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 (D^2NN) and its physical implementations; in Section 3, we first formulate the forward functions in D^2NN 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.

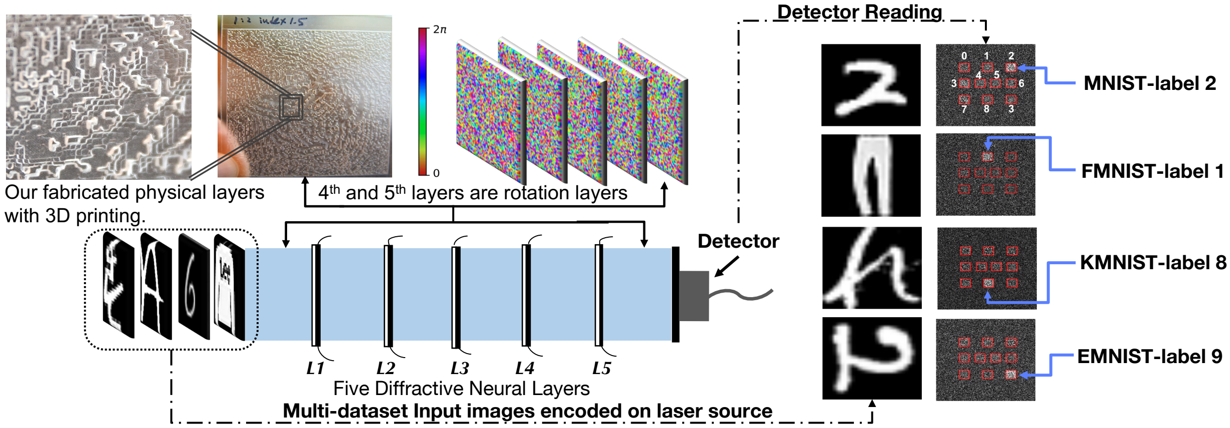


Figure 1: Overview of Rubik DONN system, consisting of (1) laser source that encodes the input images, (2) diffractive layers with trainable phase parameters (weights), which are the non-configurable passive optical devices fabricated with 3D printing and stacked together as DONN systems, and (3) detectors that reads the output. The four inference results are collected with specific rotations patterns shown in Figure 2.

2 Background

Diffractive Deep Neural Networks (D^2NN). 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* (D^2NNs) , 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 D^2NN 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 D^2NNs, 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.

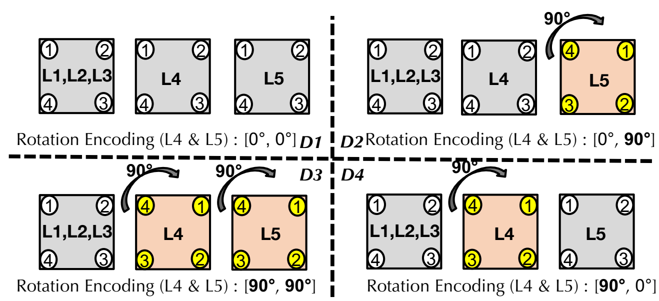


Figure 2: Example of a four-task RubikONNs rotation architecture.

Once the training of a D^2NN system is completed on the digital computation platform, the trained D^2NN 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, D^2NNs 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 D^2NN systems, we propose a novel neural architecture, namely **RubikONNs**, that utilizes the physical rotation properties of existing D^2NN systems to realize MTL with few overheads, in which case, a single-task D^2NN 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 D^2NN

D^2NN system is designed with three major components (Figure 1): (1) laser source encoding the input images, (2) diffractive 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.,

$$f^1(x,y) = \iint f^0(x_0,y_0)h(x-x_0,y-y_0,z)dx_0dy_0 \quad (1)$$

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

$$h(x,y,z) = \frac{\exp(ikz)}{i\lambda z}\exp\{\frac{ik}{2z}(x^2+y^2)\} \quad (2)$$

where $i = \sqrt{-1}$, λ 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.,

$$U^1(\alpha,\beta) = U^0(\alpha,\beta)H(\alpha,\beta,z) \quad (3)$$

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.,

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

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.,

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

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., iFFT$(U(\alpha, \beta))$ in Equation 4.

As a result, in a multiple diffractive layer constructed D^{2}NN 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,

$$\begin{aligned} I(f^0(x,y),W) = &\text{DiffMod}(\text{DiffMod}(\text{DiffMod}(\text{DiffMod} \\ &(\text{DiffMod}(f^0(x,y),W_1(x,y)),W_2(x,y)), \\ &W_3(x,y)),W_4(x,y)),W_5(x,y)) \end{aligned} \quad (6)$$

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 `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 3^{rd} 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 `argmax` on the detector. With the one-hot represented ground truth class t, the loss function L can be acquired with **MSELoss** as,

$$L = \| \text{Softmax}(I) - t \|_2 \quad (7)$$

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 D^{2}NNs 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 D^{2}NN 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_1 indicates the rotation pattern of L5. Thus, for the first task, $s_0s_1 = 00$, both layers are unchanged; for the second task, $s_0s_1 = 01$, L5 will be rotated clockwise 90°; for the third task, $s_0s_1 = 11$, both layers will be rotated clockwise 90°; for the fourth task, $s_0s_1 = 10$, only L4 will be rotated clockwise 90°. The forward function is expressed as follows:

$$I = \text{DiffMod}(\text{DiffMod}(\underset{1\le i\le 3}{I}(f^0, W_i), \begin{cases} W_4), W_5), & s_0s_1 = 00 \\ W_4), \text{Rot}(W_5)), & s_0s_1 = 01 \\ \text{Rot}(W_4)), \text{Rot}(W_5)), & s_0s_1 = 11 \\ \text{Rot}(W_4)), W_5), & s_0s_1 = 10 \end{cases} \tag{8}$$

We take $s_0s_1 = 01$ 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

$$I_{01} = \text{DiffMod}(((\underset{1\le i\le 4}{I}(f^0, W_i), \text{Rot}(W_5)) \tag{9}$$

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

$$W_{n,n} = \begin{pmatrix} w_1^1 & w_2^1 & \cdots & w_n^1 \\ w_1^2 & w_2^2 & \cdots & w_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_1^n & w_2^n & \cdots & w_n^n \end{pmatrix} L_{n,n} = \begin{pmatrix} l_1^1 & l_2^1 & \cdots & l_n^1 \\ l_1^2 & l_2^2 & \cdots & l_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ l_1^n & l_2^n & \cdots & l_n^n \end{pmatrix}$$

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

$$I_{00} = \begin{pmatrix} l_1^1 * w_1^1 & l_2^1 * w_2^1 & \cdots & l_n^1 * w_n^1 \\ l_1^2 * w_1^2 & l_2^2 * w_2^2 & \cdots & l_n^2 * w_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ l_1^n * w_1^n & l_2^n * w_2^n & \cdots & l_n^n * w_n^n \end{pmatrix}$$

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

$$\text{Rot}(W_5) = \begin{pmatrix} w_1^n & \cdots & w_1^2 & w_1^1 \\ w_2^n & \cdots & w_2^2 & w_2^1 \\ \vdots & \ddots & \vdots & \vdots \\ w_n^n & \cdots & w_n^2 & w_n^1 \end{pmatrix}$$

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

$$I_{01} = \begin{pmatrix} l_1^1 * w_1^n & \cdots & l_{n-1}^1 * w_1^2 & l_n^1 * w_1^1 \\ l_1^2 * w_2^n & \cdots & l_{n-1}^2 * w_2^2 & l_n^2 * w_2^1 \\ \vdots & \ddots & \vdots & \vdots \\ l_1^n * w_n^n & \cdots & l_{n-1}^n * w_n^2 & l_n^n * w_n^1 \end{pmatrix}$$

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

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

while $i \le$ *training iterations* **do**

- $\boldsymbol{W_{1,2,3,4}}^i \leftarrow \boldsymbol{W_0}^i$;
- $\boldsymbol{W_1}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4 \boldsymbol{W}_{R_i}^5\}$;
  $\boldsymbol{W_1}^{i+1} \xleftarrow{D1} \boldsymbol{W_1}^i - \eta\nabla\boldsymbol{W_1}^i$; ▷ training w.r.t task 1 (D1) w/o rotation.
- $\boldsymbol{W_2}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate}(\boldsymbol{W}_{R_i}^5)\}$;
  $\boldsymbol{W_2}^{i+1} \xleftarrow{D2} \boldsymbol{W_2}^i - \eta\nabla\boldsymbol{W_2}^i$; ▷ task 2 (D2) update w 5^{th} layer rotated 90°
- $\boldsymbol{W_3}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \texttt{rotate}(\boldsymbol{W}_{R_i}^5)\}$;
  $\boldsymbol{W_3}^{i+1} \xleftarrow{D3} \boldsymbol{W_3}^i - \eta\nabla\boldsymbol{W_3}^i$; ▷ task 3 (D3) update w $4{,}5^{th}$ layer rotated 90°.
- $\boldsymbol{W_4}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$;
  $\boldsymbol{W_4}^{i+1} \xleftarrow{D4} \boldsymbol{W_4}^i - \eta\nabla\boldsymbol{W_4}^i$; ▷ task 4 (D4) update w 4^{th} layer rotated 90°.
- $\boldsymbol{W_2}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \boldsymbol{W}_{R_i}^4, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^5)\}$; $\boldsymbol{W_3}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate-back}(\boldsymbol{W}_{R_i}^4), \texttt{rotate-back}(\boldsymbol{W}_{R_i}^5)\}$;
  $\boldsymbol{W_4}^i \leftarrow \{\boldsymbol{W}_{C_i}^{1,2,3}, \texttt{rotate}(\boldsymbol{W}_{R_i}^4), \boldsymbol{W}_{R_i}^5\}$; ▷ reversely rotating virtual models for aggregation.
- $\boldsymbol{W_0}^{i+1} \leftarrow (\boldsymbol{W_1^{i+1}} + \boldsymbol{W_2^{i+1}} + \boldsymbol{W_3^{i+1}} + \boldsymbol{W_4^{i+1}})/4$;
- $i \leftarrow i + 1$

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 D^{2}NN 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 $W_C^{1,2,3}$, that will not be rotated during training and inference, and the rotations layers are denoted as W_R^4 and W_R^5. First, RotAgg algorithm initializes one model for aggregation, and four virtual models to store temporary updates w.r.t specific rotation patterns and dataset (line 1). 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^i_1 with the rotation pattern of $[0°, 0°]$ as shown in Figure 2. The second update will then be performed w.r.t to task 2 dataset D2, where the virtual model W^i_2 will be 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) 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 averagely aggregates the weights from all four virtual models (line 9), and return W^{i+1}_0 for next iteration or as final model.

3.4 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^i after the training of the first task. Next, the weights will be 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 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.

Algorithm 2: Sequential Rotation Training Algorithm (**RotSeq**) for training RubikONNs.

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

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

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

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

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

5 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}^{1,2,3}_{C_i}, \boldsymbol{W}^4_{R_i}, \texttt{rotate}(\boldsymbol{W}^5_{R_i})\}$; ▷ 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 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}^{1,2,3}_{C_i}, \texttt{rotate}(\boldsymbol{W}^4_{R_i}), \boldsymbol{W}^5_{R_i}\}$; ▷ 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 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}^{1,2,3}_{C_i}, \boldsymbol{W}^4_{R_i}, \texttt{rotate-back}(\boldsymbol{W}^5_{R_i})\}$; ▷ 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 $\quad \boldsymbol{W}^i \leftarrow \{\boldsymbol{W}^{1,2,3}_{C_i}, \texttt{rotate-back}(\boldsymbol{W}^4_{R_i}), \boldsymbol{W}^5_{R_i}\}$; ▷ 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**

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 200×200, i.e., the size of the diffractive layers and the total detector plane is 200×200. The input image whose original size is 28×28 will be enlarged to 200×200 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 20×20 (Figure 1), where the sums of intensity of these ten regions are equivalent to a 1×10 vector in `float32` type. The final prediction results will be then generated using `argmax`.

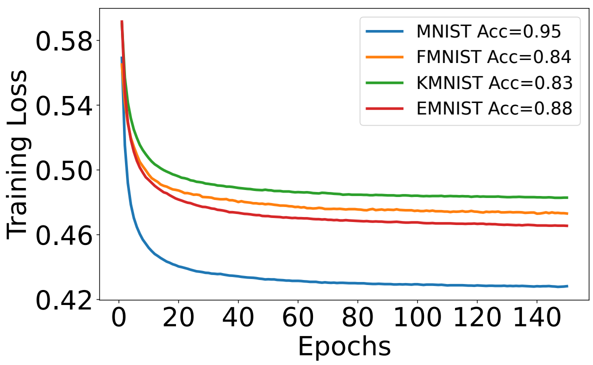
Training Parameters and Datasets. To evaluate the proposed RubikONNs architecture and RotAgg and RotSeq training algorithms, we select four public image classification datasets, including 1) *MNIST-10* (MNIST) ([LeCun, 1998],) 2) *Fashion-MNIST* (FMNIST) ([Xiao *et al.*, 2017]), 3) *Kuzushiji-MNIST* (KMNIST) ([Clanuwat *et al.*, 2018]), and 4) *Extension-MNIST-Letters* (EMNIST) ([Cohen *et al.*, 2017]), 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 the D^2NN physical system, with 48000 training examples and 8000 testing examples. In the rest of this section, the total number of training iterations of all experiments is set to 150. Within each training iteration, each dataset is trained with 1 training epoch. The learning rate in the training process is set to be 0.01 for all experiments cross all four datasets using `Adam`. The implementations are constructed using `PyTorch v1.5.1`. All experiments are conducted on a Nvidia 2080 Ti GPU.

4.1 Evaluations of RubikONNs with RotAgg and RotSeq Algorithms

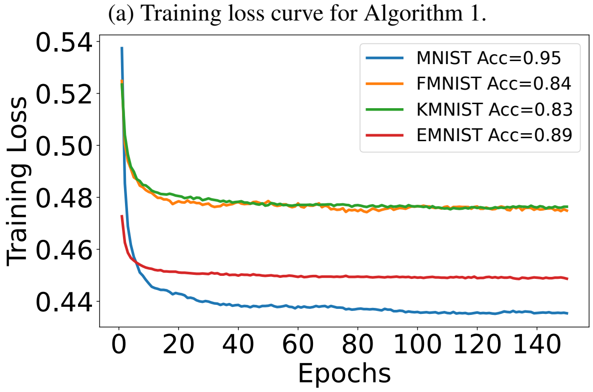
RotSeq and Training Permutations, and RotAgg. We first evaluate RotSeq algorithm (Alg. 2) on MTL using the four selected datasets. As discussed in Section 3, the performance of RotSeq can vary with different gradient update orders (i.e., lines 4–10 in Alg. 2). Therefore, we evaluate RotSeq algorithm 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

Algorithm	Permutation	MNIST$_{D1}$	FMNIST$_{D2}$	KMNIST$_{D3}$	EMNIST$_{D4}$
RotSeq (Alg. 2)	D1 → D2 → D3 → **D4**	0.9533	0.8384	0.8275	**0.8885**
	D4 → D1 → D2 → **D3**	0.9539	0.8394	**0.8353**	0.8811
	D3 → D4 → D1 → **D2**	0.9539	**0.8430**	0.8270	0.8824
	D2 → D3 → D4 → **D1**	**0.9558**	0.8386	0.8272	0.8834
RotAgg (Alg. 1)	n/a	0.9524	0.8412	0.8272	0.8819
[Li *et al.*, 2021]	n/a	0.9532	0.8370	0.8277	0.8464
BaselineMTL	n/a	0.9262	0.8176	0.7430	0.8174

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.



(a) Training loss curve for Algorithm 1.



(b) Training loss curve for Algorithm 2 with the permutation D1→D2→D3→D4.

Figure 3: Training loss curves with 150 epochs for two algorithms.

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 D^{2}NNs architecture is to simply train a D^{2}NNs 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 D^{2}NNs architecture. Specifically, [Li *et al.*, 2021] proposes a novel D^{2}NN 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 D^2NN 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[1]. 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.

$$\text{Acc-Efficiency Metric} = \frac{Acc_{\texttt{MTL}}}{Acc_{\texttt{baseline}}} \cdot \frac{Cost_{\texttt{MTL}}}{Cost_{\texttt{baseline}}};\quad Cost = \text{\#. Detectors or } \mu W\text{/fps/task} \tag{10}$$

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 4^{th} and 5^{th} 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 4^{th} layer $90°$ and rotating 5^{th} 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., 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 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 the 4^{th} layer is designed to be rotated for a given task, it rotates $90°$; and 5^{th} 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 $\mathbf{C}_5^2$=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 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 (5^{th}) 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 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. Additional outputs of 4^{th} and 5^{th} layers w.r.t other three datasets are shown in Figure 5, which further confirms that RubikONNs is able to successfully encode four differ-

[1]The layer fabrication cost is ∼\$100 and detector cost is \$1,500 – \$10,000. We formulate the cost of \$100 as unit 1, thus, the layer cost for a 5-layer ONN is 5 and one detector cost is 10 for the cost efficiency estimation.

	[Lin *et al.*, 2018]				[Li *et al.*, 2021]				**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.(%)	96.4	86.5	86.1	90.9	95.6	83.1	81.4	84.3	95.2	84.1	82.7	88.2
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}				$\mathbf{1.7}\times10^{-7}$			

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).

Rotation Angle	**RotAgg** (Alg. 1) w rot layers = 4^{th}, 5^{th}			
	MNIST	FMNIST	KMNIST	EMNIST
$90°$, $90°$	0.9524	0.8412	0.8272	0.8819
$180°$, $180°$	0.9532	0.8514	0.8313	0.8809
$270°$, $270°$	0.9527	0.8353	0.8240	0.8845
$90°$, $-90°$	0.9518	0.8427	0.8272	0.8879
$90°$, $180°$	0.9514	0.8413	0.8227	0.8851

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

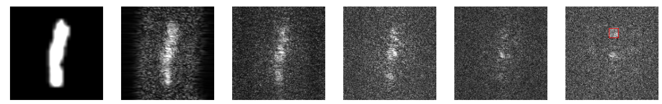
Rotated Layers	**RotAgg** (Alg. 1)			
	MNIST	FMNIST	KMNIST	EMNIST
4^{th}, 5^{th}	0.9524	0.8412	0.8272	0.8819
1^{st}, 2^{nd}	0.9536	0.8490	0.8199	0.8813
1^{st}, 5^{th}	0.9510	0.8473	0.8210	0.8865

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

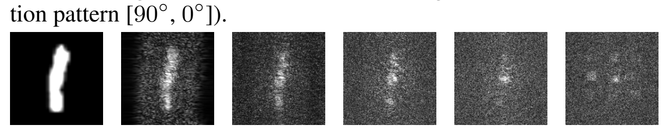
ent forward functions, which are properly optimized for four tasks using the proposed training algorithms.

5 Conclusions

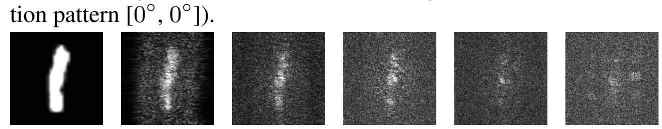
This work proposes a novel optical neural architecture **RubikONNs** architecture, which utilizes the physical properties of optical computing systems to encode multiple feed-forward functions by rotating the non-reconfigurable hardware system. To optimize the MTL performance of RubikONNs, two novel 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).



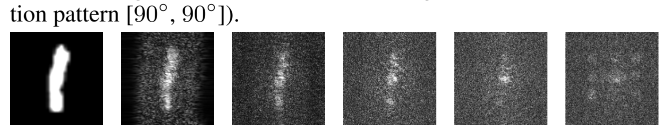
(a) Evaluating MNIST ”1” with model designed for MNIST (rotation pattern [$90°$, $0°$]).



(b) Evaluating MNIST ”1” with model designed for FMNIST (rotation pattern [$0°$, $0°$]).

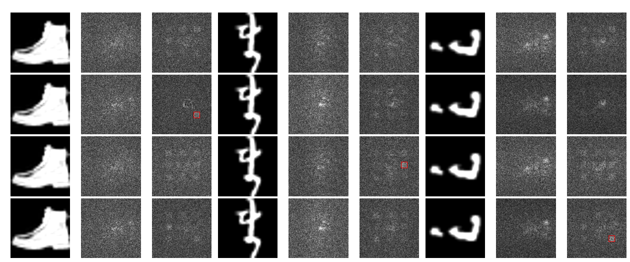


(c) Evaluating MNIST ”1” with model designed for KMNIST (rotation pattern [$90°$, $90°$]).



(d) Evaluating MNIST ”1” with model designed for EMNIST (rotation pattern [$0°$, $90°$]).

Figure 4: Visualization of light propagation patterns at inference with MNIST image ”1” as input, using all four rotations of RubikONNs.



(a) FMNIST label ”ankle boot”. (b) KMNIST label ”き”. (c) EMNIST with label ”j”.

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

Acknowledgements

This work is funded by National Science Foundation (NSF) under NSF-2008144, NSF-2019306, NSF-2019336, and NSF-2047176.

References

[Chen *et al.*, 2022a] Ruiyang Chen, Yingjie Li, Minhan Lou, Jichao Fan, Yingheng Tang, Berardi Sensale-Rodriguez, Cunxi Yu, and Weilu Gao. Physics-aware machine learning and adversarial attack in complex-valued reconfigurable diffractive all-optical neural network. *Laser & Photonics Reviews*, page 2200348, 2022.

[Chen *et al.*, 2022b] Ruiyang Chen, Yingjie Li, Minhan Lou, Cunxi Yu, and Weilu Gao. Complex-valued reconfigurable diffractive optical neural networks using cost-effective spatial light modulators. In *CLEO: Applications and Technology*, pages JTh3B–56. Optica Publishing Group, 2022.

[Clanuwat *et al.*, 2018] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. 2018.

[Cohen *et al.*, 2017] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[Duan *et al.*, 2023] Zhengyang Duan, Hang Chen, and Xing Lin. Optical multi-task learning using multi-wavelength diffractive deep neural networks. *Nanophotonics*, 2023.

[Feldmann *et al.*, 2019] J Feldmann, N Youngblood, C David Wright, H Bhaskaran, and WHP Pernice. All-optical spiking neurosynaptic networks with self-learning capabilities. *Nature*, 569(7755):208–214, 2019.

[Gao *et al.*, 2021] Weilu Gao, Cunxi Yu, and Ruiyang Chen. Artificial intelligence accelerators based on graphene optoelectronic devices. *Advanced Photonics Research*, 2(6):2100048, 2021.

[Gu *et al.*, 2020] Jiaqi Gu, Zheng Zhao, Chenghao Feng, Mingjie Liu, Ray T Chen, and David Z Pan. Towards area-efficient optical neural networks: an fft-based architecture. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 476–481. IEEE, 2020.

[Gu *et al.*, 2022] Jiaqi Gu, Hanqing Zhu, Chenghao Feng, Zixuan Jiang, Mingjie Liu, Shuhan Zhang, Ray T Chen, and David Z Pan. Adept: Automatic differentiable design of photonic tensor cores. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 937–942, 2022.

[Jouppi *et al.*, 2017] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter Performance Analysis of a Tensor Processing Unit. *Int'l Symp. on Computer Architecture (ISCA)*, pages 1–12, 2017.

[LeCun, 1998] Yann LeCun. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. Accessed: 2022-05-21.

[Li and Yu, 2021] Yingjie Li and Cunxi Yu. Late breaking results: physical adversarial attacks of diffractive deep neural networks. In *DAC*, 2021.

[Li *et al.*, 2021] Yingjie Li, Ruiyang Chen, Berardi Sensale Rodriguez, Weilu Gao, and Cunxi Yu. Multi-task learning in diffractive deep neural networks via hardware-software co-design. *Scientific Reports*, pages 1–9, 2021.

[Li *et al.*, 2022] Yingjie Li, Ruiyang Chen, Weilu Gao, and Cunxi Yu. Physics-aware differentiable discrete codesign for diffractive optical neural networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[Lin *et al.*, 2018] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[Lou and *et al.*, 2023] Minhan Lou and *et al.* Effects of interlayer reflection and interpixel interaction in diffractive optical neural networks. *Optics Letters*, 2023.

[Mengu *et al.*, 2020] Deniz Mengu, Yair Rivenson, and Aydogan Ozcan. Scale-, shift-, and rotation-invariant diffractive optical networks. *ACS photonics*, 8(1):324–334, 2020.

[Mengu *et al.*, 2023] Deniz Mengu, Anika Tabassum, Mona Jarrahi, and Aydogan Ozcan. Snapshot multispectral imaging using a diffractive optical network. *Light: Science & Applications*, 12(1):86, 2023.

[Seshadri *et al.*, 2022] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. An evaluation of edge tpu accelerators for convolutional neural networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pages 79–91. IEEE, 2022.

[Shen *et al.*, 2017] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 11(7):441, 2017.

[Strubell *et al.*, 2019] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[Tait *et al.*, 2017] Alexander N Tait, Thomas Ferreira De Lima, Ellen Zhou, Allie X Wu, Mitchell A Nahmias, Bhavin J Shastri, and Paul R Prucnal. Neuromorphic photonic networks using silicon photonic weight banks. *Scientific reports*, 7(1):1–10, 2017.

[Tang *et al.*, 2023] Yingheng Tang, Princess Tara Zamani, Ruiyang Chen, Jianzhu Ma, Minghao Qi, Cunxi Yu, and Weilu Gao. Device-system end-to-end design of photonic neuromorphic processor using reinforcement learning. *Laser & Photonics Reviews*, 17(2):2200381, 2023.

[Xiao *et al.*, 2017] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[Yan *et al.*, 2022] Tao Yan, Rui Yang, Ziyang Zheng, Xing Lin, Hongkai Xiong, and Qionghai Dai. All-optical graph representation learning using integrated diffractive photonic computing units. *Science Advances*, 8(24):eabn7630, 2022.

[Yin *et al.*, 2022] Jiaqi Yin, Zhiru Zhang, and Cunxi Yu. Exact memory-and communication-aware scheduling of dnns on pipelined edge tpus. In *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, pages 203–215. IEEE, 2022.

[Yin *et al.*, 2023] Jiaqi Yin, Yingjie Li, Daniel Robinson, and Cunxi Yu. Respect: Reinforcement learning based edge scheduling on pipelined coral edge tpus. *Design Automation Conference (DAC'23)*, 2023.

[Ying *et al.*, 2020] Zhoufeng Ying, Chenghao Feng, Zheng Zhao, Shounak Dhar, Hamed Dalir, Jiaqi Gu, Yue Cheng, Richard Soref, David Z Pan, and Ray T Chen. Electronic-photonic arithmetic logic unit for high-speed computing. *Nature communications*, 2020.