Team Number:	apmcmlz2400169

2024 APMCM Wuyue Cup

Feature Selection and Image Classification Model Optimization Based on QUBO Model

Summary

Background

This study addresses two pivotal challenges in machine learning tasks: feature selection and hyperparameter optimization. By integrating quantum optimization techniques, an efficient solution is proposed to tackle these issues. As data complexity continues to grow, identifying the optimal feature subset from high-dimensional datasets and fine-tuning model parameters have become critical barriers to achieving optimal machine learning performance. Traditional optimization methods, such as exhaustive search or random search, often suffer from high computational costs and limited scalability. Quantum optimization, particularly approaches based on Quadratic Unconstrained Binary Optimization (QUBO) models, leverages the unique properties of quantum mechanics, such as superposition and quantum tunneling, to efficiently explore complex solution spaces. This offers a novel pathway for enhancing machine learning model performance.

Addressing Feature Selection in Problem 1

For the feature selection task, the study employs the German Credit Scoring dataset as a testbed. The feature selection problem is formulated as a QUBO optimization task, solved using the Simulated Annealing algorithm. This approach identifies an optimal subset of features, which is subsequently used to train a Random Forest model. By applying this optimized feature subset, the study achieves a balance between reduced model complexity and high classification accuracy. Experimental results demonstrate that the optimized model achieves accuracy ranging from 77% to 84%. This underscores the method's efficacy and applicability, particularly for mixed-type datasets comprising categorical and numerical features.

Addressing Hyperparameter Optimization in Problem 2

To tackle the hyperparameter optimization challenge, the study focuses on the Fashion

MNIST dataset, using the QUBO model to optimize key hyperparameters of a Convolutional Neural Network (CNN). These hyperparameters include learning rate, dropout ratio, batch size, and the number of convolutional kernels. By employing the Quantum Annealing algorithm, the study rapidly searches for the globally optimal parameter combinations.

The optimized CNN model achieves a classification accuracy of 90.26% on the Fashion MNIST dataset. This result not only significantly outperforms traditional hyperparameter optimization methods, such as grid search and random search, but also validates the potential and advantages of quantum optimization in deep learning tasks.

Implications and Future Directions

This research highlights the practical value of quantum optimization in machine learning, providing an efficient solution to feature selection and hyperparameter optimization. Furthermore, it opens new avenues for the deeper integration of quantum computing and machine learning. As quantum computing hardware continues to advance, the proposed method holds promise for application to higher-dimensional and more complex datasets in diverse domains, such as financial risk assessment, medical data analysis, supply chain optimization, and natural language processing. Future work could explore multi-objective optimization or the incorporation of additional constraints to further enhance model performance and better meet real-world application needs. By synergizing with the rapid evolution of deep learning models, these advancements could push the boundaries of what is achievable in data-driven optimization.

Conclusion

This study provides a scalable and intelligent solution to high-dimensional data analysis and optimization problems, demonstrating significant theoretical and practical value. By bridging quantum optimization and machine learning, it lays a robust foundation for future exploration in this transformative interdisciplinary field.

Keywords: Feature Selection, QUBO (Quadratic Unconstrained Binary Optimization) model, Random Forest Model, CNN (Convolutional Neural Network), Hyperparameter Optimization, Quantum Annealing.

Contents

1 Introduction

1.1 Problem Background

Quantum computing has demonstrated significant potential in addressing complex problems and efficiently processing large-scale datasets, offering capabilities that surpass those of classical computing systems. When synergistically integrated with artificial intelligence (AI), quantum computing has the capacity to drive transformative advancements and revolutionary innovations.

Overall, the introduction of the QUBO model helps reduce the complexity of model training, optimize computational approaches, and improve both the performance and accuracy of models. Quantum computing's optimization of traditional model training significantly accelerates AI development, showcasing immense potential for transformative advancements in the field.

1.2 Restatement of the Problem

Considering the background information and restricted conditions identified in the problem statement, we need to solve the following problems:

- **Problem 1**
  Employs QUBO modeling to address the feature selection problem in the German credit scoring dataset, whose objective is to identify an optimal subset of features that minimizes the total number of selected features while simultaneously maximizing the classification performance.

- **Problem 2**
  Choose a specific dataset and develop a suitable deep learning model with an appropriate architecture. Subsequently, convert the associated optimization problem into a QUBO formulation and solve it using the simulated annealing algorithm available in the Kaiwu SDK.

1.3 Our Work

- **Problem 1**
  We propose an efficient feature selection and classification approach that integrates quantum optimization and machine learning, applied to the modeling task of the German credit scoring dataset. By formulating the feature selection problem as a QUBO (Quadratic Unconstrained Binary Optimization) model and solving it using the simulated annealing algorithm provided by the Kaiwu SDK, an optimal subset of features is selected from high-dimensional data. The selected feature subset is then trained and evaluated using a random forest classification model to assess its performance.

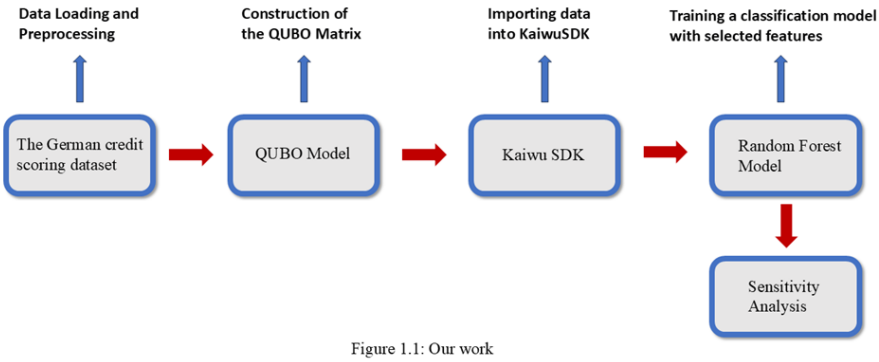

Figure 1.1: Our work

- **Problem 2**

  To address the second problem, we selected a classic image classification dataset, Fashion MNIST. A baseline classification was performed using a CNN model, which involves four hyperparameters that influence the model's classification accuracy and results. Considering that classification accuracy varies across datasets under different hyperparameter combinations, we first constructed a QUBO model to identify the hyperparameter combination yielding the highest accuracy for this model. The optimal hyperparameter combination was then incorporated into the CNN model, forming the QUBO-CNN model. Subsequently, the dataset was used to train the model, and its accuracy was evaluated.

  By comparing the QUBO-CNN model with models constructed using other hyperparameter selection methods, we observed significant improvements in both accuracy and training speed. This training process leveraged the advantages of quantum computing to enhance the image classification capability of the deep learning model.

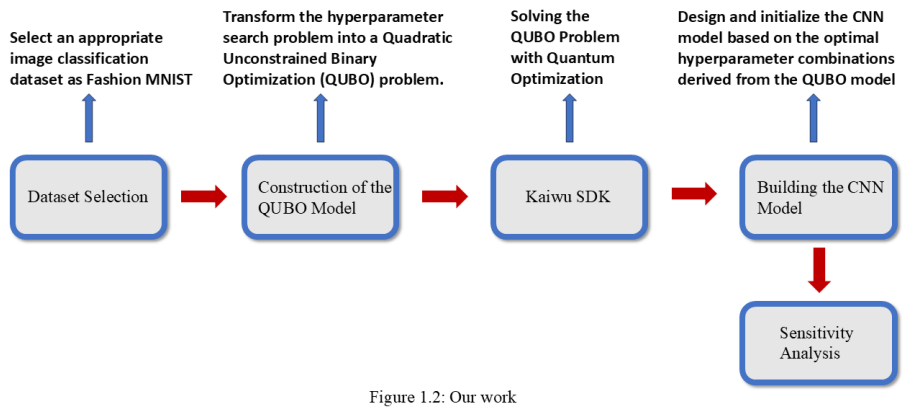

Figure 1.2: Our work

2 Assumptions and Justifications

- **Data Quality Assumption:**

The input dataset has been properly cleaned and preprocessed, containing no significant missing values or outliers.
All features (columns) in the dataset contribute meaningfully to explaining the target variable (label).

- **Data Distribution Assumption:**
  The sample size of the dataset is sufficiently large to support the training and testing of the random forest model.
- **QUBO Matrix Assumption:**
  The QUBO matrix is correctly constructed, with the weights of the classification error term (alpha) and feature count penalty term (beta) reasonably set.
  The solution provided by the Kaiwu SDK accurately reflects the optimization results of the QUBO matrix, and the solution vector selects an optimal or near-optimal subset of features.
- **Model Applicability Assumption:**
  The random forest classifier is suitable for the data after feature selection and can capture the significant nonlinear relationships in the data.

3 Random Forest Model Based on QUBO Model Optimized Feature Selection for Problem 1.

3.1 Data Description

The German Credit Scoring dataset encompasses 20 attributes, consisting of 7 numerical features and 13 categorical features, utilized for the assessment of individual credit risk, categorized as 'good' or 'bad'. Below is a concise description of the dataset:

Numerical Features:

- **Duration in month**: The duration of the loan in months.
- **Credit amount**: The amount of the loan.
- **Installment rate**: The percentage of the installment payment relative to disposable income.
- **Present residence since**: The number of years at the current residence.
- **Age in years**: The age of the individual.
- **Number of existing credits**: The quantity of credit accounts currently held.
- **Number of people being liable**: The number of individuals for whom the individual is financially responsible.

Categorical Features:

The dataset includes categorical features such as account status, credit history, purpose of the loan, savings account, and occupation type. These features are typically represented by qualitative labels (e.g., A11 signifies an account status of less than 0 DM).

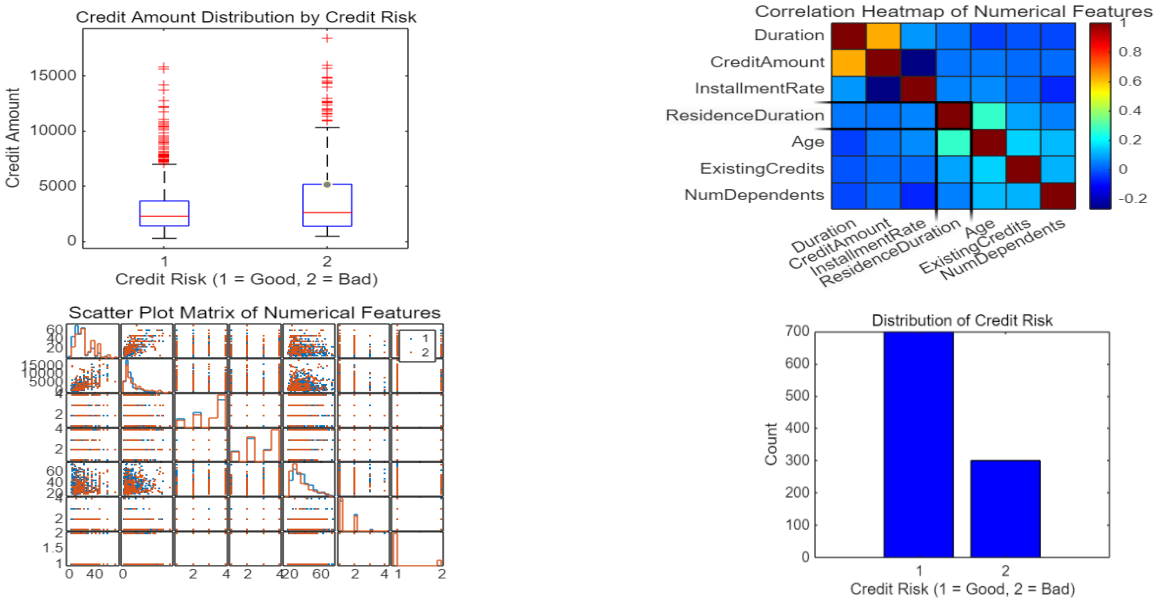

Figure 3.1

3.2 Construction of Model

The German credit scoring dataset is a well-known dataset in artificial intelligence. It includes features that describe various characteristics of individuals, as well as their credit status (good or bad). This dataset is ideal for evaluating feature selection techniques. Quantum computing provides a novel approach to feature selection. By transforming the problem into a Quadratic Unconstrained Binary Optimization (QUBO) form, quantum computing or classical techniques like simulated annealing can accelerate the solution process. By selecting the most relevant features, the performance of predictive models can be improved, and model complexity and the risk of overfitting can be reduced.

Formulate the QUBO Model

We formulate the feature selection problem as a QUBO model with the following objective function:

$$\min_{\mathbf{x}} \quad \alpha \cdot \mathbf{x}^T \mathbf{C}\mathbf{x} + \beta \cdot \| \mathbf{x} \|_1 \tag{1}$$

Where:

$x \in \{0,1\}^n$: Binary decision variables indicating whether a feature is selected.

C: Symmetric matrix encoding feature relevance and classification performance, derived from feature-label correlations.

$\| x \|_1 = \sum_{i=1}^{n} |x_i|$: Penalty term for the number of selected features.

To achieve the goal of minimizing the number of features while maximizing classification accuracy, we incorporated a combination of classification performance

weight α and feature quantity weight β in the construction of the QUBO model.

α: Weight for classification performance

β: Weight for sparsity, encouraging fewer features to be selected.

The classification error matrix $\mathbf{C}$ is defined as:

$$C = |\text{corr}(X,y)| \cdot |\text{corr}(X,y)|^T + \lambda \tag{2}$$

Train a Classification Model

After identifying the optimal subset of features:

1. Split the data into training and testing sets.
2. Train a Random Forest (RF) classifier using the selected features. RF is chosen due to its robustness and ability to handle both categorical and numerical features.
3. Evaluate the model using metrics such as accuracy and confusion matrix.

The Random Forest model is defined as:

$$f(x) = \text{majority\_vote}(T_1(x), T_2(x), ..., T_M(x)) \tag{3}$$

Where $\boldsymbol{Ti(x)}$ are the decision trees in the forest.

Additionally, we compute the Out-of-Bag (OOB) error:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\hat{y}_i^{\text{OOB}} \neq y_i) \tag{4}$$

3.3 The Model Solution Formula

In the aforementioned model construction process, we selected the Random Forest model as the primary classification model and transformed the German credit scoring dataset into a QUBO matrix to identify the optimal and most discriminative features for model training. Subsequently, we will delve into the model-solving process in greater detail through an in-depth analysis of the corresponding code implementation.

Construction and solving of the QUBO matrix

In general, solving QUBO matrices often involves directly employing traditional simulated annealing methods for analysis. However, traditional algorithms are characterized by slow convergence speeds, difficulties in parameter tuning, and a tendency to get trapped in local optima, which makes them less advantageous for solving large-scale feature classification problems. Therefore, to improve model scalability and efficiency, we adopt the following steps:

First, the German credit scoring dataset is transformed into a 20-dimensional QUBO ma-

trix using the objective function defined in Section 4.2. Each element a_{ij} represents the correlation between different features. By leveraging actual data and grid search methods (informed by cloud data analysis), we determine suitable combinations of classification performance weight α and feature quantity weight β, which are then incorporated into the code. This approach ensures sensitivity and aims to achieve the highest classification accuracy with the minimal number of features, ultimately forming the QUBO matrix.

Using Simulated Annealing, the goal is to find an optimal solution $x*$ such that:

$$x^* = \arg\min_x (\alpha \cdot x^T Cx + \beta \cdot \| x \|_1) \tag{5}$$

The matrix form of the objective function can be expanded as:

$$\min_{\mathbf{x}} \left(\alpha \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_i x_j + \beta \cdot \sum_{i=1}^{n} |x_i| \right) \tag{6}$$

Where:

C_{ij}: Element in the i , j-th position of the matrix C

$x_i, x_j \in \{0, 1\}$: Decision variables.

2. Secondly, the generated QUBO matrix is first stored in a CSV file and then uploaded directly to the Kaiwu SDK cloud computing platform to compute the solution vector. The computation resulted in nine local optima under different conditions, with a computation speed of 1.12 ms. Based on the requirement to use as few features as possible for training, we selected the solution vector with the minimal number of "1"s (i.e., the solution with the fewest selected features). This solution was saved in the designated file qubo_solution.log.

Set the initial solution x_0 and initial temperature T_0. For the current solution x_k, compute the objective function value:

$$F(x^k) = \alpha \cdot (x^k)^T Cx^k + \beta \cdot \| x^k \|_1 \tag{7}$$

Use random perturbations or other optimization strategies to find a better solution x_{k+1}. Stop optimization when convergence conditions are met (e.g., temperature is sufficiently low or the objective function value stabilizes), and return the optimal solution x^*.

The optimal solution is expressed as:

$$x^* = \arg \min_{x \in \{0,1\}^n} (\alpha \cdot x^T Cx + \beta \cdot \| x \|_1) \tag{8}$$

The resulting solution vector has a dimensional structure of [1:20], where the selected

features correspond to the decision variables. At this point, the construction and of the QUBO matrix are complete.

Training the Random Forest Model with Features Selected by QUBO

After solving the QUBO model and obtaining the optimal subset of features by Kaiwu SDK, these selected features can be used to train a Random Forest (RF) model.

Random Forest (RF) is an ensemble learning method that combines multiple decision trees. The goal is to minimize the classification error by leveraging the features selected by the QUBO model. The RF classification model can be represented as:

$$f(x) = \text{majority\_vote}(T_1(x), T_2(x), ..., T_M(x)) \tag{9}$$

Where:

$T_i(x)$: The i-th decision tree in the ensemble.

M: The total number of decision trees.

$\text{majority\_vote}(\cdot)$: Aggregates predictions from all trees to determine the final class label.

Let x^* represent the optimal solution obtained from the QUBO model. The selected features are:

$$X_{\text{selected}} = X[:, x^*] \tag{10}$$

Where:

X: The original feature matrix.

X_{selected}: The subset of features where $x^* = 1$

The training objective of the RF model is to minimize the misclassification error on the training set:

$$L_{\text{RF}} = \frac{1}{N} \sum_{i=1}^{N} I(f(x_i) \neq y_i) \tag{11}$$

During training, Random Forest uses bootstrap sampling, leaving some data samples out of the training process (out-of-bag samples). These samples are used to estimate the OOB error:

$$\text{OOB Error} = \frac{1}{N} \sum_{i=1}^{N} I(\hat{y}_i^{\text{OOB}} \neq y_i) \tag{12}$$

Model Training Process

The selected solution vector is $Solution$ [1,1,1,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0,0,0], indicating that ten features, which have the most significant impact on the classification of this dataset, were selected (derived using the quantum annealing algorithm). This effectively reduced the model complexity by 50%, helping to eliminate redundant and extraneous data and minimizing interference. Subsequently, the following code is used to split the dataset into training and testing sets in an 8:2 ratio (Figure 4.3.1).

Next, the following code (Figure 4.3.2) is used to iteratively train a Random Forest model with the selected features, using the default parameter $numtree = 100$. After the training is completed, the remaining 20% of the test set is used to evaluate the model's performance, yielding the final classification accuracy.

```
% 划分训练集和测试集
cvFinal = cvpartition(y, 'HoldOut', 0.2);
trainIdx = training(cvFinal);
testIdx = test(cvFinal);

trainX = X_selected(trainIdx, :);
trainY = y(trainIdx);
testX = X_selected(testIdx, :);
testY = y(testIdx);
```

Figure 3.3.1

```
% 训练随机森林模型并记录训练过程
disp('训练随机森林模型...');
numTrees = 100;
templateTreeObj = templateTree();
finalModel = fitcensemble(trainX, trainY, 'Method', 'Bag', ...
    'NumLearningCycles', numTrees, ...
    'Learners', templateTreeObj, ...
    'ClassNames', unique(trainY));
```

Figure 3.3.2

In the prediction section, we use the trained RF model to predict labels for the testing set:

$$\hat{y} = f(X_{\text{test, selected}}) \tag{13}$$

In the model evaluation section, we compute classification accuracy by:

$$\text{Accuracy} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} I(\hat{y}_i = y_i) \tag{14}$$

Here is a sample output from one of the experiments (classification accuracy across multiple experiments ranges between 77% and 84%):

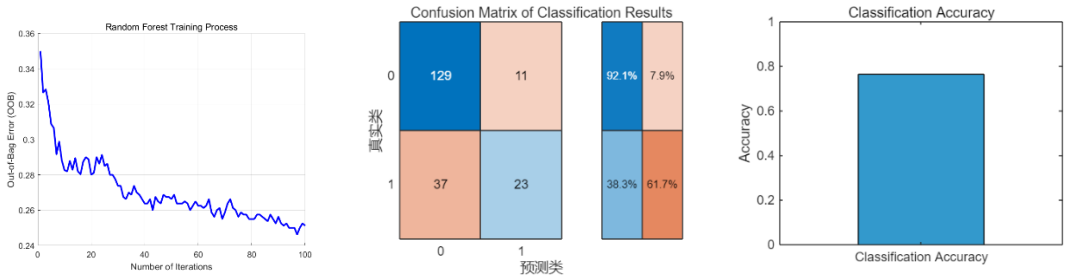

Figure 3.3.3

This modeling process combines the advantages of quantum optimization and machine learning by selecting the optimal feature subset through the QUBO model and achieving efficient and accurate classification using the Random Forest classification model.

4 CNN Model with Hyperparameter Optimization Based on the QUBO Model for Problem 2

4.1 Data Description

The Fashion MNIST dataset is a widely used benchmark for machine learning and deep learning models, consisting of grayscale images of fashion items. The dataset contains 70,000 images in total. It is divided into 60,000 training samples and 10,000 testing samples. Each image is a 28×28 pixel grayscale image, resulting in 784 features per image when flattened. There are 10 classes, each representing a type of clothing or accessory. The classes are: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.

4.2 Construction of Model

Firstly, our fundamental approach involves utilizing the QUBO model to optimize the hyperparameters of a CNN-based deep learning model, thereby improving training efficiency and classification accuracy. Furthermore, the optimization performance of the best hyperparameter combination obtained through solving the QUBO matrix will be compared with other commonly used methods, such as GridSearch and RandomizedSearchCV. Accordingly, it is necessary to develop a corresponding CNN model and a QUBO model tailored to this dataset (using Python programming).

Construction of CNN Image Classification Model:

A complete analysis of constructing a CNN model involves the following stages: First, the input layer is utilized to ingest the image dataset. The convolutional layers are responsible for extracting graphical features, which are then mapped to classification probabilities using predefined weight matrices in the fully connected layers. These probabilities are subsequently passed to the classification layer, where they are transformed into a probability distribution to perform classification.

The ReLU activation function is employed to introduce nonlinearity, enhancing the model's ability to process sparse features in the image data. The linear transformation outputs from the fully connected layers are subjected to the ReLU function to compute activation values. These activation values are then processed in the pooling layer to reduce the resolution of the feature maps. This pooling step autonomously extracts the most significant features, reducing both the dimensionality and computational complexity of the data.

The processed activation values are further input into a Dropout layer to prevent overfitting and subjected to batch normalization to mitigate gradient anomalies. Finally, the use of loss functions and optimizers facilitates parameter optimization during training, iteratively updating the model's parameters to minimize loss. These interconnected processes ensure the efficient training and accurate performance of the CNN model.

1. Construction of the Input Layer

The following function constructs the input layer:

$$X \in \mathbb{R}^{28\times 28} \tag{15}$$

Where:

X: Represents the input image, with its size corresponding to the resolution. For the Fashion MNIST dataset, the size is 28×28.

In the Fashion MNIST dataset, each pixel value is a grayscale intensity ranging from 0 to 255.

Normalization Process：

$$X' = \frac{X}{255} \tag{16}$$

The range of pixel values after normalization is [0, 1].

.

2. Construction of the Convolutional Layer

The following function defines the construction of the convolutional layer:

$$h_{i,j,k} = \sigma\left(\sum_{m,n} W_{m,n,k} \cdot x_{(i+m),(j+n)} + b_k\right) \tag{17}$$

Where:

$h_{i,j,k}$: Activation of the feature map at position (i,j) in the k-th channel.

(Used to construct the next feature map, representing the extracted feature patterns.)

$W_{m,n,k}$: Convolutional filter (kernel) values for the k-th feature map.

(The convolutional kernel, also known as the weight matrix.)

$x_{(i+m),(j+n)}$: Input pixel values in the local receptive field.

b_k: Bias term for the k-th feature map.

σ: Activation function, such as ReLU.

Extract local features from images in the Fashion MNIST dataset to generate multiple feature maps. Given an input size of 28×28, the size of the resulting feature maps is calculated as:

$$\text{Feature Map Size} = \frac{\text{Input Size} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} + 1 \tag{18}$$

The size of the convolutional kernel and other parameters will be optimized subsequently using the QUBO matrix. The feature maps are derived from the Fashion MNIST dataset.

3. Batch Normalization

Batch normalization is applied to the channel dimensions of the feature maps generated from the aforementioned operations. Since the input image data is processed through fully connected layers combined with activation functions to produce activation values, normalization is performed for each channel's activation values by calculating the mean and variance across all pixels within the batch.

The modeling formula is as follows:

$$\hat{h} = \frac{h - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{19}$$

$$h' = \gamma \hat{h} + \beta \tag{20}$$

Where:

μ: Mean of the current batch.

σ^2: Variance of the current batch.

γ, β: Learnable parameters.

h': Final activation value.

Function:

1. Standardizes activation values to prevent gradient explosion or vanishing.
2. Accelerates convergence and enhances model stability.

4. Activation Function

It is used to convert the output of the fully connected layer into activation values. The formula is as follows:

$$\sigma(x) = \max(0, x) \tag{21}$$

Introduces nonlinear analytical capabilities, effectively handling the sparse features of images in the Fashion MNIST dataset.

5. Pooling Layer

To simplify the model, pooling is used to extract the maximum activation value within a sliding window, thereby capturing the primary features while reducing noise and computational load. The intuitive effect is reflected in the reduction of feature map dimensions, enhancing the

model's capability to handle dimensionality reduction for higher-resolution images.

The formula is as follows:

$$h_{i,j,k}^{\text{pool}} = \max_{m,n}(h_{(i+m),(j+n),k}) \tag{22}$$

6. Dropout Processor

$$h_i' = \begin{cases} h_i & \text{with probability } 1-p \\ 0 & \text{with probability } p \end{cases} \tag{23}$$

Where:

H_i**:** Input activation value.

p: Dropout rate.

7. Fully Connected Layer

For each layer of the neural network, the input x undergoes a linear transformation through the convolutional kernel (weight matrix) W and bias b:

$$y = \text{softmax}(Wx + b) \tag{24}$$

Where:

X**:** Input features or activation values from the previous layer (typically a flattened feature map).

W**:** Weight matrix, representing the connection strength between the inputs and the neurons.

b**:** Bias vector, adjusting the result of the linear transformation.

The output of the fully connected layer is used in the activation function to compute activation values.

8. The categorical cross-entropy loss function for classification is defined as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log\hat{y}_{i,c} \tag{25}$$

Function**:** Calculates the difference between predicted values and true values, guiding the dynamic optimization of parameters.

9. The weight update in Stochastic Gradient Descent (SGD) is:

$$W \leftarrow W - \eta \nabla_W L \tag{26}$$

Where:

η: Learning Rate

Function**:** Optimizes the model parameters using gradient descent to minimize the loss function.

At this point, the basic structure of the CNN deep learning model has been established. The next step involves using the QUBO model to optimize the CNN in the context of the specific dataset. The objective is to perform a combinatorial search over four hyperparameters associated with the CNN model: learning rate (η), dropout rate (p), the number of convolutional kernels, and batch size. The goal is to identify the combination that achieves the highest classification accuracy and optimal performance. The resulting hyperparameter configuration will then be integrated into the previously constructed base model for further optimization.

Formulate the QUBO Model

First, the objective function addressed by the QUBO model is defined as:

$$f(x) = x^T Q x + c^T x \tag{27}$$

Where:

X**:** Binary decision variable representing the selection or non-selection of hyperparameters. If $x = 1$, the hyperparameter is selected; otherwise, it is not.

Q**:** Symmetric matrix, also known as the QUBO matrix, representing the quadratic relationships between hyperparameters. The element at position (i,j) reflects the cooperative or conflicting effects between candidate hyperparameter values x_i and x_j. The diagonal elements (linear penalties) and off-diagonal elements (interaction penalties) are initialized to construct the matrix.

c**:** Linear weight vector reflecting the cost or reward of selecting each individual hyperparameter value.

$$f(x) = \sum_i c_i x_i + \sum_{i \neq j} Q_{ij} x_i x_j \tag{28}$$

Where:

The linear term represents the penalty or reward for selecting a specific hyperparameter value.

The quadratic term accounts for the interactions between different hyperparameter values and applies penalties accordingly.

To simplify computation and ensure the efficient solution of the QUBO matrix, we select four hyperparameters, each with commonly used default values, as candidate hyperparameter values for constructing the QUBO matrix, as shown below:

Learning Rate

The learning rate controls the step size of the gradient descent process. The optimization objective is to penalize selections that deviate from the expected value (e.g., 0.01). The candidate values for the learning rate are: $[0.001, 0.01, 0.1]$.

Dropout Rate

The dropout rate is used for regularization to prevent overfitting. The optimization objective is to favor the selection of a moderate dropout rate (e.g. 0.3) to balance model complexity and regularization effectiveness. The candidate values for the dropout rate are: $[0.1, 0.3, 0.5]$.

Batch Size

The batch size refers to the number of samples used in each training iteration. The optimization objective is to balance computational efficiency and training stability. The candidate values for the batch size are: $[32, 64, 128]$.

Number of Convolutional Kernels

The number of convolutional kernels determines the number of features extracted in each convolutional layer. The optimization objective is to enhance feature extraction capabilities while controlling model complexity. The candidate values for the number of convolutional kernels are: $[16, 32, 64]$.

```
59 # Define hyperparameter candidate values
60 print('Defining hyperparameter candidate values...')
61 learning_rates = [0.001, 0.01, 0.1]  # Learning rates
62 dropout_rates = [0.1, 0.3, 0.5]  # Dropout probabilities
63 batch_sizes = [32, 64, 128]  # Batch sizes
64 num_filters = [16, 32, 64]  # Number of filters
```

Figure 4.2.1

This can be transformed into a binary vector group, resulting in a total of twelve decision variables. These variables represent the selection of hyperparameter values and their interactions. However, to ensure that exactly one value is selected for each hyperparameter, constraints must be introduced:

$$\sum_{i \in \text{Learning Rate}} x_i = 1, \quad \sum_{i \in \text{Dropout}} x_i = 1, \quad \ldots \tag{29}$$

This constraint ensures that only one value can be selected from each group of candidates, thereby constructing a QUBO matrix (with dimensions equal to the total number of decision variables). Finally, relevant optimization techniques are used to solve for the optimal solution vector, which identifies the selected candidate value for each hyperparameter. This optimal combination is then integrated into the previously constructed CNN model for training.
At this point, the primary model construction is complete, and the process transitions to the solution phase.

Adding Penalty Terms:
Penalty terms are added to the QUBO matrix to reflect the cost of deviating from the desired values of the hyperparameters. Each hyperparameter's candidate values are associated with penalties that encourage the optimization process to select values close to the desired settings.

1.Learning Rate(line72-76): For each candidate learning rate, a penalty is calculated as the absolute difference from the desired value of 0.01. This penalty is then placed on the diagonal of the QUBO matrix to ensure that the learning rate values close to 0.01 are favored.

```
72 # Add optimization objectives (penalty terms)
73 # Add penalty for learning rate
74 for i, rate in enumerate(learning_rates):
75     penalty = abs(rate - 0.01)
76     Q[i, i] = penalty
```

Figure 4.2.2

2.**Dropout Rate(line78-82):** Similarly, for the dropout rates, penalties are calculated based on the deviation from the desired value of 0.3.

```
78 # Add penalty for dropout probability
79 offset = len(learning_rates)
80 for i, rate in enumerate(dropout_rates):
81     penalty = abs(rate - 0.3)
82     Q[offset + i, offset + i] = penalty
```

Figure 4.2.2

3.**Batch Size(line84-88):** The batch sizes are penalized based on their deviation from the desired value of 64, with the penalty being a fraction of the deviation to account for the relative size of the batch.

```
84 # Add penalty for batch size
85 offset += len(dropout_rates)
86 for i, size in enumerate(batch_sizes):
87     penalty = abs(size - 64) / 64
88     Q[offset + i, offset + i] = penalty
```

Figure 4.2.3

4.**Number of Filters(line90-94):** The number of filters is penalized based on the deviation from the desired value of 32, with a similar relative penalty applied.

```
90 # Add penalty for number of filters
91 offset += len(batch_sizes)
92 for i, num_filter in enumerate(num_filters):
93     penalty = abs(num_filter - 32) / 32
94     Q[offset + i, offset + i] = penalty
```

Figure 4.2.4

5.**Saving the QUBO Matrix(line96-101):**

The constructed QUBO matrix is then saved to a CSV file, which is subsequently uploaded to the Kaiwu cloud platform for computation.

(Line60-73, Q2S2CNN_QUBO_HyperparameterOptimization.py)Once the solution vector is parsed, the selected hyperparameters are used to build and train the CNN model. The model is then trained using the FashionMNIST dataset with the specified hyperparameters, and its performance is evaluated on the test set.Meanwhile, we need to avoid 0 solutions of QUBO matrix.

The use of the QUBO solution vector in this manner allows the CNN model to be optimized for the specific task of image classification on the FashionMNIST dataset, potentially leading to improved performance compared to using default or suboptimal hyperparameter settings.

```
96 # Save QUBO matrix to CSV file
97 qubo_csv_path = 'qubo_matrix.csv'
98 df = pd.DataFrame(Q)
99 df.to_csv(qubo_csv_path, index=False, header=False)
100 print(f'QUBO matrix has been saved to file: {qubo_csv_path}')
101 print('Please upload this file to the Kaiwu cloud platform for computation.')
```

Figure 4.2.5

4.3 The Model Solution Formula

Solving the QUBO Model Using the Quantum Annealing Algorithm

Considering the high speed and accuracy of modern quantum annealing algorithms in solving QUBO matrices, we deployed a local Kaiwu SDK solver in Python and integrated it into the code. After constructing the corresponding QUBO matrix through model design, the matrix was fed into the solver, which provided the optimal solution vector. The solution time is 0.04 ms.

By comparing the positions of the selected candidates in the solution vector with the hyperparameter candidates defined during model construction, the optimal hyperparameter combination was determined as follows:

Learning Rate: 0.001

Dropout Rate: 0.3

Batch Size: 64

Number of Convolutional Kernels: 64

6.2	0.8	0.8	0	0.5	0.6	0.4	0.3	0.3	0.1	0	0.5	0.3	0.4	0.1	0.2	0.1
0.8	5.5	0.5	0	0.3	0.4	0.2	0.2	0.2	0.1	0	0.3	0.2	0.2	0	0.1	0.1
0.8	0.5	5.5	0	0.4	0.4	0.3	0.2	0.2	0.1	0	0.3	0.2	0.3	0	0.1	0.1
0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
0.5	0.3	0.4	0	5.2	0.3	0.2	0.1	0.1	0	0	0.2	0.1	0.2	0	0.1	0.1
0.6	0.4	0.4	0	0.3	5.3	0.2	0.1	0.2	0	0	0.3	0.2	0.2	0	0.1	0.1
0.4	0.2	0.3	0	0.2	0.2	5.1	0.1	0.1	0	0	0.2	0.1	0.1	0	0.1	0
0.3	0.2	0.2	0	0.1	0.1	0.1	5.1	0.1	0	0	0.1	0.1	0.1	0	0	0
0.3	0.2	0.2	0	0.1	0.2	0.1	0.1	5.1	0	0	0.1	0.1	0.1	0	0	0
0.1	0.1	0.1	0	0	0	0	0	0	5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
0.5	0.3	0.3	0	0.2	0.3	0.2	0.1	0.1	0	0	5.2	0.1	0.2	0	0.1	0
0.3	0.2	0.2	0	0.1	0.2	0.1	0.1	0.1	0	0	0.1	5.1	0.1	0	0	0
0.4	0.2	0.3	0	0.2	0.2	0.1	0.1	0.1	0	0	0.2	0.1	5.1	0	0.1	0
0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
0.2	0.1	0.1	0	0.1	0.1	0.1	0	0	0	0	0.1	0	0.1	0	5	0
0.1	0.1	0.1	0	0.1	0.1	0	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.1	0.1	0.1	0	0.1	0.1	0	0	0	0	0	0.1	0	0	0	0	0
0.3	0.2	0.2	0	0.1	0.1	0.1	0.1	0.1	0	0	0.1	0.1	0.1	0	0	0

Figure 4.2.6

(Specific solution 'qubo_matrix.csv_01_20241203005707.log' by Kaiwu
Task ID: tackjhg6llgdq5sc72878jn47y2bjgkh
Date of Created: 2024-12-03 00:54:44
Date of Solved: 2024-12-03 00:57:07
Running Period: 0.004ms
Results: 10)

Optimizing the CNN Model by the Derived Optimal Hyperparameter Combination

The optimal hyperparameter combination was integrated into the CNN model. The model was then trained using the Fashion MNIST dataset, and the classification results were compared against the ground truth to evaluate classification accuracy. The accuracy achieved was as high as 90.26%.

To conduct a comparative analysis, we also tested the unoptimized CNN model and models optimized using classical algorithms, such as GridSearch and RandomizedSearchCV, for hyperparameter tuning (the detailed procedures are omitted here, as they are intended solely for comparison with the primary model). The unoptimized model achieved an accuracy of 60% to 75%, demonstrating relatively poor precision. The comparison results, including those of the other methods, are summarized in the following chart:

The optimal hyperparameter combination was integrated into the CNN model. The model was then trained using the Fashion MNIST dataset, and the classification results were compared against the ground truth to evaluate classification accuracy. The accuracy achieved was as high as 90.26%.

To conduct a comparative analysis, we also tested the unoptimized CNN model and models optimized using classical algorithms, such as GridSearch and RandomizedSearchCV, for hyperparameter tuning (the detailed procedures are omitted here, as they are intended solely for comparison with the primary model). The unoptimized model achieved an accuracy of 60% to 75%, demonstrating relatively poor precision. The comparison results, including those of the other methods, are summarized in the following chart:

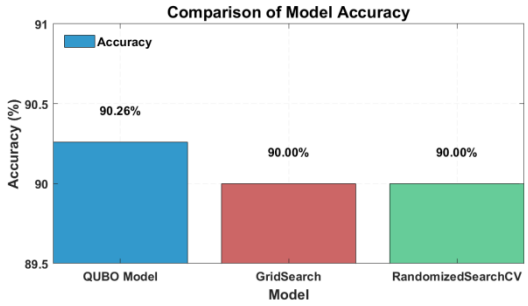

Figure 4.3

This demonstrates that the QUBO model exhibits the highest optimization capability and plays a crucial role in optimizing the CNN model and improving its accuracy. Fundamentally, it leverages the quantum superposition and tunneling effects inherent in the quantum annealing algorithm, which accelerates model solving, reduces computational complexity, expands the hyperparameter search space, and optimizes the deep learning model effectively.

5 Sensitivity Analysis of the Model for Problem 1

Tip: The objective of the model in Problem 2 is to identify the optimal hyperparameter combination. Sensitivity analysis has already been conducted during the model-solving process and will not be elaborated further here.

In classification and feature selection model construction, various fundamental parameters can significantly influence the training outcomes. For instance, in CNN models, the number of convolutional neurons; in the aforementioned QUBO matrix, the penalty coefficients α and β; and in Random Forest models, the initial number of decision trees, all affect the model's training performance. Therefore, in typical scenarios, grid search or Bayesian optimization is often employed to identify the optimal parameter combinations that yield the highest training accuracy. This process relies on performing sensitivity analysis to evaluate the impact of different coefficients on the results, which often involves substantial computational effort due to the need to compare results across multiple training iterations.

In this study, to simplify the process, we preprocessed the dataset and uploaded it to a cloud computing platform, where grid search was used to identify potentially optimal combinations of penalty coefficients. Considering the requirement to maximize classification accuracy while minimizing the number of selected features, we chose the combination of $(\alpha,\beta)=(10,0.01)$ for direct implementation in the code. This approach improved the rationality of the QUBO model construction.

However, a limitation of this study is that we did not include the initial number of decision trees in the Random Forest model as part of the optimization process. This aspect could be further improved in future research by employing methods like grid optimization or Bayesian optimization to enhance overall model performance.

As shown in Figure 7.1, the accuracy of the Random Forest model for this dataset ranges between 72% and 83%, which aligns closely with the results of our multi-model approach. Furthermore, our model demonstrates significant improvements in feature optimization and computational speed, both of which are largely attributed to the contributions of sensitivity analysis and parameter tuning.

In summary, the optimal combination of parameters derived from sensitivity analysis greatly enhanced the model's accuracy. This underscores the importance of evaluating parameter interactions and their impact on performance in constructing efficient and precise models.

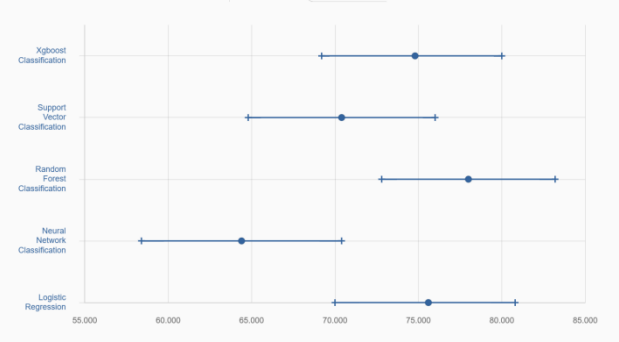

Figure 5.1

6 Model Evaluation and Further Discussion

6.1 Strengths

Problem 1:

1. Sensitivity Analysis Using Quantum Optimization Tools and Cloud Computing: Advanced quantum annealing algorithms are employed to solve QUBO matrices, enhancing feature selection efficiency. Additionally, grid search on cloud platforms is utilized to identify the optimal penalty term combination, quantifying the impact of feature selection on classification performance within the context of the dataset.

2. Optimization of QUBO Matrices with Penalty Coefficients:
Dataset features are selected as decision variables, while classification weights and feature quantity weights are introduced to construct a QUBO model. The trade-off between the two is mathematically modeled by proportionally combining classification performance penalties and feature quantity penalties to form the final QUBO matrix. This approach aims to reduce the number of selected features while improving classification performance, thereby minimizing errors.

3. Effective Overfitting Prevention:
A random forest model is used for training, leveraging its unique Bagging and random feature selection mechanisms to enhance resistance to overfitting. The QUBO model further reduces the likelihood of overfitting through feature compression. Lastly, the use of a visualized out-of-bag error curve provides effective overfitting monitoring,

facilitating model parameter optimization and training process refinement.

4. Strong Generalization Capability:
The model demonstrates high adaptability to similar optimization problems, such as those involving datasets like the Iris dataset.

5. Low Model Complexity:
With fewer features in tabular datasets, the constructed QUBO matrix has a small dimension, resulting in low linear computational cost. Additionally, only the selected features are used in model training, significantly reducing the training complexity of the random forest model.

6. Comprehensive Evaluation:
The model performs exceptionally well in feature selection and classification tasks, combining the quantum optimization advantages of the QUBO model with the robust performance of random forests. The modeling framework offers both theoretical innovation and practical applicability, particularly for feature selection and classification tasks in high-dimensional data. Moreover, the balance between runtime efficiency and classification accuracy ensures strong applicability in real-world scenarios. Future work can focus on optimizing QUBO model weight parameters and feature selection algorithms to accommodate more complex scenarios.

Problem 2：

1. Efficiency：
QUBO optimization significantly reduces the time required to find optimal hyperparameters compared to traditional methods like GridSearch or RandomizedSearchCV.

2. **Effectiveness：**
The high test set accuracies indicate that QUBO can effectively explore the hyperparameter space to find settings that maximize model performance.

3. Scalability:
QUBO's ability to handle complex optimization problems makes it a promising approach for larger and more complex models where traditional methods become impractical.

6.2 Weaknesses

Problem 1:

1. **Expand Constraints of the QUBO Model**
The current model focuses solely on optimizing feature quantity and classification performance, without considering other practical constraints such as computational

resources or feature interpretability. Incorporating additional linear inequality constraints can enhance the model's applicability to real-world scenarios.

2. **Enhance the Classification Model**
   While the Random Forest model serves as a reliable baseline, alternative methods such as Support Vector Machines or deep learning could provide superior classification performance in certain contexts. Integrating these models into the workflow could further improve the overall accuracy and adaptability.

3. **Scalability and Quantum Efficiency Analysis**
   The code is designed for small-scale problems (with approximately 20-30 features), but for larger problems with more features, it may require compressing or partitioning the QUBO matrix to ensure computational feasibility. Conducting a detailed analysis of the quantum algorithm's time complexity could provide insights for further optimization and scalability improvements.

Problem 2：

1. **Complexity:**
   Setting up and understanding QUBO models may require expertise in quantum computing, which could be a barrier for some users.

2. **Resource Availability:**
   Access to quantum computing resources is necessary to execute QUBO optimization, which might not be readily available to all researchers or practitioners.

6.3 Further Discussion

Problem 1:

The code demonstrates clear logic and operability in feature selection modeling, achieving a fusion of quantum optimization and machine learning through the QUBO model and Random Forest. However, there is still room for improvement in the areas of deep feature selection optimization, automated parameter tuning, and result validation.

By further refining the analysis of inter-feature relationships, enhancing optimization algorithms, and improving experimental design, the robustness and generalizability of the model can be significantly enhanced.

Problem 2:

This model has broad prospects, which are reflected in the following aspects:

Technological Advantages: By integrating quantum annealing with CNN models, the approach demonstrates superior efficiency and performance compared to traditional methods.

Application Potential: The model has wide applicability in areas such as image classification, hyperparameter optimization, and feature selection.

Innovative Perspective: It explores the integration of quantum computing and deep learning, providing an important reference for future research and practical applications.

However, the model also faces challenges, including limitations in quantum hardware, scalability to more complex datasets, and the complexity of QUBO construction. By improving algorithms, enhancing hardware capabilities, and exploring automated QUBO construction methods, the model is expected to deliver greater value in a wider range of fields.

7 Conclusion

Problem 1:

For Problem 1, this study reformulates the feature selection task of the German Credit Scoring dataset as a QUBO optimization problem and integrates it with the Simulated Annealing algorithm, significantly enhancing the efficiency of feature selection and classification accuracy. In this research, the QUBO model formalizes the complex feature selection problem into a mathematical optimization framework, utilizing the Simulated Annealing algorithm to efficiently identify the optimal subset in the feature space. The optimized feature subset is then used to train a Random Forest model. While reducing the complexity of the model, the classification performance remains at a high level, with experimental results showing an accuracy range of 77% to 84%. These findings demonstrate the adaptability and feasibility of this optimization approach in practical applications. Additionally, the model's ability to handle mixed-type data (including categorical and numerical features) further validates its generality and flexibility.

From a future perspective, the QUBO model, combined with modern optimization algorithms, offers significant potential for development. As quantum computing hardware continues to advance and optimization algorithms evolve, this approach will become increasingly efficient and robust when dealing with higher-dimensional and more complex datasets. In the future, this method could extend beyond credit scoring to fields such as financial risk management, medical data analysis, and supply chain optimization—domains that involve high-dimensional feature data. Furthermore, incorporating advanced classification algorithms such as Support Vector Machines and deep learning could further enhance model performance. Moreover, introducing additional constraints or multi-objective optimization would make the model more aligned with real-world scenarios, enabling it to address complex decision-making problems more effectively.

Overall, this study not only highlights the significant advantages of quantum optimization methods in feature selection and classification tasks but also provides a critical reference for

integrating quantum computing with traditional machine learning. It lays a foundation for exploring the intersection of optimization and modeling across broader fields. This interdisciplinary approach addresses existing challenges and paves the way for more efficient and intelligent solutions for future decision-making processes.

Problem 2：

In summary, our model integrates quantum annealing optimization with Convolutional Neural Networks (CNN), demonstrating its potential in the fusion of deep learning and quantum computing through its success on the Fashion MNIST dataset. By leveraging quantum annealing to optimize hyperparameters such as learning rate, dropout rate, batch size, and the number of convolutional kernels, the model achieved an accuracy of 90.26%, significantly outperforming traditional hyperparameter search methods like GridSearch and RandomizedSearchCV while improving search efficiency. The quantum tunneling and superposition properties of quantum annealing allow the model to bypass local optima in high-dimensional solution spaces and identify global optimal solutions, providing a novel approach to complex optimization problems.

The model has broad application prospects. It is not only suitable for image classification tasks, such as medical imaging analysis and object recognition in security fields, but can also be extended to other deep learning tasks, such as natural language processing and reinforcement learning, and optimization problems, such as feature selection, financial portfolio optimization, and path planning. Additionally, the universality of quantum annealing and the flexibility of the QUBO model make it adaptable to a variety of problems. However, current limitations in quantum hardware regarding variable scale and the complexity of constructing QUBO matrices may pose challenges for larger-scale and more complex problems.

In the future, advancements in quantum computing hardware will increase the scale of supported variables and computational capabilities, enabling this approach to be applied to more complex datasets (e.g., ImageNet) and tasks. Furthermore, the deeper integration of quantum computing and deep learning could give rise to a complete quantum deep learning framework, fully quantizing the optimization and training processes to provide efficient solutions across industries.

This model lays the foundation for the practical application of quantum computing in deep learning, advancing the field while providing a critical reference point for future research in quantum artificial intelligence.

8 References

[1]Wang, B., Shui, H., Wang, S., Hu, F., & Wang, C. (2021). A review of quantum annealing theory and its applications. *Science China: Physics, Mechanics & Astronomy*, *51*(8), 1–13.

[2] Pastorello, D., & Blanzieri, E. Quantum Annealing Learning Search for QUBO Problems. Quantum Information Processing, 2019, 18(9): 241.

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[4] Jiang, Q., Xie, J., & Ye, J. (2018). *Mathematical Modeling* (5th ed.). Higher Education Press.

[5] Frank R. Giordano, Maurice D. Weir, William P. Fox. A First Course in Mathematical Modeling. Brooks/Cole, 2013.

[6] Zhuo Jinwu. MATLAB in Mathematical Modeling Applications. Tsinghua University Press, Beijing, 2013.