QML FOR CONSPICUITY DETECTION IN PRODUCTION

‹ WOMANIUM | QUANTUM ›

Womanium Quantum+AI Project

Adriano Lusso

Jalal Naghiyev

1. Team presentation
2. Tasks schedule
3. Task 1,2 and 3: Basic training
4. Task 4: quantum model for trigonometrical function regression
5. task 5: binary and multi-classification for conspicuity detection

TEAM PRESENTATION

Adriano Lusso

www.linkedin.com/in/adrianolusso

1. Computer Science student at Universidad Nacional del Comahue, Argentina.
2. Young quantum software researcher, with poster acceptances at international conferences.
3. Work over QAOA, Zero-Noise Extrapolation, QML and delegated quantum computing.

Jalal Naghiyev

www.linkedin.com/in/adrianolusso

1. masters in AI at ITMO University.
2. Junior Machine Learning Researcher interested in QML, LLM and Tensor Network.
3. Previous work on LLMs, currently working on Tensor Networks for QC.

TASKS SCHEDULE

1

Familiarize yourself with Pennylane, using its tutorials codebook.

2

Work through Pennylane's tutorial of Variational Classifier, and implement it yourself.

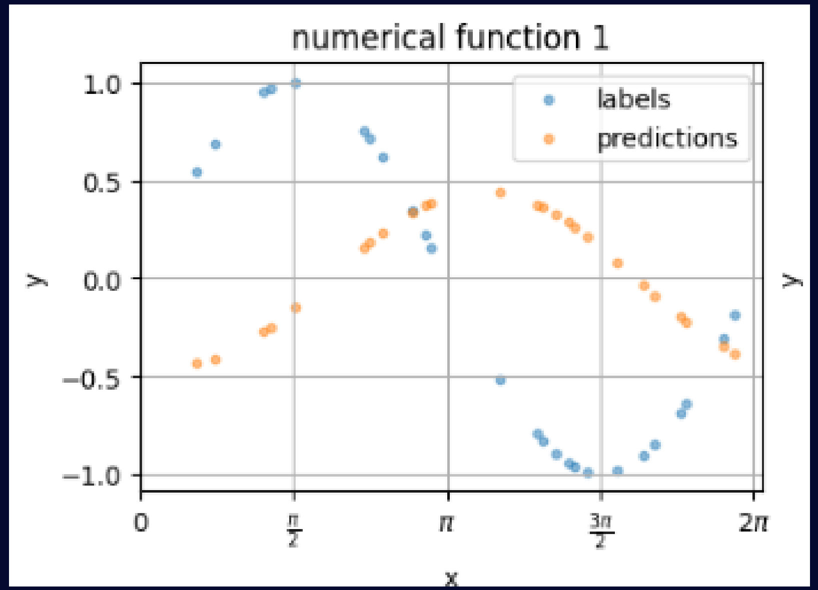
3

Work through Pennylane's tutorial of Quanvolutional Neural Networks (QCNN), and implement it yourself.

TASKS SCHEDULE

4

Develop your own model and use it to learn the sine function on the interval [0, 2pi]

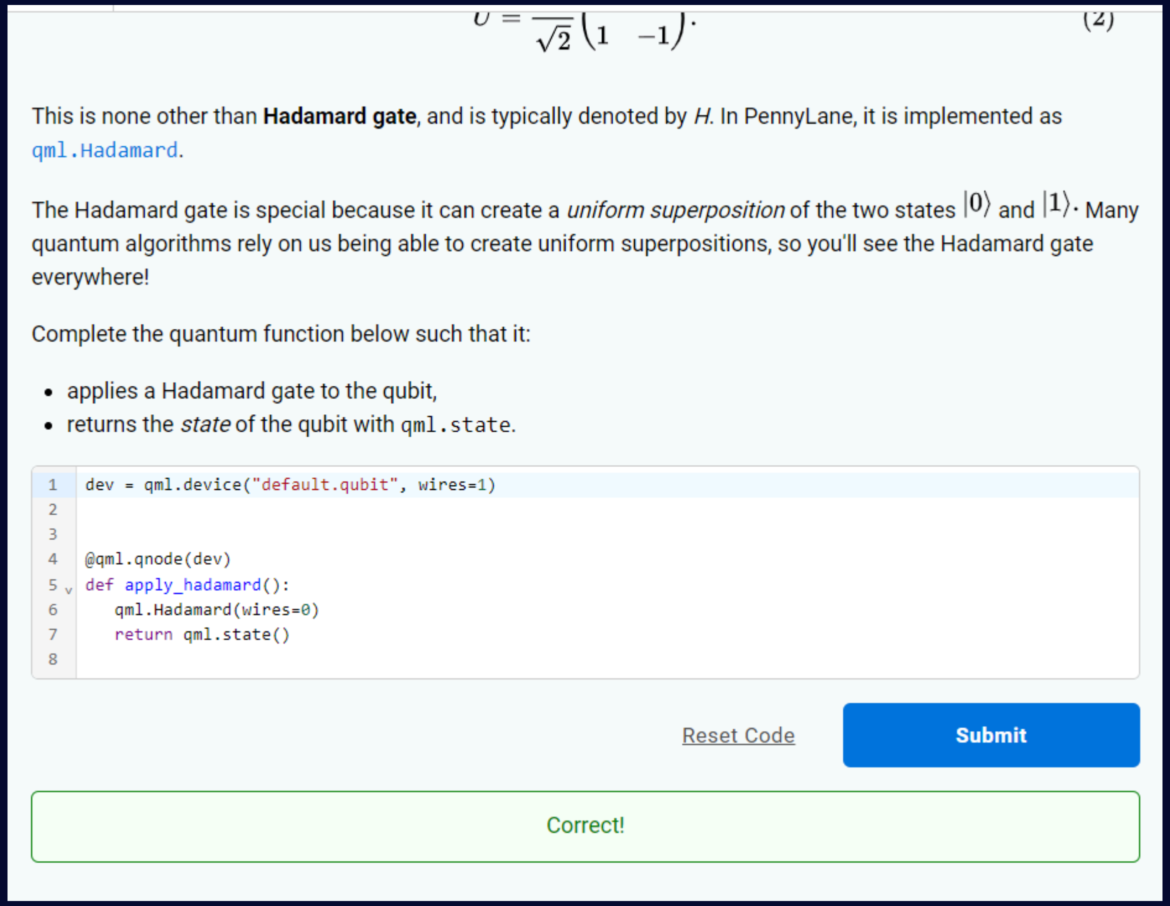


5

Implement a QML model to detect a defective production part.

[1]

BASIC TRAINING


$$U = \frac{1}{\sqrt{2}}\begin{pmatrix} \cdots \\ 1 & -1 \end{pmatrix}. \quad (2)$$

This is none other than **Hadamard gate**, and is typically denoted by *H*. In PennyLane, it is implemented as `qml.Hadamard`.

The Hadamard gate is special because it can create a *uniform superposition* of the two states $|0\rangle$ and $|1\rangle$. Many quantum algorithms rely on us being able to create uniform superpositions, so you'll see the Hadamard gate everywhere!

Complete the quantum function below such that it:

- applies a Hadamard gate to the qubit,
- returns the *state* of the qubit with `qml.state`.

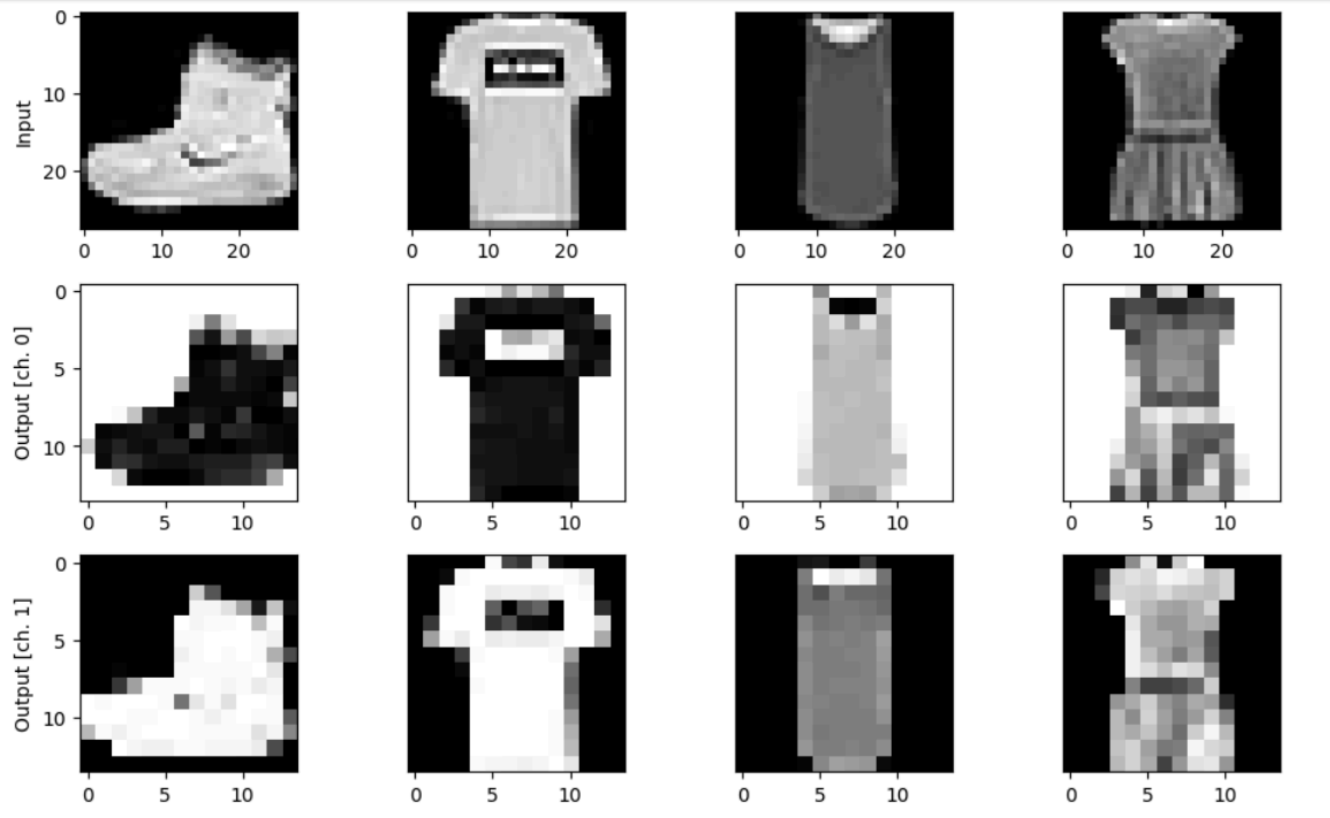
```
dev = qml.device("default.qubit", wires=1)


@qml.qnode(dev)
def apply_hadamard():
    qml.Hadamard(wires=0)
    return qml.state()

```

Reset Code Submit

Correct!

A code exercise from task 1

A convolutional layer applied to Fashion-MNIST, from task 3.

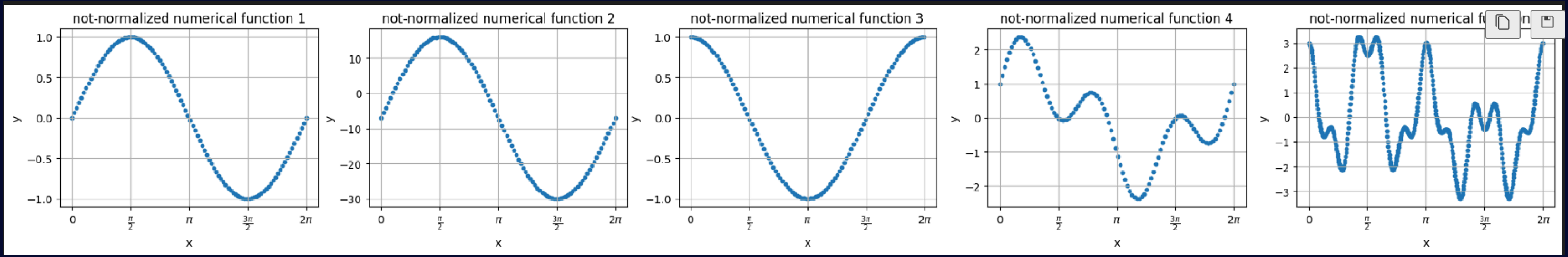


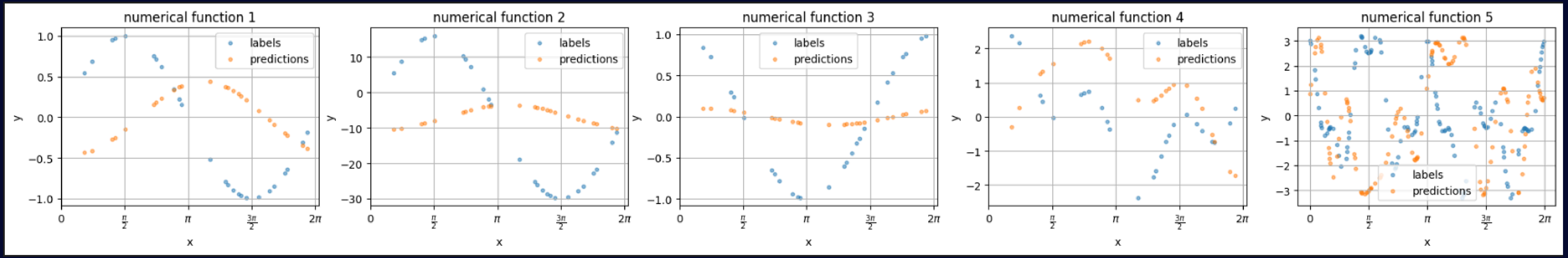
An optimisation of the variational classifier, from task 2.

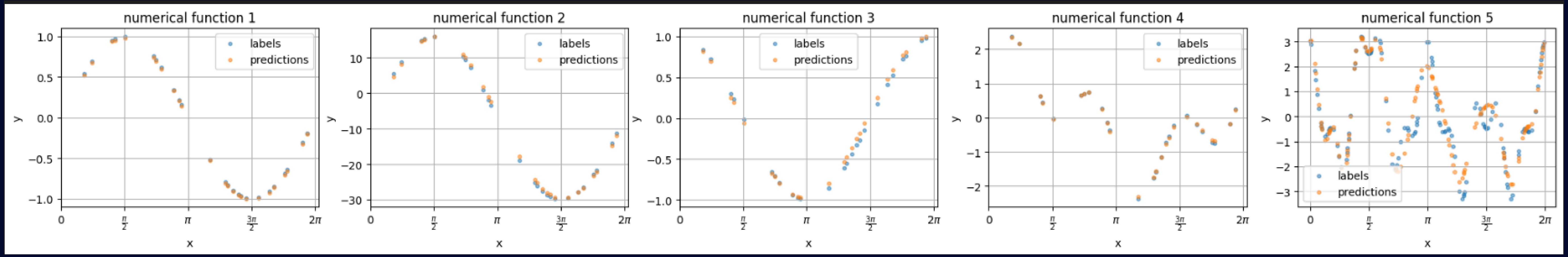
```
Iter:    1 | Cost: 1.8947039 | Accuracy: 0.6000000
Iter:    2 | Cost: 1.1471254 | Accuracy: 0.6000000
Iter:    3 | Cost: 0.9513056 | Accuracy: 0.6000000
Iter:    4 | Cost: 0.9555529 | Accuracy: 0.6000000
Iter:    5 | Cost: 1.1499226 | Accuracy: 0.4000000
Iter:    6 | Cost: 1.1009132 | Accuracy: 0.6000000
Iter:    7 | Cost: 1.1729802 | Accuracy: 0.6000000
Iter:    8 | Cost: 0.9643383 | Accuracy: 0.4000000
Iter:    9 | Cost: 0.5004217 | Accuracy: 1.0000000
Iter:   10 | Cost: 0.1876641 | Accuracy: 1.0000000
Iter:   11 | Cost: 0.0380154 | Accuracy: 1.0000000
Iter:   12 | Cost: 0.0190455 | Accuracy: 1.0000000
Iter:   13 | Cost: 0.0274739 | Accuracy: 1.0000000
Iter:   14 | Cost: 0.0559909 | Accuracy: 1.0000000
Iter:   15 | Cost: 0.0421825 | Accuracy: 1.0000000
Iter:   16 | Cost: 0.0274221 | Accuracy: 1.0000000
Iter:   17 | Cost: 0.0211031 | Accuracy: 1.0000000
Iter:   18 | Cost: 0.0097764 | Accuracy: 1.0000000
Iter:   19 | Cost: 0.0034671 | Accuracy: 1.0000000
Iter:   20 | Cost: 0.0023080 | Accuracy: 1.0000000
Iter:   21 | Cost: 0.0017877 | Accuracy: 1.0000000
Iter:   22 | Cost: 0.0023925 | Accuracy: 1.0000000
Iter:   23 | Cost: 0.0027881 | Accuracy: 1.0000000
Iter:   24 | Cost: 0.0029316 | Accuracy: 1.0000000
Iter:   25 | Cost: 0.0037595 | Accuracy: 1.0000000
...
Iter:   37 | Cost: 0.0001552 | Accuracy: 1.0000000
Iter:   38 | Cost: 0.0001866 | Accuracy: 1.0000000
Iter:   39 | Cost: 0.0002715 | Accuracy: 1.0000000
Iter:   40 | Cost: 0.0001781 | Accuracy: 1.0000000
```

QUANTUM MODEL FOR TRIGONOMETRICAL FUNCTIONS REGRESSION

```
def function_predictor(weights, bias, x):
    return circuit(weights, x) + bias

dev = qml.device("default.qubit")
@qml.qnode(dev)
def circuit(weights, x):
    qml.Hadamard(wires=0)

    for layer_weights in weights:
        layer(layer_weights,x)

    return qml.expval(qml.PauliZ(0))

def layer(weights,x):
    qml.RZ(x + weights[0],wires=0)
    qml.RX(weights[1],wires=0)
```



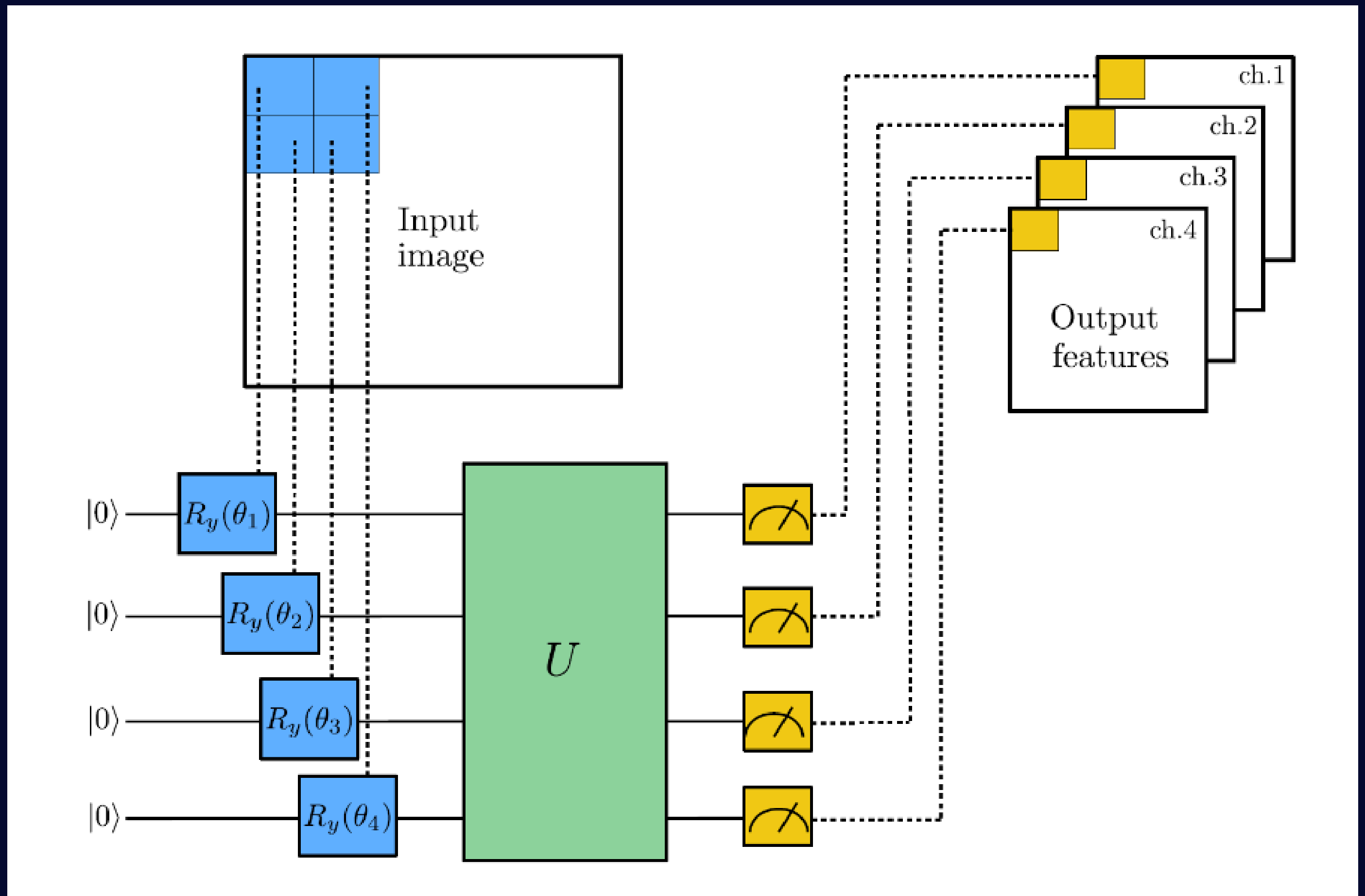




Womanium Quantum+AI Project

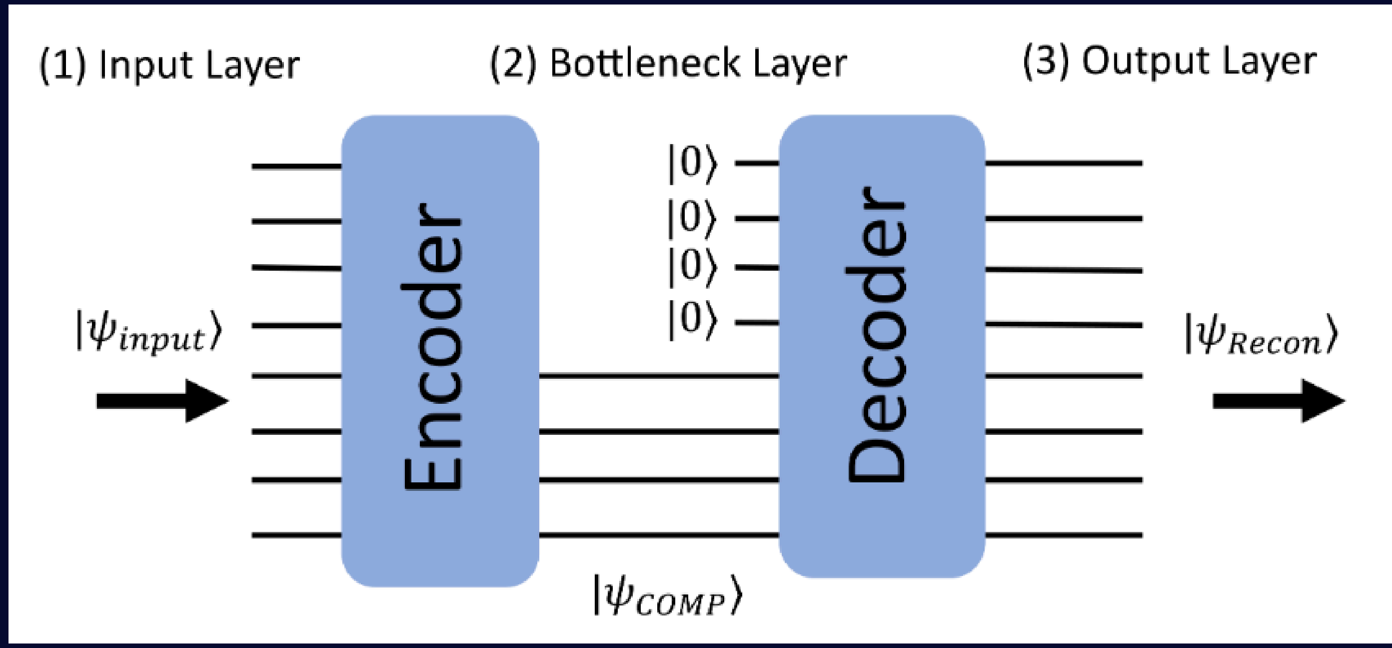
BINARY AND MULTI-CLASSIFICATION FOR CONSPICUITY DETECTION

QCNN (MULTI-CLASSIFICATION)

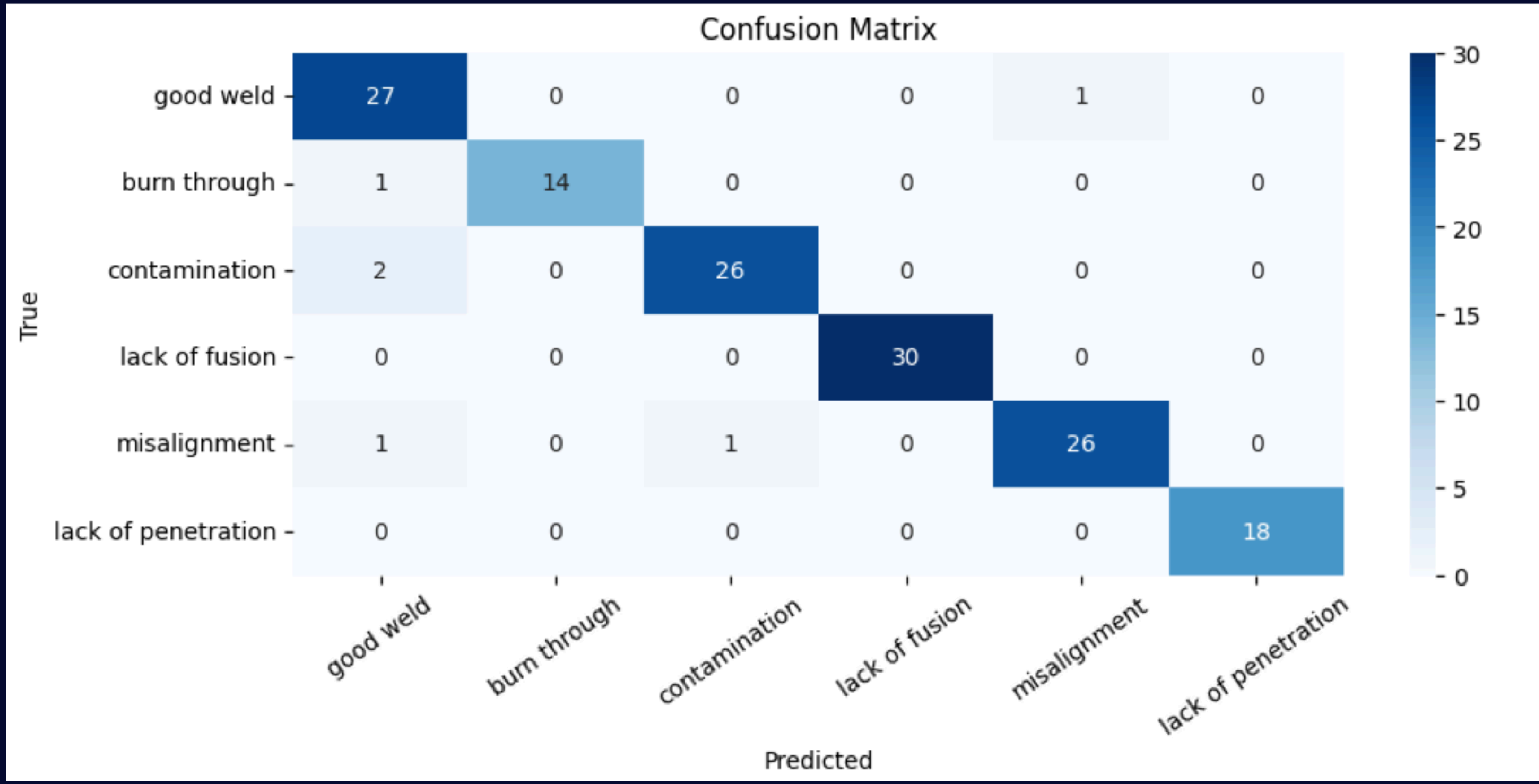


[2]

QUANTUM VARIATIONAL AUTOENCODER (BINARY CLASSIFICATION)

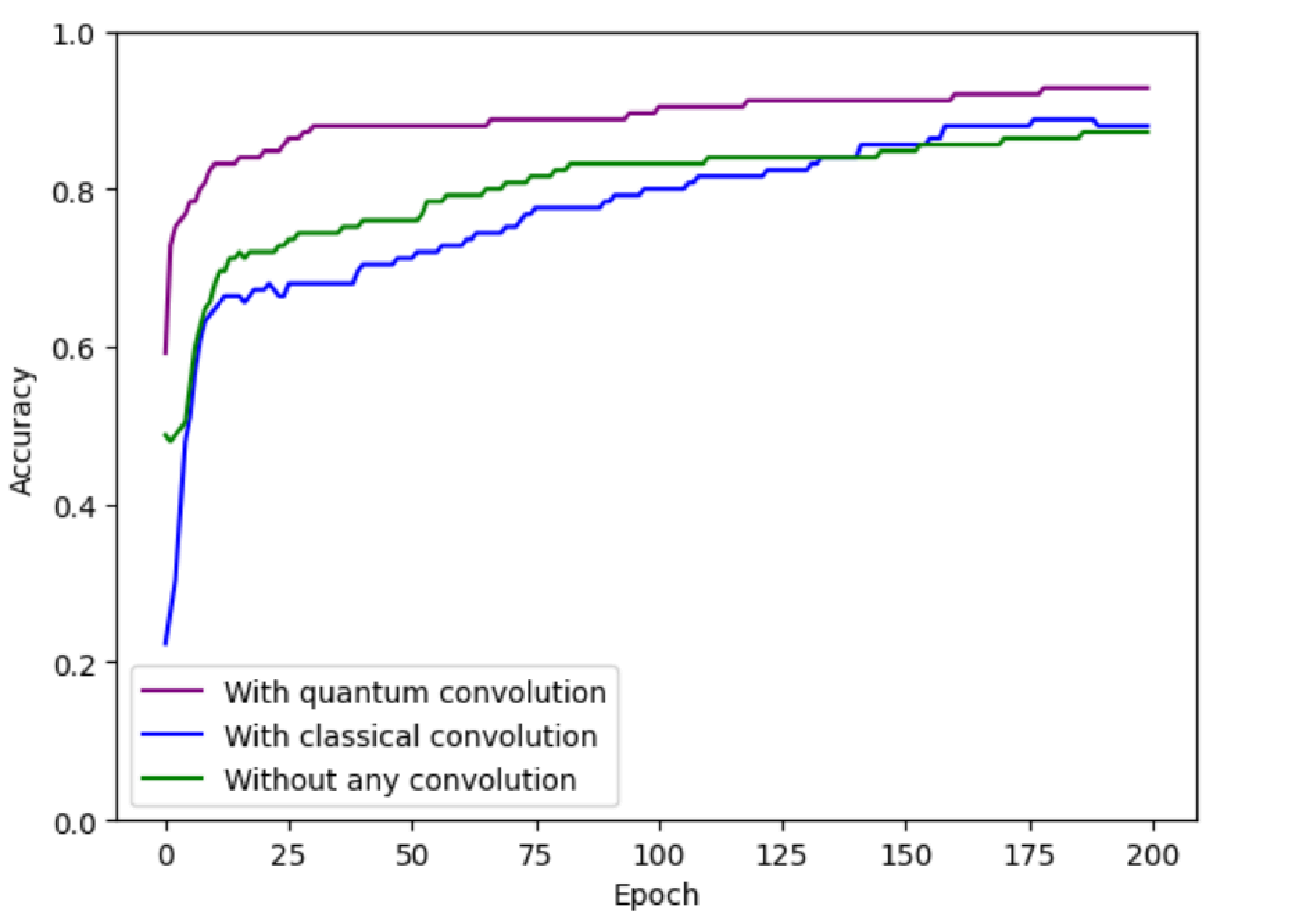


[3]

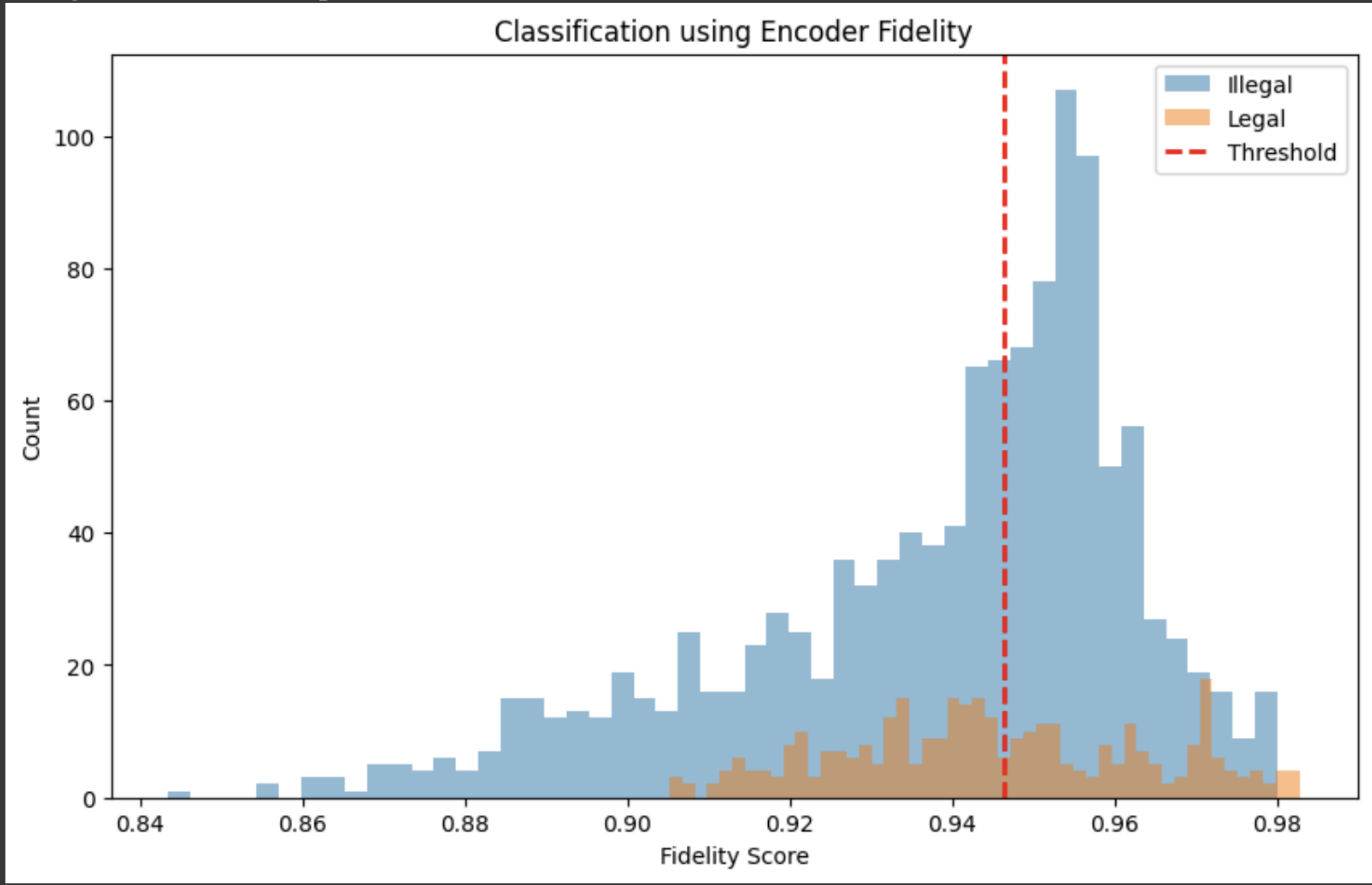


confussion matrix

accuracy history, compared to other classical models



```
Train Accuracy: 0.4672544002532959
Test Accuracy: 0.43988269567489624
Illegal Data Accuracy: 0.5232273936271667
```




```
Parameters:
Trash Bits: 5
Data Bits: 5
EPR Pairs: 1
Layers: 2
Training Epochs: 200
Batch Size: 5
Learning Rate: 0.01
```

REFERENCES

[1] https://www.kaggle.com/datasets/danielbacioiu/tig-aluminium-5083?resource=download
[2] https://pennylane.ai/qml/demos/tutorial_quanvolution/
[3] https://qiskit-community.github.io/qiskit-machine-learning/tutorials/12_quantum_autoencoder.html#8.-Applications-of-a-Quantum-Autoencoder