P2W: From Power Traces to Weights Matrix - An Unconventional Transfer Learning Approach

Roozbeh Siyadatzadeh, **Fatemeh Mehrafrooz**,

Nele Mentens, and **Todor Stefanov**
Leiden Institute of Advanced Computer Science (LIACS) Leiden University, The Netherlands

s.r.siyadatzadeh@liacs.leidenuniv.nl

Abstract

The rapid growth of deploying machine learning (ML) models within embedded systems on a chip (SoCs) has led to transformative shifts in fields like healthcare and autonomous vehicles. One of the primary challenges for training such embedded ML models is the lack of publicly available high-quality training data. Transfer learning approaches address this challenge by utilizing the knowledge encapsulated in an existing ML model as a starting point for training a new ML model. However, existing transfer learning approaches require direct access to the existing model which is not always feasible, especially for ML models deployed on embedded SoCs. Therefore, in this paper, we introduce a novel unconventional transfer learning approach to train a new ML model by extracting and using weights from an existing ML model running on an embedded SoC without having access to the model within the SoC. Our approach captures power consumption measurements from the SoC while it is executing the ML model and translates them to an approximated weights matrix used to initialize the new ML model. This improves the learning efficiency and predictive performance of the new model, especially in scenarios with limited data available to train the model. Our novel approach can effectively increase the accuracy of the new ML model up to 3 times compared to classical training methods using the same amount of limited training data.

1 Introduction

The advent and subsequent evolution of machine learning (ML) technologies have impacted various aspects of modern life, notably in fields like healthcare, autonomous vehicles, and cybersecurity [28]. However, one of the primary challenges in the ML domain is the gathering of relevant, high-quality data for training an ML model for a specific task. This is because of several reasons such as data confidentiality, high cost, time, and effort of data gathering [10]. One way to address this challenge is to use the knowledge encapsulated in existing ML models that are used for another related task, and to optimize and fine-tune these models to make them suitable for our new task. Such process is known as transfer learning [42].

Despite the clear benefits of transfer learning, a significant issue persists: it is not always easy to find a publicly available well-trained model to start from. For example, it is relatively easy to find a model on the internet for classifying simple objects, but it is not as easy to find a well-trained model for some classes of medical tasks due to several reasons, such as privacy and the expenses associated with such models. This lack of publicly available well-trained models for a specific class of tasks is even more severe in the domain of embedded systems on a chip (SoCs) for ML. For such SoCs, the design of a suitable ML model is impacted by limitations in resources and other requirements to make the model suitable for a specific kind of embedded SoC. These factors often prevent the full realization of the potential benefits of transfer learning in the embedded SoC domain.

Although public availability of well-trained ML models suitable for transfer learning in the embedded systems domain is limited, there are embedded SoCs everywhere that are running such ML models for a wide variety of tasks. The problem is that the general public does not have direct access to the models inside these chips and cannot benefit from the model's knowledge by utilizing conventional transfer learning approaches that require direct access to ML models.

Therefore, in this paper, we present a proof-of-concept of a novel unconventional transfer learning approach called **P2W**. Our approach facilitates the transfer of knowledge encapsulated in a relevant well-trained neural network, running inside an embedded SoC, to a new neural network. P2W enables effective training of the new neural network model to perform new similar tasks. This is done by measuring the power consumption of the SoC while it is running the ML model. We take multiple power traces with different input data and analyze them with the help of an Encoder-Decoder Deep Neural Network (EDNN). Through such power trace analysis we extract, to a large extent, the knowledge of the well-trained ML model in the form of an *approximated* weights matrix. This matrix serves as foundational knowledge to start training new ML models. By initializing the new models with the approximated weights matrix (i.e., transferring the knowledge) and further training them with a limited amount of data, we obtain new well-trained ML models that can perform the same or related tasks very well.

The main novel contributions of the paper can be summarized as follows:

- We propose an unconventional transfer learning approach which circumvents the traditional barriers for researchers/developers of ML models related to the acquisition of high-quality data for training, in sensitive domains such as healthcare for example, where data availability is often restricted due to privacy/ethical concerns. By extracting encoded knowledge in the form of an approximated weights matrix, purely from power consumption traces, researchers/developers can transfer this knowledge to new models without access to sensitive information.
- To support our approach, we introduce a new power analysis technique that utilizes an `EDNN` to translate power traces into an approximated weights matrix.
- Given a limited training dataset, we evaluate the effectiveness of our P2W transfer learning approach by comparing the accuracy of an ML model trained with and without the use of P2W. Our experimental results show that after training the ML model with our P2W approach, we are able to improve the initial average accuracy of the model, which is 37% using only the limited training dataset, to 97% with the aid of P2W followed by fine-tuning with the same dataset.

The remainder of this paper is organized as follows. Section 2 discusses related work, after which Section 3 provides an overview of some existing concepts on which our P2W approach relies. Section 4 presents our P2W transfer learning approach in details. In Section 5, we evaluate our approach based on several metrics to demonstrate its practical applicability and usefulness. Finally, Section 7 concludes this paper.

2 Related Work

This section provides an overview of the most relevant related work, divided into three distinct categories: Power Analysis techniques, Machine Learning on Embedded Systems, and Transfer Learning approaches.

Power Analysis: Power analysis techniques have been pivotal in reverse engineering of neural networks, where researchers employ timing, power, and electromagnetic (EM) data to reveal intricate details like activation functions and network structures [3]. Wang et al. [34] explore how to deduce the structure of deep neural network (DNN) models deployed within in-memory computing (IMC) systems through power analysis. Investigations in [37] collect voltage and current data to infer model structures, and [38] demonstrate an attack on FPGA-based DNN accelerators using EM leakage. In [9], the authors present a novel approach of utilizing cache timing side channels that offers insights into DNN structures. Our new power analysis technique complements the aforementioned techniques by generating an approximated weights matrix directly from power traces, which the existing techniques do not provide, as they focus mainly on uncovering the DNN model structure. In contrast, our focus is on transferring as much as possible knowledge, encapsulated in a DNN, via the approximated weights matrix.

Machine Learning on Embedded Systems: ML models deployed on embedded systems find use in a wide range of fields, including medical imaging and autonomous driving. Innovative applications of ML models include GFUNet, which integrates Fourier Transform with U-Net architecture for medical image segmentation [16]. Edge computing benefits from a long short-term memory (LSTM) model on TensorFlow Lite for gesture recognition in wearable devices as demonstrates in [4]. A lightweight combination of LSTM and a multi-layer perceptron (MLP) model has enabled real-time electrocardiogram (ECG) anomaly detection for internet of things (IoT) devices [29]. Additionally, MLP models have been optimized for medical decision-making within medical IoT (MIoT) using boosting and dimension reduction for efficient predictions [15]. Tiny convolutional neural networks (CNNs) on low-power devices have demonstrated the robustness of transfer learning in autonomous driving [26], while MLP models have shown efficacy in early breast cancer detection [26]. In contrast, our work harnesses the power of `EDNN` models to perform analysis of power traces captured for SoCs, thereby enabling the generation of approximated weights matrices for transfer learning, which is a novel application of `EDNNs` in the transfer learning and embedded systems domain.

Transfer Learning: Transfer learning addresses the challenge of scarce or difficult-to-obtain training data for ML models by leveraging data from a related but different source domain,

els pre-trained on large datasets can be fine-tuned for specific tasks with limited data, such as adapting general image recognition models to medical image analysis [13, 41]. Human activity recognition benefits from transferring knowledge across different sets of activities, improving model accuracy when applied to new activities [7]. In software defect detection, transfer learning allows models trained on data from one software project to be applied to another, even with different metrics [19]. Additionally, multi-language text classification uses transfer learning to adapt models for different languages, enabling improved performance in scenarios with limited labeled data in the target language [40, 39]. Other applications include environmental monitoring, where models trained on data from one geographical region are adapted to another, and financial forecasting, where models utilize data from different market conditions to enhance predictive accuracy.

These transfer learning approaches typically require direct access to a fully functional ML model and its parameters and coefficients, which is not always feasible. In contrast to existing transfer learning approaches, our P2W transfer learning approach does not require direct access to such parameters and coefficients because it utilizes power traces to extract such information. Thus, P2W expands the applicability of transfer learning to scenarios where direct access to ML models is not feasible.

3 Background

In our P2W transfer learning approach, we utilize a variety of methods and models, and in this section, we provide an overview of the three most crucial among them. These include the *Power Analysis* method, the *EDNN* model, and *F1-score*.

3.1 Power Analysis

Power analysis is a method that exploits variations in the power consumption of a processing system during operation. The method analyzes the power consumption pattern to extract (sensitive) internal data and computational states [12]. Initially developed for probing cryptographic systems, power analysis has now expanded its applications, finding relevance in fields such as neural networks [27].

Differences in the power consumption patterns are at the core of power analysis. For instance, it is possible to monitor the power consumption while a device performs cryptographic operations and use this data to infer secret keys or internal states. The power consumption P at any time t can be expressed as:

$$P(t) = \sum_{i=1}^{n} a_i \cdot P_i(t) + P_{\text{static}} \tag{1}$$

where a_i represents the activity factor of the i-th internal component, $P_i(t)$ is the dynamic (data-dependent) power consumption, and P_{static} is the static power consumption. By observing the variance in $P(t)$, particularly during data-dependent operations, it is possible to discern patterns that correlate with the processed data [12].

3.2 Encoder-Decoder Deep Neural Network (EDNN)

An Encoder-Decoder Deep Neural Network (EDNN), is a specialized form of a deep neural network designed to learn efficient representations (or encodings) of input data. The architecture of an EDNN is divided into two main parts: the *encoder*, which condenses the input data into a compressed representation or code, and the *decoder*, which attempts to reconstruct the input data from this compressed code.

The process of training an EDNN is fundamentally centered around minimizing a loss function, which quantifies the discrepancy between the original input data and the generated data produced by the network. The loss function can be expressed as follows [20]:

$$\mathrm{Loss}(X, \widehat{X}) = \frac{1}{N}\sum_{i=1}^{N} \|f(x_i;\phi) - \hat{x}_i\|^2 \tag{2}$$

In Equation 2, X and $\widehat{X}$ denote the sets of input and target data respectively, where each $x_i \in X$ is an individual input data point and $\hat{x}_i \in \widehat{X}$ is its corresponding target. The function $f(x_i;\phi)$ represents the output of the neural network for the input x_i, parameterized by ϕ, which encapsulates the trainable parameters of the network (including weights W and biases b). The term $\|f(x_i;\phi) - \hat{x}_i\|^2$ calculates the squared Euclidean norm, summing the squared differences between elements of the predicted output and target data. This norm is then averaged across all N data points in the dataset to compute the mean squared error. Training an EDNN involves iteratively optimizing the network's parameters (ϕ) to reduce the loss function. This optimization process can be expressed as:

$$\phi^{(\mathrm{new})} = \mathrm{Optimize}\left(\phi^{(\mathrm{old})}, \nabla\,\mathrm{Loss}\right) \tag{3}$$

Equation 3 outlines an iterative optimization strategy aimed at enhancing the network's capacity to accurately reconstruct the input data. During each iteration, the parameters of the network are updated ($\phi^{(\mathrm{new})}$) based on the gradient of the loss function ($\nabla\,\mathrm{Loss}$) and the previous parameter values ($\phi^{(\mathrm{old})}$). The choice of the optimization strategy can vary, encompassing algorithms such as Gradient Descent, Adam, etc [6]. Through this iterative process, the network fine-tunes its parameters to minimize the reconstruction error or loss, thereby improving the fidelity of the data reconstruction and enhancing the model's overall performance on unseen data.

3.3 F1-score

Unlike accuracy, which can be misleading if a model performs well only on one class, the F1-score helps ensure that the model's performance is reliable across all classes [31]. It is crucial to report F1-score in various tasks, such as medical classification, because it provides a balanced measure of precision and recall. The F1-score is defined as:

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

In Equation 4, precision is the proportion of true positive predictions among all positive predictions, while recall measures the proportion of actual positives that are correctly identified. This balance is essential when both false positives and false negatives carry significant consequences [32]. For instance, in medical classification tasks, a false positive might lead to unnecessary treatments or interventions, while a false negative could result in missed diagnoses and lack of essential treatment.

4 The P2W Approach

In this section, we describe our unconventional transfer learning approach in detail. The approach is based on the following assumptions. First, we assume that we only have a small dataset, D_{small}, which does not contain sufficient data samples to train an ML model from scratch and to achieve high accuracy. Also, we assume that we have an embedded device with a *target SoC* containing a programmable processor core which runs a *target ML model* having the knowledge we want to transfer. We can neither reprogram the target SoC nor look at the target ML model inside. The publicly available information we have about the target model is its type (MLP or CNN) and its topology. For example, there are many ARM Cortex-M/A based SoCs, known for their affordability and low power consumption, making them ideal for embedded machine learning applications [25, 17]. In addition, manufacturers of embedded devices typically use ML models inside such SoCs, taken from publicly available libraries like TinyML [35], STM32 model zoo [2], and others. However, the manufacturers often use proprietary training datasets and training approaches to fine-tune these models in order to obtain a specific model which is proprietary as well.

Second, we assume that we can acquire a clone of the aforementioned target SoC without any ML model inside, and we can easily (re-)program this *clone SoC* and experiment with it. This is feasible because well-established SoC manufacturers provide affordable and easy-to-use HW/SW development kits for their popular SoCs used in many embedded devices. For example, Microchip, STMicro, and others sell evaluation and prototyping boards with SoCs based on ARM Cortex-M/A series of processor cores [18, 1].

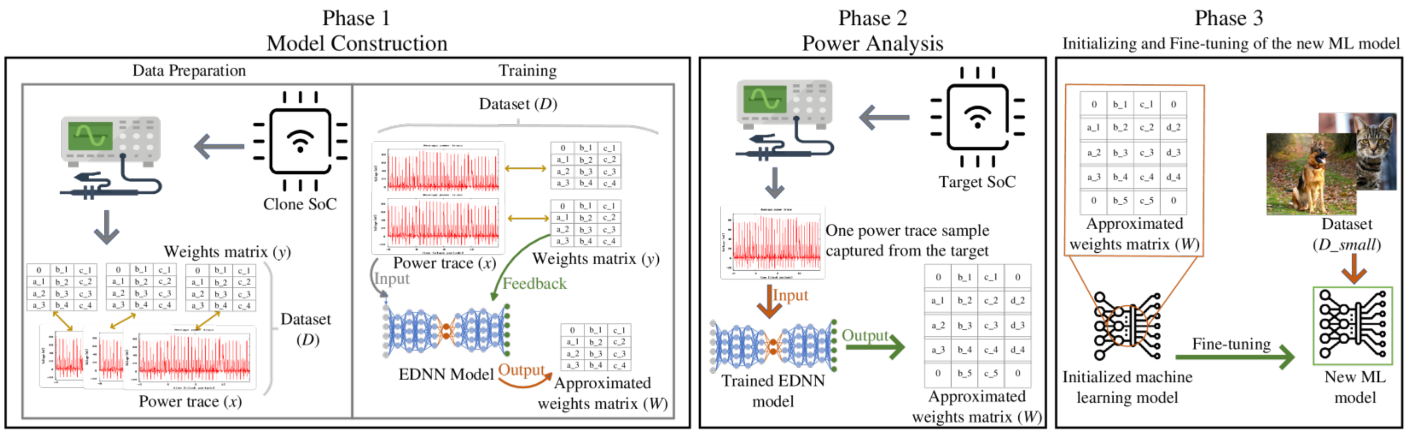


Figure 1: High-level overview of the phases in the P2W transfer learning approach, including EDNN Model Construction, EDNN-based Power Analysis, and Initializing and Fine-tuning of the new ML model.

Considering the above assumptions, Figure 1 illustrates the overall workflow of our P2W approach. The main goal of P2W is to transfer knowledge from a well-trained target ML model (MLP or CNN), running inside a target SoC, to a new ML model in order to establish a basis for transfer learning, and then to fine-tune this new model for another related task using the small dataset D_{small}. We do this transfer by analyzing power traces, obtained from the target SoC running the well-trained ML model, with the help of an EDNN. Our P2W approach consists of three phases.

In PHASE 1, described in detail in Section 4.1, we construct the EDNN which tries to learn the relationship between the power consumption of the aforementioned target SoC and the weights of the target ML model inside. Since, we can neither reprogram the target SoC nor look at the target ML model inside, we use the aforementioned clone SoC and the publicly available information about the model type and topology to perform the EDNN construction in two steps called `Data Preparation` and `Training`. In the `Data Preparation` step , we obtain a training dataset $D = \{d_1, d_2, ..., d_{|D|}\}$ for the EDNN. Every $d_i \in D$ is a pair $d_i = (x_i, y_i)$ where x_i is the power consumption trace captured when the clone SoC runs an inference task using a *surrogate ML* *model* with a set of weights y_i. This surrogate model is of the same type and has the same topology as the target ML model. By reprogramming the clone SoC multiple times to run the surrogate ML model with different sets of weights, we obtain dataset D. It is used in the `Training` step where a standard supervised learning method is applied to train the `EDNN`.

In PHASE 2, called `EDNN-based Power Analysis`, we use the trained `EDNN` to extract knowledge from the target SoC. To achieve this goal, only one power trace x, captured from the target SoC running the target ML model, is fed into the trained `EDNN` constructed in PHASE 1. The `EDNN` translates power trace x into weights matrix W, whose elements are an approximation of the weights and biases of the target ML model. Our power analysis is inspired by the general ideas and principles briefly introduced in Section 3.1.

Finally, in PHASE 3, called `Initializing and Fine-tuning`, we train a new ML model using the aforementioned small dataset D_{small} and the approximated weights matrix W obtained in PHASE 2. First, we initialize the new ML model with W, and then we further train/fine-tune this model using D_{small}. The fine-tuning process involves optimizing the weights and biases to adapt the new ML model to perform a specific inference task.

In the rest of this section, we explain PHASE 1 and PHASE 3 of our P2W approach in more detail.

4.1 Phase 1: EDNN Model Construction

As previously mentioned, PHASE 1 consists of two steps. In the first step, `Data Preparation`, the goal is to prepare dataset D for training the `EDNN` model. To create this dataset, we deploy the surrogate ML model on the clone SoC and enable it to perform inference. However, the surrogate ML model must have been trained to perform an inference task similar to the task performed by the target ML model. For example, if we want to transfer the knowledge of a target ML model performing a medical image classification task, then we have to deploy a surrogate ML model on the clone SoC that has been trained to perform similar tasks such as object image classification, animal image classification, etc.

The next action in step `Data Preparation` involves inputting a single random data sample to be processed by the clone SoC running the surrogate ML model, and capturing the SoC's power consumption from the start to the end of the inference process for this random data sample, thereby obtaining the corresponding power trace. It is important to note that the input data sample selected randomly must remain the same throughout the entire process, in both PHASE 1 and PHASE 2. The sampling rate used to capture the power trace depends on the SoC's clock frequency. A suitable sampling rate can be determined experimentally for each case. Our studies show that it is better to keep the sampling rate not lower than $1/3$ of the SoC's clock frequency. However, training the EDNN with long and high resolution power traces in dataset D may increase the computational complexity. To address this issue, we reduce the length of the power traces using a method called Principal Component Analysis (PCA) [33]. The primary advantage of PCA is its ability to convert a large set of correlated variables, such as millions of samples in the power trace, into a smaller set of uncorrelated variables known as principal components. This process effectively retains the most significant features of the original dataset, ensuring that essential characteristics of the power trace are preserved, thereby maintaining the integrity of the signal for subsequent analysis.

The surrogate ML model has to be deployed on the clone SoC multiple times in order to obtain dataset D. Each time we deploy and run the surrogate ML model, we have to use the same randomly selected input data sample and a different set of weights y_i in order to capture a new power trace x_i during the model inference.

After dataset D is created, the `Training` step of PHASE 1 commences. As illustrated in Figure 1, we use D to train an EDNN model including a convolutional encoder that receives data as input and a convolutional decoder that transforms the encoder's output. The topology of the EDNN depends on the accuracy which we want to achieve, the quality of the captured data in dataset D , and the training method which we use to train the EDNN. Based on dataset D containing pairs $d_i = (x_i, y_i)$, we use the standard supervised learning process, described in Section 3.2, to train the EDNN. In this context, x_i represents X in Equation 2, while y_i represents $\widehat{X}$ in the same equation.

```
**Input:** $S = \{s_1, ..., s_{|S|}\}, r, surML, genML, \theta$
**Output:** $genML$
$acc = 0$; $k = 1$; $Sub \leftarrow \varnothing$;
**while** $acc < \theta \wedge k \leq |S|$ **do**
    $Sub \leftarrow Sub + s_k$;
    **for** $s \in Sub$ **do**
        **for** $i = 1$ $to$ $r$ **do**
            $surML \leftarrow$ Train($surML, s$);
            Deploy $surML$ onto clone SoC;
            $x \leftarrow$ Capture power during inference;
      // Generating the weights matrix
            $y \leftarrow$ Coefficients_to_Matrix($surML$);
            $D \leftarrow D + (x, y)$;

        end for

    end for
```

```
17      $(acc, genML) \leftarrow$ Train$(genML, D)$;
18      $k = k + 1$;
19
20   end while
21  return genML;
22
23   Function Coefficients_to_Matrix(surML):
24      $y \leftarrow \emptyset$; $j = 0$;
25      for $l \in surML.Layers$ do
26          for $n \in l$ do
27              $z = 0$;
28              for $c \in n$ do
29                  $y[j][z] = c$;
30                  $z = z + 1$;
31              end for
32              $j = j + 1$;
33          end for
34
35      end for
36
37      return $y$;
38
39
40
```

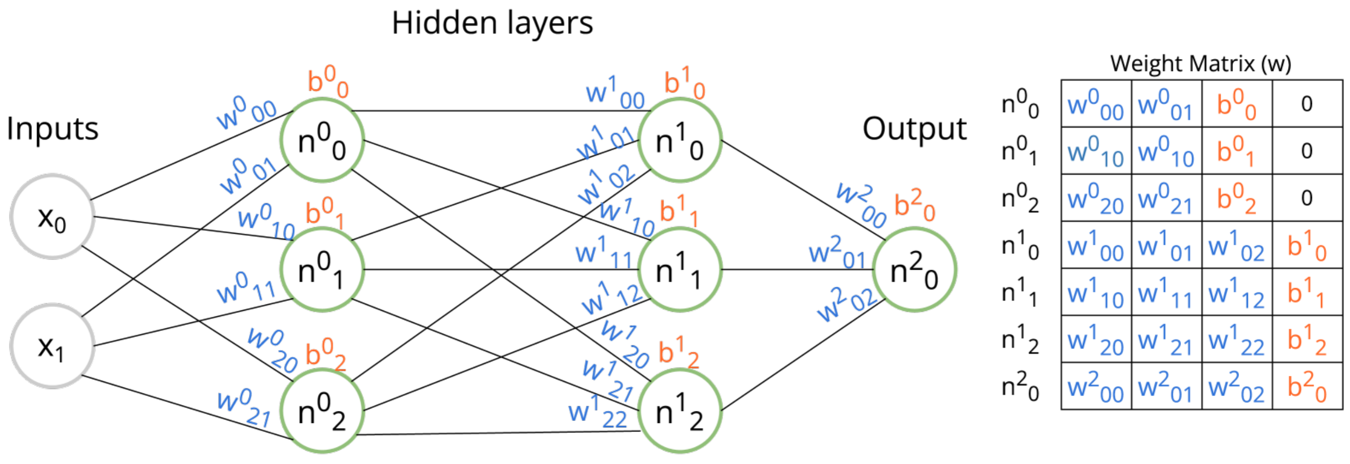
Algorithm 1 Dataset Preparation and EDNN training

Algorithm 1 details the `Data Preparation` step (lines 5-15) for creating dataset D and the `Training` step (line 16) to train the EDNN with the goal of achieving an accuracy of θ (line 4). This algorithm takes set $S = \{s_1, ..., s_{|S|}\}$, a collection of available datasets needed to train surrogate ML model $surML$, and r, which indicates the number of times the training process is repeated for the surrogate ML model with each dataset $s_i \in S$. The two other inputs of the algorithm, namely $surML$ and $genML$, represent the surrogate model and the EDNN model, respectively, both initialized with some initial weights. The last input is the required accuracy θ for the EDNN. The output of Algorithm 1 is the trained EDNN $genML$.

In line 3 of Algorithm 1, three variables are initialized: acc, the current accuracy of $genML$; k, a counter keeping track of the involved datasets used to train $surML$; and Sub, a subset of S initialized to an empty set $\emptyset$.

In lines 4-18, a while loop iterates until the current accuracy of $genML$ exceeds θ or k exceeds $|S|$, indicating that there are no more datasets in S to continue the training with. In line 5, one dataset $s_k \in S$ is added to Sub. Subsequently, from lines 6 to 15, the algorithm repeats the training of $surML$ for each dataset $s \in Sub$, thereby expanding dataset D for r times. In line 8, the surrogate ML model *surML* is trained with dataset s. In line 9, the trained model is deployed to the clone SoC, and after deployment, the power trace x of the clone SoC is captured during inference (line 10). In line 12, all coefficients of *surML* are put in the weights matrix y with the help of function $Coefficients\_to\_Matrix()$. Subsequently, in line 13, the pair (x, y) containing the power trace x and the corresponding weights matrix y is added to dataset D. Finally, in line 16, the *genML* is trained with dataset D, and the current accuracy of *genML* is saved in *acc*.

The behavior of the function $Coefficients\_to\_Matrix()$, used in line 12, is detailed in lines 20-32. This function takes a surrogate model *surML* as input and returns a matrix y containing all coefficients of *surML*. Within this function, lines 22-31 contain three nested loops. The first loop iterates over each layer l of *surML*. The second loop iterates over each filter/neuron n in layer l. The innermost loop iterates over each coefficient c in filter/neuron n. In line 26, each coefficient c is added to the weights matrix y. An example of the output matrix y from the function $Coefficients\_to\_Matrix()$ applied to an MLP model is shown in Figure 2. The MLP model consists of four layers, where the first layer contains 2 inputs, both the second and third layers consist of 3 neurons each, and the last one is the output. In the output weights matrix y, each row contains the weights and bias from a specific filter/neuron n_j^i where i is the index of the layer and j is the index of the node within that layer. For example, n_2^0 represents neuron number two in layer 0, and w_{21}^0 represents the coefficient number 1 of neuron n_2^0.



n^0_0	w^0_{00}	w^0_{01}	b^0_0	0
n^0_1	w^0_{10}	w^0_{10}	b^0_1	0
n^0_2	w^0_{20}	w^0_{21}	b^0_2	0
n^1_0	w^1_{00}	w^1_{01}	w^1_{02}	b^1_0
n^1_1	w^1_{10}	w^1_{11}	w^1_{12}	b^1_1
n^1_2	w^1_{20}	w^1_{21}	w^1_{22}	b^1_2
n^2_0	w^2_{00}	w^2_{01}	w^2_{02}	b^2_0

Figure 2: An example of a weights matrix construction.

4.2 Phase 3: Initialization and Fine-tuning

In PHASE 3, we aim at training a new ML model using transfer learning. The first step involves initializing the new ML model with the approximated weights matrix W, which is generated by the EDNN in PHASE 2. Since we keep the topology of the surrogate ML model, the target ML model, and the new ML model the same across all three phases, the shape of the approximated weights matrix W matches the shape of the new ML model's weights matrix. This means that every weight and bias in the new ML model has a corresponding value in the approximated weights matrix W and this value is used for initialization. Such initialization with corresponding values taken from W infuses the new ML model with pre-established knowledge acquired from the target ML model inside the target SoC. This creates a foundation for the new ML model to achieve levels of accuracy comparable with the target ML model.

Following the initialization step, the new ML model is fine-tuned with dataset D_{small} to obtain the final model as follows:

$$newML_{\text{final}} = \text{Train}(newML_{\text{init}(W)}, D_{\text{small}}) \tag{5}$$

where `Train` can be any supervised learning approach suitable to train and optimize the initialized new ML model ($newML_{\text{init}}$) utilizing dataset D_{small}. Recall that D_{small} does not contain a sufficient number of data samples to successfully train a randomly initialized ML model from scratch and to achieve high accuracy. However, the initialized new ML model with W has a great potential to be trained for its task using this small dataset D_{small} and achieve high accuracy because a significant amount of knowledge has been transferred from the target ML model through W. Finally, strategies such as adaptive learning rate adjustments and early stopping can be employed during this training to fine-tune the model's parameters faster.

5 Evaluation

In this section, we evaluate our P2W approach. The evaluation focuses particularly on two core aspects: the accuracy of the EDNN models obtained in PHASE 1 and used in PHASE 2, and the overall performance of the new ML models obtained after the fine-tuning in PHASE 3. First, we discuss our experimental setup concerning the surrogate ML models (PHASE 1), the EDNN models (PHASE 1 and PHASE 2), and the datasets D_{small} (PHASE 3) in Section 5.1, Section 5.2, and Section 5.3, respectively. This is followed by the evaluation of the P2W approach in Section 5.4, Section 5.5, and Section 5.6.

5.1 Surrogate ML Models and Power Trace Capturing

In our experiments, we utilize two surrogate ML models, called Model 1 and Model 2, each given as input *surML* to Algorithm 1. These models are two different fully connected neural networks, i.e., MLP networks with different topologies, that are trained in Line 8 of Algorithm 1 to perform different classification tasks.

Model 1: This surrogate ML model is trained to perform binary classification using the ECG and Heart Disease datasets described in Table 1. The model has one input layer followed by four fully connected hidden layers. The first hidden layer consists of 128 neurons and employs the ReLU activation function. The subsequent three hidden layers have 64 neurons each. Also, the model has an output layer with a single neuron utilizing the Sigmoid activation function.

Model 2: This surrogate ML model is trained to perform ternary classification using the Blood Pressure dataset described in Table 1. The model is a modified version of Model 1 where the number of neurons in the second hidden layer is increased to 128 and the output layer contains three neurons corresponding to the three classes.

In Table 1, each dataset is characterized by four main attributes. The first attribute is `Size`, representing the number of data samples in each dataset. The second attribute is `Features`, indicating the length of each sample in bytes. The third attribute is `Classes`, denoting the number of classes associated with each dataset. The last attribute is `Chunks`, showing the number of chunks, created from each dataset. In addition, Table 1 shows that the datasets are grouped based on their number of classes, thereby forming two groups, where one group consists of datasets associated with 2 classes and the other with 3 classes.

Table 1: Summary of the datasets utilized in the training of the surrogate ML models in PHASE 1.

Dataset	**Size**	**Features**	**Classes**	**Chunks**	**Ref**
ECG	109k	188	2	100	[11]
Heart Disease	319k	16	2	310	[5]
Blood Pressure	1k	50	3	10	[24]

The collection of all chunks belonging to a group of datasets is given as input $S = \{s_1, \dots, s_{|S|}\}$ to Algorithm 1. Every $s_i \in S$ is a chunk which is used to train the surrogate ML models separately as shown in Line 8 of the algorithm. Moreover, input r of the algorithm is set to 30. Thus, we repeat multiple times the training of any surrogate model $surML$ with each chunk of data s (Lines 7-8). During each training round, to address the problem of overfitting, a dropout strategy is integrated with a rate of 0.3 applied after each fully connected layer. This strategy is critical for enhancing the model's generalization capabilities and maintaining consistent performance across different datasets.

In Lines 9-10 of Algorithm 1, to capture power traces during the inference of trained surrogate model $surML$ deployed on the clone SoC, we use the ChipWhisperer platform [21], with the ChipWhisperer-Huskey as the power capture device and the CW313 SAM4S board as the clone SoC. The board is equipped with a 32-bit ARM® Cortex®-M4 RISC processor and allows for a power sampling frequency of 13 MS/s at a resolution of 12 bits per power sample.

Finally, the aforementioned chunk-based repetitive training of surrogate models **Model 1** and **Model 2** with capturing of power traces result in one dataset D containing 2000 distinct pairs (x, y) and another one containing 2800 distinct pairs, respectively. Every dataset D is partitioned and used as follows: 75% used for training the EDNN model, 20% dedicated to testing it, and 5% reserved for validation.

5.2 EDNN Models and Training Parameters

We construct and train our EDNN model in Line 16 of Algorithm 1 using PyTorch [23], primarily thanks to its flexibility and performance. The EDNN topology, given as input $genML$ to Algorithm 1, consists of two parts: Encoder and Decoder. Recall that the purpose of the trained EDNN is to translate a one-dimensional (1D) power trace x into a two-dimensional (2D) weights matrix y (Section 4.1).

Encoder Topology: The encoder part is designed to process a 1D input array of size 1024 elements, where the array represents a power trace. The encoding part starts with a fully connected layer containing 256 neurons followed by a sequence of three 1D convolutional layers. Each of these layers utilizes a kernel with a size of 4 and a stride of 1. The first convolutional layer contains 256 filters, focusing on extracting mid-level features from the input. It is succeeded by a layer with 128 filters. The final convolutional layer, equipped with 64 filters, completes the feature extraction process. ReLU is used as the activation function in convolutional layers.

Decoder Topology: The decoder part takes as an input the features extracted by the encoder part. The decoder part starts with two sequentially arranged 1D transposed convolutional layers, designed to upscale the input feature maps while enhancing spatial details. The first of these layers consists of 256 filters, employs a kernel with a size of 5 and a stride of 1, and incorporates activation function ReLU for non-linearity. The second transposed convolutional layer, employs a kernel with a size of 4 and a stride of 1, has 128 filters, aiming at refining the feature maps. The feature maps are processed by a linear (fully connected) layer whose number of neurons has to match the size of the 2D output weights matrix. This layer is crucial for reshaping the decoder's output to its final dimensions.

Training parameters: Using the EDNN model topology described above, two different EDNN models $genML$ are constructed by executing Algorithm 1 two times, each time with a different surrogate ML model, given as input $surML$, and with an input value of $\theta = 85\,\%$. The surrogate ML models **Model 1** and **Model 2**, described in Section 5.1, are used for this purpose. When Algorithm 1 is executed with $surML$ = **Model 1**, the EDNN training performed in Line 16 is carried out for 100 epochs and when $surML$ = **Model 2** the training is carried out for 130 epochs. A batch size of 100 and a learning rate of 0.001 are employed for all training epochs. We utilise the mean squared error loss function to quantitatively measure the difference between the actual and predicted weights matrices. The Adam optimizer is used during the training thanks to its adaptability and efficiency [8]. To mitigate the risk of overfitting and ensure the model robustness, a dropout mechanism with a rate of 0.5 is applied after each transposed convolutional operation during the training.

5.3 Balanced Datasets D_{small}

Table 2: Summary of the small datasets D_{small} utilized in PHASE 3 for fine-tuning.

Dataset	Classes	Size	Features	Acc	Ref
EEG	2 classes	22	19	31%	[43]
Diabetes	2 classes	45	9	43%	[30]
Sleep	3 classes	150	13	40%	[14]

In this section, we introduce the datasets D_{small} used for the fine-tuning in PHASE 3. Recall that our P2W approach assumes that D_{small} does not contain a sufficient number of data samples to successfully train a randomly initialized ML model from scratch and to achieve high accuracy. We confirm this by showing and discussing a set of experiments that evaluate the accuracy of new ML models trained with D_{small} only.

Table 2 refers to three datasets (column `Ref`), where the EEG and Diabetes datasets contain data samples belonging to two classes, and the Sleep dataset contains samples belonging to three classes. It should be noted that these original datasets are relatively large and balanced. A balanced dataset is a dataset within which all classes have (almost) the same number of data samples. In order to perform experiments showing the performance of P2W in scenarios where we do not have access to a sufficient number of data samples, we use small batches of data samples taken from these large balanced datasets. Thus, whenever we refer to balanced D_{small} in our experiments, it means a small balanced batch of samples from these datasets. The number of data samples in the small batches is mentioned in the `Size` column of Table 2. The `Features` column indicates the length of each sample in bytes.

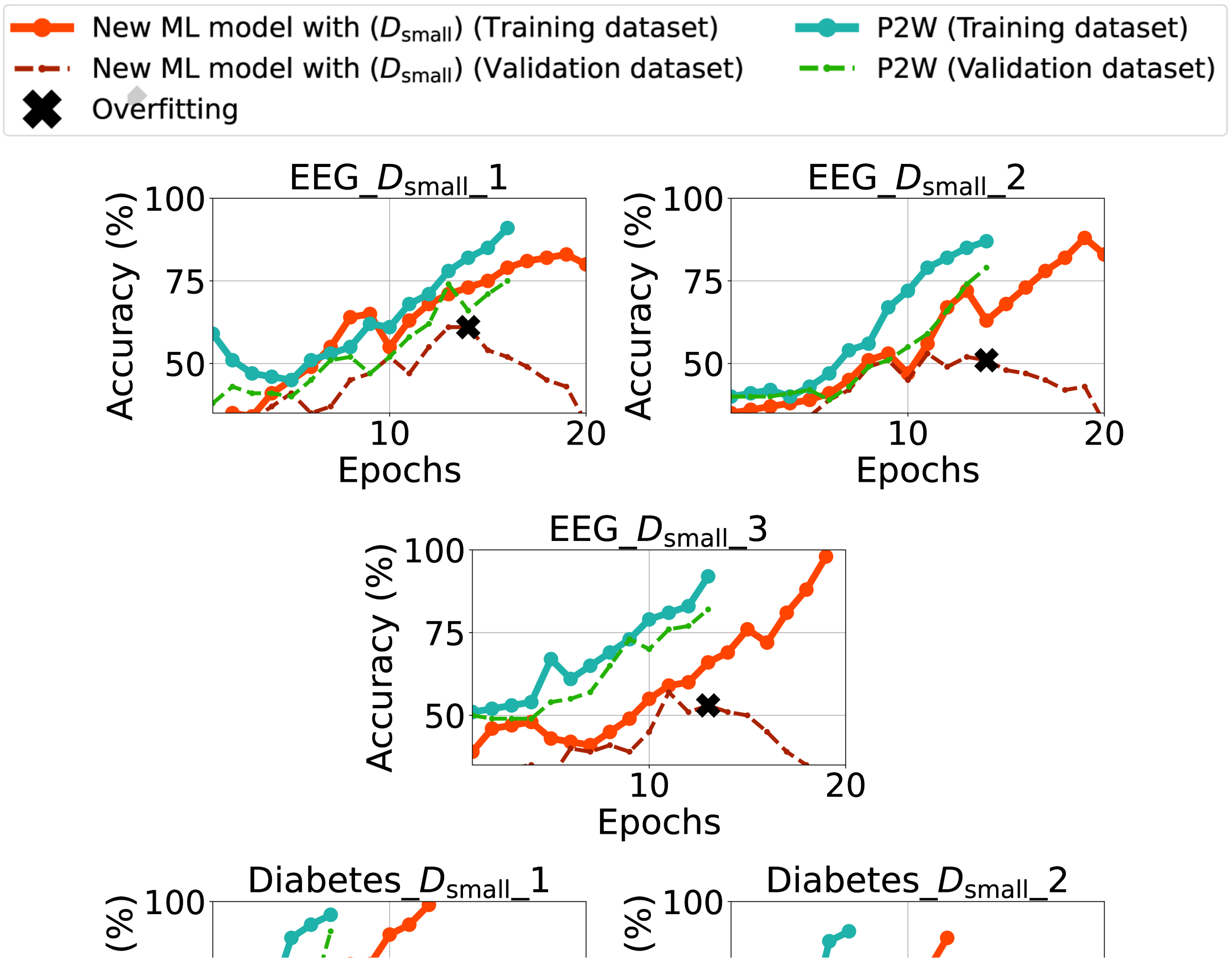
To confirm that the number of data samples in the aforementioned datasets D_{small} is not sufficient to train a new ML model from scratch and achieve high (acceptable) accuracy, we take 20 randomly selected and balanced D_{small} datasets and one *test dataset* from each large dataset (EEG, Diabetes, and Sleep). We take randomly initialized ML models that have the same topology as **Model 1** and **Model 2**, described in Section 5.1, and train them using the D_{small} datasets. For example, Model 2 is trained with the 20 randomly selected and balanced D_{small} datasets from the large Sleep dataset. After each training round, we check the accuracy of the model with the corresponding *test dataset* and take the average over the 20 rounds. The average accuracy is reported in the `Acc` column of Table 2 and is below 43% for all datasets D_{small}. Such low accuracy clearly indicates that the number of data samples (column `Size`) in our datasets D_{small} is not sufficient to train a new ML model from scratch and achieve acceptable accuracy on unseen data.

5.4 Accuracy Evaluation of the EDNN Models

To evaluate the quality of the approximated weights matrices generated by our `EDNN` models, we apply P2W with all three phases discussed in Section 4, but without fine-tuning in PHASE 3. In PHASE 1, we utilize two surrogate ML models, **Model 1** and **Model 2**, to create two datasets D and to train two separate `EDNN` models using Algorithm 1. All these actions are described in Section 5.1 and 5.2.

In PHASE 2, which is the power analysis in P2W, we use the two EDNN models and three target SoCs. Each target SoC runs a well-trained target ML model, whose knowledge we are interested to transfer to a new ML model using one of the EDNNs. By experimenting with three target ML models, we evaluate the EDNNs in three different scenarios. In the first two scenarios the well-trained target ML models perform EEG binary classification and Diabetes binary classification, respectively. For these two scenarios, the first EDNN is obtained in PHASE 1 by utilizing surrogate **Model 1**. In the third scenario, the well-trained target ML model performs Sleep ternary classification, and the second EDNN is obtained by utilizing surrogate **Model 2**.

Each of the three well-trained target ML models runs on the CW313 SAM4S board which is our target SoC in the three scenarios. This board is part of the same ChipWhisperer platform, introduced in Section 5.1, because we use this platform to capture power traces from the target SoC. In PHASE 2, while each of the target ML models is performing inference, we capture one power trace from the target SoC and feed the power trace to the corresponding EDNN. The EDNN generates an approximated weights matrix for the target model.



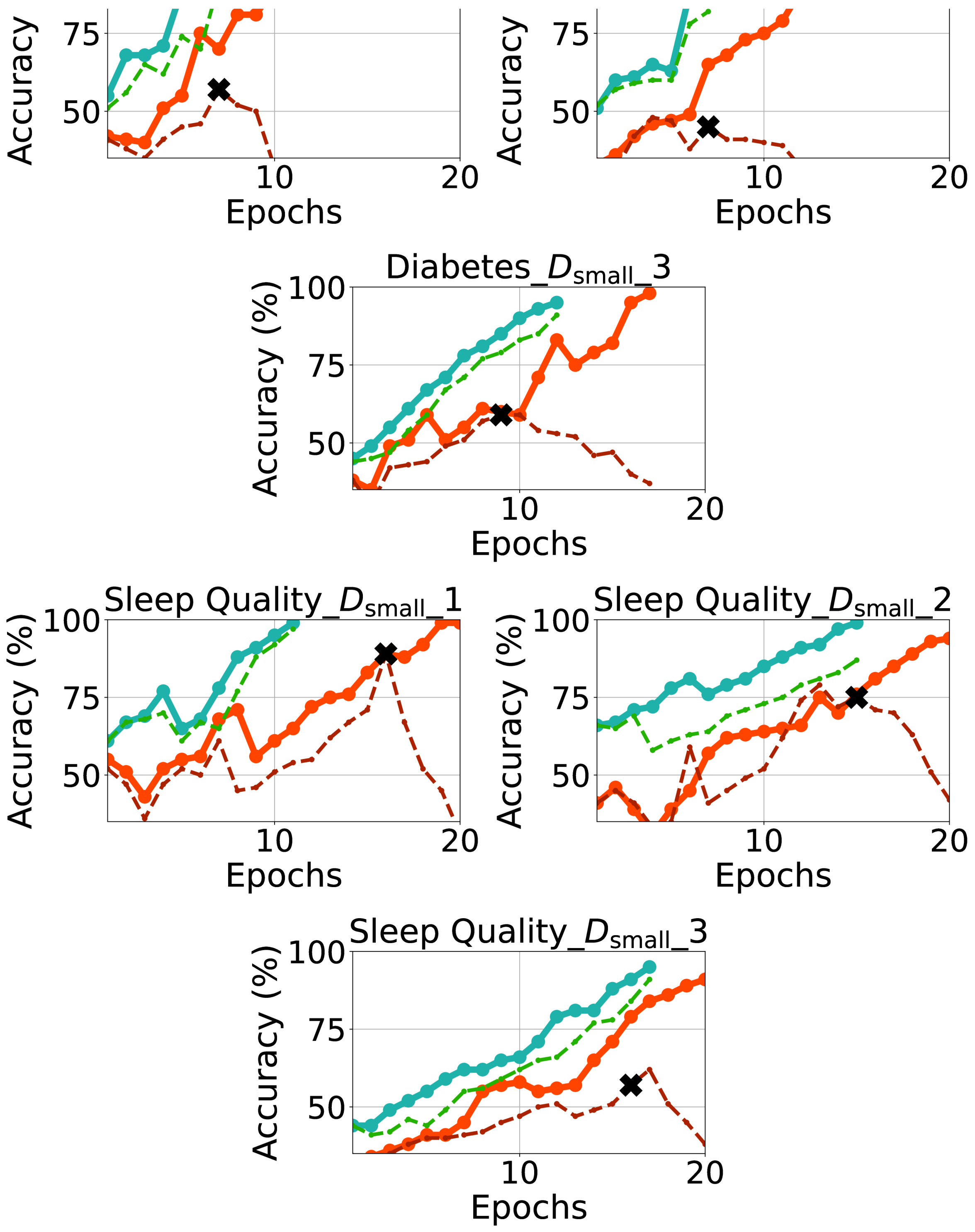


Figure 3: Accuracy of new ML models trained using only D_{small} vs. new ML models obtained using the P2W approach.

After the aforementioned PHASE 2, we obtain three approximated weights matrices (one for every scenario) and we initialize three new ML models with the matrices in PHASE 3. In order to evaluate the quality of the approximated weights matrices generated by the EDNNs, we perform a comparison between the accuracy of the target ML models and the new ML models initialized with the matrices. In this way, we evaluate how much knowledge had been transferred from a target ML model to a new ML model with the goal of performing the same classification task as the target model. For this comparison, we test the accuracy of the target and new models by inferring the data samples in the corresponding *test datasets* described in Section 5.3. These test datasets are unseen by both the target and new ML models.

Figure 4 summarizes the results from our comparison experiment. The horizontal axis shows the three scenarios with the corresponding classification tasks. Each bar represents the accuracy of the models. The gray bars show the average accuracy (column `Acc` in Table 2) of the new ML models trained only with balanced D_{small}. The blue bars show the average accuracy of the new ML models initialized with the approximated weights matrix generated by the `EDNNs` (P2W without fine-tuning). The red bars show the accuracy of the target ML models running within the target SoC.

As can be seen in Figure 4, the average accuracy of the new ML models trained only with balanced D_{small} (gray bars) is too low compared to the target ML models accuracy (red bars). This comparison is also another proof for the fact that datasets D_{small} are too small and insufficient to train a new ML model from scratch. Now, let us analyze the average accuracy of the new ML models initialized with the approximated weights matrices generated by the `EDNNs` – see the blue bars in Figure 4. These bars show that utilizing weights produced by the `EDNNs` results in ML models with a notable increase in inference accuracy compared to the gray bars. Although this average accuracy is not as high as the accuracy of the target ML models (the red bars), it indicates that our EDNN-generated weights are non-random and meaningful in the sense that a large amount (more than 60%) of the knowledge in the target ML models has been transferred successfully to the new ML models via the approximated weights matrices. This fact underscores the efficacy of the `EDNN` models in generating viable approximated weights matrices that could be used as a good starting point to continue the training and fine-tuning of the new ML model to achieve a higher level of accuracy. This will be demonstrated by the experiments described in the next section.

5.5 Performance Evaluation of P2W with Balanced Datasets D_{small}

As mentioned in Section 5.4, the initialization of the new ML models with the approximated weights matrices could be a good starting point for further training and fine-tuning in PHASE 3 with the final goal of obtaining highly accurate new ML models using our proposed P2W transfer learning approach. To continue with the training and fine-tuning of the initialized ML models, we use the small datasets D_{small} introduced in Section 5.3. It is important to note that we use these small datasets to highlight the effectiveness of our P2W approach in scenarios with very limited data availability for training.

To visualize the effectiveness of our P2W approach, Figure 3 shows some of the obtained training and validation accuracy results for the new ML models trained with P2W and the new ML models trained only with the datasets D_{small}. This figure contains nine plots, each corresponding to an ML model trained once with P2W and another time only with one of the D_{small} datasets.

For each plot in Figure 3, the horizontal axis represents the number of epochs, while the vertical axis shows the accuracy. The blue-dotted line and the green-dashed line visualize the evolution of the training and validation accuracy of the new ML model trained with P2W, respectively, across the training epochs. Conversely, the orange-dotted line and the red-dashed line visualize the evolution of the training and validation accuracy of the new ML model trained only with D_{small}, respectively, across the training epochs. Additionally, the point at which the new ML model trained only with D_{small} starts overfitting during training is marked with a black cross.

The results presented in Figure 3 show a clear pattern when a randomly initialized ML model is trained only with D_{small}. As the number of epochs increases, (1) the training accuracy (orange-dotted line) continues to improve, but (2) the validation accuracy (red-dashed line) drops at a certain point (black cross). This indicates that the model's ability to generalize and perform accurate classification on unseen data decreases due to overfitting (black cross). On the other hand, for the models trained with P2W, both training accuracy and validation accuracy keep increasing together, achieving a higher level of accuracy even with a smaller number of epochs.

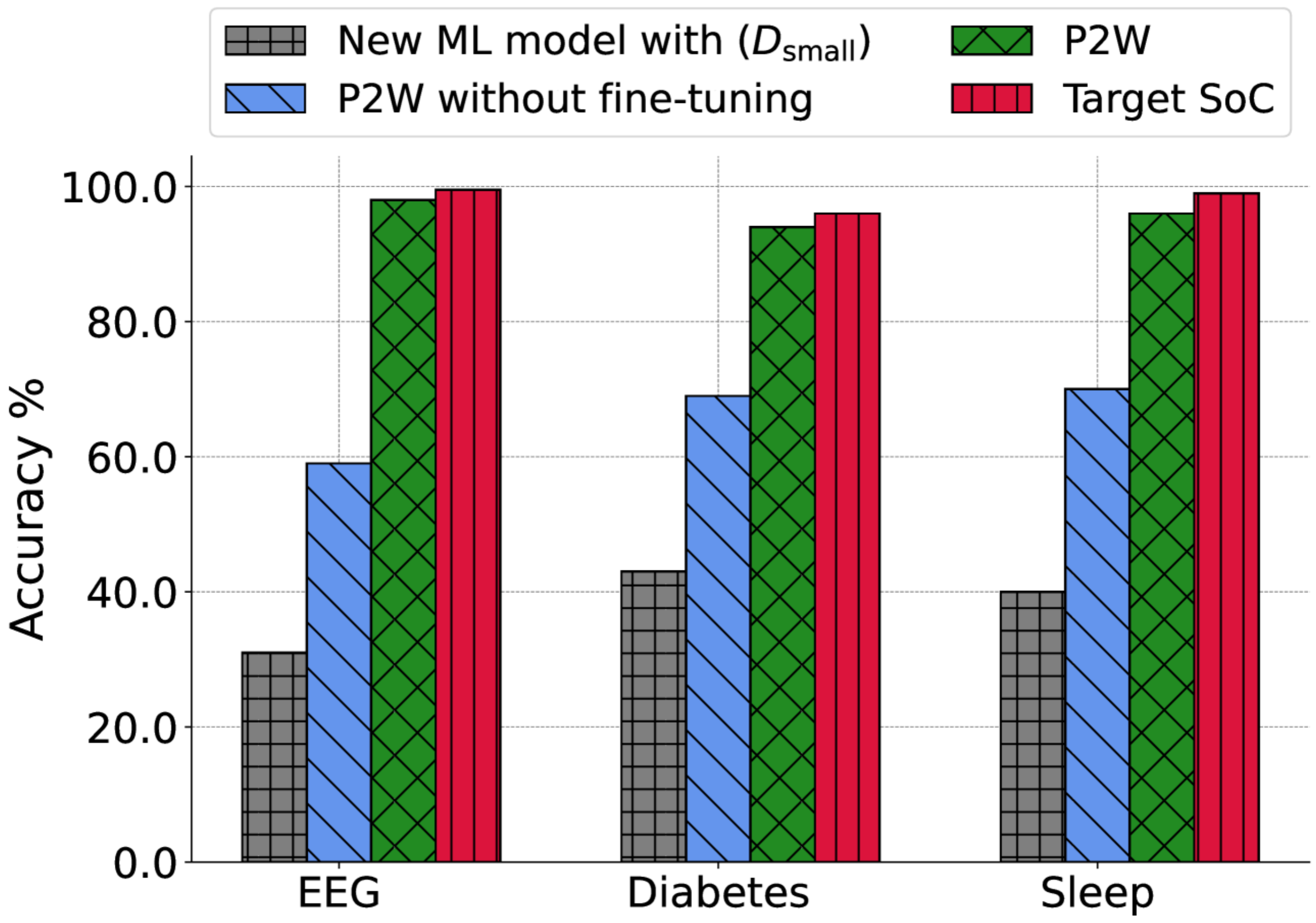


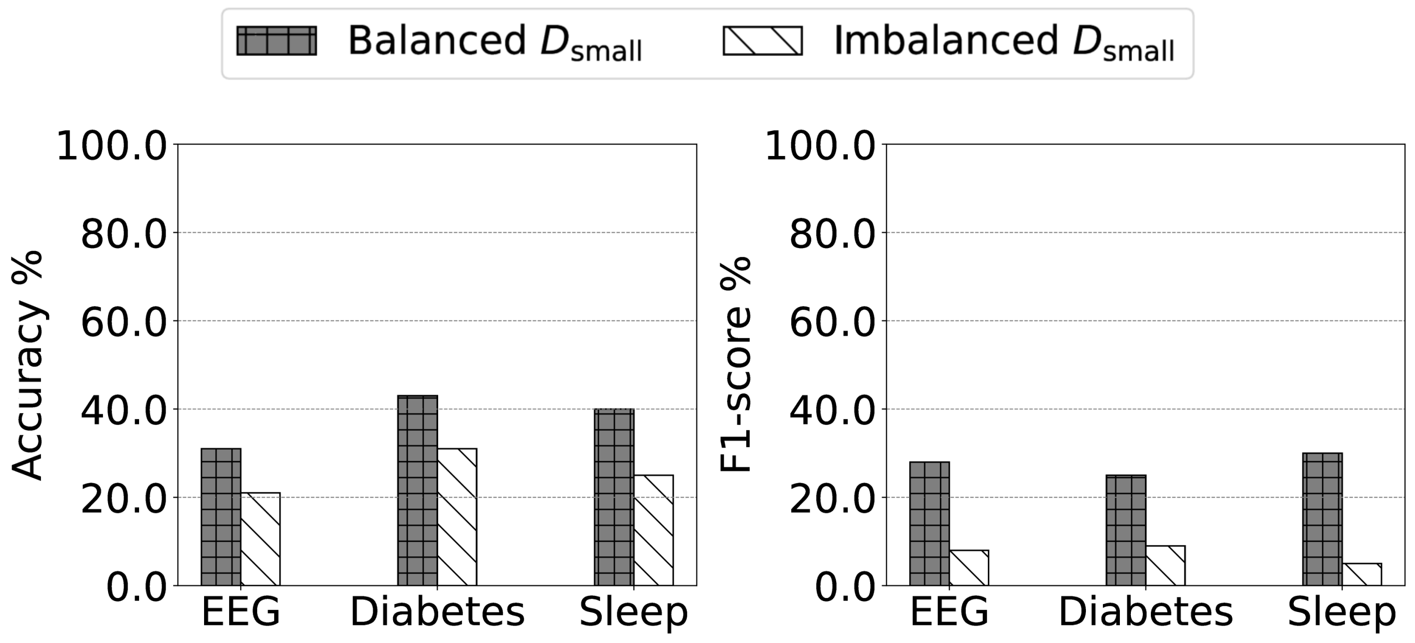
Figure 4: Accuracy of new ML models and target ML models performing EEG, Diabetes, and Sleep classification tasks.

After training and fine-tuning the initialized ML models with datasets D_{small}, we check the average accuracy of the final new ML models trained with P2W, shown with the green bars in Figure 4, by using the corresponding *test datasets* (Section 5.3). Comparing the green bars with the other bars in the figure, we see that new ML models, trained only with D_{small} (gray bars) or obtained only by initialization with the approximated weights matrices (blue bars), have much lower accuracy compared to the new ML models obtained with our P2W transfer learning approach (green bars). On the other hand, as shown in Figure 4, the accuracy of the new ML models obtained with P2W is still a bit lower than the accuracy of the target ML models (red bars). The reason is twofold: (1) it is practically impossible to transfer 100% of the knowledge from the target ML models to the new ML models via the approximated weights matrices generated from power traces; (2) the training/fine-tuning of the models in PHASE 3 is performed with small/limited datasets D_{small} that are assumed to be the only datasets available to the user of our P2W approach.

Nevertheless, the green bars in Figure 4 clearly indicate that new ML models, obtained by P2W, achieve accuracy comparable to the corresponding target ML models. Moreover, in contrast to the target ML models, these new ML models are fully open and available to any user of P2W for further (re-)use, e.g., deployment of the new ML models on different computing platforms and in different application scenarios, fine-tuning them further to perform different tasks, etc.

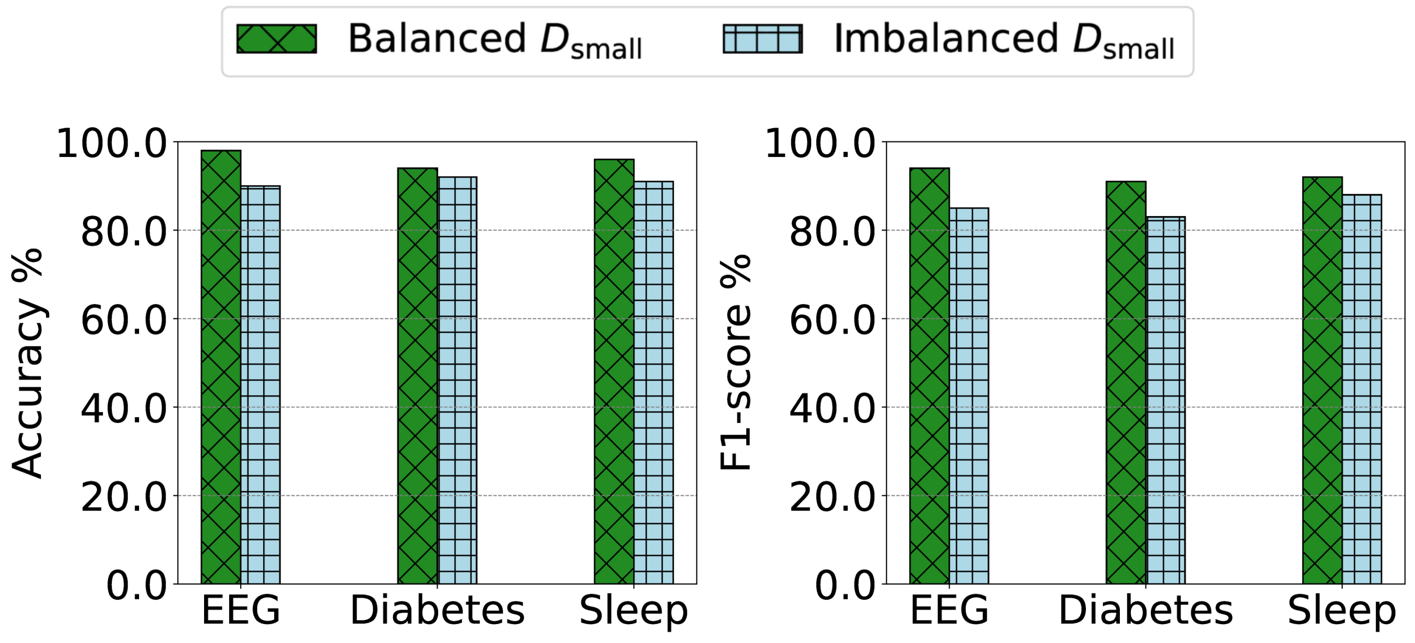
5.6 Performance Evaluation of P2W with Imbalanced Datasets D_{small}

In Section 5.5, we evaluate the performance of our P2W approach in typical scenarios, i.e., scenarios with available *balanced* datasets D_{small}. However, the availability of such balanced datasets may not be always guaranteed due to the limited size of D_{small}. Therefore, in this section, we evaluate the performance of P2W in extreme scenarios, i.e., scenarios with *imbalanced* D_{small}. The goal is to investigate the robustness of P2W to imbalanced training data. An imbalanced dataset is a dataset within which one or some of the classes have a much greater number of data samples than the others. Such imbalance can cause biased training of a new ML model, thus unreliable predictive accuracy of the model.



(a) Accuracy without P2W

(b) F1-score without P2W



(c) Accuracy with P2W

(d) F1-score with P2W

Figure 5: Accuracy and F1-score of ML models obtained with and without P2W using balanced or imbalanced D_{small}.

The imbalanced D_{small} datasets for this evaluation are constructed by randomly taking data samples from the large datasets, listed in column `Ref` of Table 2, such that in the constructed EEG, Diabetes, and Sleep D_{small} datasets, the number of samples in one of the class is $2\times$ greater than in the other classes. We obtain new ML models with P2W that are fine-tuned with the imbalanced D_{small} datasets as well as new ML models trained only with the same imbalanced datasets. Then, we check the average accuracy and F1-score of the models. We conduct the same experiments using the balanced D_{small} datasets, discussed in Section 5.3, and compare all results. Note that we check both the accuracy and F1-score because, unlike the accuracy, which can be misleading if a model performs very well only on one class, the F1-score, as we discussed in Section 3.3, helps ensure that the model's predictive performance is reliably evaluated across all classes [31].

The results of the aforementioned experiments are depicted by the four plots in Figure 5. In all plots, the horizontal axis shows the three classification tasks (EEG, Diabetes, and Sleep) performed by the obtained new ML models, and the vertical axis shows the model's accuracy or F1-score. For each model, there is a pair of bars where the left bar indicates the accuracy or F1-score of the new ML model trained/fine-tuned with balanced D_{small} datasets, and the right bar indicates the same for imbalanced D_{small} datasets. Figure 5(a) and 5(b) show the accuracy and F1-score of the new ML models trained only with the D_{small} datasets. Figure 5(c) and 5(d) show the same metrics for the new ML models obtained with P2W including fine-tuning with the same D_{small} datasets.

By comparing the white bars with the gray bars (our reference points) in Figure 5(a) and 5(b), it can be seen that when a new ML model is trained only with a small imbalanced datasets D_{small}, the model's accuracy and F1-score drop significantly by up to 32% and 72%, respectively. However, when our P2W approach is utilized including fine-tuning with the same imbalanced datasets, the accuracy and F1-score drop only by up to 5% and 7%, respectively, as indicated in Figure 5(c) and Figure 5(d) (compare the blue bars with the green bars as reference points). The results in Figure 5 clearly indicate that the new ML models obtained with P2W not only achieve much higher accuracy and F1-score across all three classification tasks compared to the new ML models obtained without P2W, but they also show that our P2W approach is very robust to imbalanced training data, thereby delivering new ML models with high and unbiased predictive accuracy.

6 Discussion

In this section, we discuss the limitations and future work concerning our P2W approach.

Limitations: In the first two paragraphs of Section 4, we explain our assumptions regarding P2W together with real-world examples for each assumption to show that P2W is realistic in several scenarios. However, the P2W approach is not very effective in scenarios where some of these assumptions are not true. For example, in order to deploy P2W, access to the power supply of the target SoC is needed, which might not always be the case. Nevertheless, for embedded devices, it is realistic that potential users are in the vicinity of the device and the power supply or even own the device. Moreover, instead of measuring the power consumption, similar results can be achieved by measuring the electromagnetic emanation of the chip, which can be done at a distance from the chip [12].

Another limitation is that the clone SoC needs to be of the exact same type as the target SoC. For broadly used embedded SoCs, it is not a problem to find suitable clone SoCs, as explained in Section 4, but for niche applications or exotic devices, it is more difficult to apply P2W.

Finally, P2W requires that there is prior knowledge on the topology of the target ML model. If this is not the case, the technique cannot be applied. To mitigate this limitation, a preliminary power consumption measurement on the target SoC can be performed in order to learn about the type of ML model that is used, as has been shown in related work [34, 3]. Moreover, for certain applications, it is commonly known which ML topologies are being deployed.

Future work: We plan to study the effects of the complexity of very large ML models within embedded devices on power traces. Although ML models utilized in embedded SoCs are typically not large, the scalability of the P2W approach to very large models could be a valuable path for future research. Additionally, as we evaluated P2W on SoCs with an ARM processor inside, exploring the effectiveness of P2W on different types of hardware, such as FPGAs and GPUs, is a promising direction for future research. Given the parallel processing nature of FPGAs and GPUs, it is beneficial to study how they affect power traces in reflecting the relationships and patterns related to ML model computations.

7 Conclusions

This paper presents P2W, an unconventional transfer learning approach for ML models. Our approach is useful for scenarios where direct access to the coefficients (weights and biases) of existing ML models is not feasible and there is only a relatively small dataset available to train a new ML model from scratch. P2W translates a power trace, captured from an embedded SoC running a target ML model inside, to a weights matrix by utilizing an encoder-decoder deep neural network. This weights matrix approximates the coefficients of the target ML model and is used to initialize a new ML model, thereby performing transfer learning from the target to the new ML model.

Experimenting with relatively small training datasets, we evaluate the effectiveness of our P2W approach by comparing the accuracy of a new ML model trained with and without using P2W.

Our experimental results show that after training the model with our P2W approach, we are able to improve the initial 37% accuracy of the model, trained with only a relatively small dataset, to 97% when P2W is utilized including fine-tuning with the same dataset. These results clearly indicate that new ML models, obtained by P2W, achieve very high predictive accuracy.

In addition, we perform an evaluation of P2W in extreme scenarios where we only have small imbalanced datasets, i.e., datasets within which one or some of the classes have a much larger number of data samples than the others. The results of this evaluation show that for a new ML model trained with only a small imbalanced dataset, the accuracy and F1-score drop significantly by up to 32% and 72%, respectively. However, when P2W is utilized including fine-tuning with the same imbalanced dataset, the accuracy and F1-score drop only by up to 5% and 7%, respectively. These results suggest that our P2W approach is very robust to imbalanced training data, thereby delivering new ML models with high and unbiased predictive accuracy.

References

[1] STM32 MCU Developer Zone - STMicroelectronics.

[2] STMicroelectronics – STM32 model zoo, November 2023. original-date: 2023-01-10T13:58:28Z.

[3] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 515–532, USA, 2019. USENIX Association.

[4] Brian Coffen and Md.Shaad Mahmud.
Tinydl: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor.
In *2020 IEEE International Conference on E-health Networking, Application & Services (HEALTHCOM)*, pages 1–6, 2021.

[5] ZUBKOV GEORGY.
Blood Pressure Data for disease Prediction.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep learning.
MIT press, 2016.

[7] Maayan Harel and Shie Mannor.
Learning from multiple outlooks.
In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 401–408, Madison, WI, USA, 2011. Omnipress.

[8] Esraa Hassan, Mahmoud Y. Shams, Noha A. Hikal, and Samir Elmougy.
The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study.
Multimedia Tools and Applications, 82(11):16591–16633, May 2023.

[9] Weizhe Hua, Zhiru Zhang, and G. Edward Suh.
Reverse engineering convolutional neural networks through side-channel information leaks.
In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Amelia Jiménez-Sánchez, Shadi Albarqouni, and Diana Mateus.
Capsule networks against medical imaging data challenges.
In *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 150–160, Cham, 2018. Springer International Publishing.

[11] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh.
Ecg heartbeat classification: A deep transferable representation.
In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE, June 2018.

[12] Paul Kocher, Joshua Jaffe, and Benjamin Jun.
Differential power analysis.
In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[13] B. Kulis, K. Saenko, and T. Darrell.
What you saw is not what you get: Domain adaptation using asymmetric kernel transforms.
In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, page 1785–1792, USA, 2011. IEEE Computer Society.

[14] Laksika Tharmalingam.
Sleep Health and Lifestyle Dataset.

[15] Shin-Jye Lee, Ching-Hsun Tseng, G.T.–R. Lin, Yun Yang, Po Yang, Khan Muhammad, and Hari Mohan Pandey.

A dimension-reduction based multilayer perception method for supporting the medical decision making.
Pattern Recognition Letters, 131:15–22, 2020.

[16] Penghui Li, Rui Zhou, Jin He, Shifeng Zhao, and Yun Tian.
A global-frequency-domain network for medical image segmentation.
Computers in Biology and Medicine, 164:107290, 2023.

[17] Ioan Lucan Orășan, Ciprian Seiculescu, and Cătălin Daniel Căleanu.
A brief review of deep neural network implementations for arm cortex-m processor.
Electronics, 11(16), 2022.

[18] microchip.
Evaluation Boards | Microchip Technology.

[19] Jaechang Nam and Sunghun Kim.
Heterogeneous defect prediction.
In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 508–519, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Michael A Nielsen.
Neural networks and deep learning, volume 25.
Determination press San Francisco, CA, USA, 2015.

[21] Colin O'flynn and Zhizhang Chen.
Chipwhisperer: An open-source platform for hardware embedded security research.

In *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*, pages 243–260. Springer, 2014.

[22] Sinno Jialin Pan and Qiang Yang.
A survey on transfer learning.
IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359, 2010.

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala.
PyTorch: An Imperative Style, High-Performance Deep Learning Library.
Curran Associates Inc., Red Hook, NY, USA, 2019.

[24] PAVAN BODANKI.
Blood Pressure Data for disease Prediction.

[25] Plumerai.
Plumerai wins MLPerf Tiny 1.1 AI benchmark for microcontrollers again | Plumerai Blog.

[26] Tanzeel Sultan Rana, Imran Saleem, Rabia Naseer Rao, Maryam Shabbir, and Laiba Wahid Chaudhry.
Comparative analysis of breast cancer detection using cutting-edge machine learning algorithms (mlas).
Innovative Computing Review, 3(1), Jun. 2023.

[27] Maria Méndez Real and Rubén Salvador.
Physical side-channel attacks on embedded neural networks: A survey.

ArXiv, abs/2110.11290, 2021.

[28] Iqbal H. Sarker, Asif Irshad Khan, Yoosef B. Abushark, and Fawaz Alsolami.
Internet of things (iot) security intelligence: A comprehensive overview, machine learning solutions and research directions.
Mobile Networks and Applications, 28(1):296–312, Feb 2023.

[29] Gawsalyan Sivapalan, Koushik Kumar Nundy, Soumyabrata Dev, Barry Cardiff, and Deepu John.
Annet: A lightweight neural network for ecg anomaly detection in iot edge sensors.
IEEE Transactions on Biomedical Circuits and Systems, 16(1):24–35, 2022.

[30] J W Smith, J E Everhart, W C Dickson, W C Knowler, and R S Johannes.
Using the ADAP learning algorithm to forecast the onset of diabetes mellitus.
In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265. 1988.

[31] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz.
Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation.
In Abdul Sattar and Byeong-ho Kang, editors, *AI 2006: Advances in Artificial Intelligence*, pages 1015–1021, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[32] Abdel Aziz Taha and Allan Hanbury.
Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool.
BMC Medical Imaging, 15(1):29, Aug 2015.

[33] Alaa Tharwat.

Principal component analysis-a tutorial.
International Journal of Applied Pattern Recognition, 3(3):197–240, 2016.

[34] Ziyu Wang, Fan-hsuan Meng, Yongmo Park, Jason K. Eshraghian, and Wei D. Lu.
Side-channel attack analysis on in-memory computing architectures.
IEEE Transactions on Emerging Topics in Computing, pages 1–13, 2023.

[35] Pete Warden and Daniel Situnayake.
Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers.
O’Reilly Media, 2019.

[36] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang.
A survey of transfer learning.
Journal of Big Data, 3(1):9, May 2016.

[37] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang.
Open dnn box by power side-channel attack.
IEEE Transactions on Circuits and Systems II: Express Briefs, 67(11):2717–2721, 2020.

[38] Kota Yoshida, Takaya Kubota, Mitsuru Shiozaki, and Takeshi Fujino.
Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis.
In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 318–318, 2019.

[39] Joey Zhou, Sinno Pan, Ivor Tsang, and Yan Yan.

Hybrid heterogeneous transfer learning through deep learning.
Proceedings of the AAAI Conference on Artificial Intelligence, 28(1), Jun. 2014.

[40] Joey Tianyi Zhou, Ivor W. Tsang, Sinno Jialin Pan, and Mingkui Tan.
Multi-class heterogeneous domain adaptation.
Journal of Machine Learning Research, 20(57):1–31, 2019.

[41] Yin Zhu, Yuqiang Chen, Zhongqi Lu, Sinno Pan, Gui-Rong Xue, Yong Yu, and Qiang Yang.
Heterogeneous transfer learning for image classification.
Proceedings of the AAAI Conference on Artificial Intelligence, 25(1):1304–1309, Aug. 2011.

[42] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou.
Transfer learning in deep reinforcement learning: A survey.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(11):13344–13362, 2023.

[43] Igor Zyma, Sergii Tukaev, Ivan Seleznov, Ken Kiyono, Anton Popov, Mariia Chernykh, and Oleksii Shpenkov.
Electroencephalograms during mental arithmetic task performance.
Data, 4(1), 2019.