Opto-Electronic Advances

Supplementary information

2021, Vol. 4, No. 11

DOI: 10.29026/oea.2021.200060

All-optical computing based on convolutional neural networks

Kun Liao[1], Ye Chen[1], Zhongcheng Yu[1], Xiaoyong Hu[1,2]*, Xingyuan Wang[3]*, Cuicui Lu[4], Hongtao Lin[5]*, Qingyang Du[6], Juejun Hu[6] and Qihuang Gong[1,2]

[1]State Key Laboratory for Mesoscopic Physics & Department of Physics, Collaborative Innovation Center of Quantum Matter, Beijing Academy of Quantum Information Sciences, Nano-optoelectronics Frontier Center of Ministry of Education, Peking University, Beijing 100871, China; [2]Collaborative Innovation Center of Extreme Optics, Shanxi University, Taiyuan 030006, China; [3]College of Mathematics and Physics, Beijing University of Chemical Technology, Beijing 100029, China; [4]Beijing Key Laboratory of Nanophotonics and Ultrafine Optoelectronic Systems, School of Physics, Beijing Institute of Technology, Beijing 100081, China; [5]College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou 310027, China; [6]Department of Materials Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

*Correspondence: XY Hu, E-mail: xiaoyonghu@pku.edu.cn; XY Wang, E-mail: wang_xingyuan@mail.buct.edu.cn;
HT Lin, E-mail: hometown@zju.edu.cn

This file includes:

Supplementary information for this paper is available at https://doi.org/10.29026/oea.2021.200060

Section 1: Convolutional neural networks

Convolutional Neural Network (CNN) is one of the representative algorithms of deep learning, which is a type of Feed-forward Neural Networks including convolution computation and usually has a depth structure. They are widely applied in the fields of natural language processing and image recognition[1]. CNNs usually process data in the form of multiple arrays: one dimensional arrays for signals and sentences, two dimensional arrays for images and audio spectrums and three dimensional arrays for videos and volumetric images[2]. Generally, CNNs contain convolutional layers, pooling layers and fully-connected layers. In the convolutional layers there are convolutional kernels, padding and active functions, which are all parameters of the convolutional layers. The unique advantages of high precision in image processing benefit from the properties of local perception and parameter sharing of CNNs, maintaining high accuracy under translation, rotation and scaling of samples.

Given that in our networks we need to input a single light or a sequence of light signals into the network, CNNs naturally meet our demands because of their ability to protect signals from distortion. Moreover, comparing to fully-connected layers, convolutional layers require only local connections and thus avoid large numbers of waveguide crossings. The difference between a 1D convolutional layer and a fully-connected layer is showed in Fig. S1. A fully-connected layer uses different weights in every connection, while in a convolutional layer a convolutional kernel, which has shared weights in it, is used for addressing all signals in the current layer.

For the simplest 1D convolutional layer with stride = 1 (the step size of each movement is 1 in signal processing) and only one input channel, the relationship between the input and the output can be described as:

$$y_i = f(bias + \sum_{j=0}^{k-1} weight_j \times x_{i+j}) \ , \tag{S1}$$

where k is the size of the kernel, x is the input sequence, y is the output sequence, the bias is a constant for y_i, and $f(x)$ is called the activation function which is usually nonlinear. However, for a fully-connected layer with the same structure, the relationship between x and y changes into:

$$y_i = f\left(bias + \sum_{j=0}^{l-1} weight_{ij} \times x_j\right) \ , \tag{S2}$$

where l is the length of the input sequence, which is irrelevant in a convolutional layer. We use 1D convolutional networks to solve transcendental equations since shared weights in each layer are enough to control the network. We also disable bias and $f(x)$ considering that our devices are linear.

For 2D convolutional layers, the basic operation principle is similar to that of 1D convolutional layer and can be described with matrices. For instance, if the input $\boldsymbol{C}_{\rm in}$ is a matrix with a size of 3×3 and the kernel $\boldsymbol{K}$ is a matrix with a size of 2×2, then the output $\boldsymbol{C}_{\rm out}$ can be calculated as (stride = 1):

$$\boldsymbol{C}_{\rm in} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \boldsymbol{K} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \ , \tag{S3}$$

$$\text{we define: } \boldsymbol{A}_1 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \boldsymbol{A}_2 = \begin{bmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{bmatrix}, \boldsymbol{A}_3 = \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}, \boldsymbol{A}_4 = \begin{bmatrix} a_{22} & a_{23} \\ a_{31} & a_{33} \end{bmatrix} \ ,$$

$$\boldsymbol{C}_{\rm out} = \begin{bmatrix} \boldsymbol{A}_1 : \boldsymbol{K} & \boldsymbol{A}_2 : \boldsymbol{K} \\ \boldsymbol{A}_3 : \boldsymbol{K} & \boldsymbol{A}_4 : \boldsymbol{K} \end{bmatrix} \ , \tag{S4}$$

where $\boldsymbol{A}_n : \boldsymbol{K} = \sum_i \sum_j \boldsymbol{A}_{nij} \boldsymbol{K}_{ij}$. We use 2D convolutional layers to train our CNNs for the multifarious logic gate operators and half-adder, because shared weights cannot meet our demands in these two scenarios. Thus, we transform the input sequence from $\begin{bmatrix} r_0 & r_1 & \cdots \end{bmatrix}$ into $\begin{bmatrix} r_0 & r_1 & 0 & 0 & \cdots \\ 0 & 0 & r_1 & r_2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$ and input it into CNNs. After the training process, we only extract the weights corresponding to the non-zero positions. In this way, we can both avoid the problem that waveguides cannot cross and maximize the ability to control neural networks. We apply stochastic gradient descent (SGD) in our optimizer as it exhibits good performances in the CNN learning process.

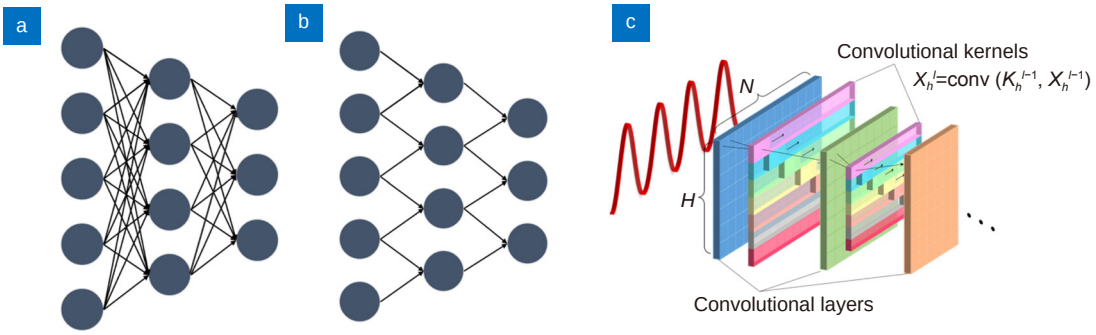

Fig. S1 | Structure comparison of CNN with fully-connected network neural network. (**a**) The structure of a fully-connected network having three fully-connected layers. (**b**) The structure of a 1D convolutional network having three 1D convolutional layers. (**c**) Schematic diagram of convolution neural network consisting of convolutional layers and convolutional kernels. *H* and *N* in the diagram represent the height and width of the processed signal, respectively. The signal between layers is transmitted through a convolutional operation $X_h^l = \mathrm{conv}(K_h^{l-1}, X_h^{l-1})$, Where *h* represents the sequence number of the signal to be processed, X_h^l and X_h^{l-1} represent the l^{th} and $(l-1)^{th}$ layer's convolutional signal states respectively, and K_h^{l-1} represents the convolutional kernel.

Section 2: Method of adjusting weight ω

We use a weight modulator (WM) positioned next to the transmission waveguide (TW) to adjust weight. WM is a small segment of silicon waveguide which has the same width as that of TW (to ensure efficient coupling). Through controlling the length (a) of WM and the gap width (b) between the WM and the waveguide, we can adjust magnitude and phase of the weight.

As stated in optical waveguide theory, when light waves propagate in two coupled waveguides, power transfer between the two waveguides periodically. When the two waveguides are close enough and the modes propagating through the two waveguides are similar, the two waveguides can strongly couple to each other. If the two waveguides are identical and each only supports a single mode, as in the case of our WM and TW, complete power transfer occurs between the two coupled waveguides.

Here we use L, a, and b to denote beat length in the coupled waveguide system, coupling length of two waveguides, and gap width between the two waveguides, respectively. The amount of light power remaining in TW is dictated by the ratio a/L, while a is readily adjusted by changing b. As Fig. S2 shows, with increasing b, L increases monotonically. We can therefore adjust the magnitude of weight ω simply via choosing suitable a and b values (Fig. 2(b)).

The beating length changes with b, and the phase accumulation scales with beating length. So, through controlling a and b, we can adjust phase of weight ω. As Fig. 2(c) shows, by choosing suitable a and b, phase of weight ω can change from 0 to 2π. In this work, we choose to adjust the magnitude of weight ω, while keeping phase in an almost constant range. The magnitude of ω can be tuned from 0.3 to 0.95 when the variation range of the phase ω is ±0.09 rad. Another alternative method is to vary the width of the WM waveguide, while keeping at equaling to the beat length to realize the

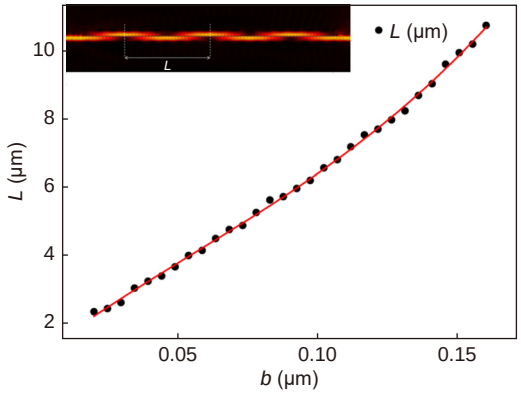

Fig. S2 | Regulation mechanism of proposed weight modulator. The relationship between L and b, where the black points is the simulation result, and the red line is the fit result. The illustration is the simulation result of the electric field distribution.

effective phase modulation.

A more efficient way to achieve a photonic computer is to maintain the magnitude of weight ω close to 1 (which minimizes insertion loss of the entire photonic circuit), while adjusting the phase in the optical neural network. This condition can be accomplished by choosing a as an integer multiple of beat length L while varying b to obtain the target phase of ω. As Fig. S3 shows, using this approach, we can keep the magnitude of ω close to 1 and changing the phase of ω from 0 to 2π.

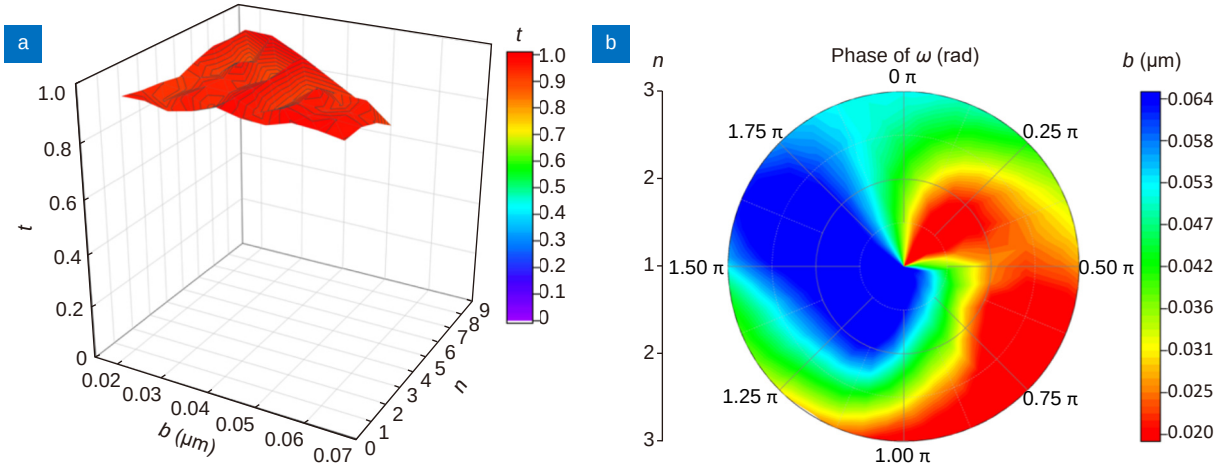

Fig. S3 | Results of weight modulation in specific designed case. By regulating the gap width b between two waveguides and integer multiple of 2L (n=2 mL, m is an integer). (**a**) The magnitude of weight ω keeps near 1. (**b**) The phase of weight ω can be continuously adjusted from 0 to 2π.

Section 3: All-optical transcendental equation solvers

Training details

PyTorch, a custom package in Python was used to construct the theoretical modeling of our optical neural networks. The calculation was based on 1D CNN used for the equation solver. We apply stochastic gradient descent (SGD) in the learning process to compute the parameters and minimize the loss function related to the model's performance. By changing the parameter k of the equation with a step size of 0.1, 12 training samples were obtained, which were used to train the weight ω. 2,000 iterations of the algorithm were performed to optimize the network weights that can implement the predictive solution function. After that, parameter k is chosen to increase 0.17 per step (to be different from the training samples) to execute the task of the equation solver. Two arms of the "Y" structure waveguide correspond to two kinds of weights, and the bifurcated waveguide is used as an element structure to extend the equal longitudinal weight and variable lateral weight, thus two kinds of weights for each layer are obtained. The three-layer network has a total of six kinds of weights, and one kind of weight in the third layer gives a negative value, indicating that the phase difference π is introduced.

Network performances with k =1.84 and 2.35

In order to show the practical performance of the equation solver, we experimentally demonstrated three of the predicted transcendental equations with different parameters k as 1.67, 1.84 and 2.35 respectively. Three structures similar to the one shown in Fig. 1(c) were fabricated, each containing a different set of waveform discretization layers to generate light intensity distributions in the feeding waveguides into the 3-layer optical CNN corresponding to the three k values. We make the maximum values of output intensity corresponding to the solutions of the equation. To illustrate the performance of the equation solver, we compare the experimental results with theoretical predictions. Here we demonstrate the rest of two results where k =1.84 and 2.35 (Fig. S4). As we can see from the figure, the result shows that our transcendental equation solvers have high accuracy. The maximum deviation between the actual solutions and the target solutions is no more than 5%, and most of them are no more than 3%.

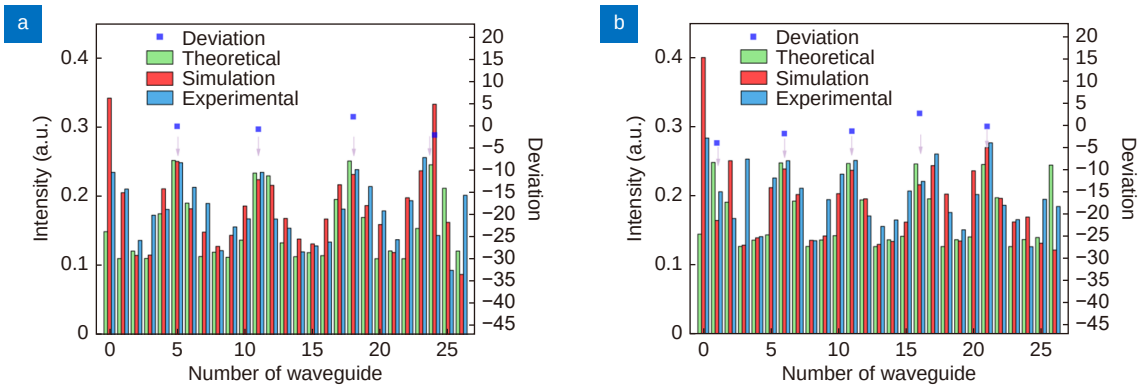

Fig. S4 | All-optical transcendental equation solver with different parameters *k*. (**a**) Distribution of output light intensity in the discretized waveguide interval (k = 1.84). (**b**) Distribution of output light intensity in the discretized waveguide interval (k = 2.35). The arrows in the figure refer to the location of the solutions. The horizontal axis is the number of discrete waveguides, the vertical axis on the left represents the output signal intensity, and the vertical axis on the right represents the deviation between the experimental output signal and the theoretical output signal.

Section 4: Multifarious logic gate operators

Sixteen logic functions (representing exhaustive combinations of output results corresponding to all four possible input signals 11, 10, 01, and 00 is 2^4= 16) have been realized in seven structures by optimizing every different weight of the network and the combination between weights and control bits. Each structure can execute 3 or 4 logic functions with appropriate control bit inputs. Here we present the results obtained on six of the seven structures (except the one discussed in the main text). As we can see from Fig. S5(a, c, e, g, i, k), when the weights and control bits of the network are varied, multifarious logic gate operators are realized. The experimental results show that the high contracts are maintained in each structure capable of performing multiple logic functions. Distinguishability of 0 and 1 indicates that multifunctional logic can be implemented on one structure (Fig. S5(b, d, f, h, j, l)).

Section 5: Fault tolerance of our networks

Optical neural network (ONN) offers excellent fault tolerance which is essential to scalable and defect-tolerant manufacturing of the all-optical chip. Taking the transcendental equation solver as an example, we mainly studied impact of two types of imperfections, weight deviation and waveguide fracture on solution deviation. Mean absolute error (MAE) is utilized to quantify the solution deviation:

$$MAE = \frac{\sum |R_i - R_{i0}|}{N \times M}, \tag{S5}$$

M is the number of output waveguides; N is the number of predicted samples (predicted samples are defined as transcendental equations of identical general form with different parameters); R_{i0} is the sequence number of the output waveguide that is supposed to have max intensity for a predicted sample; R_i is the sequence number of the output waveguide that has max intensity when given an input of the predicted sample. The summation covers all predicted samples, divided by N and M to get the MAE.

In traditional computer neural networks, the transmission of information between "neurons" depends on the weight of their connections with each other. Similarly, by adjusting the amplitude and phase of the light field in the transmission process, ONNs can realize the information transmission between layers. Therefore, tolerance of ONN to weight variations is crucial, which directly reflects the stability of ONNs. In our ONNs, since we only concern about the relative distribution of light intensity among output waveguides, only the phase difference and the amplitude ratio of two weights in each neuron layers can have an impact on the accuracy of the results. As shown in Fig. S6, by changing the amplitude ratio and phase difference of the weights in three layers separately, simulated variations in the MAE of the outputs are presented.

Another possible cause of undesired deviation is propagation loss variation in the waveguides. For simplicity and without losing generality, here we consider the extreme case where a waveguide segment is completely damaged such

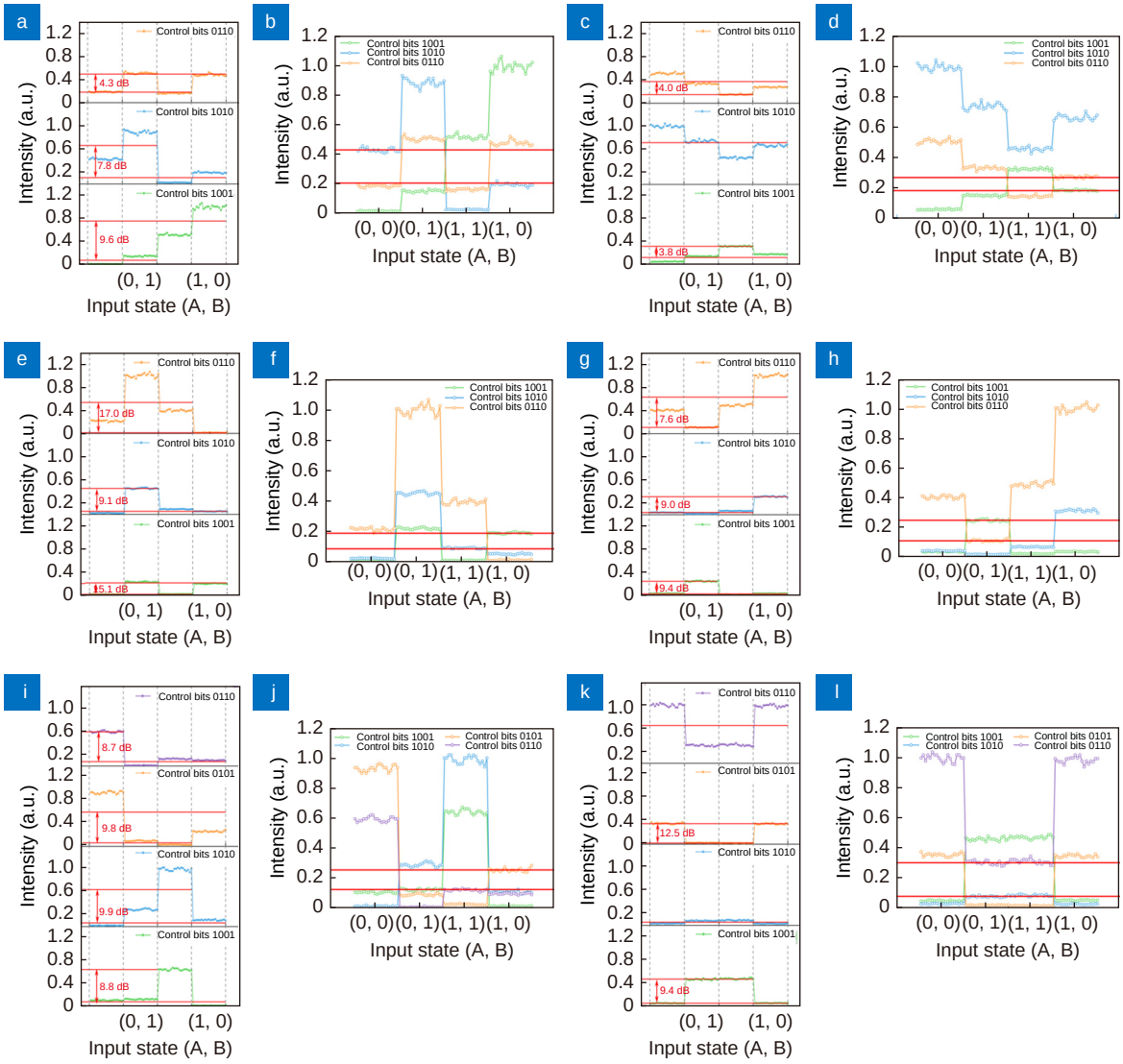

Fig. S5 | Multifarious logic gate operators. (**a, c, e, g, i, k,**) 0–1 intensity distribution of three/four logic gates corresponding to three/four kinds of control bits. (**b, d, f, h, j, l,**) Overlaid responses associated with three/four logic functions in a single structure corresponding to a, c, e, g, i, k, respectively. The top red line shows the minimum value of logic state "1", and the bottom red line shows the maximum value of logic state "0".

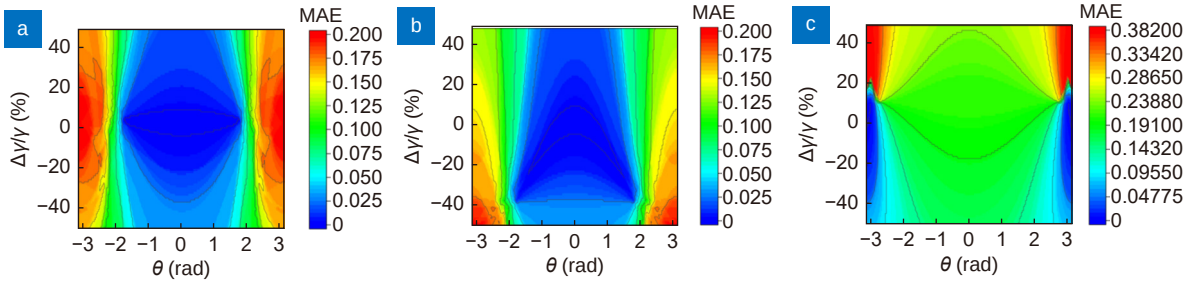

Fig. S6 | Analysis of weight deviation tolerance. (a, b, c) Represent the test results of changing the weights of the first, second and third layer, respectively. $\frac{\Delta\gamma}{\gamma}$ is the relative change of the amplitude ratio, and θ is the phase difference in each neuron layers. Different colors in the figures correspond to different MAE values.

that no optical signal can be transmitted into the neurons of the next layer. We found that the effect was only observed around the damaged waveguide, and almost no effect was observed in waveguides far away from the damaged one. For instance, in the predicted sample where $k = 2.18$, it should light up the waveguides numbered 2, 8, 13, 19, and 24. When the first waveguide or the thirteenth waveguide in layer 1, layer 2, or layer 3 is damaged, we obtain outputs as shown in

Table S1. In the table, waveguide numbers in blue indicate that they should have max intensity but do not due to presence of the defective waveguide, whereas waveguide numbers in red imply that they should not have max intensity but they do. The light intensity distribution of output waveguide is shown in Fig. S7(a). It can be seen that when a waveguide in layer 1 or layer 2 is damaged, the error propagates to layer 3. Consequently, the output light intensity at positions near the damaged waveguide is increased. Moreover, the impact of waveguide damage occurring in layer 2 on the distribution of output light intensity is significantly greater than that of waveguide damage occurring in layer 1. Therefore, if the damaged waveguide is exactly at the position of the predicted sample's solution, which means that it is supposed to have max intensity, the waveguide damage will not have a significant effect on the positions and number of solutions. Otherwise, it tends to produce an output at an incorrect location or mask the original output. However, for waveguides of layer 3, since output waveguides are behind them directly, if the damaged waveguide is located on the position of solutions, the solution will go missing; otherwise, the solution will not be affected (see Fig. S7(b)).

Table S1 | Outputs of a predicted sample under different conditions of waveguide fracture (k=2.18).

	Number of the fractured waveguide	
Layer	1	13
1	1, 2, 8, 13, 19, 24	2, 8, 11, 13, 19, 24
2	1, 2, 8, 13, 19, 24	2, 8, 13, 19, 24
3	2, 8, 13, 19, 24	2, 8, 12, 13, 14, 19, 24

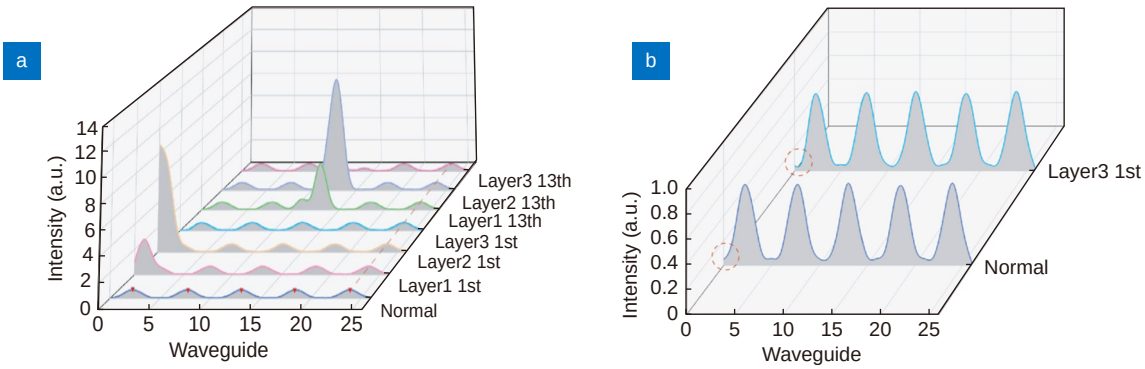

Fig. S7 | Analysis of waveguide damage tolerance. (**a**) Simulated light intensity distribution of output waveguides when the first or the thirteenth waveguide is damaged. The vertical axis corresponds to the output light intensity, the left horizontal axis labels the output waveguide number, and the right horizontal axis indicates the position of the damaged waveguide. (**b**) Comparison of light intensity distribution when the first waveguide in layer 3 is damaged with that without waveguide damage. The damage of the first waveguide in layer 3 does not affect the output at the second waveguide.

Section 6: Power consumption of optical neural network devices

Here we derive the power required to reach a given level of accuracy/performance for each of the optical CNN devices described in the main text. We further assume that low-noise light sources and photon counting detectors are used such that the ultimate noise floor is bound by the shot noise, which follows Poisson's statistics.

For the half adder and the logic gates, the digital output is encoded in terms of light intensity. The device performance is therefore gauged by the bit error rate (BER), which is related to photon count via:

$$BER = \frac{1}{2}P_0(N) = \frac{1}{2}\frac{N^0 e^{-N}}{0!} = \frac{1}{2}e^{-N}, \tag{S6}$$

where $P_0(N)$ is the probability of a mistake. The photon number N defines the threshold photon number required on the receiver to detect a "1" bit. In our experimental result, the minimal overall error rate is found when error rate in state "0" and "1" are equal. Therefore, we define the threshold energy to be detected as "1" equal to the geometric mean of the energies of the state "1" and state "0".

Considering 3.7 dB and 2.6 dB insertion losses for logic gate and half adder respectively, and apply Eq. S6 to our experimental result, the energy consumption per bit with respect to BER is plotted as Fig. S8(a). The figure

indicates ultra-low power consumptions of 10.4 aJ/bit and 23.8 aJ/bit to achieve a 10^{-9} BER in logic gate and half adder, respectively.

For the transcendental equation solver, the performance is evaluated by the solution accuracy. Here we take the result shown in Fig. 3 in the main text as an example. The mean error of the solution (ER) can then expressed as the normalized mean deviation to the true solution x_{sol}:

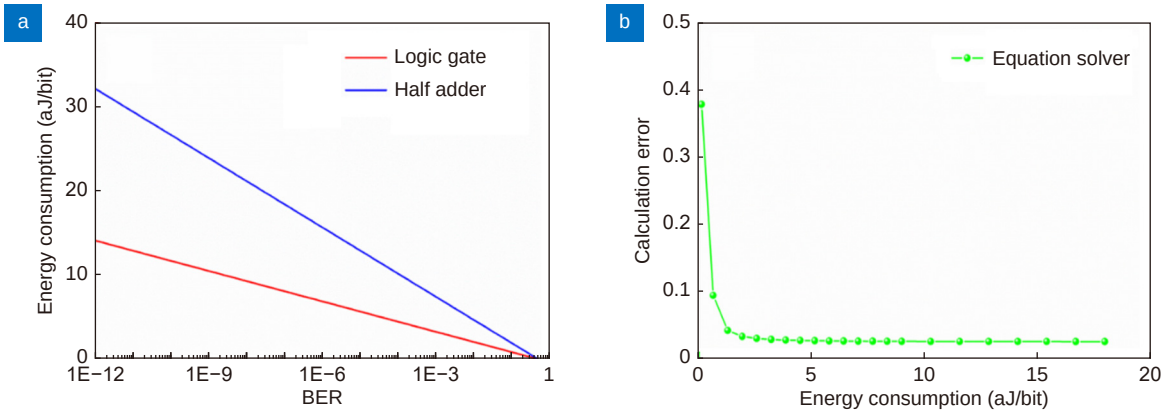

Fig. S8 | Calculation results of energy consumptions. (**a**) Energy consumption per bit of logic gate and half adder under different BER specifications. (**b**) Equation solver channel energy consumption with respect to mean error of the solution.

$$ER = P(i_{11}\ \text{is}\ \max)\frac{(x_{11}-x_{\mathrm{sol}})}{x_{\mathrm{sol}}} + P(i_{12}\ \text{is}\ \max)\frac{(x_{12}-x_{\mathrm{sol}})}{x_{\mathrm{sol}}} + \cdots + P(i_{17}\ \text{is}\ \max)\frac{(x_{17}-x_{\mathrm{sol}})}{x_{\mathrm{sol}}}\ , \tag{S7}$$

In our calculation we only account for channels 11–17 since they are in the immediate neighborhood of the correct output (channel 14). Each i_n here denote the output photon counts in channel number n, and we assume that they are independent Poisson random variables with the parameter λ_n, which equals to the simulated photon count in each channel. x_n denotes the numerical solution value that channel n corresponds to. Taking channel 11 as an example, the weighing factor which gives the probability of channel n = 11 having the highest output intensity (and hence x_{11} is regarded as the output solution from the optical neural network) is given by:

$$P(i_{11}\ \text{is}\ \max) = \sum_{i_{11}=1}^{\infty} F_{12}(i_{11})\cdots F_{17}(i_{11})\mathrm{e}^{-\lambda_{11}}\frac{\lambda_{11}^{i_{11}}}{i_{11}!}\ , \tag{S8}$$

where $F_n(i_{11})$ is the cumulative distribution function of Possion random variable of channel n and is expressed by:

$$F_n(i_{11}) = \mathrm{e}^{-\lambda_n}\sum_{j=0}^{i_{11}-1}\frac{\lambda_n^j}{j!}\ , \tag{S9}$$

Solving Eqs. S8 and S9, we numerically computed the ER with respect to channel energy consumption (photon count). The result is shown in Fig. S8(b). As indicated in the graph, increasing optical energy reduces the shot noise and enhances the solution accuracy. The shot-noise-limited mean error converges to a limit bounded by the discreteness of the network output at pulse energies above a few aJ/bit.

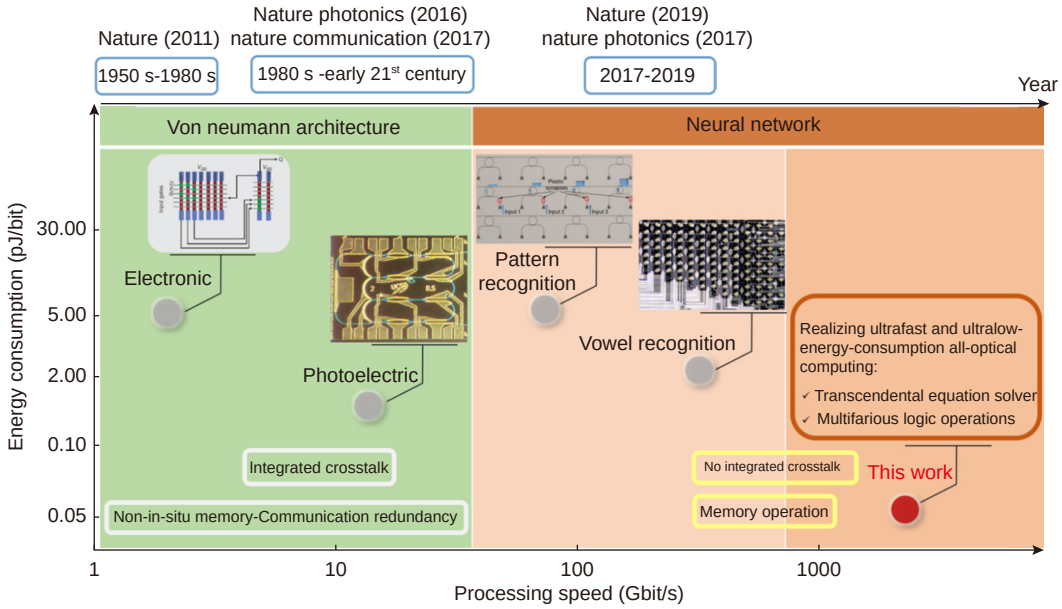

Fig. S9 | Development of on-chip integrated signal processor. Major research advances of on-chip integrated signal processors.

Section 7: Performance benchmark and significance of this work

Here, we summerize the major research advances of on-chip integrated signal processors in Fig. S9.

To benchmark the performance of our ONN, we compare the performances of our ONNs with state-of-the-art listed in Table S2.

Table S2 | Comparison of the performances of our ONNs with state-of-the-art

Comparison of the performances of our designed chip with the state-of-the-art works			
	Traditional electronic signal processing chip	Traditional photoelectric signal processing chip	This work
Predictable complex mathematical operations	×	×	√
Programmable multifarious logic operations	×	×	√
Overcoming integrated crosstalk	×	×	√
Processing speed	1~10 Gbit/s[3,4]	10~100 Gbit/s[5,6,7]	>Tbit/s
Energy consumption	1~5 pJ/bit[3,4] (1~50 mW)	0.19 pJ/bit[5,6,7] (1.9~19 mW)	50.8 fJ/bit (0.508 mW)

References

1. Fukushima K. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybern* **35**, 193–202 (1980).
2. Schmidhuber J. Deep learning in neural networks: an overview. *Neural Netw* **61**, 85–117 (2015).
3. Salahuddin S, Ni K, Datta S. The era of hyper-scaling in electronics. *Nat Electron* **1**, 442–450 (2018).
4. Yan H, Choe HS, Nam SW, Hu YJ, Das S et al. Programmable nanowire circuits for nanoprocessors. *Nature* **470**, 240–244 (2011).
5. Sun C, Wade MT, Lee Y, Orcutt JS, Alloatti L et al. Single-chip microprocessor that communicates directly using light. *Nature* **528**, 534–538 (2015).
6. Liu WL, Li M, Guzzon RS, Norberg EJ, Parker JS et al. A fully reconfigurable photonic integrated signal processor. *Nat Photonics* **10**, 190–195 (2016).
7. Marpaung D, Yao JP, Capmany J. Integrated microwave photonics. *Nat Photonics* **13**, 80–90 (2019).