Hybrid Convolutional Optoelectronic Reservoir Computing for Image Recognition

Philip Jacobson
Mizuki Shirao
Kerry Yu
Guan-Lin Su
Ming C. Wu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-251
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-251.html

December 8, 2021

Hybrid Convolutional Optoelectronic Reservoir Computing for Image Recognition

Philip Jacobson, *Student Member, IEEE,* Mizuki Shirao, Kerry Yu, Guan-Lin Su, and Ming C. Wu *Fellow, IEEE*

***Abstract*—Photonic delay-based reservoir computers (RC) have emerged as an attractive high-speed, low-power alternative to traditional digital hardware for AI. We demonstrate experimentally a novel hybrid RC scheme in which input data is first preprocessed through several convolutional layers, either trained or untrained, digitally to generate novel feature maps. These random feature maps are then processed through an optoelectronic implementation of delay-based RC. Using the MNIST dataset of handwritten digits, experiments of our proposed hybrid scheme achieve classification error of 1.6% using untrained convolutions, and an error of 1.1% using trained convolutions, results comparable to that of state-of-the-art machine learning algorithms. Additionally, our experimental implementation can offer a potential 10× decrease in model training time, compared to that of common digital alternatives.**

***Index Terms*—reservoir computing, photonics, image classification, convolutional neural networks**

I. Introduction

RECENT advancements in machine learning and in particular, deep learning, have allowed neural network models to achieve impressive performance on a wide range of tasks [1]–[3]. However, these increasingly complex models require significant computing resources to train, with training typically performed on large clusters of graphics processing units (GPUs) or tensor processing units (TPUs) [4]. This has resulted in a newfound demand for lightweight alternatives to neural networks which can be deployed in edge computing applications.

One method which has attracted significant research interest lately is known as Reservoir Computing (RC) [5]–[7]. Reservoir computing draws inspiration from traditional Recurrent Neural Networks (RNN), a variant of artificial neural networks which include recurrent connections between nodes in the hidden layers . RC similarly exploits this architecture, however instead of a series of ordered hidden layers stacked one upon the other, they are instead replaced by a "reservoir", a collection of nodes whose weights and connections are initialized randomly. At training time, weights within the reservoir are left untrained; instead, only the single output layer has its weights updated, typically through a standard linear regression routine, with optional regularization [8]. The relative simplicity of this training procedure allows for a significant reduction in required computation when compared to that of a standard RNN.

The simplicity of the RC architecture has made it an ideal candidate for implementation in hardware. Photonics is a particularly attractive platform, due to the potential of fast processing speeds and low power consumption [9]–[13]. However, implementation of large-scale spatial reservoirs remains challenging, due to the difficulty of coupling a large number of nonlinear optical elements. Reservoirs built with photonic integrated circuits remain relatively small in scale (limited to tens of nodes) [13]–[15], whereas using free space optics, though able to incorporate a larger number of nodes, sacrifices the compactness of an integrated solution [16], [17].

Because of this problem, an alternative to classical RC, known as delay-based RC, has gained significant traction recently [18]. As opposed to a spatial reservoir, delay-based RC instead uses a fully temporal reservoir, where physical nodes are replaced with so-called virtual nodes, created through time division multiplexing. These virtual nodes are processed serially through a single physical node with delayed feedback; more virtual nodes can be stored in the system memory through simply increasing the length of the delay. Optoelectronic [19]–[21] and fully optical [15], [22], [23] implementations of delay-based RC have achieved strong performance on several benchmark tasks, such as time series prediction and voice recognition. However, the relative structural simplicity arising from the lack of full spatial coupling (i.e. lack of connectivity between all nodes in the reservoir) has made scaling delay-based RC to tackle more complex tasks challenging.

Image recognition, a task central to machine learning and computer vision, has remained relatively under-explored in the RC literature because of these inherent limitations. With recent advancements in deep learning, classical image processing techniques using hand-crafted features have ceded ground to convolutional neural networks (CNNs), which have been able to achieve state-of-the-art performance on most tasks [1], [24]. CNNs are a class of neural networks which convolve input images with trained weights to extract useful features for classification, allowing the neural network to learn on its own the most useful features, rather than relying on more traditional feature engineering.

Our previous work introduced a new hybrid RC model to leverage the power of CNNs [25]. In this model, images undergo preprocessing through several convolutional layers with untrained weights, mimicking the feature extraction process of a CNN, but instead generating randomized feature maps. Applying this novel method to classifying images in the MNIST handwritten digit dataset, we are able to achieve a nearly 1% test error in simulation, in-line with the performance of CNN models. In this paper, we report three novel results: first, we introduce a further variation of this model, in which

we use weights trained through backpropagation within the convolutional layers, achieving performance better than that of a CNN in simulation. Second, we demonstrate the first experimental implementation of this hybrid RC scheme based on an optoelectronic delay-line reservoir computer, achieving performance consistent with simulations results. Third, we perform a timing analysis, showing that our proposed method offers the potential for a 10× decrease in training time through leveraging randomized convolutions in comparison to a CNN.

II. Hybrid Reservoir Computing

A. Delay-Based RC

Broadly speaking, the architecture of a reservoir computer can be described in three stages: the input layer, the reservoir, and the output layer. In the temporal model, the input data to the single physical node is created from raw pixels through a process known as *masking*. In this work, we generate a 2-dimensional mask matrix, typically a random sparse matrix, to be matrix multiplied by the flattened images. The dimension of this matrix is $N_{virtual\ nodes} \times N_{image\ dimensionality}$, allowing for the number of virtual nodes generated per image to be adjusted through changing the dimension of the mask matrix (the total number of virtual nodes stored in memory depends on both the mask matrix and the delay length). In general, virtual nodes can be stored across multiple delay lengths by simply letting the reservoir circulate for longer, allowing for scaling to a higher number of virtual nodes without directly modifying the system hardware. This masked input is then processed sequentially through the single node. The reservoir response, $x(t)$, is given as:

$$\tau_f \frac{dx(t)}{dt} + x(t) = G_{loop} f(x(t-\tau_D) + u_{in}(t) + V_{bias}) \quad (1)$$

where f is the nonlinear activation of the physical node, τ_D is the delay time, G_{loop} is the gain of the entire circulation, $u_{in}(t)$ is the masked input data, and V_{bias} is a constant bias. A low pass filter with time constant τ_f is used to effectively limit the memory of the system. The continuous reservoir response is sampled discretely to generate the virtual node values, $X(k)$, where k denotes discrete time. The final output layer is a linear mapping of the virtual node values to the desired predicted labels, $\hat{Y}$:

$$\hat{Y} = W^{opt} X(k) \quad (2)$$

where W^{opt} are the trained readout weights. For the task of image classification, the true class labels Y are encoded through one-hot encoding; i.e. Y is an $N_{samples} \times N_{classes}$ matrix, with a binary encoding used to denote the true class of a given image. Classification of the predicted labels $\hat{Y}$ is done with a winner-take-all approach, in which the class with the highest value is chosen to be the model prediction. Training is done using the ridge regression routine [26], where the equation for W^{opt} is given by:

$$W^{opt} = \underset{W}{argmin}\ ||Y - WX(k)||^2 + \lambda ||W||^2 \quad (3)$$

where λ is a regularization constant used to prevent overfitting. This optimization is performed offline, after all of the input data has been circulated through the reservoir.

B. Convolutional RC

Convolutional neural networks have convincingly supplanted traditional computer vision techniques for the majority of image classification tasks [1], [3], [27]. The building blocks of CNNs consist primarily of two types of hidden layers: convolutional layers, used to extract useful features from input images, and fully-connected layers, used to perform the actual classification on the extracted features [28]. Pooling layers, a type of hard-coded layer used to down-sample feature maps, are often interspersed between convolutional layers to reduce the dimensionality of feature maps.

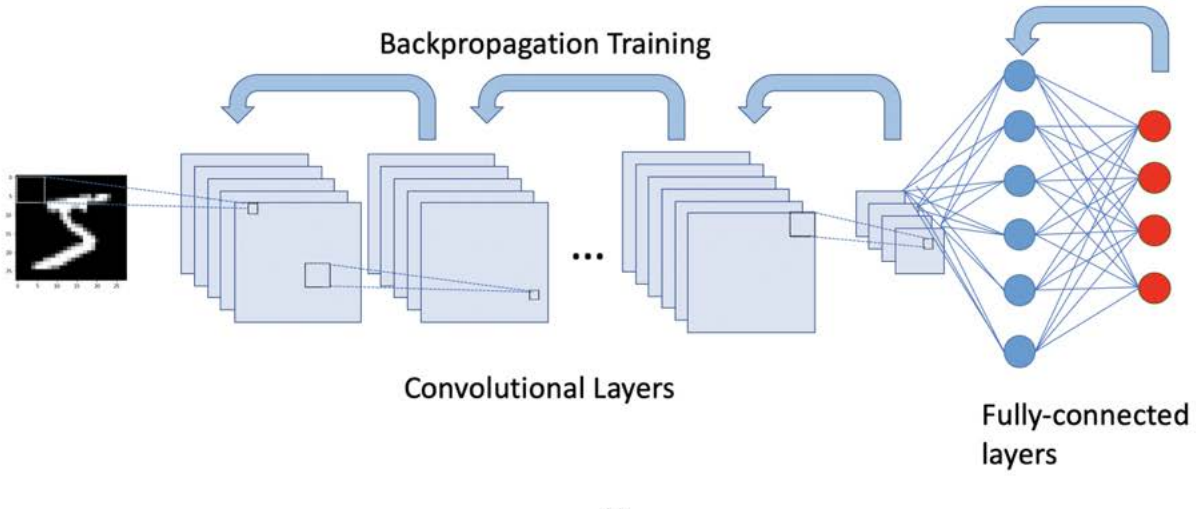
To capture the power of convolutional layers' feature extraction, we have introduced a new form of RC which combines standard delay-based RC with preprocessing images through convolutional layers [25]. After passing through the series of convolutional and pooling layers, the output feature maps are flattened row-wise into a single vector, which is then multiplied by the random mask matrix to generate the input nodes. Thus, each input node becomes a random linear combination of the output convolutional features, the exact nature of which depends on the mask matrix. A comparison of our proposed scheme with standard CNNs and RC is shown in Fig. 1. Normally, the weights within convolutional layers are trained through backpropagation to allow the CNN to learn the most optimal features for classification. We consider two models for our hybrid scheme: one which maintains the training of convolutional layers through backpropagation, and another in which we instead use convolutional layers which are randomly initialized, but then left untrained. The latter method, instead of extracting learned features, generates randomized feature maps to preserve the fast training speed of RC. Several examples of these random features are shown in Fig. 2. The number of feature maps generated is controlled through the final convolutional layer's width. After passing through these layers, all of the feature maps are flattened into a single feature vector and multiplied with a mask matrix before being processed through the reservoir computer itself.

C. Experimental Implementation

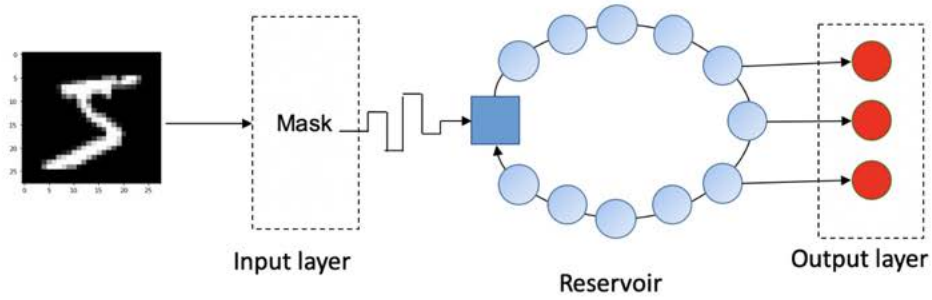
1) Digital preprocessing: The first phase of our hybrid RC model is processing the input images through convolutional layers, followed by input masking. This step is done entirely in the digital domain on a standard laptop computer[1]. The preprocessing scheme is implemented in Python using the TensorFlow library, with the matrix masking performed using a standard linear algebra package.

2) Optoelectronic RC: To implement the delay-based reservoir computer, we use a well-known implementation based on the optoelectronic oscillator (OEO) model [20], [29]. A schematic of our experimental testbed is shown in Fig. 3. The experimental set-up consists of a 1550 nm distributed feedback laser (Gooch & Housego DS-7009) modulated by a 40 Gb/s $LiNbO_3$ Mach-Zehnder modulator (Fujitsu FTM7937EZ). The modulated signal passes through an optical delay line before being converted to the electrical domain with a high-speed

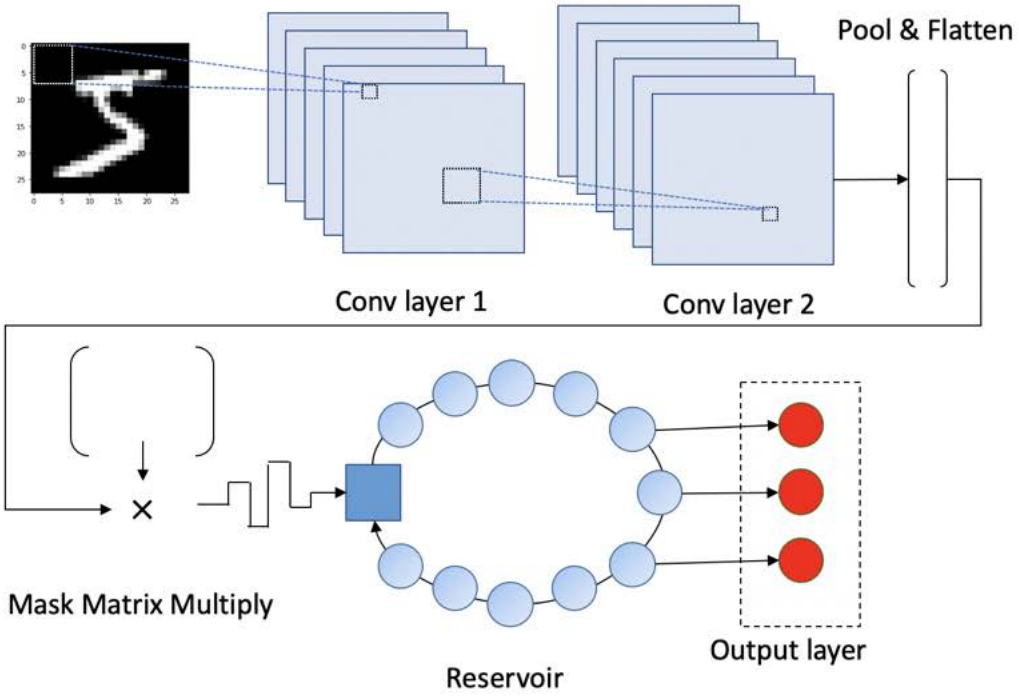
[1]MacBook Pro, 2.6 GHz 6-Core Intel Core i7



(a)



(b)



(c)

Fig. 1. (a) Prototypical CNN architecture with illustration of backpropagation training. (b) Standard delay-based RC architecture. (c) Architecture of our proposed hybrid RC scheme, illustrating preprocessing through untrained convolutional layers followed by delay-based RC. Figure adapted from [25].

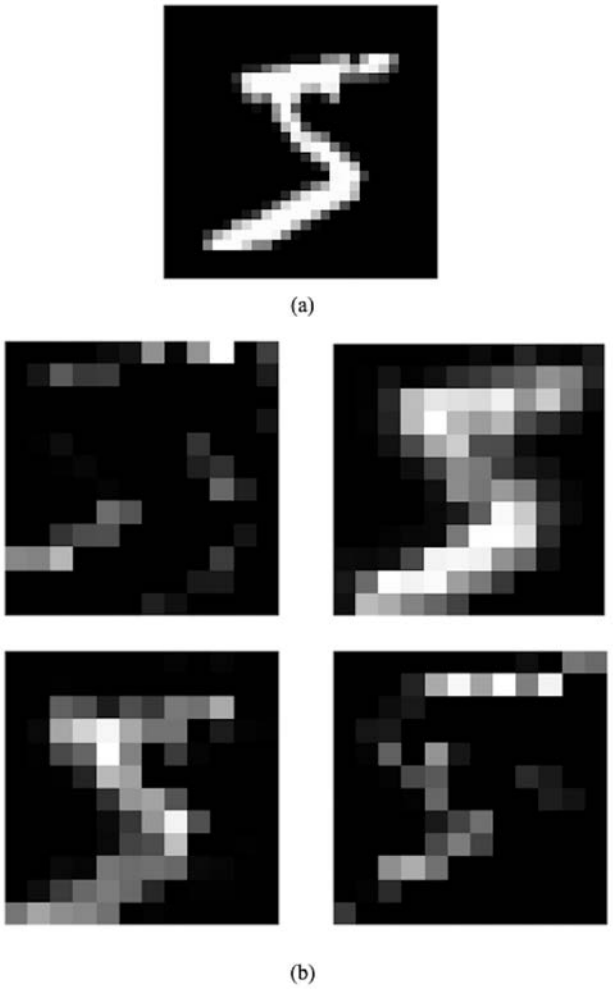


Fig. 2. (a) Example image from MNIST dataset of digit "5". (b) Several feature maps generated by forward pass through two untrained convolutional layers.

photodiode (MACOM P-18A). After passing through a low-noise transimpedance amplifier and low-pass filter, the analog signal is sampled by an analog-to-digital converter (ADC) with 12 bit resolution attached to a field programmable gate array (FPGA) mezzanine card (Zipcores FMC-DSP Rev. B). The reservoir response is added digitally to the input signal within the FPGA's programmable logic, before being output through a digital-to-analog converter (DAC) used to drive the modulator input. The bandwidth of this system is limited by the ADC/DAC, which have a 125 MSa/s sampling rate. For the experiments in this paper, we use a virtual node spacing is 80 ns, allowing for a total information processing speed of 12.5 MHz.

3) FPGA Design: To handle the data I/O of our system, we use a PicoZed board with a Zynq-7000 system on a chip (SoC). Preprocessed input data is transmitted from the PC to the board via 1 Gbps ethernet, which interfaces directly with the on-chip processing system (PS). The input data is stored in DRAM memory until ready to be processed by the experiment. The ADC/DAC module interfaces with the SoC through the chip's programmable logic (PL), where the input data is processed

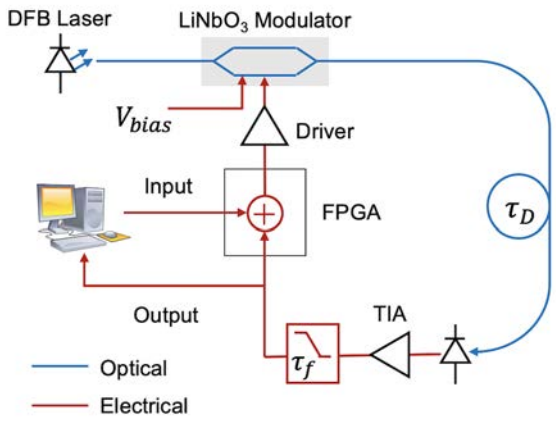


Fig. 3. Schematic illustrating optoelectronic reservoir computer used in experiments. Single wavelength distributed feedback laser is modulated by a Mach-Zehnder modulator, before passing through a delay line. Photodiode converts signal to electrical domain; FPGA sums input with feedback to drive modulator.

through a custom DSP block to perform feedback addition before being output through the DAC. The reservoir response is measured through the ADC, where virtual node values are computed through the averaging of 8 neighboring ADC samples in the PL in order to reduce sensitivity to noise. Virtual node samples are similarly stored in DRAM memory until being transmitted back to the PC via ethernet to perform the linear regression. Data transfer between the PS and PL is done using a standard AXI DMA module. Because of limits on the DRAM memory capacity, the input data is processed in blocks of 4 MB, during which input data is sent to the FPGA, the inputs are circulated through the reservoir, and virtual node samples are sent back to the PC. Because only a fraction of an image is stored in the reservoir memory at a given time, we are able to forego resetting the reservoir between images without degrading performance, allowing for faster processing. This process is repeated until all of the input nodes are sent through the reservoir. A diagram of our FPGA architecture is shown in Fig. 4.

III. MNIST Classification: Experimental Results

To experimentally demonstrate our proposed RC scheme, we use the well-known MNIST handwritten digit classification task. The MNIST dataset consists of 70000 (60000 training, 10000 test) grayscale 28×28 images of handwritten digits 0-9; the goal of the task is to correctly identify the written number [30]. We consider two benchmarks for this task: first the LeNet-5 CNN, a well-known CNN architecture adapted for this problem, which achieves a 0.95% test error [24]. Modern neural networks with even deeper architectures have further improved accuracy on MNIST to a few fractions of a percent [31], however these exceed the training capabilities of a standard CPU and so are less useful to us as benchmarks. Additionally, we also consider an Extreme Learning Machine (ELM), a variant of neural network containing layers of

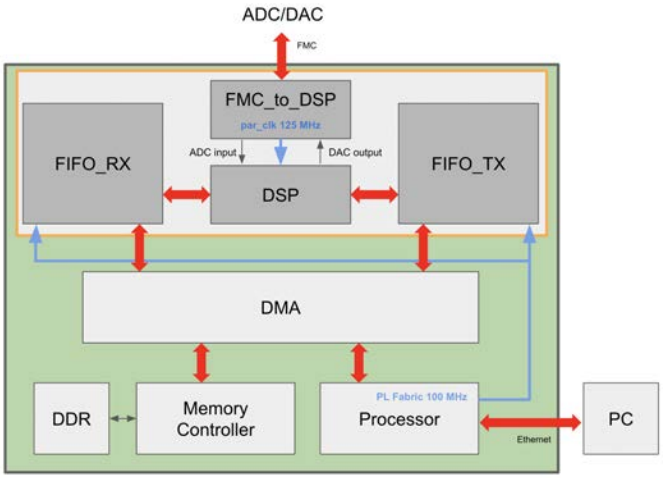


Fig. 4. Block diagram of the FPGA configuration used in our experiments.

untrained neurons, as a benchmark [32]. We use an ELM containing two convolutional layers and one fully-connected layer with 15000 neurons, all of which are left untrained, to serve as a memoryless corollary to our hybrid RC scheme. This benchmark achieves a 1.4% test error on the MNIST task.

A. Parameter Optimization

Our model has several adjustable parameters that need to be optimized, both physical and digital. We explore these primarily through experiments, running scaled-down experiments of 1000 virtual nodes per image to measure performance.

1) Convolutional Preprocessing: Within the convolutional layers used to generate input feature maps, there are several parameters we can adjust: the number of layers, the width of the layers, convolutional kernel size, and the inclusion of pooling layers. Unlike in trained CNN models, performance improvements from adding more untrained convolutional layers level off relatively quickly; for this task, we use only two layers. The width of the layers, representing the number of convolutional kernels included within the layer, similarly has a small effect on performance once a small threshold has been passed. In our experiments, we use layers of width 32 and 64, respectively. We use a 3×3 convolutional kernel within each of these layers, consistent with many modern CNN models [33]. Additionally, we include a third 2×2 max-pooling layer as part of this preprocessing scheme, as we found it to significantly increase test accuracy while further reducing dimensions and processing time. Processing images through these three layers produces a flattened vector of dimension 9216 x 1.

2) Random Matrix Masking: The random mask matrix, used to generate the input nodes to the reservoir computer from the preprocessed feature maps, has several adjustable parameters associated with it. The first of these, the matrix density, denotes the number of non-zero elements in the matrix, representing the fraction of matrix elements with non-zero values. For this problem, a low matrix density of 0.01 is found to achieve optimal experimental performance. The other parameter associated with the mask matrix is the random distribution from which the non-zero elements are drawn from. From experimentation, we used a uniform random distribution with range $(-1, 1)$. After the matrix is initialized, we perform a row-wise normalization to limit the range of generated input nodes.

3) Experimental Parameters: In addition to the parameters associated with preprocessing, there are also several experimental parameters to be optimized in our optoelectronic experimental testbed. First, we consider the delay-line length and the low pass filter constant. Because the MNIST task does not inherently require memory, we found that experimental performance has a fairly weak correlation with the delay-line length, with test accuracy obtained using a 100 meter and 2 meter delay being equivalent, as long as laser power is increased to compensate for loss in the longer delay line. For convenience, we use the short 2 meter delay in the rest of our experiments. To validate the idea of using an RC (i.e. memory-dependent) approach for a static image recognition task, we also tried eliminating the system memory by simply removing the feedback addition. In this case, we found a sizable decrease in test accuracy of 0.7% from experiments in which memory is included. Thus, while the inclusion of a memory mechanism adds a small boost to performance from the introduction of a further (random) degree of freedom, storing a larger number of virtual nodes in a long delay-line has a minimal effect. This result is as we expect, since individual virtual nodes have little spacial relation between each other due to the random masking process, meaning a large memory bank does not offer a more meaningful global view of the image. We use a low pass filter with a time constant approximately equal to one times the virtual node spacing; longer time constants result in degrading performance as a result of too much averaging [18].

Also within our experiment are several parameters used to control the operating point of the dynamical system itself (i.e. whether the system is in an oscillatory regime or not). In the OEO model, these are typically captured by the single-loop gain, i.e. the small-signal gain of a single circulation through the entire system, and the modulator bias point [29]. The former can be most directly controlled through the power of the laser source, whereas the latter is simply the DC bias voltage of the Mach-Zehnder Modulator, in terms of V_π, with $1V_\pi$ corresponding to the minimum transmission of the modulator. We performed an experimental parameter sweep of these two quantities, illustrated in Fig. 5. We find that test accuracy is generally stable, except in the quadrant corresponding to high laser power and low modulator bias. This part of the parameter space is congruous with the OEO's oscillation regime, illustrating the deleterious effect of the induced oscillation on the system's performance. Thus, for this task, we can choose the system's operating point fairly liberally while maintaining fairly consistent performance. For the experiments in this paper, we used a modulator bias of $1V_\pi$ and a laser power of 3.5 mW, while driving the modulator at 0.4V.

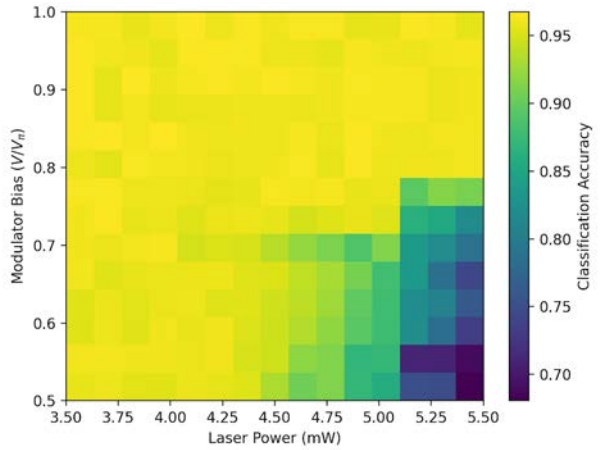


Fig. 5. Plot of experimental parameter sweep as a function of laser power and modulator bias. A modulator bias of 0 corresponds to the peak of the sinusoidal transfer function.

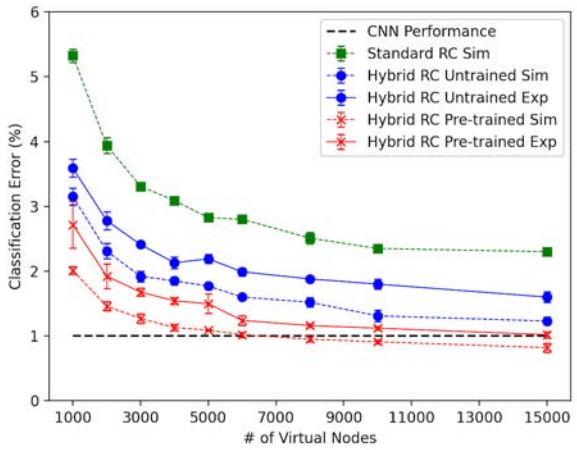


Fig. 6. Test error on MNIST task plotted as a function of virtual nodes for standard RC, hybrid untrained RC, and hybrid trained RC, both experiment and simulation.

B. Untrained Hybrid RC

First, we discuss experimental results using untrained convolutional layers for our hybrid preprocessing scheme. The main appeal of this approach is the minimization of processing time; by using completely untrained convolutional layers, the preprocessing stage is reduced to the time required to randomly initialize the weights within the layers, followed by performing a single forward pass. In this scheme, we randomly initialize the weights using TensorFlow's `glorot_uniform` initializer, which draws weights from a random uniform distribution whose interval is inversely proportional to the sum of the input and output dimensions of the layer. We performed simulations and experiments ranging in size from 1000 virtual nodes per image to 15000 virtual nodes per image, with results shown in Fig. 6. In simulation, we achieved test errors of 3.16% and 1.2%, respectively, at 1000 and 15000 virtual nodes [25], whereas we achieved 3.6% and 1.6% for for the same points in experiments. As benchmarks, we also plot performance of a standard RC scheme (i.e. no convolutional preprocessing) and a well-known CNN model, LeNet-5 . For all reservoir sizes, our hybrid scheme, even with random convolutions, improves test accuracy by more than a percentage point in comparison to the standard implementation in both experiment and theory. At 15000 virtual nodes, we find our hybrid RC scheme comes close to the 0.95% test accuracy achieved using LeNet-5 [24]. Additionally, simulations of our scheme outperform the ELM baseline, indicating the added benefit of including RC memory.

C. Trained Hybrid RC

As an alternative approach, we also consider ways of implementing our hybrid RC scheme using trained weights in the convolutional layers, rather than simply randomly initializing them. The most straightforward way of achieving this is to fully train a CNN using backpropagation, after which we take the trained convolutional layer weights for use in our preprocessing procedure. For this purpose, we use a 6-layer CNN, denoted MNIST convnet in Table I, containing the aforementioned 2 convolutional layers and one max pooling layer, followed by two fully-connected layers with 128 and 10 neurons, respectively for classification [34]. In total, this CNN model contains 1.2 million trainable parameters, with around 18000 of them contained within the two convolutional layers. Training this model to convergence achieves around 1% test error, similar to that of LeNet-5. Using the two trained convolutional layers and max pooling layer for preprocessing, we replicate the experiments in Sec. III-B, also plotted within Fig 6. In simulation, we achieve 2% and 0.8% test accuracy at 1000 and 15000 virtual nodes respectively, whereas we achieve 2.7% and 1.1% test accuracy in experiments. Particularly for smaller numbers of virtual nodes, pre-training the convolutional weights significantly improves test accuracy, as seen in the large divergence between the red and blue curves in Fig 6. We also note that at 15000 virtual nodes, our hybrid scheme in simulation actually outperforms the CNN model from which we have derived our trained weights from, as well as outperforming the LeNet-5 benchmark. Experiments nearly match these results, with test accuracy virtually indistinguishable from that of a CNN.

D. Experimental Comparison

In both the trained and untrained hybrid RC implementation, we are able to achieve test accuracy nearly equal to that of state-of-the-art CNN models for the MNIST task. Additionally, our experimental optoelectronic implementation offers further benefits compared to most digital machine learning algorithms. Most salient of these is the potential for significant increases in processing speed. The total processing time for our untrained hybrid RC implementation can be broken down into four parts: the convolutional preprocessing, the mask matrix multiplication, the reservoir circulation, and the final linear regression. Even for large reservoirs of 15000 nodes

TABLE I
COMPARISON OF HYBRID RC APPROACH WITH VARIOUS OTHER MACHINE LEARNING METHODS

Method	Test Error	Training Time*
Le-Net 5 CNN [24]	0.95%	1
MNIST Convnet	1.0%	1
Hybrid RC Untrained (Simulation)	**1.2%**	15
Hybrid RC Untrained (Experiment)	**1.6%**	**0.1**
Hybrid RC Trained (Simulation)	**0.8%**	15
Hybrid RC Trained (Experiment)	**1.1%**	**1.1**
Standard RC [25]	2.3%	0.1
ELM	1.4%	0.25
Linear Classifier [24]	7.6%	0.01

*Normalized to LeNet-5 CNN case

or more, the theoretical minimum processing time through the reservoir computer is quite small, requiring only a few seconds using a 125 MHz FPGA clock. To provide an estimate of speed increase, we compare the processing time of these four steps using our laptop's CPU, versus fully training a LeNet-5 network to convergence on the same CPU. Comparing these, our hybrid scheme can offer a potential $10\times$ increase in processing speed compared to the training of the CNN, with the processing speed bottleneck arising from the large matrix multiplication required to produced the masked inputs. The full breakdown of our calculated hybrid RC processing time is as follows: 15% is spent performing convolutional prepocessing, 55% is spent performing the random mask matrix multiplication, 10% is spent circulating the virtual nodes through the reservoir, and 20% is spent running the ridge regression optimization. In total, the CNN training time is 10 minutes, whereas our hybrid RC training routine can be condensed down into as short as approximately one minute.

The trained hybrid RC scheme does not offer this same speed advantage, given that it requires the training of a CNN to select the desired weights. Thus, the trade-off between these two approaches is between a very fast solution with slightly worse performance, versus a slower solution, but with performance even surpassing many state-of-the-art algorithms. These results comparing processing speed and performance are summarized in Table I. A potential middle ground exists in the leveraging of transfer learning, in which convolutional layer weights are taken from an already on-hand CNN model trained for a different task, from which some feature extraction information is still preserved [35]. We leave this for exploration in future work.

IV. CONCLUSION

In this paper, we expound upon our previously introduced hybrid reservoir computing scheme. This approach utilizes a combination of preprocessing through convolutional layers and a standard delay-based optoelectronic reservoir computer implementation to adapt RC to tackle tasks within the computer vision domain. We have built the first experimental demonstration of this system, based on an OEO-inspired implemnetation, and have to date performed experiments ranging in size from 1000 to 15000 virtual nodes per image on the MNIST handwritten digit classification task. Within this approach, we consider two ways of initializing the convolutional layers used in our preprocessing step: in the first, we randomly initialize the weights, and in the second, we select weights from a CNN trained through backpropagation. Experimental results have shown impressive performance using both of these methods; the first sacrifices a small amount of performance for processing speeds up to $10\times$ faster than digital solutions, whereas the second requires more time for training, but can outperform even state-of-the-art CNN models. Our experimental demonstration, using a maximum of 15000 virtual nodes, achieves a test error of 1.6% using untrained convolutions, and a test error of 1.1% using trained convolutions, compared to the 0.95% test error of the LeNet-5 CNN. With the promising results obtained using our system, further scaling the system to tackle more challenging image datasets, or tasks with more explicit temporal dependence, such as video classification, present interesting future directions for this line of work.

ACKNOWLEDGMENT

This project was supported in part by the DARPA Photonic Edge AI Compact Hardware (PEACH) program.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems (NeurIPS)*, 2012, pp. 1097–1105.
[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 2015.
[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *Int J Comput Vis*, vol. 115, p. 211252, 2015.
[4] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018.
[5] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," *Technical Report GMD Report 148, German National Research Center for Information Technology*, 2001.
[6] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
[7] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, pp. 78–80, 2004.
[8] M. Lukosevicius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, p. 127149, 2009.
[9] D. Woods and T. J. Naughton, "Photonic neural networks," *Nature*, vol. 8, pp. 257–249, 2012.
[10] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, and M. Solja, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, vol. 11, pp. 441–447, 2017.
[11] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, pp. 1004–1008, 2018.
[12] A. N. Tait, T. F. de Lima, E. Zhou, A. X. Wu, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal, "Neuromorphic photonic networks using silicon photonic weight banks," *Scientific Reports*, vol. 7, p. 7430, 2017.
[13] K. Vandoorne, P. Mechet, T. V. Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, "Experimental demonstration of reservoir computing on a silicon photonics chip," *Nature Communications*, vol. 5, p. 3541, 2014.
[14] K. Harkhoe, G. Verschaffelt, A. Katumba, P. Bienstman, and G. Van der Sande, "Demonstrating delay-based reservoir computing using a compact photonic integrated chip," *Opt. Express*, vol. 28, no. 3, pp. 3086–3096, Feb 2020. [Online]. Available: http://www.opticsexpress.org/abstract.cfm?URI=oe-28-3-3086

[15] K. Takano, C. Sugano, M. Inubushi, K. Yoshimura, S. Sunada, K. Kanno, and A. Uchida, "Compact reservoir computing with a photonic integrated circuit," *Optics Express*, vol. 26, no. 22, pp. 29 424–29 439, 2018.

[16] P. Antonik, N. Marsal, and D. Rontani, "Large-scale spatiotemporal photonic reservoir computer for image classification," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 1, pp. 1–12, 2020.

[17] R. Alata, J. Pauwels, G. Van der Sande, A. Bouwens, M. Haelterman, and S. Massar, "Phase noise robustness of a coherent spatially parallel optical reservoir," in *2019 Conference on Lasers and Electro-Optics Europe European Quantum Electronics Conference (CLEO/Europe-EQEC)*, 2019, pp. 1–1.

[18] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, "Information processing using a single dynamical node as complex system," *Nature Communications*, vol. 2, pp. 466–468, 2011.

[19] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, "Optoelectronic reservoir computing," *Scientific Reports*, vol. 2, pp. 1–6, 2012.

[20] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso, and I. Fischer, "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing," *Optics Express*, vol. 20, no. 3, p. 3241, 2012.

[21] L. Larger, A. Bayln-Fuentes, R. Martinenghi, V. S. Udaltsov, Y. K. Chembo, and M. Jacquot, "High-speed photonic reservoir computing using a time-delay-based architecture: Million words per second classification," *Physical Review X*, vol. 17, p. 011015, 2017.

[22] F. Duport, A. Smerieri, A. Akrout, M. Haelterman, and S. Massar, "Fully analogue photonic reservoir computer," *Scientific Reports*, vol. 6, pp. 1–12, 2016.

[23] C. Sugano, K. Kanno, and A. Uchida, "Reservoir computing using multiple lasers with feedback on a photonic integrated circuit," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 20, no. 1, 2020.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] P. Jacobson, M. Shirao, K. Yu, G.-L. Su, and M. C. Wu, "Image classification using delay-based optoelectronic reservoir computing," in *AI and Optical Data Sciences II*, B. Jalali and K. Kitayama, Eds., vol. 11703, International Society for Optics and Photonics. SPIE, 2021, pp. 120 – 126. [Online]. Available: https://doi.org/10.1117/12.2578062

[26] A. N. Tikhonov, A. V. Goncharsky, V. V. Stepanov, and A. G. Yagola, *Numerical Methods for the Solution of Ill-Posed Problems*. New York, NY, USA: Springer, 1995, vol. 328.

[27] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, pp. 958–963.

[28] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybernetics*, vol. 36, p. 193202, 1980.

[29] X. S. Yao and L. Maleki, "Optoelectronic oscillator for photonic systems," *IEEE Journal of Quantum Electronics*, vol. 32, no. 7, pp. 1141–1149, 1996.

[30] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*. [Online]. Available: https://ci.nii.ac.jp/naid/10027939599/en/

[31] D. Cirean, U. Meier, L. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computing*, vol. 22, no. 12, pp. 3207–3220., 2010.

[32] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[34] F. Chollet, "Keras documentation: Simple mnist convnet," Jun 2015. [Online]. Available: https://keras.io/examples/vision/mnist_convnet/

[35] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.