Hybrid Quantum-Classical Convolutional Neural Networks

Junhua Liu,[1,2,3] Kwan Hui Lim,[2] Kristin L. Wood,[3,4] Wei Huang,[5] Chu Guo,[1,*] and He-Liang Huang[6,7,8,†]

[1]*Quantum Intelligence Lab, Supremacy Future Technologies, Guangzhou 511340, China*
[2]*Information Systems Technology and Design, Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore*
[3]*SUTD-MIT International Design Center, Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore*
[4]*Engineering Product Develoment, Singapore University of Technology and Design, 8 Somapah Road, 487372 Singapore*
[5]*Guangxi Key Laboratory of Optoelectronic Information Processing, Guilin University of Electronic Technology, Guilin 541004, China*
[6]*Hefei National Laboratory for Physical Sciences at Microscale and Department of Modern Physics, University of Science and Technology of China, Hefei, Anhui 230026, China*
[7]*Shanghai Branch, CAS Centre for Excellence and Synergetic Innovation Centre in Quantum Information and Quantum Physics, University of Science and Technology of China, Hefei, Anhui 201315, China*
[8]*Henan Key Laboratory of Quantum Information and Cryptography, Zhengzhou, Henan 450000, China*

(Dated: November 11, 2019)

Deep learning has been shown to be able to recognize data patterns better than humans in specific circumstances or contexts. In parallel, quantum computing has demonstrated to be able to output complex wave functions with a few number of gate operations, which could generate distributions that are hard for a classical computer to produce. Here we propose a hybrid quantum-classical convolutional neural network (QCCNN), inspired by convolutional neural networks (CNNs) but adapted to quantum computing to enhance the feature mapping process which is the most computational intensive part of CNN. QCCNN is friendly to currently noisy intermediate-scale quantum computers, in terms of both number of qubit as well as circuit's depths, while retaining important features of classical CNN, such as nonlinearity and scalability. We demonstrate the potential of this architecture by applying it to a Tetris dataset, and show that QCCNN can accomplish classification tasks with performance surpassing the CNN benchmark.

In view of the rapid progresses in quantum computing hardware, we are entering into the era of developing quantum software to perform useful computational tasks using the noisy intermediate-scale quantum (NISQ) computers [1]. It has been recently shown that the current 53-qubit quantum computer could already solve random quantum circuit sampling problems more efficiently than the best supercomputers in the world [2]. Theoretically, this success originates from the fact that a quantum computer could output wave functions with polynomial number of quantum gate operations, which could nevertheless generate statistical distributions that are very hard for a classical computer to produce [3]. If a quantum computer could easily produce complex distributions, it is also natural to postulate that it is able to learn patterns from certain data distributions which could be very difficult for classical computers [4]. Quantum machine learning (QML) attempts to utilize this power of quantum computer to achieve computational speedups or better performance for machine learning tasks, and parameterized quantum circuits (PQCs) offer a promising path for quantum machine learning in the NISQ era [5, 6]. Compared to traditional quantum algorithms such as Shor's algorithm, PQCs based quantum machine learning algorithms are naturally robust to noise and could also benefit from quantum advantages, by taking advantage of both the high-dimensional Hilbert space of a quantum system and the classical optimization scheme. Several popular related algorithms have been proposed, including variational quantum eigensolvers (VQE) [7], the quantum approximate optimization algorithm (QAOA) [8], quantum generative adversarial networks [9, 10], and quantum classifiers [11, 12]

For QML to solve real world problems, the first step is to translate classical data, which is usually represented as a multi-dimensional array, into a quantum state. A standard way is to use a kernel function to map each element of the array into a single-qubit state, which is often referred to as qubit-encoding. The kernel function could, for example, be chosen as

$$\theta_j \rightarrow \cos(\theta_j)|0\rangle + \sin(\theta_j)|1\rangle. \tag{1}$$

For an input array of size L, the mapping would result in an L-qubit quantum state, which lives in a Hilbert space of size 2^L. For real world data with a large size, this mapping would soon become impractical for current quantum computers with less than 100 qubits. The same problem also exists in classical deep learning, which is often built from interlacing layers of linear and nonlinear functions. A straightforward way to implement the linear function is the so-called fully connected layer, which can be represented as a dense matrix connecting each neuron of the output to all the neurons of the input. When the input size is large, this approach would become inefficient due to the large matrix size. Convolutional Neural Network (CNN) [13, 14] is a very popular scheme which tries to solve this problem by replacing the fully connected layer with a convolutional layer. The convolutional layer only connects each neuron of the output to a small region (window) of the input which is referred to as a feature map, thus greatly reducing the number of parameters. CNN has demonstrated itself as one of the most successful tools in the area of computer vision [15–19], and more recently, it also found applications in Natural Language Processing [20–25].

Inspired by CNN, we propose a hybrid quantum-classical Convolutional Neural Network (QCCNN). We note that re-

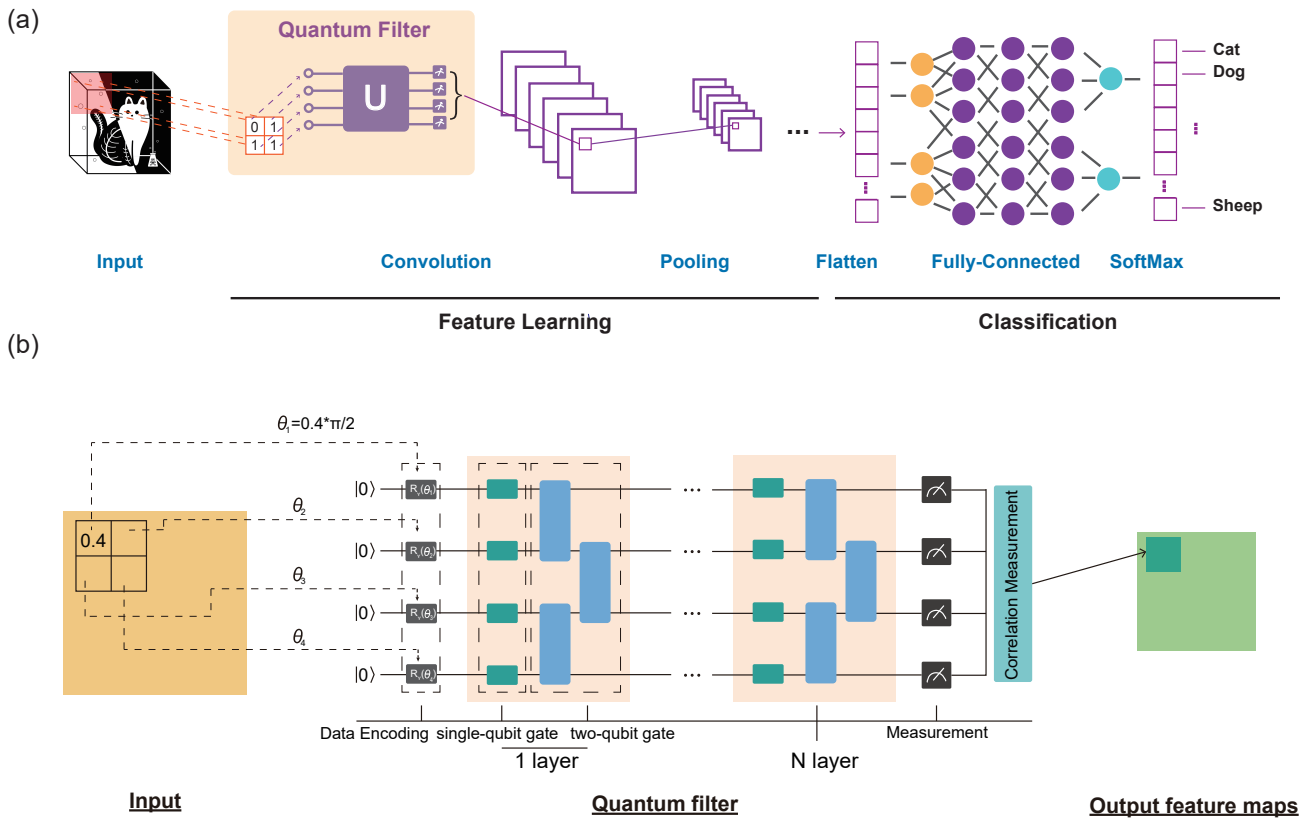

FIG. 1. (a) Hybrid quantum-classical Convolutional Neural Network (QCCNN). The input demonstrated here is a two-dimensional array, which is sent to a quantum convolutional layer of 6 filters. Each filter takes a 2×2 window, translating it into a separable 4-qubit quantum state, and evolves this state with a parametric quantum circuit. After that a correlational measurement is made on the output quantum state and a scalar is obtained. Gathering the scalar outputs, the final output of the quantum convolutional layer is a 3-dimensional array. Then a pooling layer is used to reduce the dimensionality of the data. This process could be repeated and finally ends with a fully connected layer. (b) Details of our design of parametric quantum circuit, which is made of interlaced single-qubit layer and two-qubit layers. The single-qubit layer consists of R_y gates, each containing one tunable parameter. The two-qubit layer consists of CNOT gates on nearest-neighbour pairs of qubits.

cently, a pure quantum analogy of CNN, named QCNN, was proposed to solve certain quantum many-body problems [26]. Similar to other QML algorithms, QCNN uses as many qubits as the size of the input, which makes it unlikely to be implemented on current quantum computers to solve real world problems. The central idea of QCCNN is to implement the feature map in the convolutional layer with a parametric quantum circuit, and correspondingly, the output of this feature map is a correlational measurement on the output quantum state of the parametric quantum circuit. In the following, we refer to this new structure as a quantum convolutional layer. As a result, the number of qubits required by this approach is only related to the window size of the feature map, which often ranges from 3×3 to 9×9 and is well within reach for current quantum computers. Moreover, since the output of our quantum convolutional layer is a classical array, it is straightforward to adapt the multi-layer structure as in CNN. Therefore, our QCCNN could utilize all the features of classical CNN, and at the same time, it is able to utilize the power of current NISQ computers.

To better describe our design of QCCNN (see Fig. 1(a)), we first briefly outline some basic features of CNN. CNN consists of interlaced convolutional layers and pooling layers, and ends with a fully connected layer. The primary purpose of the convolution layer is to extract features from the input data using a feature map (or filter), which is the most computational intensive step of CNN. After the convolutional layer, it is common to add a pooling layer to reduce the dimensionality of the data and prevent overfitting. A single filter maps small regions (windows) of the input to single neurons of the output, and is parameterized by an array P which has the same shape as the window. The windows are often chosen as follows. Assuming the input is a two-dimensional array A of size $v \times h$, and the predefined window size is $m \times n$, then the first window is located at the upper left conner of A, namely $A_{1:m,1:n}$ (here $a:b$ denotes the range from a to b). Then the mapping is done by the linear function

$$A_{1:m,1:n} \to \sum_{1 \le i \le m, 1 \le j \le n} A_{i,j} P_{i,j}. \tag{2}$$

Then the next window slides to the right with a stride value s, which is often chosen to be 1, until it reaches the right edge. After that it hops down to the (left) beginning of the image with the same stride value s and repeats the process until the entire image is traversed. As a result, after the evolution, the output will be a two-dimensional array of size $\frac{v-m+1}{s} \times \frac{h-n+1}{s}$. Moreover, in general one could have several filters in the same layer and the input could be a three-dimensional array. For example, for a three-dimensional array of size $v \times h \times d$, and if we have k filters with size $m \times n$, assuming the stride s, then the output array would have the shape $\frac{v-m+1}{s} \times \frac{h-n+1}{s} \times (dk)$. Generally after a convolutional layer, the output would become thiner but longer. In some situations, one would like to prevent the data to become thiner, by adding zeros around the edges of input, which is referred to as padding.

In our quantum convolutional layer, the filter is redesigned to make use of the parametric quantum circuit, which is shown in Fig. 1(b), and we refer to it as a quantum filter. A quantum filter takes windows of shape $m \times n$, maps them into quantum states $|\psi^i(A_{p:(p+m-1),q:(q+n-1)}\rangle$ of $N = mn$ qubits using Eq.(1), and then evolves the quantum state with the parametric quantum circuit $\mathcal{C}(\vec{\theta})$, such that the output quantum state $|\psi^o\rangle$ is

$$|\psi^o\rangle = \mathcal{C}(\vec{\theta})|\psi^i(A_{p:(p+m-1),q:(q+n-1)}\rangle. \tag{3}$$

After the evolution, we take the expectation value of the observable $Z^{\otimes N}$, thus the feature map can be written as

$$A_{p:(p+m-1),q:(q+n-1)} \to \langle\psi^o|Z^{\otimes N}|\psi^o\rangle. \tag{4}$$

Eq.(4) is nonlinear and thus in our quantum convolutional layer, we do not need an additional nonlinear function such as ReLU to explicitly bring in nonlinearity. It is also clear from Eq.(4) that in our approach, the minimal number of qubits required is equal to the window size. For the next window of our quantum convolutional layer, these qubits could be reused. Thus, the quantum convolutional layer is experimentally friendly and suitable for NISQ scenarios because only a few qubits are needed and no any additional usage of qRAM is required. And the quantum correlational measurement has the potential to better capture the cross-correlation inside each window. In our architecture, the pooling layers as well as the final fully connected layer are kept in the same way as CNN since they are computationally cheap, and can further induce nonlinearities.

The parametric quantum circuit we use contains interlaced single-qubit layers and two-qubit layers. The total number of gate operations of a parametric quantum circuit, denoted as L, grows only polynomially with the number of qubits N, namely $L \sim poly(N)$ gates. In our setup, the two-qubit layer consists of CNOT gates while the single qubit layer consists of rotational Y gates (R_y), which is defined as

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{5}$$

Each layer of two-qubit gates is counted as one depth. For optimization problems, it is usually helpful to provide the gradient of the loss function. The easiest way to approximately compute the gradient in our case is to use the finite difference method, which only requires forward evaluation of the loss function. Here we show how to exactly compute the gradient using auto-differentiation, which could also be implemented with a quantum computer. Since the only 'quantum' step is the feature map implemented with a parametric quantum circuit, we only need to show how to compute the gradient of Eq.(4), and then it can be imbedded into the classical back propagation process [27]. There are two cases: i) The input A is constant and the derivative against A is not required and ii) The input A is the output of previous steps and the derivative against A is required for the following the backward propagation process. In the second case, we can add a single-qubit layer of R_y into the parametric circuit, whose parameters correspond to the values of A, and then the problem reduces to the first case. Therefore, it is enough to consider the gradient of the function in Eq.(4), which is well-known to be [28]

$$\begin{aligned}\frac{\partial\langle\psi^o|Z^{\otimes N}|\psi^o\rangle}{\partial\theta_j} =& \frac{1}{2}\Big(\langle\psi^i|\mathcal{C}^\dagger(\vec{\theta}_j^+)Z^{\otimes N}\mathcal{C}(\vec{\theta}_j^+)|\psi^i\rangle \\ &-\langle\psi^i|\mathcal{C}^\dagger(\vec{\theta}_j^-)Z^{\otimes N}\mathcal{C}(\vec{\theta}_j^-)|\psi^i\rangle\Big),\end{aligned} \tag{6}$$

where $\vec{\theta}_j^{\pm}$ means to shift the j-th parameter of $\vec{\theta}$, θ_j by $\pm\frac{\pi}{2}$ respectively. Training our hybrid quantum-classical CNNs works in the same way as a regular neural network. Various gradient-based optimization techniques, such as stochastic, batch, or mini-batch gradient descent algorithms, can be used to optimize the parameters of the hybrid quantum-classical CNNs. Once the model has been trained, it can be then used to predict outputs for given inputs.

We demonstrate the potential of our QCCNN by applying it to the Tetris dataset. We create a Tetris image dataset that consists of 1000 grey-scale images with shape 3×3, in which each grey-scale image is a simulated Tetris brick (refer to a Fig. 2 for some samples). Concretely, the foreground pixels are represented by random floating numbers ranging from 0.7 to 1, whereas the background are small floating numbers ranging from 0 to 0.1. There are 5 labels, namely *S*, *O*, *I*, *T* and *L*, each of which represents a type of Tetris bricks. The dataset is further processed by randomly splitting into a training set and a testing set that contain 80% and 20% of the images, respectively. We benchmark our QCCNN against CNN with two particular structures, namely one with a single convolutional layer and another with two convolutional layers. To see the performances with a different number of labels, we create another dataset by only picking the two labels S, T out of the original training and testing data. For the single-layer structure, we use a single (quantum) convolutional layer with 5 filters with no padding, plus a pooling layer also with no padding. For the two-layer structure, we use two (quantum) convolutional layers with 2 and 3 filters respectively, plus a pooling layer with padding 1. The window shape for all the layers is 2×2, and the stride value $s = 1$. Therefore the

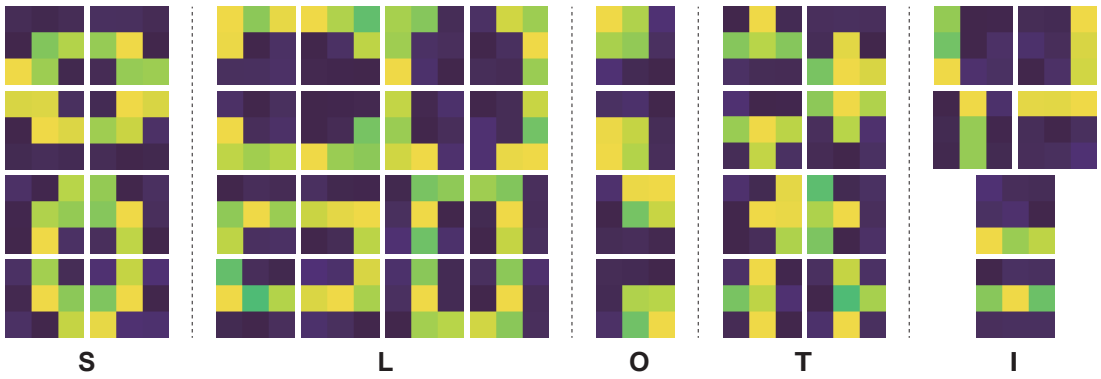

FIG. 2. Some samples of the Tetris dataset. The dataset contains 1000 gray-scale images with shape 3×3. In the dataset, there are five types of Tetris bricks labeled with S, L, O, T, I, which have 8, 16, 4, 8, and 6 possible configurations in the grey-scale images respectively. For each image, the foreground pixels are represented by random floating numbers ranging from 0.7 to 1, whereas the background are small floating numbers ranging from 0 to 0.1.

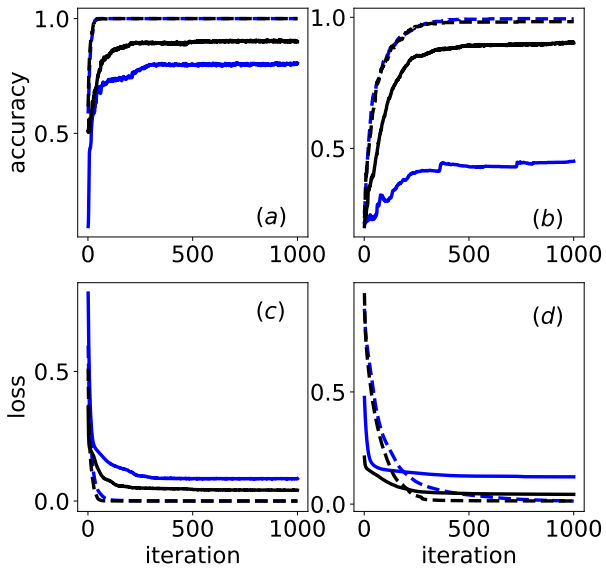

FIG. 3. Accuracy and loss as a function of the number of iterations. In all the figures the blue line represents the result of one-layer CNN and the black line represents the result of two-layer CNN, the blue dashed line represents the result of one-layer QCCNN and the black dashed line represents the result of two-layer QCCNN. We have used the optimizer ADAM [29] and an initial learning rate of 0.01. The results are averaged over 10 random simulations. (a) Accuracy in case of 2 labels. (b) Accuracy in case of 5 labels. (c) Loss in case of 2 labels. (d) Loss in case of 5 labels.

number of qubits fed to the quantum filter is 4, and the the depth of the parametric quantum circuit is set as 4. During 1000 iterations, we compute the accuracy on the the testing data and store the values of the loss function, which is chosen as mean square loss. In Fig. 3(a,c), we plot the accuracy and loss values for the 2-label case. While in Fig. 3(b,d) we plot the accuracy and loss values for the 5-label case. We can see that QCCNN can reach almost 100% accuracy for both the two structures we have used, and it can reach much lower loss values for both cases compared to its classical counterpart. Benefiting from the high-dimensional nature of the quantum system, the advantages of QCCNN become more transparent when the number of labels increases from 2 to 5. We can also see that the 5-label case takes more iterations to converge than the 2-label case, and that QCCNN with a two-layer structure converges faster than the single-layer structure, especially in the 5-label case, which indicates that for complex problems, better performance could be achieved by deeper architectures.

In summary, we present a hybrid quantum-classical Convolutional Neural Network which could be used to solve real world problems with current quantum computers. As a quantum machine learning algorithm inspired by classical CNN, QCCNN keeps the features of CNN such as the nonlinearity, locality of the convolutional layer, as well as extensibility to deep structures. Moreover, the generalized feature map with a parametric quantum circuit is able to explore the correlations of neighbouring data points in a exponentially large linear space, hopefully allowing our algorithm to capture the patterns in the dataset more efficiently or precisely with a quantum computer. We also present methods to exactly compute the gradient of the loss function which could be implemented on a hybrid quantum-classical architecture. We demonstrate our approach on the Tetris dataset and show the potential of our approach to reach better precision for classification problems.

Note added. During the preparation of this work, we notice a similar work which uses qRAM [30], which was carried out independently.

Acknowledgments. The numerical simulation is done by the open source variational quantum circuit simulator VQC [31]. C.G. acknowledges support from National Natural Science Foundation of China under Grants No. 11504430 and No. 11805279. H.-L. H. acknowledges support from the

Open Research Fund from State Key Laboratory of High Performance Computing of China (Grant No. 201901-01), National Natural Science Foundation of China under Grants No. 11905294, and China Postdoctoral Science Foundation.

* guochu@supremacyfuture.com
† quanhhl@ustc.edu.cn

[1] J. Preskill, Quantum **2**, 79 (2018).
[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, Nature **574**, 505 (2019).
[3] A. W. Harrow and A. Montanaro, Nature **549**, 203 (2017).
[4] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nature **549**, 195 (2017).
[5] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New Journal of Physics **18**, 023023 (2016).
[6] M. Benedetti, E. Lloyd, and S. Sack, arXiv:1906.07682 (2019).
[7] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. Obrien, Nature communications **5**, 4213 (2014).
[8] E. Farhi, J. Goldstone, and S. Gutmann, arXiv:1411.4028 (2014).
[9] P.-L. Dallaire-Demers and N. Killoran, Physical Review A **98**, 012324 (2018).
[10] S. Lloyd and C. Weedbrook, Physical Review Letters **121**, 040502 (2018).
[11] M. Schuld and N. Killoran, Physical Review Letters **122**, 040504 (2019).
[12] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Nature **567**, 209 (2019).
[13] K. Fukushima, Biological cybernetics **36**, 193 (1980).
[14] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, in *Shape, contour and grouping in computer vision* (Springer, 1999) pp. 319–345.
[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012) pp. 1097–1105.
[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, International journal of computer vision **115**, 211 (2015).
[17] K. Simonyan and A. Zisserman, arXiv:1409.1556 (2014).
[18] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 770–778.
[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, in *Advances in neural information processing systems* (2014) pp. 2672–2680.
[20] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org, 2017) pp. 933–941.
[21] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org, 2017) pp. 1243–1252.
[22] Y. Zhang and B. Wallace, arXiv:1510.03820 (2015).
[23] A. Kirillov, D. Schlesinger, W. Forkel, A. Zelenin, S. Zheng, P. Torr, and C. Rother, arXiv:1511.05067 (2015).
[24] S. Song, H. Huang, and T. Ruan, Multimedia Tools and Applications **78**, 857 (2019).
[25] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, arXiv:1804.09541 (2018).
[26] I. Cong, S. Choi, and M. D. Lukin, Nature Physics , 1 (2019).
[27] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, Cognitive modeling **5**, 1 (1988).
[28] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Physical Review A **98**, 032309 (2018).
[29] D. P. Kingma and J. Ba, arXiv:1412.6980 (2014).
[30] I. Kerenidis, J. Landman, and A. Prakash, arXiv:1911.01117 (2019).
[31] Available on GitHub at https://github.com/supremacyfuture/VQC.