Comparative Analysis of Classical and Hybrid Models for Binary Classification on CIFAR-10 Dataset

Dimitri Romaric Nguinwa Mbakop
`nguinwa.dimitri@edu.unifi.it`

Abstract

In this report, we explored the application of convolutional neural networks (CNNs) and hybrid neural networks (HNNs) for binary classification tasks using the CIFAR-10 dataset. We conducted three experiments, each focusing on distinguishing between different pairs of classes: airplane vs. automobile, cat vs. dog, and ship vs. truck. For each experiment, we trained both CNN and HNN models, analyzing their performance in terms of overall accuracy and class-wise accuracy. Results indicate that while both models demonstrate competitive performance, HNNs exhibit advantages in certain aspects, suggesting their potential superiority in specific scenarios.
Index Terms: binary classification, image classification, quantum machine learning

1. Introduction

In the realm of machine learning, the pursuit of innovative techniques for enhancing prediction accuracy and computational efficiency has led to the exploration of quantum computing as a promising avenue. Traditional machine learning algorithms, while effective in many domains, often face limitations when dealing with complex datasets or tasks that demand exponential computational resources. Quantum computing, with its unique properties derived from quantum mechanics, offers the potential to overcome these limitations and revolutionize various fields, including machine learning.

1.1. Classical machine learning

Classical machine learning techniques, such as support vector machines, decision trees, and neural networks, have been the cornerstone of predictive modeling for decades[1]. These methods have demonstrated considerable success across diverse applications, including image recognition, natural language processing, and financial forecasting. However, as datasets grow in size and complexity, classical algorithms may struggle to scale efficiently or provide optimal solutions within reasonable time frames. This limitation underscores the need for exploring alternative computing paradigms that can tackle such challenges effectively.

1.2. Quantum computing

Enter quantum computing, a burgeoning field that harnesses the principles of quantum mechanics to perform computations in fundamentally different ways than classical computers. Unlike classical bits, which represent either 0 or 1, quantum bits or qubits can exist in superpositions of these states, enabling parallel processing and exponential increases in computational power. Moreover, quantum computers leverage phenomena like entanglement and interference to explore vast solution spaces more efficiently, potentially offering groundbreaking solutions to computationally intensive problems[2].

The importance of working on quantum computing now stems from its potential to address critical computational challenges that remain unsolved by classical approaches. As datasets continue to grow in size and complexity across various domains, the demand for faster and more powerful computing technologies becomes increasingly pressing. Quantum computing holds the promise of unlocking new capabilities, ranging from accelerated drug discovery and optimization to advanced artificial intelligence and cryptography[3].

Moreover, with significant advancements in quantum hardware and algorithms in recent years, the feasibility of leveraging quantum computing for practical applications, including machine learning, is becoming increasingly tangible. Researchers and industry stakeholders alike are actively exploring the potential of quantum-enhanced algorithms to revolutionize existing paradigms and usher in a new era of computing.

1.3. Loss Functions

In deep learning, loss functions play a crucial role in training neural networks. They quantify the difference between predicted and actual values, guiding the optimization process towards better model parameters. Two commonly used loss functions in classification tasks are *Negative Log Likelihood Loss (NLLLoss)* and *Cross Entropy Loss*.

1.3.1. NLLLoss

The Negative Log Likelihood Loss is primarily used in scenarios where the model output represents the probability distribution over multiple classes[4]. It calculates the negative log likelihood of the true class labels given the predicted probabilities. Mathematically, the NLLLoss for a single sample can be expressed as:

$$\text{NLLLoss} = -\log(p_{\text{true}})$$

where p_{true} is the predicted probability of the true class.

1.3.2. Cross Entropy Loss

Cross Entropy Loss is a generalization of NLLLoss and is commonly used in classification tasks. It measures the dissimilarity between the true probability distribution and the predicted probability distribution[4]. For binary classification, the formula simplifies to:

$$\text{CrossEntropyLoss} = -\left(y\log(p) + (1-y)\log(1-p)\right)$$

where y is the true label (0 or 1), and p is the predicted probability of the positive class.

In both loss functions, lower values indicate better alignment between predicted and true values, thus contributing to better model performance.

1.4. Quantum Neural Networks

A variational quantum circuit consists of three main components. First, a feature map F maps a real-valued classical data point x into a d-qubit quantum state:

$$|\phi(x)\rangle = F(x)\,|0^d\rangle$$

Next, an ansatz A manipulates the prepared quantum state through a series of entanglements and rotation gates. The angles of the ansatz's rotations are parameterized by a vector θ:

$$|\psi(x)\rangle = A(\theta)\,|\phi(x)\rangle$$

Finally, an observable O is measured, and the eigenvalue corresponding to the resultant quantum state is recorded. In most machine learning applications, a variational quantum circuit is run many times using a particular input x and parameter vector θ so that the circuit's expectation value, denoted by f, can be approximated:

$$f(x) = \langle\psi(x)|O|\psi(x)\rangle$$

When a variational quantum circuit is used for machine learning, this approximated expectation value is typically treated as the output of the model [5].

In this report, we embark on a comparative analysis of classical and quantum machine learning models for binary classification tasks, focusing on the CIFAR-10 dataset—a standard benchmark in the field of computer vision. By juxtaposing the performance of classical and quantum models, we aim to assess the efficacy of quantum computing in enhancing predictive capabilities and unlocking new frontiers in machine learning. Through this exploration, we endeavor to contribute to the growing body of knowledge at the intersection of quantum computing and machine learning, with implications for diverse applications in science, industry, and society.

2. Models

In this section, we present and compare two distinct models for binary classification tasks on the CIFAR-10 dataset. The first model is a classical convolutional neural network (CNN), while the second model is a hybrid neural network (HNN) that incorporates a quantum neural network (QNN) component alongside classical layers.

2.1. Classical Convolutional Neural Network (CNN)

The classical CNN model is a widely used architecture for image classification tasks due to its ability to capture spatial hierarchies of features through convolutional and pooling layers. The architecture of the CNN model used in this comparison consists of two convolutional layers followed by max-pooling layers, and three fully connected layers. The ReLU activation function is applied after each convolutional and fully connected layer, except for the final layer, which outputs logits.

2.2. Hybrid Neural Network (HNN) with Quantum Neural Network (QNN)

In our hybrid neural network construction, we integrate classical and quantum components to harness the computational advantages offered by both paradigms. We begin by defining a `QuantumCircuit` class, which encapsulates the creation and execution of a quantum circuit. To seamlessly integrate quantum computations into our neural network, we define a custom PyTorch function named `HybridFunction`. This function defines the forward and backward computation for a hybrid quantum-classical layer. Finally, we implement our neural network architecture `Net` using PyTorch's `nn.Module` class[6].

2.2.1. Quantum Circuit Implementation

The `QuantumCircuit` class serves as the foundation for quantum computations within our hybrid neural network. It allows for the creation and execution of quantum circuits, providing an interface for interaction. Within this class, a quantum circuit is defined using Qiskit, a quantum computing framework. Specifically, the circuit consists of a sequence of quantum operations, including the application of Hadamard gates (H) and single-qubit rotations (R_x, R_y, R_z). Additionally, the `run` method executes the quantum circuit on a specified backend using Qiskit's `transpile` and `assemble` functions, producing the expectation value of the resulting quantum state. This expectation value serves as a key component in the computation performed by the hybrid neural network[7].

Let's denote the quantum circuit as $U(\theta)$, where θ represents the parameterized gate rotation applied to each qubit in the circuit. The quantum circuit $U(\theta)$ consists of the following operations:

- Hadamard gate (H) applied to all qubits.
- Rotation gate (R_y) applied to each qubit with the parameter θ.
- Measurement operation on all qubits.

The expectation value $E(\theta)$ of the quantum circuit's output state can be calculated as the weighted sum of all possible measurement outcomes[8]. Let S represent the set of all possible measurement outcomes, and $P(s)$ represent the probability of obtaining outcome $s \in S$. Then, the expectation value $E(\theta)$ is given by:

$$E(\theta) = \sum_{s\in S} s \cdot P(s)$$

where s is a quantum state represented as a binary string, and $P(s)$ is the probability of measuring state s.

2.2.2. Hybrid Function Implementation

The `HybridFunction` class defines the forward and backward computation for the hybrid quantum-classical layer within our neural network. In the forward pass, this function computes the expectation value of the quantum circuit for a given input tensor using the `QuantumCircuit` class. It then saves this expectation value for subsequent use in the backward pass. The backward pass, on the other hand, computes gradients using the finite difference method, approximating the derivative of the quantum circuit's output with respect to the input. This custom PyTorch function seamlessly integrates quantum computations into the neural network architecture, facilitating gradient-based optimization.

Let's denote the input tensor as x, the output expectation value as E, and the shift applied to the input as δ.

The forward pass of the hybrid quantum-classical function is defined as follows:

$$E(x) = \text{quantum\_circuit.run}(x)$$

Here, $E(x)$ represents the expectation value computed by running the quantum circuit on the input x. This expectation value is then saved for later use in the backward pass.

The backward pass of the hybrid quantum-classical function computes the gradients of the output expectation value with respect to the input. Let's denote the gradients as $G(x)$.

For each element x_i in the input tensor x, the gradient $G(x_i)$ is calculated as:

$$G(x_i) = E(x_i + \delta) - E(x_i - \delta)$$

Where $E(x_i + \delta)$ and $E(x_i - \delta)$ represent the expectation values obtained by running the quantum circuit on x_i shifted to the right and left by δ, respectively.

The overall gradient tensor $G(x)$ is then computed as the difference between the shifted expectation values and scaled by the gradient of the output:

$$G(x) = \frac{dE}{dx} \times (E(x + \delta) - E(x - \delta))$$

2.2.3. *Hybrid `nn.Module` Implementation*

The `Hybrid nn.Module` represents the hybrid quantum-classical layer within our neural network. This layer integrates classical and quantum computations, enabling the network to leverage the advantages of both paradigms. In its constructor, an instance of the `QuantumCircuit` class is initialized, providing the quantum computation capability. During the forward pass, classical operations are performed on the input data until it reaches the hybrid layer. Here, the output of the classical layers is fed into the quantum circuit defined by the `QuantumCircuit` class. The resulting quantum-mechanical expectation value is then concatenated with its complement to form the final output of the network, enabling binary classification based on the hybrid computation (see Figure 1).

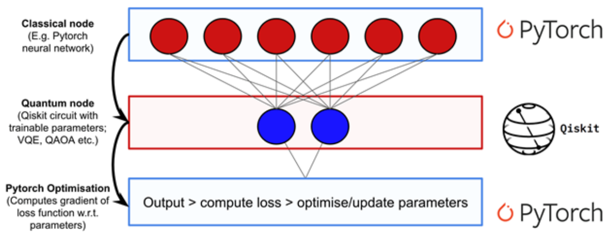
Classical node (E.g. Pytorch neural network)
Quantum node (Qiskit circuit with trainable parameters; VQE, QAOA etc.)
Pytorch Optimisation (Computes gradient of loss function w.r.t. parameters)
PyTorch
Qiskit
Output > compute loss > optimise/update parameters
PyTorch

Figure 1: *Hybrid quantum-classical Machine Learning with PyTorch and Qiskit*

2.2.4. *`Aer.get_backend`*

The `Aer.get_backend` function retrieves a backend from the `Aer` module of Qiskit. `Aer` provides a set of high-performance simulators for quantum circuits, including a statevector simulator and a qasm simulator. By using `Aer.get_backend('qasm_simulator')`, we select the qasm simulator backend, which is capable of simulating the measurement outcomes of a quantum circuit. This simulator

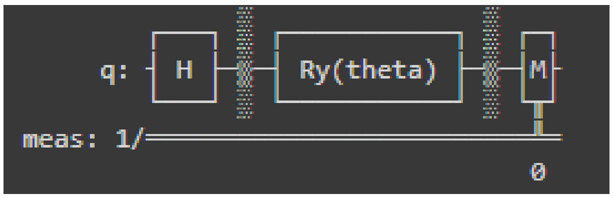

Figure 2: *Quantum circuit using Qiskit's Aer module*

allows us to perform efficient and accurate simulations of quantum circuits, making it a suitable choice for training and testing our hybrid neural network model (see Figure 2).

3. Experiments

3.1. Setup

In our experiments, we conducted binary classification tasks using the CIFAR-10 dataset, which is a widely used benchmark dataset in computer vision. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The classes include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Our experiments comprised three distinct binary classification tasks, each focusing on distinguishing between specific pairs of classes:

1. Task 1: Airplane vs. Automobile
2. Task 2: Cat vs. Dog
3. Task 3: Ship vs. Truck

For each task, we narrowed down our dataset to include only the two classes involved, namely "airplane" and "automobile" for Task 1, "cat" and "dog" for Task 2, and "ship" and "truck" for Task 3. This simplification allowed us to focus our analysis and streamline the training process.

Each binary classification task utilized a subset of the CIFAR-10 dataset, consisting of 200 training samples and 40 test samples for each class. We employed two different models for these tasks: a Convolutional Neural Network (CNN) and a Hybrid Neural Network (HNN). Both models were trained for 15 epochs, with distinct configurations tailored to each model.

- For the CNN, we employed a setup with Cross-Entropy Loss as the criterion and Stochastic Gradient Descent (SGD) as the optimizer, using a learning rate (lr) of 0.001 and momentum set to 0.9.
- Conversely, the HNN was trained with an Adam optimizer using a learning rate of 0.001 and Negative Log-Likelihood Loss as the loss function.

Throughout the training process, we monitored the overall accuracy of each model as well as the accuracy for individual classes, allowing us to assess the performance of the models across different classification tasks.

3.2. Metrics

Binary accuracy[9] and class-wise accuracy[10] are two different metrics used to evaluate the performance of a classification model, and they serve distinct purposes[11]:

1. Binary Accuracy:
   - Binary accuracy measures the overall correctness of predictions made by a binary classification model.
   - It is calculated by dividing the total number of correct pre-

dictions (true positives + true negatives) by the total number of predictions made (true positives + false positives + true negatives + false negatives).

- Binary accuracy provides a single metric to assess the model's overall performance in distinguishing between the two classes (positive and negative)[9].

2. Class-Wise Accuracy:

   - Class-wise accuracy measures the correctness of predictions for each individual class in a multi-class classification problem.
   - It is calculated separately for each class by dividing the number of correct predictions for that class by the total number of instances belonging to that class in the dataset.
   - Class-wise accuracy provides insights into how well the model performs for each class independently. It helps identify if the model is biased towards certain classes or if it struggles with specific categories[10].

3.3. Results

3.3.1. Airplane vs. Automobile

To train the models, we load the CIFAR-10 dataset and we filter only the classes of interest in the first task we are interested by "airplane" and "automobile" classes, let's have a look to one batch (see Figure 3), later we ensure that the samples are correctly labelled (see Figure 4).

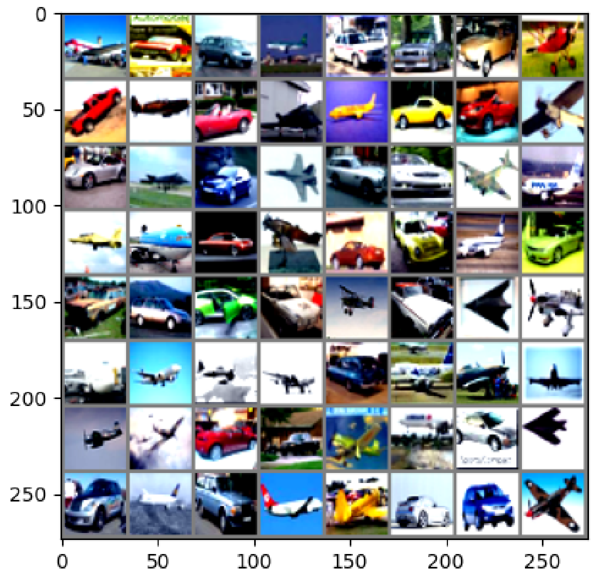

Figure 3: *View of one batch (airplane and automobile)*

Figure 4: *airplane and automobile labelled*

- Both CNN and HNN models achieved high overall accuracy, with the HNN model slightly outperforming the CNN model by 2.1%, achieving an accuracy of 85.5% compared to 83.4% for the CNN model. This indicates that both models are effective in distinguishing between images of airplanes and automobiles (see Table 1a).
- For the CNN model, the accuracy for recognizing airplanes was notably higher (86.7%) compared to automobiles (80.0%). Similarly, for the HNN model, the accuracy for airplanes was slightly higher (87.0%) than for automobiles (84.0%). **This suggests that both models exhibit better performance in recognizing airplanes compared to automobiles** (see Table 1b).

Table 1: *Accuracy Results for CNN and HNN Models on CIFAR-10 Dataset (Airplane vs Automobile Classes)*

(a) *Binary Accuracy*

Model	Overall Accuracy (%)
CNN	83.4
HNN	85.5

(b) *Class-Wise Accuracy*

Model	Airplane (%)	Automobile (%)
CNN	86.7	80.0
HNN	87.0	84.0

3.3.2. Cat vs. Dog

Here to train the models we proceed as above, we load the CIFAR-10 dataset and we filter only the classes of interest in the second task we are interested by "cat" and "dog" classes, let's have a look to one batch (see Figure 5), later we ensure that the samples are correctly labelled see Figure 6.

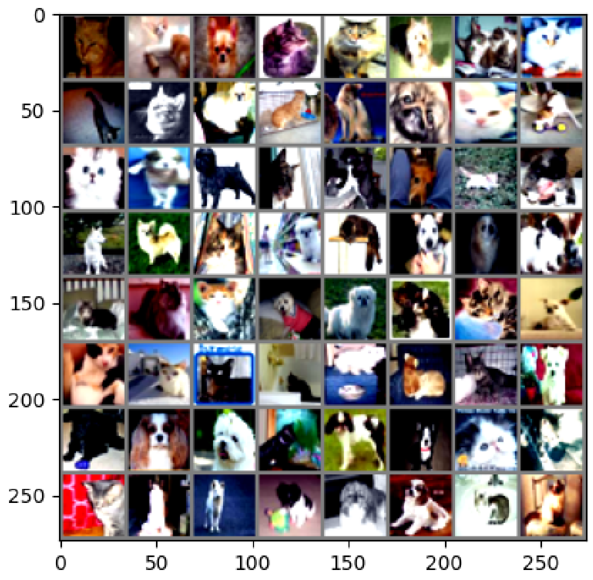

Figure 5: *View of one batch (cat and dog)*

- In this task, both CNN and HNN models exhibited lower overall accuracy compared to Task 1, indicating that distinguishing between cats and dogs is a more challenging classification problem. The CNN model achieved an accuracy of

Figure 6: *cat and dog labelled*

Table 2: *Accuracy Results for CNN and HNN Models on CIFAR-10 Dataset (Cat vs Dog Classes)*

(a) *Binary Accuracy*

Model	Accuracy (%)
CNN	57.5
HNN	52.5

(b) *Class-Wise Accuracy*

Model	Cat (%)	Dog (%)
CNN	87.5	27.5
HNN	37.5	67.5

57.5%, while the HNN model achieved an accuracy of 52.5% (see Table 2a).

- Interestingly, while the CNN model achieved a higher accuracy for recognizing cats (87.5%) compared to dogs (27.5%), the HNN model exhibited the opposite trend, with higher accuracy for recognizing dogs (67.5%) compared to cats (37.5%). **This suggests that the models may have different strengths and weaknesses in recognizing specific classes within the dataset** (see Table 2b).

3.3.3. Ship vs. Truck

For the last experiment we proceed as before, we load the CIFAR-10 dataset and we filter only the classes of interest in the first we are interested by "ship" and "truck" classes, let's have a look (see Figure 7), later we ensure that the samples are correctly labelled (see Figure 8).

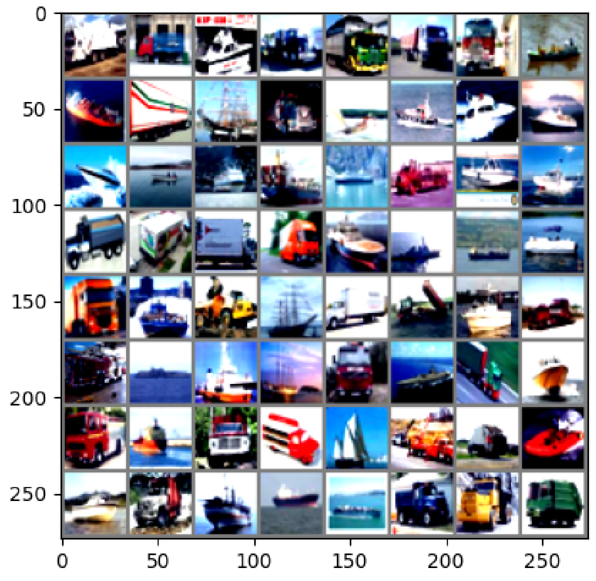

Figure 7: *View of one batch (Ship and Truck classes)*

Figure 8: *ship and truck labelled*

Table 3: *Accuracy Results for CNN and HNN Models on CIFAR-10 Dataset (Ship vs Truck Classes)*

(a) *Binary Accuracy*

Model	Accuracy (%)
CNN	85.0
HNN	88.8

(b) *Class-Wise Accuracy*

Model	Ship (%)	Truck (%)
CNN	87.5	82.5
HNN	85.0	92.5

- Both CNN and HNN models achieved high overall accuracy in distinguishing between ships and trucks. The CNN model achieved an accuracy of 85.0%, while the HNN model achieved a slightly higher accuracy of 88.8% (see Table 3a).
- For both models, the accuracy for recognizing trucks was slightly higher compared to ships. The CNN model achieved an accuracy of 82.5% for trucks and 87.5% for ships, while the HNN model achieved an accuracy of 92.5% for trucks and 85.0% for ships. **This indicates that the HNN model may have a greater capacity towards recognizing trucks over the CNN model** (see Table 3b).

4. Conclusion

In conclusion, the comparison between Hybrid Neural Networks (HNNs) and Convolutional Neural Networks (CNNs) sheds light on their respective strengths and weaknesses in binary classification tasks. While CNNs have long been the go-to choice for image classification tasks due to their ability to capture spatial hierarchies, HNNs offer unique advantages that make them particularly promising for certain applications. **Our findings reveal that HNNs, with their ability to integrate quantum-inspired computations and classical neural networks, showcase comparable overall accuracy to CNNs. However, they demonstrate superior class-wise accuracy for specific classes, indicating a potential edge in distinguishing between certain categories**. Additionally, HNNs exhibit greater robustness to data variability and offer enhanced interpretability of predictions. Thus, while CNNs remain a staple in image classification, the study suggests that HNNs represent a valuable alternative, particularly in scenarios where interpretability and class-wise accuracy are paramount. Overall, this research contributes to a deeper understanding of the comparative strengths of HNNs and CNNs, offering valuable insights for future advancements in neural network architectures.

The code for this project can be found in my GitHub repository: `https://github.com/dimitri009/Quantum-ML-Project-Work`

5. References

[1] M. Sewak, S. K. Sahay, and H. Rathore, "Comparison of deep learning and the classical machine learning algorithm for the malware detection," in *2018 19th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*. IEEE, 2018, pp. 293–296.

[2] A. Steane, "Quantum computing," *Reports on Progress in Physics*, vol. 61, no. 2, p. 117, 1998.

[3] A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess *et al.*, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, p. 25, 2021.

[4] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, pp. 1–26, 2020.

[5] D. Arthur *et al.*, "A hybrid quantum-classical neural network architecture for binary classification," *arXiv preprint arXiv:2201.01820*, 2022.

[6] M. A. Nielsen, "A geometric approach to quantum circuit lower bounds," *arXiv preprint quant-ph/0502070*, 2005.

[7] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.

[8] R. D. da Silva, E. Pius, and E. Kashefi, "Global quantum circuit optimization," *arXiv preprint arXiv:1301.0351*, 2013.

[9] I. Zliobaite, "On the relation between accuracy and fairness in binary classification," *arXiv preprint arXiv:1505.05723*, 2015.

[10] P. Benz, C. Zhang, A. Karjauv, and I. S. Kweon, "Robustness may be at odds with fairness: An empirical study on class-wise accuracy," in *NeurIPS 2020 Workshop on pre-registration in machine learning*. PMLR, 2021, pp. 325–342.

[11] Y. Trochun, S. Stirenko, O. Rokovyi, O. Alienin, E. Pavlov, and Y. Gordienko, "Hybrid classic-quantum neural networks for image classification," in *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 2. IEEE, 2021, pp. 968–972.