Rubik’s Optical Neural Networks: Multi-task Learning with Physics-aware Rotation Architecture

Yingjie Li, Weilu Gao, Cunxi Yu
University of Utah
{yingjie.li, weilu.gao, cunxi.yu}@utah.edu

Abstract

Recently, there are increasing efforts on advancing optical neural networks (ONNs), which bring significant advantages for machine learning (ML) in terms of power efficiency, parallelism, and computational speed. With the considerable benefits in computation speed and energy efficiency, there are significant interests in leveraging ONNs into medical sensing, security screening, drug detection, and autonomous driving. However, due to the challenge of implementing reconfigurability, deploying multi-task learning (MTL) algorithms on ONNs requires re-building and duplicating the physical diffractive systems, which significantly degrades the energy and cost efficiency in practical application scenarios. This work presents a novel ONNs architecture, namely, RubikONNs, which 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. To optimize MTL performance on RubikONNs, two domain-specific physics-aware training algorithms RotAgg and RotSeq are proposed. Our experimental results demonstrate more than 4× improvements in energy and cost efficiency with marginal accuracy degradation compared to the state-of-the-art approaches.

1 Introduction

Recently, use of Deep Neural Networks (DNNs) shows significant advantages in many applications, including large-scale computer vision, natural language processing, and data mining tasks. 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 [Jouppi et al., 2017; Yin et al., 2022; Seshadri et al., 2022; Yin et al., 2023]. More importantly, it is identified that training large DNN models produces significant carbon dioxide, e.g., recent studies estimated 626,000 pounds of planet-warming carbon dioxide, equal to the lifetime emissions of five cars, produced in training Transformer network [Strubell et al., 2019]. As models grow bigger, their demand for computing increases, as well as the carbon footprint produced by those computations. To address these challenges and make the computation more eco-friendly, there has been a significant trend in building novel high-performance DNNs platforms, especially the increasing efforts on implementing novel DNNs in optical domain, i.e., optical neural network (ONNs) that mimic conventional feed-forward neural network functions using light propagation [Lin et al., 2018; Gu et al., 2020; Ying et al., 2020; Shen et al., 2017; Chen et al., 2022a; Chen et al., 2022b; Li et al., 2022; Li and Yu, 2021; Duan et al., 2023]. Unlike directly accelerating conventional DNNs, algorithms for training and deploying ONNs need to be customized in order to precisely represent the whole physics properties of light propagation. Specifically, the equivalent numerical representations of inputs, intermediate results, and propagation functions in optical domain are complex values and complex-valued functions. Additionally, due to the limitations from nature physics, implementing reconfigurability and deploying multi-task learning (MTL) algorithms on many ONNs systems requires re-building and duplicating the physical hardware systems, which significantly degrades the energy and cost efficiency in practical application scenarios.

This work proposes a novel architecture 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. With the realization of MTL in optical systems, the computational carbon footprint can be significantly reduced while maintaining the system performance. The paper is organized as follows: in Section 2, we introduce Diffractive Deep Neural Networks (D2NN) and its physical implementations; in Section 3, we first formulate the forward functions in D2NN systems for MTL. Furthermore, to optimize the MTL performance of RubikONNs, we propose two novel domain-specific physics-aware training algorithms, RotAgg and RotSeq; in Section 4, we demonstrate four-task MTL on RubikONNs with implementation cost and energy efficiencies improved more than 4×. Finally, a comprehensive RubikONNs design space exploration analysis and explainability are provided to offer concrete design methodologies for practical uses.
2 Background

Diffractive Deep Neural Networks (D2NN). Recently, there are increasing efforts on optical neural networks and optical computing based DNNs hardware, which bring significant advantages for machine learning systems in terms of their power efficiency, parallelism, and computational speed, demonstrated at various optical computing systems by [Mengu et al., 2020; Lin et al., 2018; Feldmann et al., 2019; Shen et al., 2017; Tait et al., 2017; Li et al., 2021; Gu et al., 2022; Gao et al., 2021; Tang et al., 2023; Lou and et al., 2023]. Among them, free-space diffractive deep neural networks (D2NNs), which is based on the light diffraction and phase modulation of the light signal provided by diffractive layers (L1-L5 in Figure 1), featuring millions of neurons in each layer interconnected with neurons in neighboring layers. This ultrahigh density and parallelism make this system possess fast and high throughput computing capability. Additionally, the D2NN system is implemented with passive optical devices, where the devices function without additional maintaining power required, thus significantly reducing the consumption power for solving deep learning tasks with orders of magnitude energy efficiency advantages over low-power digital devices ([Lin et al., 2018; Mengu et al., 2023; Li et al., 2021; Chen et al., 2022a; Li et al., 2022]). More importantly, [Lin et al., 2018; Li et al., 2021; Chen et al., 2022a; Mengu et al., 2020; Li et al., 2022] demonstrated that diffractive propagation controlled by phase modulation are differentiable, which means that such parameters can be optimized with conventional backpropagation algorithms using conventional automatic differentiation (autograd) engine implemented in modern compilers such as PyTorch and Tensorflow.

In conventional DNNs, forward propagation is computed by generating the feature representation with floating-point weights associated with each neural layer. While in D2NNs, such floating-point weights are encoded in the phase modulation of each neuron in diffractive phase masks, which is acquired by and multiplied onto the light wavefunction as it propagates through the neuron. Similar to conventional DNNs, the final output class is predicted based on generating labels according to a given one-hot representation, e.g., the max energy reading over the output signals of the last layer observed by detectors. Specific examples of the system at training and inference can be found in the next section.

Once the training of a D2NN system is completed on the digital computation platform, the trained D2NN is deployed on the optical platform with non-configurable fabricated phase masks such as 3D printed phase masks, as diffractive layers for all-optical inference. Thus, D2NNs lack reconfigurability for the weight parameters, which will bring significant energy and system cost overhead in practical application scenarios, especially for MTL.

3 Approach

To overcome the aforementioned limitations of existing D2NN systems, we propose a novel neural architecture, namely RubikONNs, that utilizes the physical rotation properties of existing D2NN systems to realize MTL with few overheads, in which case, a single-task D2NN system can be used to encode multiple feed-forward functions by rotating its underlying structure just like rotating a Rubik’s Cube.

3.1 Forward Function for a Single-task D2NN

D2NN system is designed with three major components (Figure 1): (1) laser source encoding the input images, (2) diffrac-
tive layers encoding trainable phase modulation, and (3) detectors capturing the output of the forward propagation. Specifically, the input image is first encoded with the laser source. The information-encoded light signal is diffracted in the free space between diffractive layers and modulated via phase modulation at each layer. Finally, the diffraction pattern after light propagation w.r.t light intensity distribution will be captured at the detector plane for predictions.

From the beginning of the system, the input information (e.g., an image) is encoded on the coherent light signal from the laser source, its wavefunction can be expressed as \( f^0(x_0, y_0) \). The wavefunction after light diffraction from the input plane to the first diffractive layer over diffraction distance \( z \) can be seen as the summation of the outputs at the input plane, i.e.,

\[
f^1(x, y) = \int \int f^0(x_0, y_0) h(x - x_0, y - y_0, z) dx_0 dy_0
\]  (1)

where \((x, y)\) is the coordinate on the receiver plane, i.e., the first diffractive layer, \( h \) is the impulse response function of free space. Here we use Fresnel approximation, thus the impulse response function \( h \) is

\[
h(x, y, z) = \exp \left( \frac{ikz}{i\lambda z} \right) \exp \left( \frac{ik}{2z} \left(x^2 + y^2\right) \right)
\]  (2)

where \( i = \sqrt{-1} \), \( \lambda \) is the wavelength of the laser source, \( k = 2\pi/\lambda \) is free-space wavenumber.

Equation 1 can be calculated with spectral algorithm, where we employ Fast Fourier Transform (FFT) for fast and differentiable computation, i.e.,

\[
U^1(\alpha, \beta) = U^0(\alpha, \beta) H(\alpha, \beta, z)
\]  (3)

where \( U \) and \( H \) are the Fourier transformation of \( f \) and \( h \) respectively.

After light diffraction, the wavefunction resulting in Equation 3 \( U^1(\alpha, \beta) \) is first transformed to time domain with inverse FFT (iFFT). Then the phase modulation \( W(x, y) \) provided by the diffractive layer is applied to the light wavefunction in time domain by matrix multiplication, i.e.,

\[
f^2(x, y) = \text{iFFT}(U^1(\alpha, \beta)) \times W_1(x, y)
\]  (4)

where \( W_1(x, y) \) is the phase modulation in the first diffractive layer, \( f^2(x, y) \) is then the input light wavefunction for the light diffraction between the first diffractive layer and the second diffractive layer.

We enclose one computation round of light diffraction and phase modulation at one diffractive layer as a computation module named DiffMod, i.e.,

\[
\text{DiffMod}(f(x, y), W) = L(f(x, y), z) \times W(x, y)
\]  (5)

where \( f(x, y) \) is the input wavefunction, \( W(x, y) \) is the phase modulation, \( L(f(x, y), z) \) is the wavefunction after light diffraction over a constant distance \( z \) in time domain, i.e., \( \text{iFFT}(U(\alpha, \beta)) \) in Equation 4.

As a result, in a multiple diffractive layer constructed D²NN system, the forward function can be computed iteratively for the stacked diffractive layers. For example, for the 5-layer system shown in Figure 1, the forward function can be expressed as,

\[
I(f^0(x, y), W) = \text{DiffMod}(\text{DiffMod}(\text{DiffMod}(\text{DiffMod}(
\text{DiffMod}(f^0(x, y), W_1(x, y)), W_2(x, y)), W_3(x, y)), W_4(x, y)), W_5(x, y))
\]  (6)

where \( f^0(x, y) \) is the input wavefunction to the system and \( W_{1-5} \) is phase modulation provided at each diffractive layer.

The final diffraction pattern w.r.t the light intensity \( I \) in Equation 6 is projected to the detector plane. We can design arbitrary detector patterns for classes in different tasks by setting the corresponding coordinates of the detector region at the full detector plane for each class by the user’s definition. For example, for MNIST datasets, the output plane is divided into ten detector regions to mimic the output of conventional neural networks for predicting ten classes. The final class will be produced by \( \text{argmax} \) function with the ten intensity sums of the ten detector regions as input. For example, in Figure 1, based on the label indices of the ten detector regions for image “2”, we can see that the 3rd region on the first row has the highest energy. Then, the predicted class is class “2”. Similarly, the predicted classes “1”, “8”, and “9” of other three datasets can be generated by applying \( \text{argmax} \) on the detector. With the one-hot represented ground truth class \( t \), the loss function \( L \) can be acquired with \( \text{MSELoss} \) as,

\[
L = \tau \text{ Softmax}(I) - t \parallel_2 \tag{7}
\]

Thus, the whole system is designed to be differentiable and compatible with conventional automatic differential engines.

### 3.2 RubikONNs Architecture for MTL

To deal with multiple tasks with minimum system overhead, an ideal system should be designed to encode different forward functions without changing the single-task system. 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 3D printing. However, as demonstrated by [Lin et al., 2018; Li et al., 2021], the layers are portable in D²NNs and they are in square shapes. This means that we can 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 will be different since the weights of the model are changed. In optical domain, this means that the 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 D²NN systems. 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.

To illustrate the rotation architecture RubikONNs, an example of encoding four tasks with the last two layers (L4, L5) as rotation layers is shown in Figure 2. The designed rotation type applied to these layers is the rotation angle (clockwise 90° in this example). To summarize the forward function of
RubikONN architecture, we introduce two Boolean variables to indicate the rotation patterns of L4 and L5 layers. When $s_0 = 1$, L4 will be rotated clockwise 90°, otherwise, L4 will remain unchanged; similarly, $s_3$ indicates the rotation pattern of L5. Thus, for the first task, $s_0 s_3 = 00$, both layers are unchanged; for the second task, $s_0 s_3 = 01$, L5 will be rotated clockwise 90°; for the third task, $s_0 s_3 = 11$, both layers will be rotated clockwise 90°; for the fourth task, $s_0 s_3 = 10$, only L4 will be rotated clockwise 90°. The forward function is expressed as follows:

$$I = \text{DiffMod}(I, 1 \leq i \leq 3, f^0, W_i), \text{DiffMod}(W_4, \text{Rot}(W_5)), s_0 s_3 = 00$$

where $s_0 s_3 = 00$, both layers are unchanged.

We take $s_0 s_3 = 01$ as an example for the mathematical analysis w.r.t the rotation in the system, where only the fifth layer is rotated for 90°. Thus, the forward function is

$$I_{01} = \text{DiffMod}((I, 1 \leq i \leq 3, f^0, W_i), \text{Rot}(W_5))$$

(9)

where DiffMod($f(x, y), W = L(f(x, y), z) \times W(x, y)$, $x \in [n, n], y \in [n, n]$ as shown in Equation 5, assuming $W_5$ is trained as $W_{n, n}$ and the diffraction result is $L_{n, n}$, i.e.,

$$W_{n, n} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{pmatrix} \quad L_{n, n} = \begin{pmatrix} l_{11} & l_{12} & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & l_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix}$$

Thus, the wavefunction after DiffMod($f, W$) without rotation, i.e., $I_{00}$, is calculated as

$$I_{00} = \begin{pmatrix} l_{11} + w_{11} & l_{12} + w_{12} & \cdots & l_{1n} + w_{1n} \\ l_{21} + w_{21} & l_{22} + w_{22} & \cdots & l_{2n} + w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} + w_{n1} & l_{n2} + w_{n2} & \cdots & l_{nn} + w_{nn} \end{pmatrix}$$

When the rotation pattern applied to $W_5$ is 90°, the corresponding Rot($W_5$) is

$$\text{Rot}(W_5) = \begin{pmatrix} w_{1n} & w_{2n} & \cdots & w_{nn} \\ w_{11} & w_{21} & \cdots & w_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{nn} \end{pmatrix}$$

While $L(f, z)$ remains the same, the corresponding rotated $I_{01}$ is thus altered to

$$I_{01} = \begin{pmatrix} l_{11} + w_{11} & l_{21} - w_{11} & \cdots & l_{1n} + w_{1n} \\ l_{12} + w_{22} & l_{22} + w_{22} & \cdots & l_{2n} + w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{1n} - w_{nn} & l_{2n} - w_{nn} & \cdots & l_{nn} + w_{nn} \end{pmatrix}$$

As a result, by rotating the weights matrix $W$ (phase modulation) to different angles, the information-carried light signal is modulated with different applied phase modulations w.r.t different datasets. The full propagation figures in Figure 4 show the light patterns when different rotation patterns are applied to the same input light signal.

Algorithm 1: Rotated Aggregation Algorithm (RotAgg) for training RubikONNs.

Result: $W = \{W^1_{8, 4}, W^2_{8, 4}, W^3_{8, 4}\}$ for the rotation model
1 Initialization: Weights $W_0 = \{W^{1, 2, 3}_{c, 3, W_{R_{1, 4}}, W_{R_{2, 4}}}, W^{1, 2, 3}_{R_{1, 4}}\}$ for the model
2 while $i \leq \text{training iterations}$ do
3 $W_{1, 2, 3, 4} \leftarrow W_0$;
4 $W_i \leftarrow \{W^{i, 1, 2, 3, 4}_{c, 3, W_{R_{1, 4}}, W_{R_{2, 4}}}, W^{i, 1, 2, 3, 4}_{R_{1, 4}}\}$;
5 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
6 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
7 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
8 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
9 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
10 $W_i = \{W^{i, 1, 2, 3, 4}_{c, 3, \text{rotate}(W_{R_{1, 4}})}, \text{rotate}(W_{R_{2, 4}})\}$;
11 end

Note that the proposed architecture can go beyond four tasks by adding more rotation patterns. For example, each layer has four rotation states, 0°, 90°, 180°, or 270°, such that the maximum number of different forward functions is 16 with two rotation layers; when add another rotation layer, i.e., three rotation layers with four rotation states in the system, the system can deal with up to 40 tasks. Discussions about choices of rotation layers and different rotation angles are included in Section 4.

While the RubikONNs architecture enables zero-overhead MTL on D2NN systems, the training algorithms that are aware of physical rotations and light diffraction do not exist. Specifically, the training algorithms should be able to learn structural weight parameters w.r.t specific rotation patterns and given datasets. Thus, we introduce two novel MTL algorithms for training RubikONNs, i.e., rotated aggregation algorithm (RotAgg) and rotated sequential algorithm (RotSeq).

3.3 Algorithm 1: Rotated Aggregation Training

The Rotated Aggregation Training (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. We illustrate RotAgg using the same rotation designs shown in Figure 2, where the first three layers are shared layers, named as $W^c_{8, 4}$, that will not be rotated during training and inference, and the rotations layers are denoted as $W^{R}_{1, 4}$ and $W^{R}_{2, 4}$. First, RotAgg algorithm initializes one model for aggregation, and four virtual models to store temporary updates w.r.t specific rotation patterns and dataset
In every training iteration, RotAgg will first update the parameters in the four virtual models, $W_{i}^{1,2,3,4}$, 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 will be re-initialized before any gradient update, 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 $W_{1}^{1}$ with the rotation pattern of $[0^\circ, 0^\circ]$ as shown in Figure 2. The second update will then be performed w.r.t task 2 dataset D2, where the virtual model $W_{2}^{1}$ will be initialized by rotating parameters in rotation layers (L4 and L5) with the rotation pattern $[0^\circ, 90^\circ]$ (line 5). Similarly, the virtual models for task 3 (D3) and task 4 (D4) will be performed. Before the final weight aggregation, the four virtual models will be reverse-rotated back to the initial position (line 8). Finally, RotAgg averages the weights from all four virtual models (line 9), and return $W_{0}^{i+1}$ for next iteration or as final model.

### 4.2 Algorithm 2: Rotated Sequential Training

The second training algorithm Rotated Sequential Training (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 will be updated w.r.t dataset D1 without rotating the rotation layers (line 4). Unlike the RotAgg algorithm, the model will be directly updated to $W_{1}^{1}$ after the training of the first task. Next, the weights will be rotated with the rotation pattern $[0^\circ, 90^\circ]$, i.e., rotating $W_{2}^{1}$ clockwise $90^\circ$ 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 task D1. Similarly, the model will be trained in the same sequential updating fashion according to the rotation patterns designed for task 3 (line 7) and task 4 (line 9). Note that the inner loop update order can be fixed for all iterations or can be dynamically changed through the training process. Therefore, in addition to other training parameters, RotSeq could also be impacted by the inner loop update orders. In Section 4, a comprehensive analysis of the update orders is provided.

### 4 Results

**System Setups.** The model explored in our experiments is designed with five diffractive layers as it is shown in Figure 1. The system size is set to be $200 \times 200$, i.e., the size of the diffractive layers and the total detector plane is $200 \times 200$. The input image whose original size is $28 \times 28$ will be enlarged to $200 \times 200$ and encoded on the laser light signal with the wavelength = 532 nm. The physical distances between layers, first layer to source, and final layer to detector, are set to be 30 cm. The architecture is designed with the rotation patterns shown in Figure 2. The detector collects the light intensity of the ten pre-defined regions for ten classes with each size of 20 $\times$ 20 (Figure 1), where the sums of intensity of these ten regions are equivalent to a $1 \times 10$ vector in float32 type. The final prediction results will be then generated using argmax.

### 4.1 Evaluations of RubikONNs with RotAgg and RotSeq Algorithms

**RotSeq and Training Permutations, and RotAgg.** We first evaluate RotAgg algorithm (Alg. 2) on MTL using the four selected datasets. As discussed in Section 3, 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, 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, for each dataset, we can see that RotSeq offers a small accuracy boost for the given task/dataset 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 dataset is trained at the beginning of each training iteration (first task
in the training sequence), the prediction performance of this task might slightly degrade. For example, MNIST accuracy collected using model trained with D1 → D2 → D3 → D4 sequence is 0.0006/0.0006/0.0025 lower than the other three permutations. However, the model trained with RotAgg algorithm shows overall better performance and robustness since the training is not impacted by 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 its performance is not influenced by the gradient update order; (2) The algorithm is expected to be more robust than RotSeq as RotSeq has slight training bias w.r.t the gradient update order.

The training loss curves for two algorithms are shown in Figure 3. Both algorithms can converge efficiently and produce similarly decent accuracy performance, while the training loss for Algorithm 2, which trains the rotation MTL system with sequential datasets, shows more fluctuations. Due to the bias training characteristics in RotSeq that can potentially result in requiring more training setup exploration, we use RotAgg as default algorithm for RubikONN architecture exploration analysis in the rest of the result section.

**Prediction Performance Comparisons.** 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 D²NNs architecture is to simply train a D²NNs while merging the four datasets as one. Thus, we implement a straightforward baseline algorithm by extending the approach proposed by [Lin et al., 2018], where the training dataset consists of fully shuffled training samples from all four datasets, namely BaselineMTL, and its depth is set to be five and system size is set to be 200 × 200, which is the same setup as our rotation system. 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 D²NNs architecture. Specifically, [Li et al., 2021] proposes a novel D²NN architecture that utilizes transfer learning concepts from conventional neural networks, which includes 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 diffractive layers in each channel for four tasks and the same system size (200 × 200) as our system.

As shown in Table 1, we can see that by utilizing the physical rotation properties with the proposed training algorithms, RubikONNs offers better prediction accuracy for all datasets. We can see that with RotAgg and RotSeq, RubikONNs performs significantly better than both baseline approaches. For example, with RotAgg algorithm, our approach offers about 2.5% accuracy increases for MNIST and FMNIST, 6.5% increases for EMNIST and 8.4% for KMNIST, compared to BaselineMTL ([Lin et al., 2018]); compared to [Li et al., 2021], RotAgg offers 3.5% accuracy increases on EMNIST,
and performs similarly for other three tasks. This demonstrates that by utilizing the physical rotations into D2NN architecture, RubikONNs offers clear prediction improvements over other approaches, while system cost, energy consumption, and complexity into the comparisons are not yet included in the comparisons.

**Accuracy-Efficiency Comparison.** To fully evaluate the efficiency of the models regarding the system cost, complexity, and energy efficiency, we introduce an accuracy-cost evaluation metric, where hardware cost is the sum of diffractive layer cost and detector cost. In Table 2, single-task cost metric is set as the baseline (unit 1), and the improvement of the architectures is calculated using Equation 10. Note that in Table 2, the baseline results are collected using single-task implementation with five layers and 200 × 200 system size, and our results are generated using RotAgg algorithm. We can see that our approach offers more than 4.0 × and 2.0 × hardware cost efficiency improvements compared to [Lin et al., 2018] and [Li et al., 2021], respectively. Regarding energy efficiency, we evaluate the power consumption per task. Our approach demonstrates 2.7 × and 5.3 × energy efficiency improvements compared to [Lin et al., 2018] and [Li et al., 2021], respectively. Note that the power consumption of DONNs is orders of magnitude more efficient than conventional digital platforms. Thus, we only compared to DONNs baselines in this work since the advantages of DONNs over conventional DNN hardware have been demonstrated.

\[
\text{Acc-Efficiency Metric} = \frac{\text{Acc}_{\text{MTL}}}{\text{Acc}_{\text{baseline}}} \cdot \frac{\text{Cost}_{\text{baseline}}}{\text{Cost}_{\text{MTL}}};
\]

\[
\text{Cost} = \# \text{ Detectors or } \mu\text{W/fps/task}
\]

(10)

### 4.2 Design Space of RubikONNs Architecture

With the proposed architecture and training algorithms, the rotation architecture can be designed in many different variants. Specifically, the rotation angles of the rotation layers, and the index of rotation 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 other layers other than the 4th and 5th layers to be rotated. Thus, we provide experimental analysis of other variants of the proposed architecture by evaluating different rotation angles and various rotation layer selections using RotAgg algorithm.

**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 4th layer 90° and rotating 5th layer by 180°. To evaluate the impacts of rotation angles, the experiments shown in Table 3 are conducted with fixed selection of rotation layers, i.e., 4th and 5th layers. Table 3 shows the accuracy of four datasets in the model trained with RotAgg when different rotation angles are applied to 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 the 4th layer is designed to be rotated for a given task, it rotates 90°; and 5th layer is rotated 180° if needed. In general, with different rotation angles, RubikONNs shows little fluctuation in terms of accuracy.

**Analysis of Rotated Layer Selections.** Let the number of tasks be 4 and each rotation layer can only rotate clock-wise 90°, the total number of layer selections is C2=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 4. We can see that (a) the models trained with RotAgg algorithm perform almost the same, regardless of which layers are selected as rotation layers; (b) including the last layer (5th) in the rotation layers performs slightly better on average.

In summary, Table 3 and Table 4 results suggest the follows: (1) The rotation layers are preferred to be selected close to the output. (2) The prediction performance is not restricted to specific rotation angles, which offers possibility to encode more forward functions, and it is the key to enable larger number of tasks for MTL.

### 4.3 RubikONNs MTL Explainability

To understand the impacts of rotations for MTL, we measure the internal propagation of RubikONNs between the source layers and detectors. Specifically, we measure the intensity of the light in the RubikONNs at inference phase, shown in Figures 4. The visualizations of the forward propagation shown in Figures 4 are organized by applying a same image from one dataset using all rotation patterns, following the designed rotation patterns shown in Figure 2. 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 4, 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 4th layer, which is rotated clockwise 90° for MNIST and KMNIST tasks, and remains un-rotated for FMNIST and EMNIST tasks. Similarly, since the 5th layer is also designed to be rotated as well, the outputs collected by the detectors clearly show four different intensity distributions. Additional outputs of 4th and 5th layers w.r.t other three datasets are shown in Figure 5, which further confirms that RubikONNs is able to successfully encode four differ-
Table 2: Evaluations of hardware efficiency on multi-task learning compared with [Lin et al., 2018] and [Li et al., 2021], using datasets. MNIST(D1), FMNIST(D2), KMNIST(D3), and EMNIST(D4).

Table 3: Explorations with various rotation angles (clockwise) with the 4th and 5th layers as rotation layers.

Table 4: Design space explorations with different selections of rotation layers with rotation angle 90°.

5 Conclusions

This work proposes a novel optical neural architecture RubikONNs architecture, which utilizes the physical properties of optical computing systems to encode multiple feed-forward functions by rotating the non-reconfigurable hardware system. To optimize the MTL performance of RubikONNs, two novel domain-specific physics-aware training algorithms RotAgg and RotSeq are proposed, such that RubikONNs offers 4× implementation cost and energy efficiencies improvements, with marginal accuracy degradation. Finally, a comprehensive RubikONNs design space exploration analysis and explainability are provided to offer concrete design methodologies for practical uses. The ONNs have the potential to handle more complex image tasks, including image classification tasks and graph tasks [Yan et al., 2022] with outstanding energy efficiency advantage over conventional NNs (e.g., CNNs), which can already be observed with simple datasets like MNIST (~3 orders compared to CPU/GPU in our setups).
References


[Yan et al., 2022] Tao Yan, Rui Yang, Ziyang Zheng, Xing Lin, Hongkai Xiong, and Qionghai Dai. All-optical graph

