RubikONNs: Multi-task Learning with Rubik's Diffractive Optical Neural Networks

Yingjie Li
University of Utah
Weilu Gao
University of Utah
Cunxi Yu (cunxi.yu@utah.edu)
University of Utah

Research Article

Keywords:

Posted Date: April 4th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1497910/v1

RubikONNs: Multi-task Learning with Rubik's Diffractive Optical Neural Networks

Yingjie Li,[1] Weilu Gao,[1,*] and Cunxi Yu[1,*]

[1]*Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112, USA*

[*]To whom correspondence should be addressed; E-mail: weilu.gao@utah.edu; cunxi.yu@utah.edu.

Optical neural networks (ONNs) are emerging as a high-performance machine learning (ML) in terms of power efficiency, parallelism, and computational speed. There are broad interests in leveraging ONNs into medical sensing, security screening, drug detection, and autonomous driving[1]. However, it is challenging to implement reconfigurability for ONNs at specific frequency, e.g., Terehertz (THz), and thus deploying multi-task learning (MTL) algorithms on ONNs requires re-building and duplicating physical diffractive systems, which significantly degrades the energy and cost efficiency in practical application scenarios. This work presents a diffractive ONNs targeting MTL with unreconfigurable components, namely, *RubikONNs*. This architecture utilizes the physical properties of optical systems to encode multiple feed-forward functions by physically rotating the hardware similarly to rotating a Rubik's Cube. We demonstrate two domain-specific training algorithms *RotAgg* and *RotSeq* to optimize MTL performance of RubikONNs. Our analytic results demonstrate more than 4× improvements in energy and cost efficiency with marginal accuracy degradation compared to the state-of-the-art approaches[2,3] for MTL-ONNs. Moreover, we perform a comprehensive RubikONNs design space analysis and explainability, which offers concrete design methodologies for practical uses.

Introduction

Recently, the use of Deep Neural Networks (DNNs) represents the state-of-the-art approaches in many applications, including large-scale computer vision, natural language processing, and data mining tasks[1,4]. However, DNNs have substantial computational and memory requirements, which greatly limit their training and deployment in resource-constrained (e.g., computation, I/O, and memory bounded) environments [5,6]. To address these challenges, there has been a significant trend in building specific high-performance DNNs hardware accelerator platforms. Particularly, there are increasing efforts on implementing optical neural network (ONNs) that mimics conventional feed-forward neural network functions based on light propagation[2,3,7–17]. There optical hardware architectures bring significant advantages for machine learning systems in terms of their power efficiency, parallelism and computational speed. Among them, free-space diffractive optical neural networks (DONNs) based on light diffraction feature millions of neurons in each layer interconnected with neurons in neighboring layers[2,18]. This ultrahigh density and parallelism make this system possess fast and high throughput computing capability.

However, implementing reconfigurability and deploying multi-task learning (MTL) algorithms on most ONNs systems requires re-building and duplicating physical hardware systems, which significantly degrades the energy and cost efficiency in practical application scenarios. The lack of reconfigurability prohibits weight parameters sharing, which is commonly used in conventional neural networks to reduce the computation and energy cost of the neural networks[19–21]. As a result, the existing state-of-the-art DONNs bring significant energy and system cost overhead in

practical MTL application scenarios.

We present a new DONN architecture namely *RubikONNs*, which utilizes the physical properties of optical systems to encode multiple feed-forward functions by physically rotating the systems similarly as rotating a Rubik's Cube. To optimize the MTL performance of RubikONNs, we present two domain-specific algorithms, *RotAgg* and *RotSeq* in *Methods*. As a result, the RubikONNs architecture enables multi-task learning (MTL) using a single-task system, without the need of fabricating new masks. We demonstrate four-task MTL on RubikONNs with implementation cost and energy efficiencies improved more than $4\times$. Finally, we perform a comprehensive RubikONNs design space exploration analysis and explainability to offer concrete design methodologies for practical uses.

Results

Figure 1a illustrates the architecture of the RubikONNs deployed for four-task MTL image classification, which consists of three major components: (1) coherent input images, (2) diffractive layers that encode the neural layers with trainable phase parameters (weights), and (3) detectors that capture the output diffraction images. Specifically, input images are generated when a coherent source passes through image masks. The encoded light wave is diffracted in the free space between diffractive layers and modulated via phase modulation at each layer. As demonstrated in Refs[2,3,18], the output plane is divided into ten detector regions to represent ten classification classes. The final prediction class will be produced by selecting the corresponding class with maximum energy

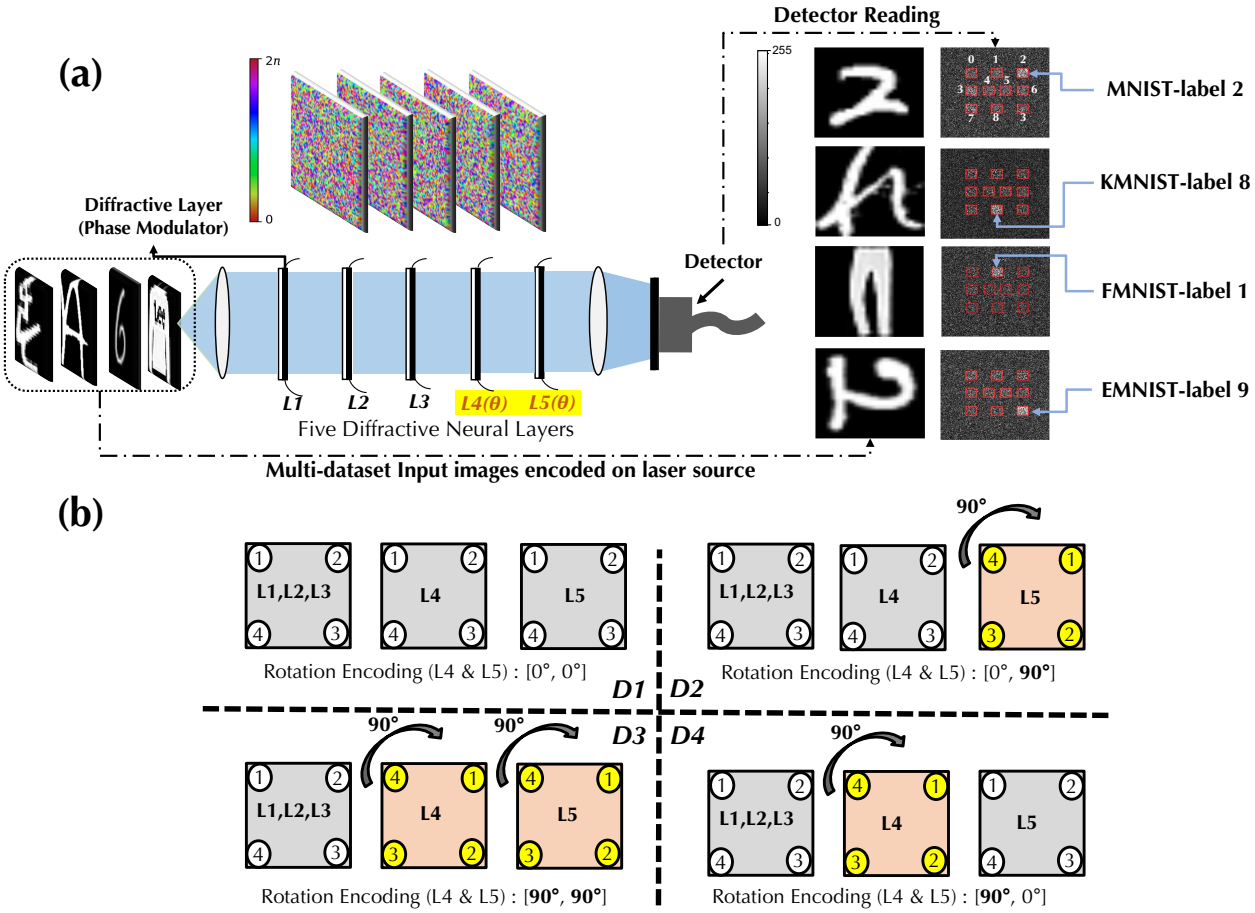

Figure 1: Overview of RubikONNs diffractive neural networks system, consisting of **(a)** laser source encoding input images, diffractive layers providing diffraction and trainable phase modulation (weights), detectors capturing the output diffraction images, and **(b)** a specific rotations patterns introduced for four-task MTL image classification.

sums of those ten detector regions. For example, in Figure 1, based on the label indices of the ten detector regions for image '2', we can see that the 3^{rd} region on the first row has the highest energy. So that the predicted class is class '2'. Similarly, the predicted classes '8', '1', and '9' of other three datasets can be generated by selecting the class with the maximum energy sum captured at the detector.

Figure 1b illustrates the key architectural innovation in RubikONNs to deal with multiple tasks with minimum system overhead, i.e, a MTL DONNs system encodes different forward functions without changing the weight parameters that are physically fabricated in a single-task. Note that the diffractive layers are mostly designed with 3D printed materials, such that the phase parameters (weights) carried by these layers are non-reconfigurable after fabrication. However, as demonstrated in Refs.[2,3], the layers are portable in DONNs and they are in square shapes. This means that we are able to rotate each layer by close-wise $90°$, $180°$, or $270°$, and place the layer back in the system without any other changes. While each layer carries specific trained phase parameters, by rotating one or multiple layers, the forward function becomes different since the weights of the model are changed. In optical domain, this means that the phase modulation of the light changes accordingly w.r.t specific rotation patterns. This offers the main motivation of designing RubikONNs that aims to enable MTL in existing single-task DONNs. As a result, RubikONNs enables MTL by simply **(1)** pulling out the layer, **(2)** rotating it to the specific rotation pattern as designed, and **(3)** plugging the layer back to the original location, without changing the rest of the system.

While the RubikONNs architecture enables zero-overhead MTL on DONNs systems, the main challenge of enabling MTL-RubikONNs is to develop domain-specific training algorithms that incorporate hardware design properties. We introduce two domain-specific MTL training algorithms, i.e., rotated aggregation algorithm (RotAgg) and rotated sequence Algorithm (RotSeq) for RubikONNs. The details of the algorithm and their implementations are included in *Methods* and Algorithms 1 and 2. We provide detailed analysis for algorithmic space exploration in Table 1. We also provide comprehensive analytic evaluations to demonstrate the efficiency and capability of RubikONNs with other two state-of-the-art domain-specific training algorithms for four-task MTL image classification tasks in Table 2. As shown in Figure 1, we can see that there are various options of rotation patterns in RubikONNs. Therefore, we provide architectural design space exploration of RubikONNs to provide more insights to understand the characterizations of RubikONNs (Table 3). Finally, to understand the impacts of rotations for MTL, we provide full propagation from the laser source all the way to the detector in RubikONNs, which offers high-level explainability of MTL learning with RubikONNs (Figures 2 and 3).

Analysis of RubikONNs on MTL

We first evaluate RotSeq algorithm (Alg. 2) on MTL using four selected datasets. As discussed in *Methods*, the performance of RotSeq can vary with different gradient update orders (i.e., lines 4–10 in Alg. 2). Therefore, we evaluate RotSeq algorithm with four different permutations of the gradient update sequences, as shown in second column of Table 1. With such recurring permutation of training orders, each task can be trained at each position in the training order. First,

Table 1: Evaluations of prediction performance on four-task multi-task learning using datasets. MNIST(D1), FMNIST(D2), KMNIST(D3), and EMNIST(D4), optimized with the proposed RotSeq and RotAgg algorithms.

Algorithm	Permutation	MNIST(D1)	FMNIST(D2)	KMNIST(D3)	EMNIST(D4)
RotSeq (Alg. 2)	D1 → D2 → D3 → **D4**	0.9440	0.8300	0.8197	**0.8936**
	D4 → D1 → D2 → **D3**	0.9477	0.8332	**0.8409**	0.8671
	D3 → D4 → D1 → **D2**	0.9538	**0.8466**	0.7923	0.8748
	D2 → D3 → D4 → **D1**	**0.9573**	0.8275	0.8038	0.8768
RotAgg (Alg. 1)	n/a	0.9535	0.8469	0.8290	0.8891
Li et al. [3]	n/a	0.9550	0.8368	0.8237	0.8430
BaselineMTL	n/a	0.9282	0.8237	0.7571	0.8316

for each dataset, we can see that RotSeq offers a small accuracy boost for the given task/dataset, which is used as the last gradient update in one RotSeq training iteration. This is because RotSeq (Alg. 2) updates the parameters in a given sequence to all training tasks, where testing accuracy is basically obtained right after the gradient updates of the last task. Second, when a given dataset is used at the beginning of each training iteration (first task in the training sequence), the prediction performance of this task might slightly degrades. For example, MNIST accuracy collected using model trained with D1 → D2 → D3 → D4 sequence is 0.0037/0.0098/0.0133 lower than the other three permutations.

As for the comparison between RotAgg and RotSeq rotation training algorithms, the model

trained with the RotAgg algorithm shows overall better performance and robustness since the training is not impacted by the orders of gradient updates. Instead, RotAgg averages the gradients obtained independently for all tasks. The advantages of RotAgg can be summarized in two: **(1)** the training hyperparameter space is much limited than RotSeq since the gradient update order does not require exploration; **(2)** The algorithm is expected to perform more robustness than RotSeq, because RotSeq includes slight training bias w.r.t the gradient update order. While using these four datasets RotSeq performs similarly to RotAgg, this bias training characteristics can potentially require more training setup exploration. Thus, in the rest of the result section, we use RotAgg as default algorithm for RubikONN architecture exploration analysis.

To fully demonstrate the effectiveness of the proposed approaches, we first compare the prediction performance with two existing approaches. First of all, a straightforward method to enable MTL on a fixed single-task DONNs architecture is to simply train DONNs while merging the four datasets as one. Thus, we implement a straightforward baseline algorithm by extending the approach in Ref.[2], where the training dataset consists of fully shuffled training samples from all four datasets, namely *BaselineMTL*. The evaluation result of this baseline algorithm is shown in the last row of Table 1. Next, we compare our approaches to a specific MTL DONNs architecture. Specifically, Ref.[3] presents a DONNs architecture that utilizes transfer learning concepts from conventional neural networks, which included shared diffractive layers (shared weights) and independent diffractive layers at the output stage for each task. To make a fair comparison, we extend that architecture to deal with four tasks and set three layers for the shared diffractive layers and two layers for the independent-trained diffractive layers in each channel for four tasks. The

system has the same system size as our RubikONNs.

Table 1 shows that by utilizing the physical rotation under our training algorithms, RubikONNs offer better or comparable prediction accuracy for all datasets than Ref.[3] and *BaselineMTL*. We can see that with RotAgg and RotSeq, RubikONNs perform significantly better than both baseline approaches. For example, RubikONNs with RotAgg algorithm offers about 2.5% accuracy increases for MNIST, 2.3% increases for FMNIST, 5.8% increases for EMNIST and 7.2% for KMNIST, compared to *BaselineMTL*[2]; compared to Ref.[3], RotAgg offers 4.6% accuracy increases on EMNIST, and performs similarly for other three tasks. This demonstrates that by utilizing the physical rotations into DONNs architecture, RubikONNs offers clear prediction improvements over other approaches, while system cost, energy consumption, and complexity into the comparisons are not yet included in comparisons.

To evaluate the efficiency of the models regarding the system cost, complexity, and energy efficiency, we introduce a accuracy-cost evaluation metric, where hardware cost is the sum of diffractive layer cost and detector cost, and it can be approximated by only the detector cost as the diffractive layer cost is significantly cheaper than the detectors. In Table 2, single-task cost metric is set as the baseline (unit 1), and the improvement of the architectures is calculated using Equation 1. Note that in Table 2, the baseline results are collected using the single-task implementation with five layers and 200×200 system size, and our results are generated using the RotAgg algorithm. We can see that our approach offers more than $4.0\times$ and $2.0\times$ hardware cost efficiency improvements compared to Ref.[2] and Ref.[3], respectively. Regarding energy efficiency, we evaluate the power

Table 2: Evaluations of hardware efficiency on multi-task learning compared with [2] and [3], using datasets MNIST(D1), FMNIST(D2), KMNIST(D3), and EMNIST(D4).

	Lin et al.[2]				*Li et al.*[3]				**This work**			
	D1	D2	D3	D4	D1	D2	D3	D4	D1	D2	D3	D4
Layer Cost	5	5	5	5	11				5			
Detector Cost*	10	10	10	10	20				10			
Acc.(%)	96.6	86.7	86.8	91.2	95.5	83.7	82.4	84.3	95.4	84.7	82.9	88.9
Cost Efficiency	1	1	1	1	2.0×	2.1×	1.9×	1.9×	**4.0**×	**4.1**×	**4.1**×	**4.1**×
Power (μW/fps/task)	4.67×10^{-7}				8.86×10^{-7}				$\mathbf{1.7}\times10^{-7}$			

consumption per task. Our approach demonstrates 2.7× and 5.3× energy efficiency improvements compared to Ref.[2] and Ref.[3], respectively.

$$\begin{aligned} \text{Acc-Efficiency Metric} &= \frac{\text{Acc}_{\texttt{MTL}}}{\text{Acc}_{\texttt{baseline}}} \cdot \frac{\text{Cost}_{\texttt{MTL}}}{\text{Cost}_{\texttt{baseline}}}; \\ \text{Cost} &= \text{\#. Detectors or } \mu W\text{/fps/task} \end{aligned} \tag{1}$$

Design Space of RubikONNs Architecture

With the RubikONNs architecture and training algorithms, the rotation architecture can be designed in many different variants. Specifically, the rotation angles of rotatable layers, and selected diffractive layers to be rotated, which are independent to all other system and algorithm specifications. For example, instead of rotating the layers clockwise 90°, the layers can also be rotated 180°

and 270° (-90°). Similarly, the architecture can also be designed by selecting layers other than the 4^{th} and 5^{th} layers to be rotated. Thus, we provide comprehensive analysis of other variants of the architecture by evaluating different rotation angles and various rotation layer selections. To limit the rotation architecture exploration space on the algorithm side for RubikONN architecture exploration, we only use RotAgg algorithm as shown in Table 3.

(a) Analysis of rotated layer selections – Let the number of tasks be 4 and each rotation layer can only be rotated clock-wise 90°. The total number of layer selections is $\mathbf{C}_5^2$=10. According to studies of conventional neural networks, the layers close to the inputs are usually very important for feature extractions, while the layers close to outputs are crucial for generating the final prediction class. Thus, we evaluate three combinations, including (1) the last two layers, (2) the first two layers, and (3) the first and the last layers. The results are shown in Table 3. We can see that the models trained with RotAgg algorithm perform almost the same, regardless of which layers are selected as rotation layers.

(b) Analysis of different rotation angles – Since each diffractive layer can rotate close-wise 90°, 180°, and 270° (-90°), the rotation angle can be independent from layer to layer, e.g., rotating 4^{th} layer 90° and rotating 5^{th} layer by 180°. To evaluate the impacts of rotation angles, we fix the selections of rotation layers, i.e., 4^{th} and 5^{th} layers. Table 3 shows the accuracy of four datasets in the model trained with RotAgg when different rotation angles are applied to the last two layers. Specifically, we evaluate two different rotation angle settings: (1) same rotation angles for both layers; or (2) different rotation angles for the two layers. For example, (90°, 180°) means that if

Table 3: Explorations with various rotation angles (clockwise) with the 4^{th} and 5^{th} layers as rotation layers, and exploration of rotated layers with $90°$, $90°$ angle.

Rotation Angle	**RotAgg** (Alg. 1) w rot lyers = 4^{th},5^{th}			
	MNIST	FMNIST	KMNIST	EMNIST
$90°$, $90°$	0.9535	0.8469	0.8290	0.8891
$180°$, $180°$	0.9537	0.8565	0.8300	0.8865
$270°$, $270°$	0.9553	0.8414	0.8300	0.8844
$90°$, $-90°$	0.9531	0.8491	0.8277	0.8875
$90°$, $180°$	0.9555	0.8376	0.8322	0.8885

Rotated Layers	**RotAgg** (Alg. 1) w rotation angle $90°$, $90°$			
	MNIST	FMNIST	KMNIST	EMNIST
4^{th}, 5^{th}	0.9550	0.8469	0.8290	0.8891
1^{st}, 2^{nd}	0.9529	0.8503	0.8296	0.8871
1^{st}, 5^{th}	0.9544	0.8466	0.8249	0.8919

the 4^{th} layer is designed to be rotated for a given task, it rotates $90°$; and 5^{th} layer rotates $180°$. In general, with different rotation angles, RubikONNs shows little fluctuation in terms of accuracy.

In summary, Table 3 results suggest that the prediction performance is not restricted to specific rotation angles or rotation layers, which offers possibility to encode more forward functions, and it is the key to enable larger number of tasks for MTL.

RubikONNs MTL Explainability

To understand the impacts of rotations for MTL, we provide the full propagation of RubikONNs between the source, layers and detectors. Specifically, we measure the intensity of the light in the RubikONNs at inference phase as shown in Figures 2 and 3. The visualizations of the forward propagation shown in Figures 2 and 3 are organized by applying the same input image from one dataset using all rotation patterns, following the designed rotation patterns shown in Figure 1b. It is known that the main idea of DNNs is that layers close to the input focus on extracting features, and layers close to the output focus on finalizing the predictions using the extracted features. The intuition of RubikONNs architecture is relatively the same, and has been demonstrated based on the propagation measurements. For example, in Figure 2, the input image is from MNIST dataset, where four complete propagation measurements are included w.r.t the rotation patterns for task MNIST, FMNIST, KMNIST, and EMNIST, respectively. We can see that the outputs of the first three layers are identical for all four cases, since the first three layers are not the rotation layers. The differences of forward propagation are observed starting from the 4^{th} layer, which is rotated clockwise $90°$ for MNIST and KMNIST tasks, and remains un-rotated for FMNIST and EMNIST

tasks. Similarly, since the 5^{th} layer is also designed to be rotated as well, the outputs collected by the detectors clearly show four different intensity distributions. This confirms that RubikONNs successfully encodes four different forward functions, which are properly optimized for four tasks using our training algorithms.

Methods

Evaluation Setups. The model explored in our evaluation is designed with five diffractive layers as it is shown in Figure 1(a). The system size is set to be 200×200, i.e., the size of the diffractive layers. The input image whose original size is 28×28 is enlarged to 200×200 that can be encoded with a coherent THz source, with the wavelength = 0.75 mm. The physical distance between layers, first layer to source, and final layer to detector, are set to be 3 cm. The architecture is designed with the rotation patterns shown in Figure 1(b). The detectors collect the intensity of the ten pre-defined regions with each size of 20×20 (Figure 1(a)), where the sums of intensity of these ten regions are equivalent to a 1×10 vector in `float32` type. The final prediction results are then generated by selecting the corresponding class with the max energy sum of the detector region.

Training Parameters and Datasets. To evaluate the RubikONNs architecture and RotAgg and RotSeq training algorithms, we select four public image classification datasets, including (1) *MNIST-10*[22] (MNIST), (2) *Fashion-MNIST*[23] (FMNIST) , (3) *Kuzushiji-MNIST*[24] (KMNIST), and (4) *Extension-MNIST-Letters*[25] (EMNIST), an extension of MNIST to handwritten letters. Specifically, for EMNIST, we customize the dataset to have the first ten classes (i.e., A-J) to match

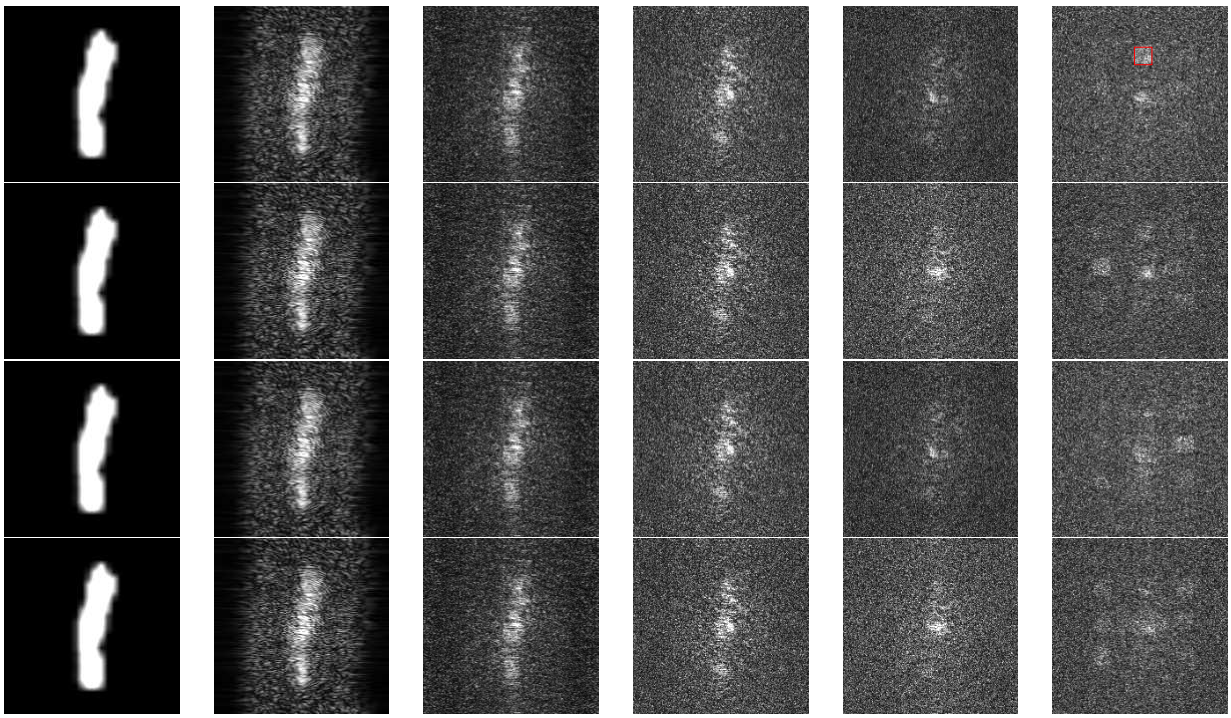

Figure 2: Visualization of light propagation measurements at inference phase with MNIST image "1" as input, using all four rotations at 4^{th} layer and 5^th layer of RubikONNs. At 4^{th} layer, for the first and third figure, the rotation angel is $0°$, for the second and forth, the rotation angel is $90°$. The 4^{th} propagation image will the same for the first and third and the same for the second and forth. Since the input example is MNIST example '1' and the model for MNIST is trained with the 4^{th} layer as $0°$, the propagation image at the 4^{th} layer will show higher contrast when the model is trained with the same rotation pattern at 4^{th} layer (the first and third). The pattern shown in the last image shows the power of the last layer in classification.

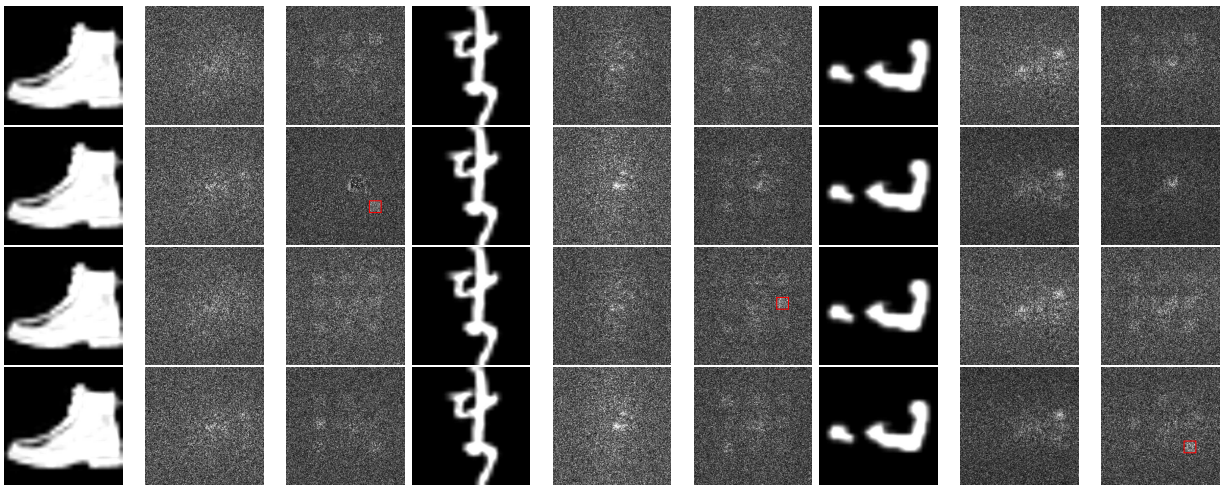

Figure 3: Visualization of light propagation measurements (input, and diffraction pattern of 4^{th} and 5^{th} layers) at inference phase with FMNIST, KMNIST, and EMNIST images, using all four rotation configurations of RubikONNs.

the DONN physical system, with 48000 training examples and 8000 testing examples. The total number of training iterations of all experiments is set to 30, where the aggregated gradients (Alg. 1 RotAgg) or sequential gradient updates (Alg. 2 RotSeq) of each iteration are generated with 5 epoch. The learning rate in the training process is set to be 0.01 for all experiments cross all four datasets using `Adam`[26] as the optimizer. The implementations are constructed using PyTorch v1.5.1[27]. All experiments are conducted on a Nvidia 2080 Ti GPU.

Algorithm 1: Rotated Aggregation Training (RotAgg) for RubikONNs. The RotAgg algorithm shown in Alg. 1 aims to update the parameters of RubikONNs by averagely aggregating gradients generated from all tasks, while the gradient of each task is computed by including the rotations in every training iteration. Therefore, the training iterations are fully aware of physical rotations of the RubikONNs architecture. Specifically, RotAgg algorithm initializes one model for aggregation,

Algorithm 1: Rotated Aggregation Algorithm (**RotAgg**).

Result: $\boldsymbol{W} = \{\boldsymbol{W}_C^{1,2,3}, \boldsymbol{W}_R^4, \boldsymbol{W}_R^5\}$ for the rotation model

1 **Initialization**: Weights $\boldsymbol{W_0}^0 = \{\boldsymbol{W_{C_0}}^{1,2,3}, \boldsymbol{W_{R_0}}^4, \boldsymbol{W_{R_0}}^5\}$ for the model ▷ Weights initialization

2 **while** $i \leq$ *training iterations* **do**

3 $\quad \boldsymbol{W_{1,2,3,4}}^i \leftarrow \boldsymbol{W_0}^i$;

4 $\quad \boldsymbol{W_1}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4 \boldsymbol{W}_{R_i}^5\}$; $\boldsymbol{W_1}^{i+1} \xleftarrow{D1} \boldsymbol{W_1}^i - \eta \nabla \boldsymbol{W_1}^i$; ▷ training w.r.t task 1 (D1) w/o rotation.

5 $\quad \boldsymbol{W_2}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate}(\boldsymbol{W}_{R_i}^5)\}$; $\boldsymbol{W_2}^{i+1} \xleftarrow{D2} \boldsymbol{W_2}^i - \eta \nabla \boldsymbol{W_2}^i$; ▷ task 2 (D2) update w 5^{th} layer rotated $90°$

6 $\quad \boldsymbol{W_3}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \texttt{rotate}(\boldsymbol{W}_{R_i}^5)\}$; $\boldsymbol{W_3}^{i+1} \xleftarrow{D3} \boldsymbol{W_3}^i - \eta \nabla \boldsymbol{W_3}^i$; ▷ task 3 (D3) update w 4,5^{th} layer rotated $90°$.

7 $\quad \boldsymbol{W_4}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$; $\boldsymbol{W_4}^{i+1} \xleftarrow{D4} \boldsymbol{W_4}^i - \eta \nabla \boldsymbol{W_4}^i$; ▷ task 4 (D4) update w 4^{th} layer rotated $90°$.

8 $\quad \boldsymbol{W_2}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^5)\}$;
$\quad\quad \boldsymbol{W_3}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^4), \texttt{rotate-back}(\boldsymbol{W}_{R_i}^5)\}$; $\boldsymbol{W_4}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$; ▷ reversely rotating virtual models for aggregation.

9 $\quad \boldsymbol{W_0}^{i+1} \leftarrow (\boldsymbol{W_1^{i+1}} + \boldsymbol{W_2^{i+1}} + \boldsymbol{W_3^{i+1}} + \boldsymbol{W_4^{i+1}})/4$;

10 $\quad i \leftarrow i + 1$

11 **end**

and four virtual models to store temporary updates w.r.t to specific rotation patterns and dataset (line 1). We illustrate RotAgg using the same rotation designs shown in Figure 1. In this case, the first three layers are common layers that are not rotated during training and inference, namely as $W_C^{1,2,3}$, and the rotations layers are denoted as W_R^4 and W_R^5. In every training iteration, RotAgg first updates the parameters in the four virtual models, $W_{1,2,3,4}^i$, where the four models are trained separately w.r.t the designed rotation patterns and the corresponding dataset (lines 4 - 7). Note that at each iteration, the virtual model is re-initialized with the initial weight parameters or parameters optimized in the previous iteration (line 3). For example, the first update is performed for task 1 w.r.t dataset D1 (line 4), where the model is rotated based on W_1^i with the rotation pattern of $[0°, 0°]$ as shown in Figure 1. The second update is then performed w.r.t to task 2 dataset D2,

where the virtual model W_2^i is initialized by rotating parameters in rotation layers (L4 and L5) with the rotation pattern $[0°, 90°]$ (line 5). Similarly, the virtual models for task 3 (D3) and task 4 (D4) are performed. Before the final weight aggregation, the four virtual models are reverse-rotated back to the initial position (line 8). Finally, RotAgg averagely aggregates the weights from all four virtual models (line 9), and return W_0^{i+1} for next iteration or as final model.

Algorithm 2: Sequential Rotation Training Algorithm (**RotSeq**).

Result: $\boldsymbol{W} = \{\boldsymbol{W}_C^{1,2,3}, \boldsymbol{W}_R^4, \boldsymbol{W}_R^5\}$ for the rotation model

1 **initialization**: Weights $\boldsymbol{W}^0$ for the model;

2 **while** *i* $\leq$ *training iterations* **do**

3 $\quad \boldsymbol{W}^i = \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \boldsymbol{W}_{R_i}^5\}$;

4 $\quad \boldsymbol{W}^i \xleftarrow{D1} \boldsymbol{W}^i - \eta \nabla \boldsymbol{W}^i$;

5 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate}(\boldsymbol{W}_{R_i}^5)\}$; ▷ re-training w.r.t task 2 (D2) w 5^{th} layer rotated $90°$.

6 $\quad \boldsymbol{W}^i \xleftarrow{D2} \boldsymbol{W}^i - \eta \nabla \boldsymbol{W}^i$;

7 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$; ▷ re-training w.r.t task 3 (D3) w 4^{th} & 5^{th} layers rotated $90°$.

8 $\quad \boldsymbol{W}^i \xleftarrow{D3} \boldsymbol{W}^i - \eta \nabla \boldsymbol{W}^i$;

9 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^5)\}$; ▷ re-training w.r.t task 4 (D4) w 4^{th} layer rotated $90°$.

10 $\quad \boldsymbol{W}^i \xleftarrow{D4} \boldsymbol{W}^i - \eta \nabla \boldsymbol{W}^i$;

11 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$; ▷ rotate back the original pattern for task 1 (D1)

12 $\quad \boldsymbol{W}^{i+1} \leftarrow \boldsymbol{W}^i$;

13 $\quad i \leftarrow i + 1$

14 **end**

Algorithm 2: Sequential Rotation Training (RotSeq) for RubikONNs. The second training algorithm RotSeq shown in Alg. 2 aims to update the parameters of RubikONNs by sequentially updating the model w.r.t a given sequence of task orders in order to incorporate the physical rotations in the training process. Here, we illustrate Alg. 2 using a specific order of updates, i.e., D1→D2→D3→D4. In the illustration example, for the first task, the model is updated w.r.t dataset

D1 without rotating the rotation layers (line 4). Unlike the RotAgg algorithm, the model is directly updated to W^i after the training of the first task. Next, the weights are rotated with the rotation pattern $[0°, 90°]$, i.e., rotating $W^5_{R_i}$ clockwise $90°$ before the gradient update process for task 2 (line 5). Note that the model rotated before training for task 2 has already been updated w.r.t D1. Similarly, the model is trained in the same sequential updating fashion according to the rotation patterns designed for task 3 (line 7) and task 4 (line 9). Therefore, in addition to other training parameters, RotSeq could also be impacted by the inner loop update orders. In *Results*, a comprehensive analysis of the update orders is provided.

Code and Data Availability

Dataset: Datasets used in this work, including select four public image classification datasets (1) *MNIST-10*[22] (MNIST), (2) *Fashion-MNIST*[23] (FMNIST) , (3) *Kuzushiji-MNIST*[24] (KMNIST), and (4) *Extension-MNIST-Letters*[25] (EMNIST), are all public datasets that are described in Methods.

Code: Please find all code in the supplementary file **RubikONN-SciRep.zip** uploaded with the submission.

- `train_single_task.py`: used for producing results in Tables 1 and 2 for Ref.2
- `train_4_single_model.py`: used for producing results as **BaselinMTL** row in Table 2.
- `train_split_4ctest.py`: used for producing results in Ref.3 in Tables 1 and 2.

- `train_4_avg.py`: This is the implementation of the proposed algorithm RotAgg (Algorithm 1).

- `train_4_seq.py`: This is the implementation of the proposed algorithm RotSeq (Algorithm 2).

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *nature* **521**, 436–444 (2015).

2. Lin, X. *et al.* All-optical machine learning using diffractive deep neural networks. *Science* **361**, 1004–1008 (2018).

3. Li, Y., Chen, R., Rodriguez, B. S., Gao, W. & Yu, C. Multi-task learning in diffractive deep neural networks via hardware-software co-design. *Scientific Reports* 1–9 (2021).

4. Jordan, M. I. & Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science* **349**, 255–260 (2015).

5. Jouppi, N. P. *et al.* In-datacenter Performance Analysis of a Tensor Processing Unit. *Int'l Symp. on Computer Architecture (ISCA)* 1–12 (2017).

6. Abadi, M. *et al.* Tensorflow: A system for large-scale Machine Learning. *Symp. on Operating System Design and Implementation (OSDI)* 265–283 (2016).

7. Gu, J. *et al.* Towards area-efficient optical neural networks: an fft-based architecture. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 476–481 (IEEE, 2020).

8. Ying, Z. *et al.* Electronic-photonic arithmetic logic unit for high-speed computing. *Nature communications* **11**, 1–9 (2020).

9. Shen, Y. *et al.* Deep learning with coherent nanophotonic circuits. *Nature Photonics* **11**, 441 (2017).

10. Mengu, D., Rivenson, Y. & Ozcan, A. Scale-, shift-and rotation-invariant diffractive optical networks. *arXiv preprint arXiv:2010.12747* (2020).

11. Feldmann, J., Youngblood, N., Wright, C. D., Bhaskaran, H. & Pernice, W. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature* **569**, 208–214 (2019).

12. Tait, A. N. *et al.* Neuromorphic photonic networks using silicon photonic weight banks. *Scientific reports* **7**, 1–10 (2017).

13. Rahman, M. S. S., Li, J., Mengu, D., Rivenson, Y. & Ozcan, A. Ensemble learning of diffractive optical networks. *arXiv preprint arXiv:2009.06869* (2020).

14. Li, Y. & Yu, C. Late breaking results: Physical adversarial attacks of diffractive deep neural networks. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 1374–1375 (IEEE, 2021).

15. Gao, W., Yu, C. & Chen, R. Artificial intelligence accelerators based on graphene optoelectronic devices. *Advanced Photonics Research* **2**, 2100048 (2021).

16. Tang, Y. *et al.* Device-system co-design of photonic neuromorphic processor using reinforcement learning. *arXiv preprint arXiv:2203.06061* (2022).

17. Wang, T. *et al.* An optical neural network using less than 1 photon per multiplication. *Nature Communications* **13**, 1–8 (2022).

18. Zhou, T. *et al.* Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit. *Nature Photonics* **15**, 367–373 (2021).

19. Caruana, R. Multitask learning. *Machine learning* **28**, 41–75 (1997).

20. Weiss, K., Khoshgoftaar, T. M. & Wang, D. A survey of transfer learning. *Journal of Big data* **3**, 1–40 (2016).

21. Hinton, G., Vinyals, O., Dean, J. *et al.* Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* **2** (2015).

22. LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

23. Xiao, H., Rasul, K. & Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

24. Clanuwat, T. *et al.* Deep learning for classical japanese literature (2018). `cs.CV/1812.01718`.

25. Cohen, G., Afshar, S., Tapson, J. & Van Schaik, A. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 2921–2926 (IEEE, 2017).

26. Kingma, D. P. & Ba, J. Adam: A method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

27. Paszke, A. *et al.* PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NIPS)* 8024–8035 (2019).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SI.pdf