August 10, 2024

QML For Conspicuity Detection

Name	Country
Martyna Anna Czuba	Poland
Hussein Shiri	Lebanon

The project focuses on conspicuity detection in production, which makes it possible to identify improvement measures for individual work steps or sub-processes at an early stage and thus optimize the production process. To do this, we analyze process data such as image data or time series to uncover deviations and weak points in production. Classical methods for analyzing such data are very time-consuming.

1 Task One

The first task is about getting familiar with pennylane from the codebooks on the pennylane.ai website. Three notebooks are recommended to be finished and all were finished by our team, "Introduction to Quantum Computing", "Single-Qubit Gates" and "Circuits with Many Qubits".

The "Introduction to Quantum Computing" section covers the basics of quantum computing, including qubits, superposition, and entanglement. It provides a foundational understanding of how quantum states are represented and manipulated. Differences between classical and quantum computing are discussed, as well as the principles of quantum mechanics that support quantum computation. The tutorials explain how quantum gates work on qubits, introduce the concept of normalization (where the sum of probabilities of all states equals one), and cover algebraic properties of quantum operations, such as combining operators and the importance of commutators.

```
U = np.array([[1,1],[1,-1]])
U = U/np.sqrt(2)
@qml.qnode(dev)
def apply_u():
    qml.QubitUnitary(U, wires=0)
    # return the state
    return qml.state()
```

Fig 1: Example of applying a custom unitary

The "Single-Qubit Gates" section focuses on the operations that can be performed on individual qubits. These gates are the building blocks of more complex quantum circuits. Implementations and visualizations of single-qubit gates using Pennylane provide hands-on experience with applying these gates to manipulate quantum states.

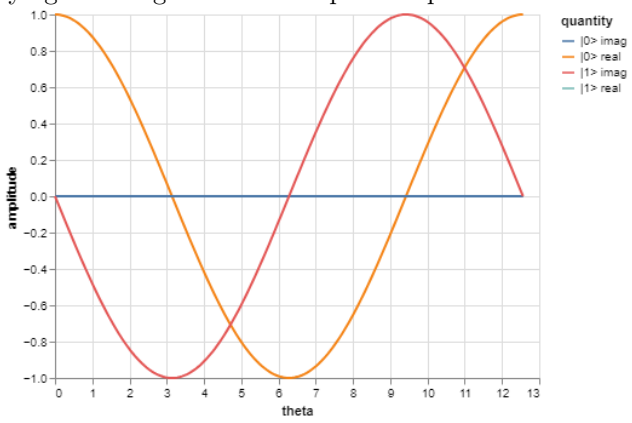


Fig 2: Evolution of quantum state for different angles of the RX gate

Figure 2 illustrates the real and imaginary components of the amplitudes of the quantum states $|0\rangle$ and $|1\rangle$ after applying the $RX(\theta)$ gate to the initial state $|0\rangle$. The $RX(\theta)$ gate rotates the state around the x-axis of the Bloch sphere by an angle θ. As θ varies from 0 to 4π, the amplitudes oscillate, reflecting the periodic nature of the rotation. The real parts show sinusoidal behavior, with $|0\rangle$ and $|1\rangle$ having a phase difference of π, illustrating the effect of the RX rotation on the qubit state.

The "Circuits with Many Qubits" section progresses to building and analyzing quantum circuits with multiple qubits. This section covers the construction of more complex circuits and the use of entanglement to create quantum states that cannot be represented classically.

The full solution can be found in the "Task 1" folder in the GitHub repository.

2 Task Two

This task required familiarizing oneself with Variational Classifiers. This topic pertains to a variational quantum machine learning model used for supervised learning. This is a popular approach that uses trainable quantum circuits as machine learning models, similar to neural networks. The circuit (also known as ansatz, parameterized circuit) is measured multiple times to estimate the expectation of some observable, and the result is interpreted as a prediction. The weights of the quantum circuit are optimized during training to minimize a cost function, similar to how weights are optimized in classical neural networks. This approach is known by different names such as variational circuits, quantum circuit learning, and parameterized circuits. This concept is visualized in Fig. 3

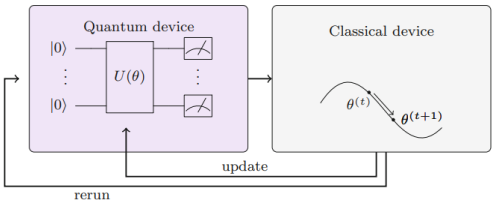


Fig 3: The idea of a hybrid quantum-classical training algorithm for variational circuits. Source: [5]

To formalise the problem, let $\mathcal{X}$ be a set of inputs and $\mathcal{Y}$ a set of outputs. Given a dataset

$$D = \{(x^1, y^1), \ldots, (x^M, y^M)\}$$

of pairs of so-called *training inputs* $x^m \in \mathcal{X}$ and *target outputs* $y^m \in \mathcal{Y}$ for $m = 1, \ldots, M$, our goal is to predict the output $y \in \mathcal{Y}$ of a new input $x \in \mathcal{X}$.

This task is divided into two main sections:

- Variational Classifiers for the Parity Function
- Variational Classifiers for Iris Classification

2.1 Variational Classifiers for the Parity Function

The first notebook called "Variational Classifier" shows that a variational circuit can be optimized to emulate the parity function

$$g : x \in \{0,1\}^{\otimes n} \rightarrow y = \begin{cases} 1 & \text{if uneven number of 1's in } x \\ 0 & \text{else.} \end{cases}$$

We performed data preprocessing by converting the labels from 0,1 to -1,1 to directly correspond to the output of the quantum circuit, which ranges from [-1,1] due to the Pauli-Z measurement. This alignment facilitates clear interpretation and seamless integration of quantum computing techniques with classical machine learning methods.We also split our dataset into training and test sets. In the mathematical form the idea of a variational classifier can be write as the expectation value of an observable O as the output of a classifier:

$$f(x;\theta) = \langle \psi(x)|U^{\dagger}\sigma_z^1 U|\psi(x)\rangle$$

, where σ_z operator applied on first qubit, U is a model circuit. The variational circuit is visualized in Fig 4.

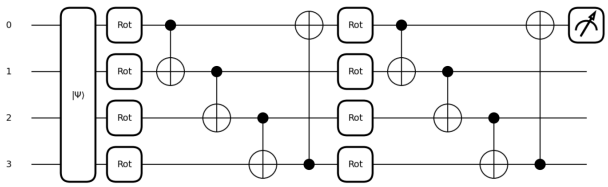


Fig 4: The circuit model using for parity classification.

This quantum circuit diagram illustrates a variational classifier using basis encoding to initialize the qubits with the input data. The circuit comprises two layers of parameterized single-qubit rotation gates (Rot) and CNOT gates in a ring topology. The final measurement step typically involves the Pauli-Z operator, with the output corresponding to the model's prediction.

For the training process, we used the Mean Squared Error (MSE) . The formula for MSE is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

,where y_i are the actual labels, $\hat{y}_i$ are the predicted values, and n is the number of samples. We employed the Nesterov Momentum Optimizer with a learning rate of 0.5. We used batches of 5 examples over 100 epochs. The small initial weight values are chosen to ensure that the training process starts smoothly and avoids numerical issues that could arise from very large or very small initial weights. This practice is based on empirical findings that suggest models train more effectively when starting with small random weights.

In our implementation, we can choose different optimizers (Gradient Descent Optimizer, Nesterov Momentum Optimizer, Adam Optimizer). Additionally, the code is written to be easily reusable and maintainable in different notebooks. This is particularly useful when the training dataset is larger.

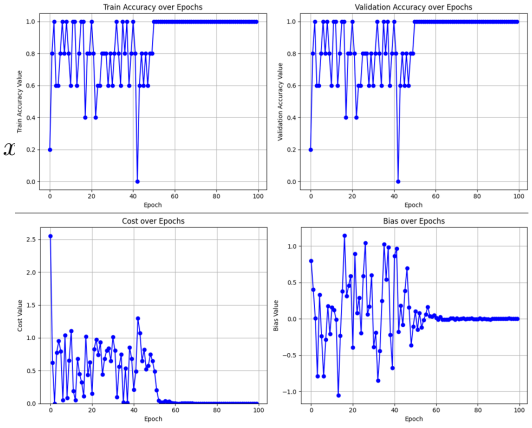


Fig 5: The plots illustrate (a) training accuracy over epochs, (b) validation accuracy over epochs, (c) cost over epochs, and (d) bias over epochs. The model demonstrates significant fluctuations in the early stages of training but eventually stabilizes, achieving 1 accuracy on both the training and validation sets and a minimal cost, indicating effective learning and generalization.

The training and validation accuracy both fluctuate significantly in the initial epochs but stabilize and reach 1 around the 50th epoch, indicating the model eventually learns and generalizes well. The cost value rapidly decreases at first, then stabilizes close to zero, reflecting effective learning. The bias value also fluctuates initially but converges around zero over time, suggesting the model parameters stabilize as training progresses. Additionally, we calculated the metrics: accuracy, precision, recall, and F1 score for the test data. All of them equal 1.

2.2 Variational Classifiers for Iris Classification

We prepared a function 'load and prepare iris data' to load the iris data, normalize the vectors, and split the data into training, validation, and test sets as part of the preprocessing stage. The data are represented as 4-dimensional real-valued vectors, and encode these inputs into 2 qubits. The Ansatz that we will use in this section is described in the previous section. We have changed state preparation method. We used the built-in function 'qml.MottonenStatePreparation'. This built-in operation prepares an arbitrary state on the given wires using a decomposition into gates developed by Mttnen et al. (2004) [3]. This state preparation stage prepares a specific quantum state using a series of RY rotation gates and CNOT gates. The RY gates apply rotations around the Y-axis, while the CNOT gates create entanglement between the qubits.This operations are shown in Fig. 6.

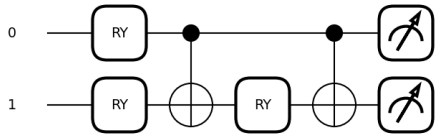


Fig 6: Quantum circuit diagram for Mottonen State Preparation.

We utilized the Nesterov Momentum Optimizer with a learning rate of 0.01. We used batches of 5 examples over 60 epochs. For this implementation, we used 2 qubits and 6 layers, with the state preparation method set to 'Mottonen'.

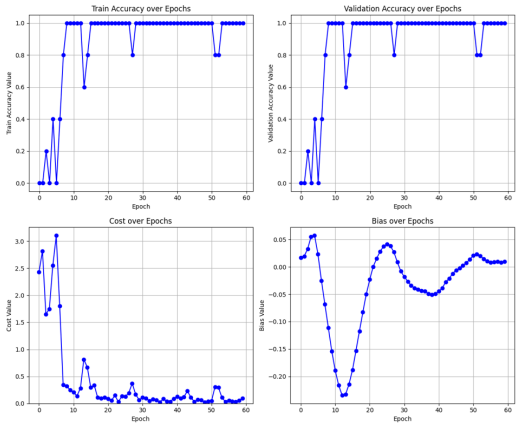


Fig 7: The plots illustrate (a) training accuracy over epochs, (b) validation accuracy over epochs, (c) cost over epochs, and (d) bias over epochs. The model demonstrates rapid learning and stabilization, achieving 1 accuracy on both the training and validation sets, and effectively minimizing the cost. The bias value converges to zero, indicating stable model parameters.

The training and validation accuracy both increase rapidly in the initial epochs, stabilizing at 1 around the 20th epoch, indicating effective learning and generalization. The cost value drops significantly within the first 10 epochs and remains low, reflecting successful minimization. The bias value fluctuates initially but converges to zero over time. Additionally, we calculated the metrics: accuracy, precision, recall, and F1 score for the test data. All of them equal 1.

2.3 Summary

Summarizing both sections, in this task we deepened our understanding of variational quantum circuits. We trained both circuits for different problems, generated plots, and calculated metrics. We also ensured that our code is clean and well-managed, making it easy to use and open to changes in the future.

Additionally, based on the article Circuit-centric quantum classifiers [4], we implemented a circuit called circuit-centric quantum classifiers. However, we did not have time to conduct experiments with this code. We plan to extend this task in the future.

3 Task Three

3.1 Classic convolution

Convolution layers are part of a neural network that works on images to perform certain task like classification, encoding, ... These layers use a kernel to perform feature extraction and reduce the image dimensions.

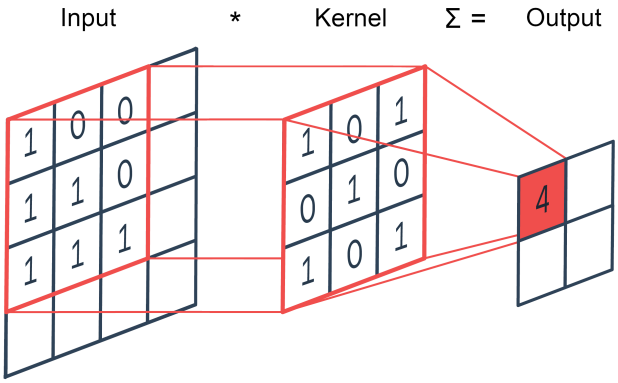


Fig 8: kernel acting on an image

For example, a neural network for image classification starts first by applying several layers, one of which is the convolution layer before reaching the final layer and performing classification. This helps in decreasing the image size and extracting the most important features for classification.

3.2 Quantum convolution

Instead of using a classical model for classification, we can use a hybrid model. This model starts first by performing "quantum convolution" called "quanvolution" instead of classical convolution layers. Then using classical layers for image classification.

In our solution we have compared the validation accuracy using the same dataset for 2 kernels of different sizes. A kernel of size 4x4 vs 2x2.

- Using a 4x4 kernel:
  Original image size: (28, 28, 1)
  Size after convolution: (7, 7, 16)

- Using a 2x2 kernel:
  Original image size: (28, 28, 1)
  Size after convolution: (14, 14, 4)

The 3 values **w x h x c**, where c is the number of channels, change according to the kernel size.

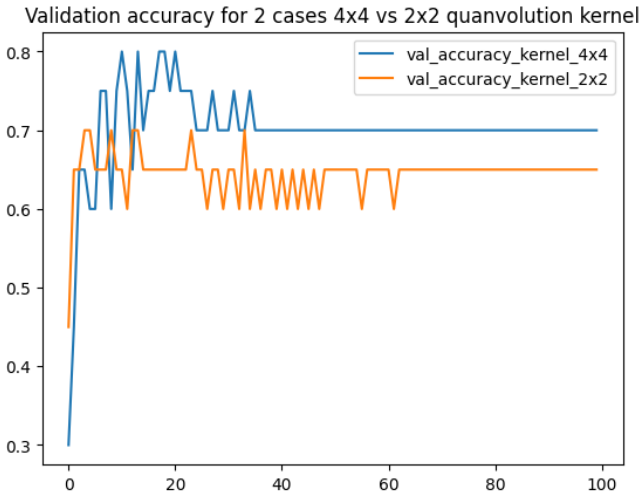


Fig 9: Validation accuracy for each epoch

This comparison shows a 5-10% increase in the validation accuracy using a 4x4 kernel compared to a 2x2. This encourages trying different kernel sizes to find the one that gives the highest accuracy after classification.

Our solution uses the following dataset and parameters:

- Dataset:
  Training: 100 images
  Testing: 20 images
  MNIST Dataset

- Other parameters:
  Epochs: 100
  Classes: 10
  Batch size: 4

Full solution is found in "Task 3" folder in the github repository.

4 Task Four

The task was to develop a model to learn the sine function on the interval $[0, 2\pi]$. We discretized the interval with a suitable number of points and used the sine values at these points as labels. The goal was to implement a Quantum Machine Learning model that reproduces the sine function values. Our solutions are based on several key articles,including Schuld, Maria et.al "The effect of data encoding on the expressive power of variational quantum machine learning models" [6] and Patrick Holzer,Ivica Turkalj "Spectral Invariance and Maximality Properties of the Frequency Spectrum of Quantum Neural Networks" [2].These articles have been invaluable, providing us with a wealth of information and serving as significant resources for our research and development..

We identified three sections in this task:

- Quantum Model 1
- Quantum Model 2
- Classical Model

4.1 Quantum Model 1

Based on the previously mentioned articles [6], [2], we learned that the expressivity of the corresponding quantum model is fundamentally limited by the data encoding. Inspired by these articles, specifically Section II [6]. A, we have chosen the following ansatz in a slightly modified form, but based on the same principles.

The generated data is in the form: $g(x) = sin(x)$. We discretized this function into 1000 points over the interval $[0, 2\pi]$ We use a single-qubit gate generator, $H = \left(\frac{1}{2}\right)\sigma_x$. Our variational circuit is of the form

$$f_\theta(x) = \langle 0|U^\dagger(x,\theta)MU(x,\theta)|0\rangle$$

,where $|0\rangle$ is a single qubit, $M = \sigma_z$, and

$$U(x,\theta) = W_\theta^{(1)} R_x(x).$$

This circuit is illustrated in Fig 9.

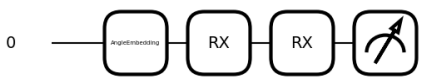


Fig 9: Variational Quantum Circuit for Learning the Sine Function

During the training process, we utilize the Gradient Descent Optimizer (learning rate 0.1) to minimize the Mean Squared Error (MSE) cost function over 30 epochs. The parameters of the quantum circuit are initially small and are iteratively updated to fit the sine function based on the training data.

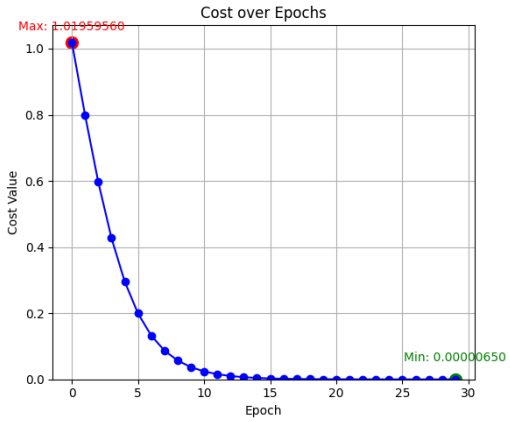


Fig 10: Cost Function Convergence During Training.

Fig 10 shows the convergence of the MSE cost function during 30 epochs of training. The maximum cost value is 1.0195960 at epoch 0, and the minimum cost value achieved is 0.00000650 at epoch 30, indicating successful fitting of the sine function to the training data.

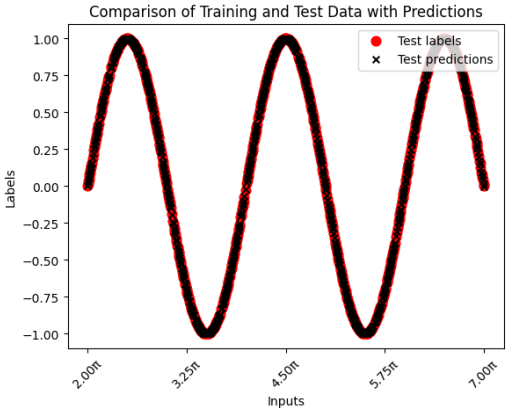


Fig 11: Evaluation of VQC Model on on Extended Interval for Sine Function.

Fig 11 compares the test labels (red circles) and test predictions (black crosses) for the sine function over the extended interval $[2\pi, 7\pi]$. The VQC model demonstrates excellent performance with a very low Mean Squared Error (MSE) of 0.0000067 on this interval. The close alignment of predictions with actual labels indicates the model's robustness and generalization capability. The MSE values are: training set - 0.0000060, test set - 0.0000064, and interval $[2\pi, 7\pi]$ - 0.0000067, confirming the model's accurate learning of the sine function.

In our code, it is possible to change the cost function to the Mean Absolute Error (MAE) by using 'cost MAE'. The logs of this training are saved in the file '1 Training Process Sin function.txt' in folder experiments.

4.1.1 Adjustment of number of points

In this section, we will select a sufficient number of data points to effectively learn the sine function. In the 'perform training function', we run the training process on data generated using different numbers of points $(1000, 100, 50, 30, 20, 10, 5, 4, 3, 2, 1)$. The test cost is always measured on the same dataset: the interval $[2\pi, 7\pi]$, sampled with 1000 points. Fig 12 illustrates the relationship between the number of data points and the corresponding training and test costs for the quantum model.

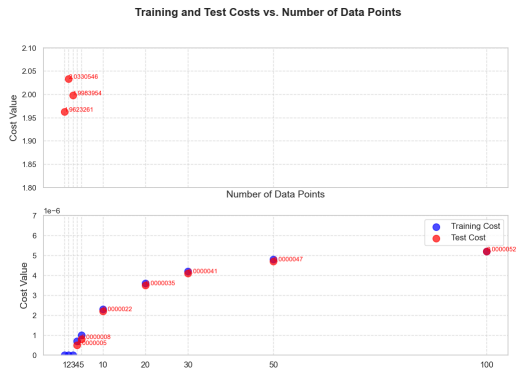


Fig 12: Analysis of Training and Test Costs vs. Number of Data Points

The parameters tend to converge to similar values as the number of data points increases, indicating stable model training. As for the cost function behavior, an increase in the number of data points results in a slight increase in the cost function, suggesting a trade-off between model complexity and the volume of training data. It is experimentally observed that with the given training parameters, the model requires only 4 data points to effectively learn the sine function For cases with 3, 2, and 1 data points, despite achieving a final cost of zero during training, the test cost remains significantly high, indicating overfitting and poor generalization performance

This analysis underscores the importance of selecting an appropriate number of data points for training quantum models. While fewer data points can lead to overfitting, an optimal number can achieve effective learning and good generalization.

4.1.2 Training for more complicated function

We evaluate the performance of this model on functions with a larger frequency spectrum. Despite numerous training sessions and hyperparameter updates, this model failed to fit the function.

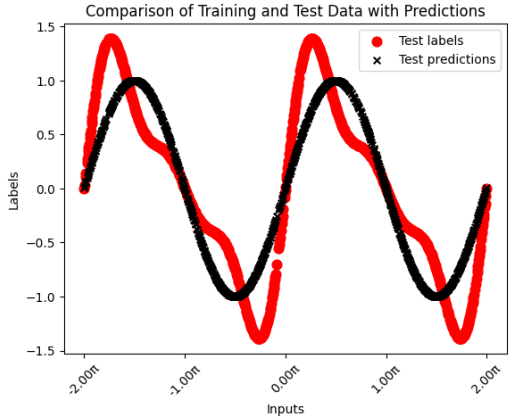


Fig 13: Evaluation of VQC Model on function with larger frequency spectrum.

Fig 13 shows the limitations of the quantum model in fitting complex functions. Above, we can observe that this quantum model is capable of fitting only a sine function. In other words, this quantum model (utilizing a single Pauli-X rotation) can only learn to fit a Fourier series of a single frequency, and this is possible only if that frequency exactly matches the scaling of the data. Regardless of the chosen weights, the single qubit model for $L = 1$ will always represent a sine function of a fixed frequency. The weights merely adjust the amplitude, vertical shift, and phase of the sine wave. Exactly as suggested by the articles [6], [2].

4.2 Quantum Model 2

We implemented the proposed model from article Schuld, Maria et.al "The effect of data encoding on the expressive power of variational quantum machine learning models" [6]. Subsequently, we extend the task and train the model on a more complex function. We visualize our results with plots and demonstrate that this model effectively learns more complex functions.

Here we slightly modify the model. The system is in exactly the same form as proposed in the article Schuld, Maria et.al "The effect of data encoding on the expressive power of variational quantum machine learning models" [6]. The architecture from this article is visualized in Fig 14.

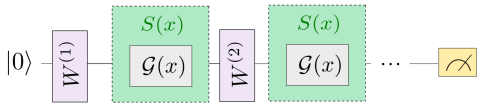


Fig 14:The general the Quantum Model. Source: [6].

We used a (univariate) quantum model $f_\theta(x)$ and follow the assumption that overall quantum circuit has the form:

$$U(x) = W^{(L+1)}S(x)W^{(L)}...W^{(2)}S(x)W^{(1)}$$

We can see that our model is splitted into 'data encoding (circuit) block1' $S(x)$ and 'trainable (circuit) block' W. We use popular strategy of encoding an input single-qubit rotations. Our variational circuit is of the form

$$f_\theta(x) = \langle 0|U^\dagger(x,\theta)MU(x,\theta)|0\rangle$$

,where $|0\rangle$ is a single qubit, $M = \sigma_z$, and

$$U(x,\theta) = W^{(2)}R_x(x)W^{(1)}$$

.

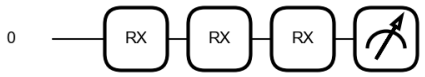


Fig 15:Second Variational Quantum Circuit for Learning the Sine Function

This model handled the problem equally well. During the training process, we utilize the Gradient Descent Optimizer (learning rate 0.1) to minimize the Mean Squared Er- ror (MSE) cost function over 30 epochs. The parameters of the quantum circuit are initially small and are iteratively updated to fit the sine function based on the training data.

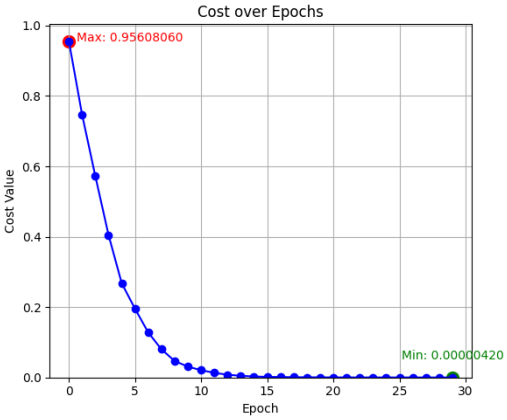


Fig 16: Cost Function Convergence During Training.

Fig 10 shows the convergence of the MSE cost function during 30 epochs of training. Comparing with the previous model, both training processes utilized the same dataset of 800 samples, with identical hyperparameters and parameter initialization, allowing for a direct comparison of the two circuits. Both circuits effectively minimized the cost function over 30 epochs, with Circuit 1 achieving a final cost of 0.0000065 and Circuit 2 reaching a slightly lower final cost of 0.0000042. The mean cost and standard deviation were nearly identical between the circuits, and the final parameters showed minimal differences, indicating that both circuits successfully and stably learned the sine function

with comparable performance. More details on this comparison can be found in the accompanying notebook. According to observations from [6], the single-qubit model with $L = 1$ consistently produces a sine function with a fixed frequency, regardless of the chosen weights. The weights influence only the amplitude, vertical shift, and phase of the sine wave.

4.2.1 Training for more complicated function

In this section, we aim to expand the frequency spectrum to a wider range.

Our target function:

$$g(x) = (0.05 + 0.05j)\, e^{ix} + (0.05 + 0.05j)\, e^{i\cdot 2x}$$

$$+ (0.05 + 0.05j)\, e^{i\cdot 3x} + (0.05 + 0.05j)\, e^{i\cdot 4x} + (0.05 + 0.05j)\, e^{i\cdot 5x}$$

Our variational circuit is of the form

$$f_\theta(x) = \langle 0|U^\dagger(x,\theta) M U(x,\theta)|0\rangle$$

,where $|0\rangle$ is a single qubit, $M = \sigma_z$, and.

$$U(x,\theta) = W_\theta^{(L+1)} \underbrace{S_L(x) W_\theta^{(L)}}_{\text{Layer L}} \cdots W_\theta^{(2)} \underbrace{S_1(x) W_\theta^{(1)}}_{\text{Layer 1}}$$

,where $L = 5$, $S(x) = e^{-ixH} = R_x(\phi)$, $W_\theta = RZ(\omega)RY(\theta)RZ(\phi)$.

The circuit is visualized in Fig. 16.

Fig 16: Variational Quantum Circuit.

During the training process, we utilize the Adam Optimizer (learning rate 0.1) over 300 epochs. Additionally, we modify the loss function slightly. We multiply our loss function by a factor of 1/2. Adding the 1/2 factor is common practice for simplifying the gradient during backpropagation.The gradients are scaled down by a factor of 2 when we include the $\frac{1}{2}$ factor in the loss function. It is important to note, that this can influence the learning process, particularly with regard to the learning rate

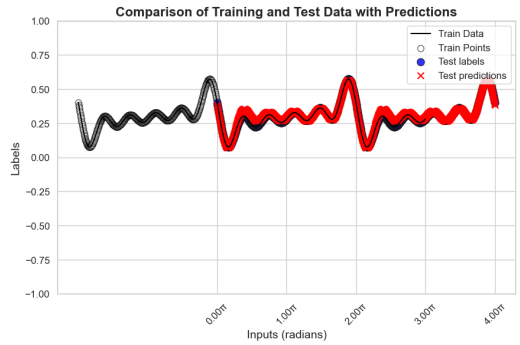


Fig 17: Evaluation of VQC Model on function with larger frequency spectrum.

Overall, Fig 17 shows that the model performs well on both training and test data, with only minor discrepancies that could be further optimized.

4.3 Classical Model

This study defines a neural network model using PyTorch to approximate the sine function, aiming to capture its underlying pattern through regression. The model consists of a simple feed-forward neural network with one hidden layer, trained using the Adam optimizer and Mean Squared Error (MSE) loss function. The network is trained over $10,000$ epochs, ultimately achieving a final loss value of 0.00084. The classical neural network model employed in this study has a total of 61 trainable parameters. These results underscore the efficiency of quantum models in achieving lower final costs with significantly fewer parameters, particularly for the problem at hand.

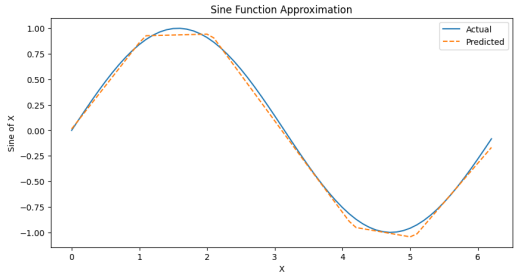


Fig. 18 :Evaluation of Neural Network Model for Sine Function Approximation.

For the quantum models, Circuit 1 achieved a final cost of 0.0000065 using just 2 parameters over 30 epochs, while Circuit 2 reached a final cost of 0.0000042 with the same number of parameters and epochs. These results demonstrate the efficiency of quantum models in attaining lower costs with significantly fewer parameters compared to the classical neural network, particularly for this specific task.

However, it is important to emphasize that this analysis does not account for several critical factors. Comprehensive evaluation should consider aspects such as computational complexity, scalability, and the robustness of each approach to draw more definitive conclusions. Further exploration in these areas is necessary to fully understand the comparative advantages and limitations of classical and quantum solutions.

4.4 Future work

Open questions:

Based on observations in the "Adjustment of Number of Points" section, why exactly does an increase in the number of data points result in a slight increase in the cost function? How does this apply to other problems? Do we know exactly how many data points are needed depending on the problem?

5 Task Five

5.1 Introduction

As task 5 is the most challenging and needs the most time, we took the following approach. We started working first on task 5 from the start of the project phase and even 1-2 weeks before. We continued working on this task in parallel with the other tasks till the end of the project phase.

The general approach for this task is summarized in the following points:

- Image Preprocessing and Exploration: Provides a comprehensive analysis and preprocessing pipeline for image data.
- Building the Dataset: We randomly select images from both the train and test folders, ensuring an equal number of images from each class to maintain a balanced dataset. For binary classification, we consolidate classes 1 through 5 into a single class, resulting in two final classes.
- Dimensionality Reduction: Given the large image size, we applied two dimensionality reduction techniquesPrincipal Component Analysis (PCA) and Autoencodersto extract the most significant features and reduce the number of variables needed for encoding in the quantum circuit.
- Data encoding: The classical data were encoded into the quantum circuit using amplitude encoding, applied to circuits of 4 and 8 qubits.
- Model Training: The model was trained using the Adam optimizer, with a comparative analysis of three different loss functions: exponential loss, binary cross-entropy loss, and focal loss.
- Performance Evaluation: To evaluate our model, we calculated accuracy, recall, precision, and F1 score

5.2 Exploring the dataset:

The initial step involves examining the dataset's contents: how the images are stored, what the labels represent, the subject matter of the images, and their dimensions.

The dataset consists of 6 labels: 1 label for non-defective images (Label 0: good weld) and 5 labels for various defects (Labels 1-5). To enable binary classification, we merged images with Labels 1-5 into a single "defective" category (Label 1), while retaining Label 0 as the "non-defective" category. The label distribution is as follows:

- Label 0 (good weld): 10,947
- Label 1 (burn through): 2,134
- Label 2 (contamination): 8,403
- Label 3 (lack of fusion): 5,035
- Label 4 (misalignment): 3,682
- Label 5 (lack of penetration): 3,053

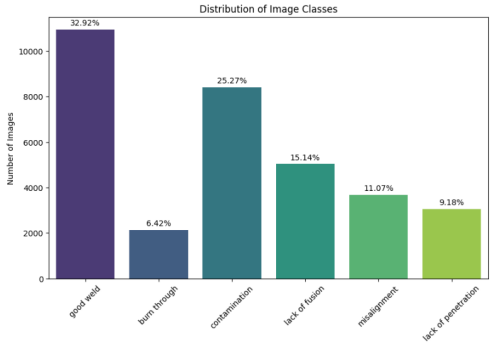
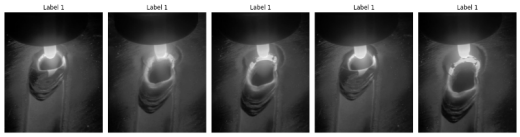


Fig 19: Class-wise distribution of images in the dataset

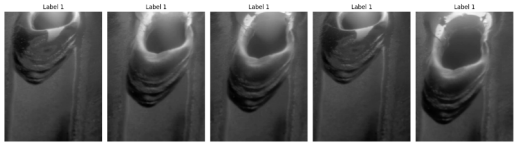
5.3 Data preprocessing and preparation:

We combined the images from the train and test folders because we observed a significant discrepancy in accuracy: the model achieved high accuracy when tested on images from the train folder, but low accuracy when tested on images from the test folder. To improve the model's ability to generalize to unseen data, we merged both folders into a single dataset. Images are then randomly selected from this combined dataset for training and testing..

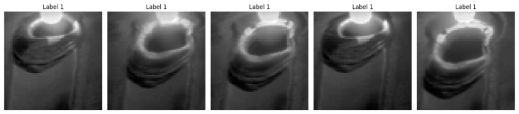
- Original images:



- Cropped images:



- Resized images after crop, 64x64:



Cropping reduces the image size and removes irrelevant parts, while resizing ensures all images are uniform in size. However, further reduction and feature extraction are necessary. For instance, a 64x64 image yields 4,096 values to encode, which would require 12 qubits using amplitude encodinga time-consuming process. Therefore, we implement two classical dimensionality reduction techniques: PCA and autoencoders.

We acknowledge that this probably had a significant impact on the model's learning process. Feature extraction techniques, such as PCA and autoencoders, are used to capture the most relevant information from the data while reducing its dimensionality, thereby improving the models efficiency and effectiveness. Additionally, more advanced techniques, such as Fast Fourier Transform (FFT), could be explored in future experiments to further enhance feature extraction and potentially provide better model performance. In the future, additional experiments on larger feature spaces may be necessary.

5.4 PCA vs autoencoder:

Principal Component Analysis (PCA) is a dimensionality reduction technique that compresses the dataset while preserving the most significant patterns. In other words, it acts as a compression tool that reduces the size of the data while retaining as much information as possible.

The key difference between PCA and autoencoders lies in their reduction techniques. PCA utilizes a linear mapping function, making it a linear dimensionality reduction method, while autoencoders apply non-linear transformations, enabling them to capture more complex relationships within the data.

In our study, we applied PCA to reduce each 64x64 image to 16 values, aligning with our goal of using amplitude encoding on a 4-qubit ansatz. We also utilized an autoencoder to reduce the image size from 64x64 to 256 values, which allows us to encode the data into an 8-qubit ansatz using amplitude encoding.

In summary, we employed two different approaches based on circuit size: one uses PCA to encode data into a 4-qubit circuit, and the other uses an autoencoder to encode data into an 8-qubit circuit, both utilizing amplitude encoding.

5.5 Quantum tensor networks:

Quantum tensor networks are structures designed to represent and manipulate quantum states, leveraging their interconnected design. In these networks, tensors take the place of traditional graph nodes, with each tensor corresponding to a 2-qubit ansatz. These tensors are linked via CNOT gates to form larger configurations, such as 4-qubit or 8-qubit anstze, which are crucial for quantum machine learning applications. This specific circuit configuration is known as the $SU(4)$ circuit.

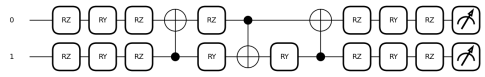


Fig 20: General element of $SU(4)$

We chose this block ansatz because of the big number of parameters so the model can learn better and due to its high entanglement using 3 CNOTs. Also, we will use the MERA quantum tensor network architecture instead of TTN. The reason for that is that MERA has extra layers and parameters which help the trainable ansatz learn better and give higher accuracy. This difference between the 2 architectures can be clearly seen in the following fig. 21.

5.6 Loss function:

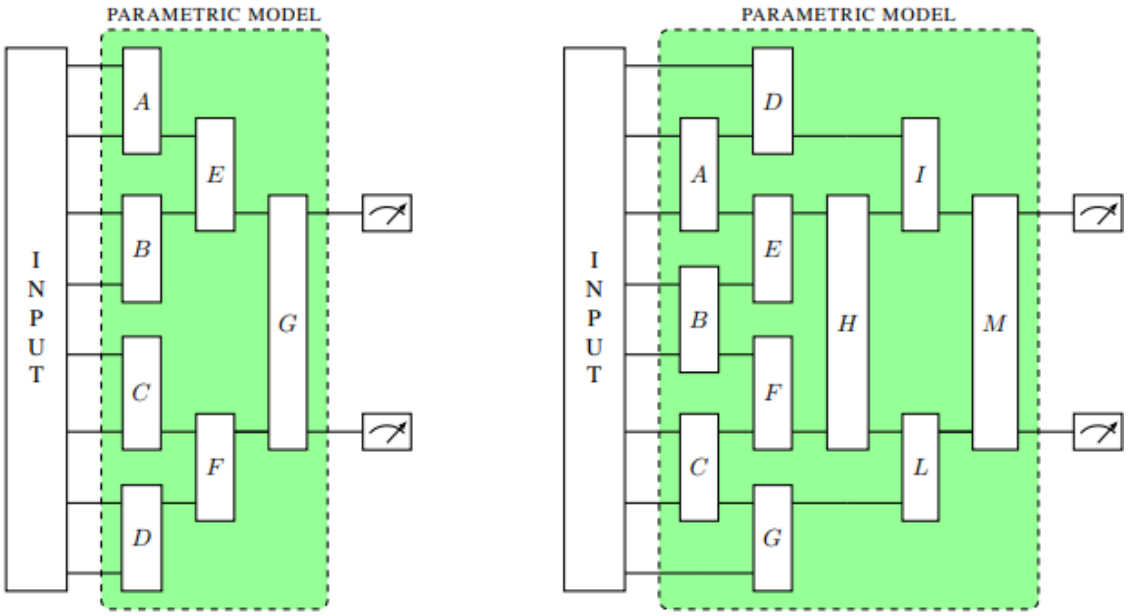
Before going in detail into the results, it is important to explain the 3 different loss function we have used.

- Exponential loss:

$$loss = \sum_i (1 + 10e^{7p_i})^{-1}$$

- Binary cross entropy loss:

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$



(a) Variational classifier based on a TTN. (b) Variational classifier based on a MERA.

Fig. 21: TTN vs MERA architecture. Source: [1]

Train set	Test set	Validation set	Loss	epochs	Validation acc	Test acc
7971,8028	1006,995	1023,977	exponential	200	88.41%	87%
7971,8028	1006,995	1023,977	cross entropy	100	83.41%	83.3%
7971,8028	1006,995	1023,977	FL, gamma=1	100	85.71%	84.85%
7971,8028	1006,995	1023,977	FL, gamma=2	100	87.46%	86.65%

Table 1: table showing the results for circuit with 4 qubits.(Binary Classification)

- Focal loss:

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

The exponential loss function was taken from the research paper Practical overview of image classification with tensor-network quantum circuits, It did not have a name so we chose exponential loss for it since it uses exponential.

This loss function focuses on optimizing all images instead of part of the image like what the binary cross entropy does. So we might get a correct label for some images but binary cross entropy keeps optimizing these correct labels instead of focusing on other images.

The focal loss is the cross entropy loss multiplied by

$$-(1-p_t)^{\gamma}$$

which helps the model to focus more on hard examples, since after easy examples have been easily classified we need more focus on the hard examples, as no need to focus more on the easy examples.

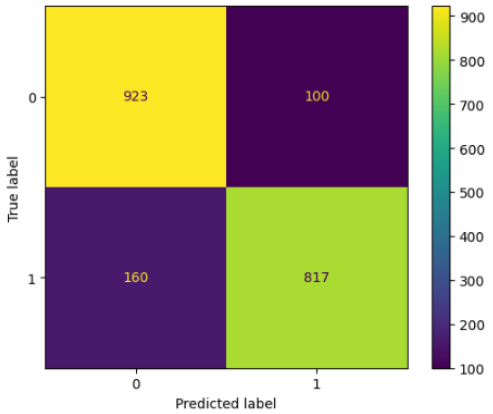
5.7 Four qubits circuit:

We now start with our first solution, after getting the images, performing preprocessing, and building our circuit we can now compare the 3 loss function after performing PCA.

```
from  Modules.Utils import apply_pca
X_reduced = apply_pca(X_set, n_components=16)
```

Now we encode the 16 values into the 4-qubits circuit using amplitude encoding and we optimize the ansatze parameters using Adam optimizer for the different loss functions.

From the table above we can see that the exponential loss gave the best validation and test accuracy's of 88.41% and 87% respectively. Although we ran it for 200 epochs, it was able to reach its maximum accuracy before 100 epochs, so we ran the other cost functions for 100 epochs.



In fact, the accuracy is not the only important metric. The damaged induced by predicting a defective model is not defective can lead to missing defects and have product failure in the end. The F1-score and the confusion matrix are some metrics to help analyze the ability of the model to predict each class 0 and 1.

The calculated f1-Scores are [0.87654321, 0.86272439] for classes [0, 1]. We can clearly see that for 7971,8028 train images (7971 for class 0 and 8028 for class 1) we had the model slightly better at predicting class 0 than class 1 in the test dataset.

5.8 Eight qubits circuit:

Using 8 qubits gives us the ability to encode more information in the circuit, also using the autoencoder allows us to find relations, especially non-linear ones that PCA might have missed when it performed reduction. In this approach we have used keras to build the autoencoder.

Steps for building the autoencoder:

- Build the encoder part

Train set	Test set	Validation set	Loss	epochs	Validation acc	Test acc
2393,2407	303,297	304,296	cross entropy	200	88.17%	86.0%
2393,2407	303,297	304,296	exponential	200	92.67%	89.83%
2393,2407	303,297	304,296	FL, gamma=1	200	89.33%	86.0%
2393,2407	303,297	304,296	FL, gamma=2	200	92.33%	80.83%

Table 2: table showing the results for circuit with 8 qubits.(Binary classification)

- Build the decoder part
- Compile and fit the model
- Predict the encoded dataset(train, validation and test)

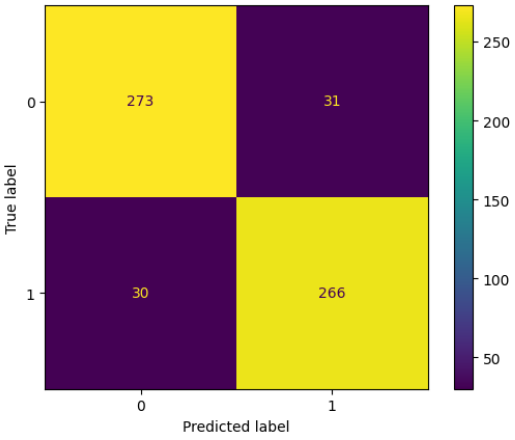
The autoencoder was trained using the mean squared error loss combined with Adam optimizer using keras.

Layer (type)	Output Shape
input_layer_2 (InputLayer)	(None, 64, 64, 1)
conv2d_4 (Conv2D)	(None, 32, 32, 32)
conv2d_5 (Conv2D)	(None, 16, 16, 64)
flatten_2 (Flatten)	(None, 16384)
dense_6 (Dense)	(None, 4096)
dense_7 (Dense)	(None, 256)
dense_8 (Dense)	(None, 4096)
reshape_2 (Reshape)	(None, 8, 8, 64)
conv2d_transpose_8 (Conv2DTranspose)	(None, 16, 16, 64)
conv2d_transpose_9 (Conv2DTranspose)	(None, 32, 32, 32)
conv2d_transpose_10 (Conv2DTranspose)	(None, 64, 64, 16)
conv2d_transpose_11 (Conv2DTranspose)	(None, 64, 64, 1)

Fig 22: The architecture of the used autoencoder.

We can see that for the second time the exponential loss function was the one that gave the best validation and test accuracy's. We had to use less training images since we have a bigger circuit of 8 qubits. Using a bigger training set will increase the training time a lot.

We show another time the plotted confusion matrix to see how good the model was at predicting each individual class defective vs non-defective.



We can see another time from the F1-score that the model was able to slightly better predict class 0 compared to class 1. The calculated F1-scores are the following [0.89950577, 0.89713322], for classes [0, 1].

5.9 Extra:

Although not required by task 5, we have worked also on checking the accuracy of the solution for multiclassification using 4,6 different labels and using only the binary cross entropy loss with PCA.

We got a lot lower results, and the maximum we were able to get was around accuracy 60% for 6 class and 80% for 4 class. The results are partially summarized in Table 3.

We suggest that using the autoencoder with exponential loss could lead to improved accuracy. Additionally, we observed a slight difference between the MERA architecture described in this paper [1] and the version implemented in PennyLane.

6 Challenges and future scope

6.1 Challenges

We encountered several challenges while working on the project::

- Implementing research papers:

  Given the high dimensionality of our images, approximately 800x800 pixels, significant additional effort was required to apply the techniques discussed in this project. Most existing studies typically focus on lower-dimensional datasets, such as 28x28 pixel images, and often utilize simplified datasets like MNIST. In contrast, our data is more complex, originating from real-world sources rather than standardized datasets, which necessitated the development of more sophisticated methods to effectively process and analyze these images.

- Training time:

  As the complexity of the problem (image) increases, our model expands in both width and depth. This growth also translates to an increase in the training time required. In the context of quantum machine learning, this means that as the image complexity rises, the quantum circuits must become more intricate, with more qubits and deeper layers of gates, leading to longer training times and greater computational demands.

- Challenges and Learning Experiences: A significant challenge was the time constraint. The project involved numerous aspects that we had to familiarize ourselves with, which proved to be both educational and broadening for our perspectives.

6.2 Future Scope

6.2.1 Binary-classification:

We are considering applying the concept from Task 4 to Task 5 by using quantum machine learning to analyze image data. The idea is to treat each row of an image as a sine-like function, extracting its frequency components through Fourier analysis and representing them as a sum of sine and cosine functions. A quantum model could be trained to learn the correct behavior of these functions by processing Fourier-type sums that capture the frequency patterns in each row.

Once trained, the model could evaluate new images by analyzing the frequency components of each row. If the image is correct, the quantum model would recognize the familiar patterns, leading to a low loss function. If the image contains anomalies, the model would detect deviations in the frequency patterns, resulting in a high loss function.

This is still just an idea without detailed implementation, but we believe its a concept worth exploring further, especially given the potential advantages of quantum models in handling complex frequency data and detecting subtle anomalies.

6.2.2 Multi-classification:

This project focused on binary classification but in some cases multi-classification is required and binary classification is not

Train set	Validation set	Loss	epochs	Validation acc
349,364,350,350,325,361	61,42,39,51,61,47	cross entropy	70	56.81%
349,364,350,350,325,361	61,42,39,51,61,47	cross entropy	100	46.18%
349,364,350,350,325,361	61,42,39,51,61,47	cross entropy	215	48.18%
349,364,350,350,325,361	61,42,39,51,61,47	cross entropy	300	59.9%%

Table 3: table showing the results for circuit with 8 qubits.(multi-class classification, 6 classes)

sufficient. For example classifying the breeds of a certain animal like dogs. Or classifying the type of a vehicle.

In our project we have shown some results when the project is expanded to 6 classes, this expantion can also be worked on to find some ways to get better accuracy.

6.2.3 Unanswered questions:

Based on observations in task 4 on the Adjustment of Number of Points section, why exactly does an increase in the number of data points result in a slight increase in the cost function? How does this apply to other problems? Do we know exactly how many data points are needed depending on the problem?

How Does Our model compare to other models? Can we find an exponential speedup in QML?

6.2.4 Startup:

Stratups in ML/AI mainly can provide consultation, new software or programs and other services to companies and individuals. Startups that try to merge AI/ML with quantum technologies can use classical machine learning to build new software to speed certain tasks related to quantum technologies like better or faster compilers for quantum computers.

Startups in this field can also focus on research in trying to find new algorithms or potential use cases with speedup in the field of quantum machine learning.

References

[1] Daniel Gonzalez, Lukasz Cincio, Mikkel Kjaergaard, and et al. Multi-class quantum classifiers with tensor network circuits for quantum phase recognition. *arXiv preprint arXiv:2110.08386*, 2021.

[2] Patrick Holzer and Ivica Turkalj. Spectral invariance and maximality properties of the frequency spectrum of quantum neural networks. *Physical Review A*, 104(3):032404, 2021.

[3] Mikko Mottonen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. *arXiv preprint quant-ph/0407010*, 2004.

[4] Maria Schuld, Alex Bocharov, Krysta Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3):032308, 2020.

[5] Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers*. Springer International Publishing, 2019.

[6] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. The effect of data encoding on the expressive power of variational quantum machine learning models. *Physical Review A*, 103(3):032430, 2021.