learning [8]. The IoT device can only utilize the local quantization scale factors and does not have access to other parameters used during training as opposed to QAT.

2) Device Resource Constraints: The collection of large sample data is restricted on IoT devices primarily due to computational and memory constraints. The forward pass latency on low-power devices is still one of the main bottlenecks for processing samples [24]. Current approaches for transfer learning send the data to the edge and cloud that do not face such resource limitations. These edge or cloud-based methods typically use data sets in high-dimensional inputs, focusing mainly on training latency. Transfer learning on low-power devices needs to address trade-offs between computational, energy, and memory constraints on low-power IoT devices. For example, in the Imagenet data set, the image size is 256x256, occupying 196 KB, almost all the RAM space of a low-power IoT device. Even with smaller image sizes like CIFAR-10, an image (32x32) will occupy 4 KB, which is a significant amount for a low-power device.

B. MicroTL Overview.

The IoT device has a pre-installed quantized neural network trained for a generic problem (source domain). The IoT device interacts with the environment and collects new sensor data. The IoT application forwards the data to the neural network for inference. MicroTL provides two main operations on top of the quantized network. First, the sample collection takes control of the hidden layer outputs and creates a local training data set. For examples, when a user using a smartwatch and engages in activities, the device creates a training set by using another sensor or asking the user through user interaction. Second, when the training set is filled with enough data, it initiates the training of appended Fully Connected (FC) layer(s). In Figure 1, we illustrate the overview of MicroTL.

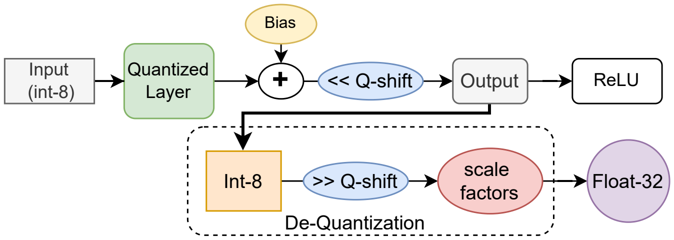


Fig. 2. Overview of the forward pass on a quantized network together with MicroTL's dequantization operation. The figure demonstrate the operation over a single input. In MicroTL we collect several outputs and perform the de-quantization during training. Q-shift is the offset because of the zero-point difference in the real and integer domains. Scale factors come from the min-max quantization.

Algorithm 1: MicroTL Algorithm

```
Batch Size: 4 / 8 / 16 / 32
Pre-install: 8-bit Neural Net
Constants: Learning rate, Momentum, Decay
// Collecting Training Data
while sensor-data d do
    Label := Receive user label $y_t$;
    Forward pass (Net, d);
    Hidden layer output := Net.layer(s).output;
    // 8-bit int format
    TrainingData $(y_i, x_i)$ := (Label, Hidden layer output);
end
// Training Process
FC := Randomized Fully Connected Layer(s)
for epoch e = 1, ... , E do
    for batch m in TrainingData do
        // 32-bit floating point format
        $Y^{(i)}, X^{(i)}$ := de-quantize (m, Q-factors, Q-shift);
        $Y^{(j)}$ := Forward (FC, $X^{(i)}$);
        delta := $-\sum_{i=1}^{m} y_i \log(y_j)$ // cross entropy loss
        Back propagation (FC, delta);
        Update weights (FC, Learning rate, Momentum,
          Decay);
    end
end
```

C. Dequantization.

In order to employ training on resource-constrained devices, we enforce the optimization and the proper placement of the dequantization process. In Figure 2, we present in detail where we collect the intermittent outputs and which operations are involved. After the convolution operation and adding bias, the matrix is shifted due to zero-point. We collect the output before applying ReLU to avoid consecutive zeros. Due to quantization, we have Q-shift, an offset of the difference between the integer domain's zero-point and real numbers. We need to reverse both the min-max scale factors between integer and real numbers and shift back and reverse the zero-point offset.

D. Pre-trained Neural Networks

MicroTL utilizes the hidden layers of the existing pre-trained neural network as feature extractors for applying transfer learning of fully connected layers. MicroTL freezes the pre-trained hidden layers and trains only the Fully-Connected (FC) layer(s) depending on the target domain, specialized to the end-user. MicroTL manages the memory allocation together with the computations of forwarding pass for FC layers. The weights of the FC layer are stored in floating-point precision. The floating-point precision enables the computation of adequate gradients for training with the back-propagation algorithm.

E. Sample Storage

Typical deep learning approaches store high-dimensional training samples on large databases. For low-power IoT devices, this will overwhelm the limited resources. One key observation is that deep hidden layers can reduce the dimensions of the input data. In Table I, we can observe that hidden layer outputs have a smaller dimension (shape) than the input data. Since we train only the weights of the last fully connected layers, in principle, we can pre-compute the outputs of the last hidden layer and store them in a lower dimension data set. MicroTL collects, during inference, the output of the last convolution or recurrent layer and stores it into 8-bit integer-format data set. The training data set is stored locally and MicroTL does not transmit any sensor data or intermittent results to the cloud. The training data is discarded after training, preserving the privacy of the user.

F. MicroTL Training Algorithm

MicroTL's training algorithm uses back-propagation with mini-batch stochastic gradient descent and cross-entropy as a loss function to train the fully connected layers. MicroTL uses 32-bit floating-point representation for the fully connected weights to achieve convergence when calculating the gradients in the backward phase. The pseudo-code version of the training process is presented in Algorithm 1. The algorithm works with batches of 8, 16, or 32, depending on the device's available memory. It de-quantizes a batch sample by dividing the Q-factors and bit-shifting with Q-shift (see Figure 2). However, a key observation is that the division is a power of 2 which can also be done using bit-shifting, and most cross-compilers can highly optimize and combine these operations. The scale factors and Q-shift are pre-calculated using min-max quantization [29]:

$$q_x = \frac{(2^n - 1)/2}{\max(\text{abs}(X.min), \text{abs}(X.max))}$$

MicroTL temporarily stores the batch for each training epoch. The learning rate, momentum, and decay are hyperparameters controlling the learning process. The training process randomly initializes the Fully Connected (FC) layer(s) weights. Next, for each epoch, MicroTL de-quantizes the batch, computes the output loss, and performs backpropagation. Finally, the training process minimizes the loss function (cross-entropy) given the hyperparameters.

IV. Performance Evaluation

This section presents the evaluation of MicroTL on low-power IoT devices. It gives answers to the contributions listed in the introduction on three specific questions: (a) Is transfer learning feasible on low-power IoT devices? (b) What is the accuracy of on-device training compared to edge or cloud-based training? (c) What is the overhead of transfer learning in terms of computation, memory, and energy consumption?

TABLE I
The neural networks used in our experiments. The table shows the shape size, and the layer output in bytes for each layer.

Pre-trained RNN on HAR-UCI-3a			**Pre-trained CNN on CIFAR-7**		
Freezed Layers	Shape	Bytes	**Freezed Layers**	Shape	Bytes
Input	(128, 9)	1,152	Input	(32, 32, 3)	3,072
Conv1D	(43, 9)	387	Conv2D	(30, 30, 64)	57,600
LSTM	(43, 32)	1,376	MaxPool	(15, 15, 64)	14,400
GRU	(43, 64)	5,504	Conv2D	(13, 13, 32)	5,408
GRU	(43, 32)	2,752	MaxPool	(6, 6, 32)	1,152
GRU	(43, 6)	258	Conv2D	(4, 4, 16)	256
Trainable Layers (MicroTL)			**Trainable Layers (MicroTL)**		
Fully Connected	(258, 32)	33,024	Fully Connected	(256 , 32)	32,768
Fully Connected	(32, 3)	384	Fully Connected	(32, 3)	384

Outline. The evaluation is divided into five parts. First, we present the implementation details. Second, we present the training set and neural network of experiments. Third, we present a comparison of MicroTL with edge and cloud-based approaches. Fourth, we present the evaluation of MicroTL on low-power devices. Fifth, we discuss our results and remaining challenges.

A. Experimental Setup

Software & hardware implementation. We define, train, and quantize each deep neural network using PyTorch in Python 3.8. The quantized pre-trained layers are listed in Table I. We utilize CMSIS-NN [14] library for the inference part on IoT devices. We implement MicroTL in C, using the Arm-gcc 9.2.1. We evaluate MicroTL on nRF-52840-DK board featuring: a 32-bit ARM Cortex-M4 with an FPU at 64 MHz, a DSP co-processor, 256 KB of RAM, and 1MB KB of flash. The board has wireless communication capability with Bluetooth Low Energy (BLE), Thread, and Zigbee. The chip of this board is widely used on low-power IoT applications like wearable and smart-watches [20]. Finally, we use the Nordic-Semiconductors Power Profiler Kit (PPK) v1.1 [23] to measure the power consumption.

Code availability. We provide the complete source code of MicroTL in a public repository.[2]

B. Training Sets and Neural Networks

We use two representative machine learning datasets for image classification and human activity recognition. For image classification, we use CIFAR-10 [13], reflecting computer vision IoT applications. For human activity recognition, we use HAR-UCI [1] reflecting typical IoT health tracker applications.

CIFAR-10. The data-set consists of 60,000 colored 32x32 images, where 50,000 are for training and 10,000 for testing. There are ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. We use seven classes (plane, car, cat, deer, dog, horse, ship), named it CIFAR-7, and use it as source domain. CIFAR-7 has 40,000 training and 7,000 testing images. We use the remaining three mutually exclusive classes (bird, frog, truck) for transfer learning as the target domain.

[2] https://github.com/chrpro/MicroTL

TABLE II
AVERAGE ACCURACY OVER THE COMPLETE TEST SET REPORTED AS A PERCENTAGE (%). UPLINK REFERS TO THE DATA WE NEED TO BE TRANSMITED IN KB.

	HAR-UCI-3b			CIFAR-3		
Training	Accuracy (%)		Uplink	Accuracy (%)		Uplink
Sample	MicroTL	Edge-TL	(KB)	MicroTL	Edge-TL	(KB)
100	78±0.4	77±0.4	115	84±0.8	83±0.3	307
200	82±0.2	80±0.2	230	88±0.3	84±0.2	614
300	83±0.1	81±0.1	346	89±0.2	84±0.2	922
400	85±0.1	82±0.1	461	91±0.2	87±0.1	1,230
500	87±0.1	83±0.1	576	92±0.1	88±0.1	1,540
600	87±0.1	83±0.1	692	92±0.1	88±0.1	1,840
ALL (Cloud-QAT)	88±0.1		4,210	93±0.1		46,100

We named the data set CIFAR-3 with 10,000 training and 3,000 test images.

HAR-UCI. The data-set is a human activity recognition data set. HAR-UCI is based on time-series of sensors (accelerometer and gyroscope) captured by smartphone (Samsung Galaxy S II), and it has six classes (activities): 1) walking, 2) walking upstairs, 3) walking downstairs, 4) sitting, 5) standing, 6) laying. The data type is a time series of signals needing preprocessing before use. Our experiments parse the data per 128 time-step and apply min-max normalization over the nine-axis (final dimension 128x9), leading to 7,300 training samples and 3,000 testing samples. We use the following three classes as the source domain (*HAR-UCI-3a*): walking upstairs, walking downstairs, and standing. HAR-UCI-3a has 3,650 samples for training and 1,500 for testing. We use the other three classes as the target domain: walking, sitting, and lying. We named the data set as *HAR-UCI-3b*, and it has 3,650 samples for training and 1,500 for testing.

Pre-trained networks. Our experiments use two pre-trained and quantized neural networks based on CMSIS-NN example models [14]. For the CIFAR image classification problem, we utilize a convolutional neural network (CNN). For the HAR-UCI we utilize a Recurrent Neural Network (RNN). We remove the last fully connected layers and append the Fully Connected (FC) layers of MicroTL. We report the shape size and the output of each hidden layer in bytes. Finally, we list the appended Fully Connected (FC) layers for transfer learning (MicroTL).

MicroTL training sets. We draw balanced (all classes equally represented) random samples of size from 100 to 600 from each training set (CIFAR-3, and HAR-UCI-3b). We split the training set to 90% for training and 10% for validation, and we repeat 50 times the training experiments of MicroTL for each training sample size. We use validation-based early stopping [10] to avoid over-fitting. We use momentum [10] to accelerate training and weight decay [10] to adapt learning rates and set the batch size to 8 for all experiments. Finally, we report the standard deviation as ±.

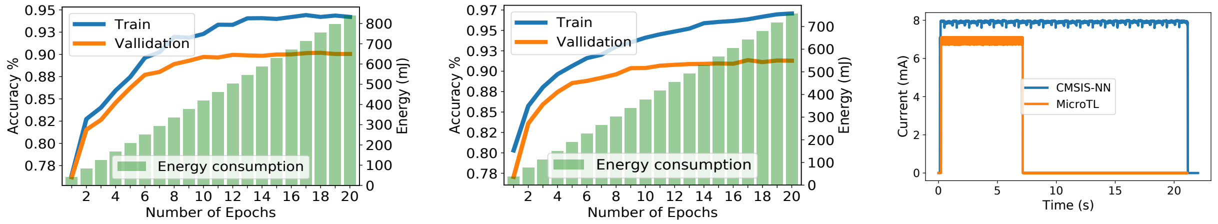
C. MicroTL Learning Accuracy

Baselines. Our experiments compare the accuracy obtained with MicroTL on low-power IoT devices (on-device learning) with the accuracy we would have obtained if we have trained in edge/cloud services. We create two baselines. **Edge-TL** is the straightforward approach to send training data and apply Transfer Learning (TL) in the edge, where a copy of the deployed neural network (before quantization) is stored. The transfer learning in the Edge-TL happens in Python libraries (e.g, TensorFlow), and the network is post-quantized to fit the memory constraints of the IoT device. **Cloud-QAT** is an approach where the device sends all training data to the cloud by applying Quantization-Aware Training (QAT) to optimize a new neural network from scratch. For both Edge-TL and Cloud-QAT, we use the same Python version, and we report the results of the network in the final form to be deployed in the IoT device.

Communication. For the data transmission of the IoT device to the edge and cloud, we assume a Low-Power Bluetooth (BLE) connection, typically available on low-power devices. We report the transmission latency using the standard throughput of 700 kbps (payload) using the default 1 Mbps mode [20]. For energy, we report the energy consumption using the average current drawn from the BLE radio evaluated in previous work [20] (using PPKv1.1) by applying 3 V, which is 7.5 mA.

Accuracy. In Table II, we present the average test accuracy of MicroTL and baselines, Edge-TL and Cloud-QAT. The accuracy is over the complete test set reported as a percentage (%). With 100 training samples, MicroTL has marginally higher accuracy (on test-set) than Edge-TL, but Cloud-QAT can achieve almost 10% higher accuracy than MicroTL. The accuracy improves with more samples, and MicroTL can reach accuracy close to Cloud-QAT on CIFAR-3 and on HAR-UCI-3b. For example, with 500 training samples, the accuracy on test set of MicroTL on CIFAR-3 is 92% close to 93% of Cloud-QAT. Similarly with HAR-UCI-3b MicroTL achieves 87% close to 88% of Cloud-QAT. Finally, we notice that transfer learning in the Edge (Edge-TL) achieves lower accuracy than MicroTL. The main reason is that the MicroTL carefully collects and de-quantizes intermittent outputs in high precision for the fully connected layers. Edge-TL uses standard post-quantization to produce the final network. On the other hand, Cloud-QAT uses quantization-aware training to simulate and duplicate the quantization effect over the full network and outperforms both Edge-TL and MicroTL with respect to accuracy, but the communication cost is higher.

MicroTL training cost. We plot in Figure 3 the average train and validation accuracy together with energy consumption for 20 epochs, using a 500 training samples, where we achieve high accuracy on MicroTL. The green bar is the total energy consumption aggregated with each training epoch. Figure 3a shows the same experiment for HAR-UCI-3b. We achieve 89% validation accuracy after 10 epochs and consume 430 mJ energy. In order to increase the accuracy by a 1-2% margin, we need to spend another 400 mJ. Figure 3b shows the same experiment for CIFAR-3. We achieve 88% validation accuracy after 10 epochs and consume 400 mJ. In order to achieve a 1% margin, we need to consume another 335 mJ.

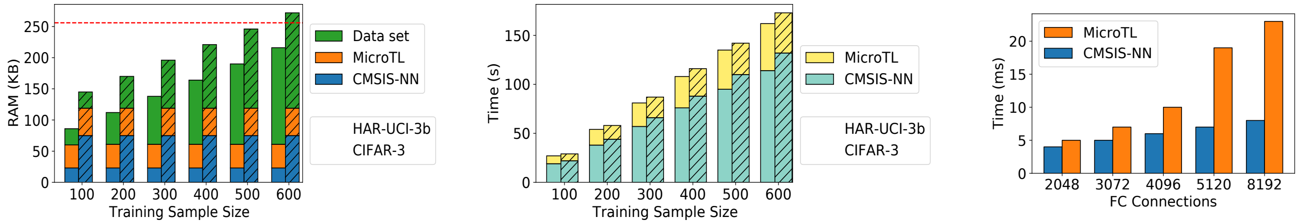


(a) MicroTL train & validation accuracy together with energy consumption on HAR-UCI-3b.

(b) MicroTL train & validation accuracy together with energy consumption on CIFAR-3.

(c) Electric current drawn during forward-pass and training on nRF-52840-DK, by applying at 3 V.

Fig. 3. Train and validation accuracy on MicroTL using 500 samples and 20 epochs. The green bar refers to the total energy consumption aggregated with each epoch. In the third figure, CMSIS-NN refers to forward-pass and MicroTL refers to training with 100 samples on CIFAR-3.Electric current.



(a) The RAM consumption consists of different parts: CMSIS-NN network, MicroTL, and training data set. The red line is the RAM capacity of the device (256KB).

(b) Training time of MicroTL compared to the forward pass. CMSIS-NN refers to forward part of the hidden layers, MicroTL refers to training time of the fully connected layers.

(c) Comparison of the inference time with a floating-point fully connected layer (MicroTL) and CMSIS-NN.

Fig. 4. Performance Evaluation of MicroTL in terms of memory, computation and energy consumption.

However, the training and validation accuracy diverges after 10 epochs, indicating that the network starts over-fitting, and in practice, the algorithm applies early stopping.

Communication cost. In Table II, we report uplink as the amount of data in KB we need to transmit to the edge and cloud (Edge-TL and Cloud-QAT baselines). Downlink is fixed in both baseline, the device needs to receive the new fully connected layers in Edge-TL, which 9 KB for CIFAR-3, 8KB for HAR-UCI-3-b. In the case of Cloud-QAT, the device needs to download a new model, which is 85 KB for CIFAR-3, and 61 KB for HAR-UCI-3-b. However, the device needs to send to the Cloud-QAT a total of 4,210 KB for HAR-UCI-3b, and 46,100 KB data for CIFAR-3. Using BLE for communication will take 50 seconds and consume 1,238 mJ for HAR-UCI-3b, and 552 seconds and consume 13,662 mJ for CIFAR-3. Compared to MicroTL with 500 training samples, achieving similar accuracy, the device will consume a total of 840 mJ and takes 40 s for HAR-UCI-3b. When training on CIFAR-3 it takes 32 s and consumes 735 mJ, on average. Overall, the training process with CIFAR-3, it takes 2.8x less time, and 3x less energy with MicroTL compared to Cloud-QAT to achieve similar accuracy, similarly with HAR-UCI-3b.

D. MicroTL Performance on low-power IoT devices

This section presents the evaluation of MicroTL in terms of memory, time, and energy consumption. We report the average peak (maximum amount) for memory consumption on RAM and flash. We report the average training time and forward-pass based on the cycle clock register. We report the average energy consumption based on the average electric current draw in mA, by applying 3 V on Power Profile Kit. We repeated each experiment 50 times, and we report the standard variation as ±.

Memory consumption. In Table III, we report the RAM and Flash consumption for each training set. We observe that Flash remains the same through all experiments as it stores only static parts and the code sections. In Figure 4a, we demonstrate the average peak memory consumption of RAM divided into three parts: the pre-trained network (CMSIS-NN), transfer learning (MicroTL), and the training data set. The different parts consume memory as follows. CMSIS-NN allocates static memory for storing the quantized network. MicroTL dynamically allocates memory for storing the weights of the fully connected layers and allocates temporary memory for the de-quantized batches during training, along with the gradient matrixes for the backpropagation. Finally, the data set is the inference output collected during the forward pass of CMSIS-NN in an 8-bit integer format.

For example, with CIFAR-3 data set, the pre-trained network (CMSIS-NN) consumes 30% of RAM, and MicroTL consumes 17% of RAM. The data set size is related to the size of the last hidden output, where on CIFAR-3 is 256 bytes (see Table I). A sample size of 100 requires storing 25,600 bytes of data (10% of the RAM), and a training sample of size 500 consumes 49% of the RAM, while a sample size of 600 together with CMSIS-NN and MicroTL exceeds the memory capacity (red dotted-line). With HAR-UCI-3b, we need 258 bytes per sample (see Table I), a data set of 100 consumes 11%, and a data set of 600 consumes 50% of RAM.

Execution time. Next, we evaluate the execution time of training (MicroTL) and the forward-pass of the pre-trained networks (CMSIS-NN). The training time highly depends on the input of fully connected layers. In all experiments, we use two fully connected layers. The first layer has the input size of the last hidden layer of the pre-trained network and an output of 32 nodes. The final layer takes the 32 nodes and creates an output of the number of classes (specific to each task). In Figure 4b, we observe that the average execution of the forward-pass on CMSIS-NN can take up to 25-27% more time than the training part of MicroTL on average. For example, with 100 training samples of CIFAR-3, the forward-pass takes 22 s while the training part of MicroTL takes 7 s on average.

Finally, in Figure 4c, we plot the inference time for a quantized fully connected layer (CMSIS-NN) compared to a floating-point (MicroTL). We can see a cost using MicroTL floating-point over fix-point CMSIS-NN. For example, with a shape of 256x32 and 8,192 neuron connections, it takes 23 ms on MicroTL compared to 5 ms on quantized CMSIS-NN. However, the latency is relatively small on the overall inference time, but as we increase the number of neurons, we expect the FC layers to affect the inference time. A complete evaluation of CMSIS-NN is out of scope, and it has been presented by others' work [14], [24].

Energy consumption. The average current drawn during training using MicroTL is lower than the forward-pass of CMSIS-NN, as illustrated in Figure 3c. MicroTL draws 7 mA on average using the FPU unit to execute the training process. CMSIS-NN draws 8 mA on average using the DSP co-processor for the inference part. The energy consumption depends on the training time, which takes longer as we increase the training data. In Table III, we report the energy consumption of MicroTL for all experiments. For example, with 600 samples, training consumption can go up to 1,008 mJ on average. However, the accuracy is not improving as we increase the training data, and we get similar accuracy with 500 training samples by spending less energy.

E. Discussion and Limitations

In this section, we discuss the trade-offs of transfer learning on low-power devices and the remaining challenges.

Hidden layers. The first trade-off we face is the depth of the hidden layers. The pre-trained network needs to have adequate depth so as the hidden layers reduce the input dimension size. However, a deeper network takes more time to execute on forward-pass. On the other hand, a smaller and faster neural network will need more memory to store the higher dimension output of the hidden layers.

Resource constraints. One of the main challenges of MicroTL is the limited resources in terms of energy and memory storage. We have observed that the training set can occupy up to 50% of RAM, and training can consume up to 1,010 mJ. However, to achieve similar accuracy with other methods, we will need to send sensor data to the edge or cloud. Besides the privacy concerns, the large number of data needed to train a deep neural network will shortly overwhelm low-power IoT devices. For example, sending all training data of CIFAR-3 will require 46 MB. It will take more time and energy to send the data using BLE communication than train on the device using MicroTL.

TABLE III
THE COMPLETE EVALUATION OF MICROTL.

	HAR-UCI-3b				**CIFAR-3**			
Sample Size	RAM (KB)	Flash (KB)	Time (s)	Energy (mJ)	RAM (KB)	Flash (KB)	Time (s)	Energy (mJ)
100	63	123	8±0.2	168±0.2	70	98	7±0.2	147±0.2
200	89	123	16±0.1	336±0.1	95	98	14±0.1	294±0.1
300	115	123	24±0.1	504±0.1	121	98	21±0.1	441±0.1
400	141	123	32±0.1	672±0.1	146	98	28±0.1	588±0.1
500	167	123	40±0.1	840±0.1	171	98	32±0.1	735±0.1
600	193	123	48±0.1	1,008±0.1	197	98	41±0.1	882±0.1

Number of classes. Our approach works for a relatively small number of classes. We can increase the number of classes by using larger Fully Connected (FC) layers, but it will lead to two issues. First, FC layers will need more data and time to train, and second, it will increase inference time. Overall, solving more complex problems with transfer learning will lead to higher energy consumption.

V. RELATED WORK

This section lists the related work regarding different methods for learning on edge devices, and other compression methods for neural networks.

Edge learning. Machine Learning (ML) is primarily associated with cloud services, but recently we have experienced a move from cloud-centric ML, where training and inference happen in the cloud, to learning on edge devices. For example, TensorFlow Lite [27] offers solutions to use on-device learning for mobile phones. Different promising on-device methods have been proposed like transfer learning [2], incremental learning [3], meta-learning [6], and few-shot learning [26]. In all these methods, the device adds new classes or makes predictions based on a limited number of samples. This paper focuses on bringing transfer learning from edge to resource-constrained devices.

Transfer learning. DeepCham [16] and IoTTL [2] have shown the feasibility of transfer learning on edge devices (e.g., Raspberry Pi). However, they overlook the challenges of low-power devices, especially the energy and memory limitations, and they do not consider quantized pre-trained networks. Moreover. their models require MBs of RAM and floating-point precision networks. In contrast, MicroTL tailors transfer learning on low-power IoT devices running on 32-64 MHz and KBs of RAM by addressing their recourse constraints, without needing extra parameters for training and by dynamically utilizing personalized data.

Federated learning. With federated learning, multiple devices train a global neural network using multiple datasets located locally on the device. Training methods of federated algorithms often use Homomorphic Encryption (HE) [11] or Secure Multiparty Computation (SMC) [5] to protect the privacy of the users. Due to this, federated learning today is commonly done on edge class devices. These requirements are demanding for low-power devices. With our work, we bring on-device learning to low-power IoT class devices. Federated learning can be used to create a joint global model, and MicroTL can personalize the model locally on the device.

Network search and compression. Other methods for compression neural network for IoT devices include: pruning [17], Network Architecture Search (NAS) [22] This paper focuses on 8-bit quantized methods due to the wide support and success on low-power devices and keeping the accuracy close to the original networks. Even though pruning and NAS can significantly reduce the network's size, they suffer from accuracy drops. Other methods using quantized neural network is feasible, for example, with half-wave Gaussian quantization [4] and low-bit neural networks [15], [28]. However, they have a significant accuracy drop.

VI. Conclusion

Inference with Deep Neural Networks (DNNs) is readily available on embedded devices. However, the current approaches assume that training and compression of DNNs happen in the cloud. These approaches overlook three key issues. First, IoT devices are deployed and interact in real-world environments, and their initial problem can change. Second, uploading personalized data to cloud service to re-train neural networks poses privacy concerns. Third, resource-constrained devices need to use energy-efficient communication with limited reliability and network bandwidth, which can delay or restring access of training samples to the cloud. This paper presents MicroTL, an approach to tackle the above challenges by tailoring Transfer Learning (TL) to resource-constrained IoT devices. MicroTL focuses on personalizing deep networks for end-users without sending sensitive IoT data to the cloud and allows IoT devices to adapt to changes in the problem domain. We evaluate MicroTL in terms of accuracy, computation, memory, and energy consumption. Notably, training with MicroTL takes 3x less energy and 2.8x less time than transmitting all data to the edge/cloud.

Acknowledgment

This work was supported by the Swedish Research Council (VR) through the project "AgreeOnIT", the Swedish Civil Contingencies Agency (MSB) through the projects "RICS2" and "RIOT", and the Vinnova-funded project "KIDSAM".

References

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones," in *European Symposium on Artificial Neural Networks (ESANN)*, 2013.

[2] A. Anjomshoaa and E. Curry, "A transfer learning framework for iot-enabled environments," in *ACM International Conference on Internet of Things Design and Implementation (IoTDI)*, 2021.

[3] J. Bai, A. Yuan, Z. Xiao, H. Zhou, D. Wang, H. Jiang, and L. Jiao, "Class incremental learning with few-shots based on linear programming for hyperspectral image classification," *IEEE Transactions on Cybernetics*, 2020.

[4] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[5] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: Programmable and efficient secure two-party computation for machine learning," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[6] Y. Chen, Y. Ma, T. Ko, J. Wang, and Q. Li, "Metamix: Improved meta-learning with interpolation-based consistency regularization," in *25th International Conference on Pattern Recognition (ICPR)*, 2021.

[7] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "Tensorflow lite micro: Embedded machine learning on tinyml systems," *arXiv:2010.08678*, 2021.

[8] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[9] W. Fang, F. Xue, Y. Ding, N. Xiong, and V. C. M. Leung, "Edgeke: An on-demand deep learning iot system for cognitive big data on industrial edge devices," *IEEE Transactions on Industrial Informatics*, 2021.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[11] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *Information Security and Cryptology (ICISC)*. Springer, 2013.

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[14] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv:1801.06601*, 2018.

[15] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," *arXiv:1707.09870*, 2017.

[16] D. Li, T. Salonidis, N. V. Desai, and M. C. Chuah, "Deepcham: Collaborative edge-mediated adaptive deep learning for mobile object recognition," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2016.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.

[18] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," in *Advances in Neural Information Processing Systems, NeurIPS*, 2020.

[19] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020.

[20] "Nordic Semiconductor technical documentation," 2019, https:// infocenter.nordicsemi.com/.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[22] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

[23] "Nordic Semiconductorpower profiler kit," 2019, https://www.nordicsemi.com/ Software-and-tools/Development-Tools/Power-Profiler-Kit.

[24] C. Profentzas, M. Almgren, and O. Landsiedel, "Performance of deep neural networks on low-power iot devices," in *CPS-IoTBench '21*. Association for Computing Machinery, 2021.

[25] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[26] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[27] "Tensorflow methods," 2021, https://www.tensorflow.org/.

[28] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[29] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research. PMLR, 2019.

[30] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-iid data: A survey," *arXiv:2106.06843*, 2021.