QUANTUM COMPUTING BASED ENSEMBLE CLASSIFICATION

Minor project report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

AYUSH RAJ	(19UECS0083)	**(VTU11836)**
ANSHUMAN RAJ	(19UECS0063)	**(VTU12952)**
RAHUL KUMAR THAKUR	(19UECS0816)	**(VTU13664)**

Under the guidance of
Dr. ARUN PANDIAN J ,M.E. Ph.D.,
ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A Grade
CHENNAI 600 062, TAMILNADU, INDIA

June , 2022

CERTIFICATE

It is certified that the work contained in the project report titled ”QUANTUM COMPUTING BASED ENSEMBLE CLASSIFICATION” by ”AYUSH RAJ (19UECS0083), ANSHUMAN RAJ (19UECS0063), RAHUL KUMAR THAKUR (19UECS0816)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Dr. Arun Pandian J
ASSOCIATE PROFESSOR
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr.Sagunthala R&D
Institute of Science & Technology
June,2022

Signature of Head of the Department
Dr. V. Srinivasa Rao
Professor & Head
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr.Sagunthala R&D
Institute of Science & Technology
June,2022

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)
AYUSH RAJ
Date: / /

(Signature)
ANSHUMAN RAJ
Date: / /

(Signature)
RAHUL KUMAR THAUR
Date: / /

APPROVAL SHEET

This project report entitled QUANTUM COMPUTING BASED ENSEMBLE CLASSIFICATION by (AYUSH RAJ (19UECS0083), ANSHUMAN RAJ (19UECS063), RAHUL KUMAR THAKUR (19UECS0816) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners **Supervisor**

Dr. Arun Pandian J , M.E. Ph.D.,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Dean & Head, Department of Computer Science & Engineering Dr.V.SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We also take this opportunity to express a deep sense of gratitude to our Internal Supervisor **Dr. ARUN PANDIAN J ,degree.,** for his cordial support, valuable information and guidance, he/she helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. C. SHYAMALA KUMARI, M.E., Ms.S.FLORENCE, M.Tech.,** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

AYUSH RAJ	**(19UECS0083)**
ANSHUMAN RAJ	**(19UECS0063)**
RAHUL KUMAR THAKUR	**(19UECS0816)**

ABSTRACT

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the classification performance. We use the forest.qvm device to simulate one QPU and the qiskit.aer device to simulate another. Each QPU makes an independent prediction, and an ensemble model is formed by choosing the prediction of the most confident QPU. The iris dataset is used in this project, consisting of three classes of iris flower.The TensorFlow quantum package was used to add the outcomes of the quantum circuits to dense layers for efficient classification. A different number of quantum filters were used in the HQCNN model and tested. A different number of quantum filters were used in the HQCNN model design. The performance of the HQCNN models with the different number of quantum filters was compared using validation and testing performance.

Keywords: Quantum Machine learning, Quantum Computing, Ensemble Classification, Iris Flower, Fully connected layer

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

IDE	Integrated Dvelopment Environment
OS	Operating system
QCNN	Quantum Convolutional Neural Network
QML	Quantum Machine Learning
QPU	Quantum Processing Unit
QVSM	Quantum Vector Support Machine
SPT	Symmetry Protected Topological

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

Quantum machine learning is a research area that explores the interplay of ideas from quantum computing and machine learning. Quantum machine learning extends the pool of hardware for machine learning by an entirely new type of computing device — the quantum computer. Information processing with quantum computers relies on substantially different laws of physics known as quantum theory We use the forest.qvm device to simulate one QPU and the qiskit.aer device to simulate another. Each QPU makes an independent prediction, and an ensemble model is formed by choosing the prediction of the most confident QPU. The iris dataset is used in this consisting of three classes of iris flower. Using a pre-trained model and the PyTorch interface, we'll see that ensembling allows the QPUs to specialize towards different classes.

In the modern viewpoint, quantum computers can be used and trained as like neural networks. We can systematically adapt the physical control parameters, such as an electromagnetic field strength or a laser pulse frequency, to solve a problem. For example, a trained circuit can be used to classify the content of images, and by encoding the image into the physical state of the device and taking measurements.

1.2 Aim of the project

To implement the ensample learning techniques using quantum computing approach. To enhance the model predictions using ensample learning on multi class classification

1.3 Project Domain

The project focuses on the Quantum Computing based ensemble classification.

1.4 Scope of the Project

Developing a novel ensample learning technique using quantum circuits and machine learning algorithms for image classification on quantum data. Develop multiple QPUs for classification applications

Chapter 2

LITERATURE REVIEW

Quantum computing is a recent computing technique inspired by quantum physics to solve computing problems using quantum state properties such as superposition, interference, and entanglement [1]. Artificial neural networks are powerful techniques to solve modern decision-making challenges such as classification, time series forecasting and natural language processing [2]. Also, advanced neural networks like Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN) need more powerful computational tools than traditional computing systems to handle complex data [3]. The combination of recent artificial.

Neural networks and quantum computing techniques may extend the decision-making capabilities of modern computers [4]. The integrated quantum states and convolutional neural network is named a hybrid quantum convolutional neural network (HQCNN). The quantum-inspired convolutional and pooling layers are the elements of the quantum filter. It performs the convolutional and pooling operations like standard CNNs[5]. Multiple numbers of quantum filters can be connected to the quantum state in the HQCNN model for the feature extraction process. The number of quantum filters for the HQCNN model should be optimized based on the dataset and architecture of the model.

[6] This research combines the quantum states and CNN with the optimized number of quantum filters to handle classification tasks. In this article, we begin with the short introduction and application of a hybrid quantum CNN. Section 2 reviews the recent developments and applications of hybrid quantum neural networks. Section 3 explains the construction of the proposed HQCNN model for the Symmetry-Protected Topological (SPT) phase classification task. Section 4 compares the performance of the proposed HQCNN using the different number of quantum filters on test data. At last, section 5 concludes the outcomes of the research with future suggestions. Quantum computing is used in numerous domains such as neural networks, cryptography and network security to improve performance and processing capabilities. This short literature survey focused on the applications of quantum computing in neural network development on classification tasks. Several neural network ap-

plications are developed using quantum circuits and states. Some of the applications are discussed in this section. The significance of the quantum inspired deep convolutional neural network (QDCNN) on image classification tasks is studied using standard datasets in [7,8]. The authors identified the QDCNN achieves better performance than the traditional CNN models on image classification applications. In [9], the authors proposed a quantum matched-filter technique and deep spiking neural network for segmentation of tumor region in the scanned images. They optimized the quantum matched-filter technique to improve the performance of the task. They achieved an average accuracy of 98.21 percent on test data. The authors in [10], proposed a remote sensed hyperspectral data classification technique using CNN model with quantum genetic technique. They used the quantum genetic technique for creating a sparse feature matrix for achieving improved performance on hyperspectral data classification. A conversational sentiment analysis technique using quantum-inspired CNN and Long short-term memory (LSTM) was proposed by the authors in [11]. The experimental result of the technique shows the importance of quantum-inspired neural network model development. In [12], the authors proposed a facial expressions detection technique using a quantum-inspired search technique. They named the technique is Quantum-inspired binary gravitational search technique to detect seven different emotions by the facial appearance. The authors in [13], proposed a quantuminspired Particle Swarm Optimization (PSO) technique for enhancing CNN models. The quantuminspired PSO optimized CNN model performs better than traditional CNN models. A Quantum Generative Adversarial Network (QGAN) to generate small gray-scale images was proposed by the authors in [14]. The image generation on high dimensional space was completed in parallel using the QGAN model. The next section briefly discussed the SPT phase dataset preparation, hybrid model development and the model training processes.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

A novel hybrid quantum convolutional neural network was proposed in this section to classify the binary class data. The TensorFlow quantum framework was used to implement the proposed HQCNN model for classification. The implementation of the proposed HQCNN was performed on a GPU environment. The implementation process consists of three steps. At first, the data collection step collects the STP phase data. In the second step, the quantum cluster was designed for the convolutional and pooling process. Finally, the HQCNN model was developed for the STP phase classification. The upcoming sub-sections were extensively discussed about each step of the implementation process.

3.2 Proposed System

To implement the ensample learning techniques using quantum computing ap- proach. To enhance the model predictions using ensample learning on multi class classification.Developing a novel ensample learning technique using quantum circuits and machine learning algorithms for image classification on quantum data. Develop multiple QPUs for classification applications

3.3 Feasibility Study

This phase examines the design's practicality, and business proposals are presented with a comprehensive design plan and some cost estimates. The feasibility study of the proposed system should be conducted during system analysis. This is to ensure that the planned system will not cause the organisation any problems. A basic understanding of the system's major conditions is required for feasibility analysis. The feasibility analysis takes into account three important factors.

3.3.1 Economic Feasibility

This project is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into their search and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

3.3.2 Technical Feasibility

This project is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.3.3 Social Feasibility

The aspect of the project is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.4 System Specification

3.4.1 Hardware Specification

- PROCESSOR:I3/INTEL PROCESSOR
- RAM:8GB
- HARDDISK:160GB

3.4.2 Software Specification

• OS: WINDOWS 10
• SERVER SCRIPT: PYTHON
• IDE: GOOGLE COLAB FOR RUNNING QUANTUM MACHINE LEARNING CODES

3.4.3 Standards and Policies

Anaconda Prompt

Anaconda prompt is a type of command line interface which explicitly deals with the ML(MachineLearning) modules.And navigator is available in all the Windows,Linux and MacOS.The anaconda prompt has many number of IDE's which make the coding easier. The UI can also be implemented in python.

Standard Used: ISO/IEC 10918-1:1994

Jupyter

It's like an open source web application that allows us to share and create the documents which contains the live code, equations, visualizations and narrative text. It can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning.

Standard Used: ISO/IEC WD TR 24772-4

Chapter 4

METHODOLOGY

4.1 General Architecture

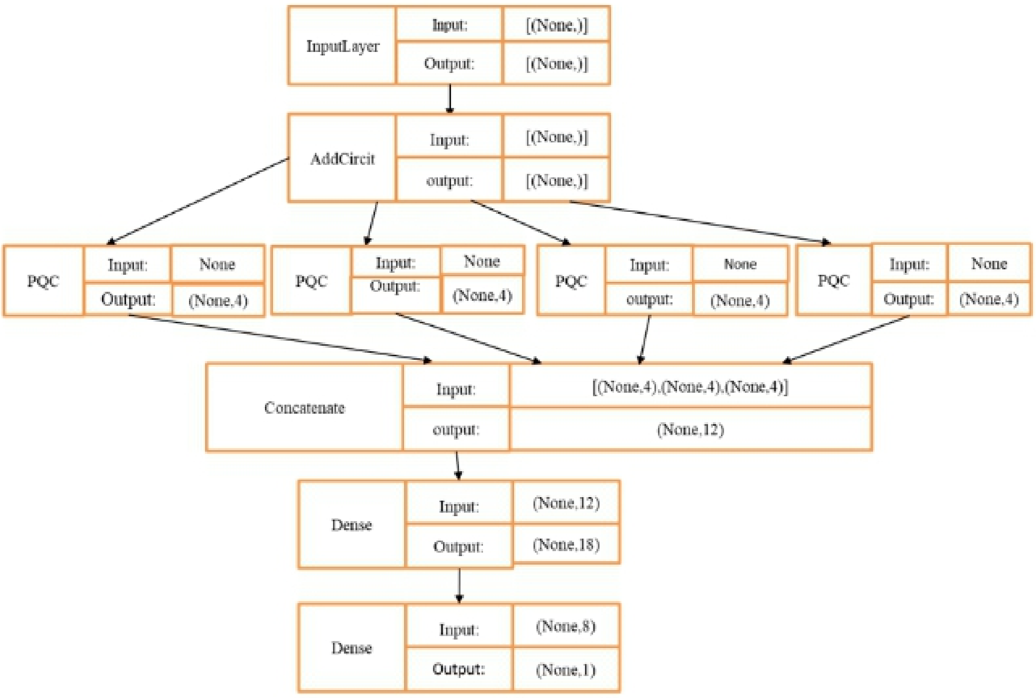


Figure 4.1: General Architecture

The HQCNN model was extended with four parallelconnected quantum filters. The model was named HQCNN4QF. The outputs of the four parallel-connected quantum filters are combined using a concatenation layer. The concatenation layer output was forwarded to the dense layers of the HQCNN-4QF model. Fig. 4.1 shows the layered architecture of the proposed HQCNN-4QF model for SPT phase data classification. The HQCNN-4QF model was trained to 300 epochs on a GPU environment. The accuracy of the HQCNN-4QF model on training and validation data

4.2 Design Phase

4.2.1 Data Flow Diagram

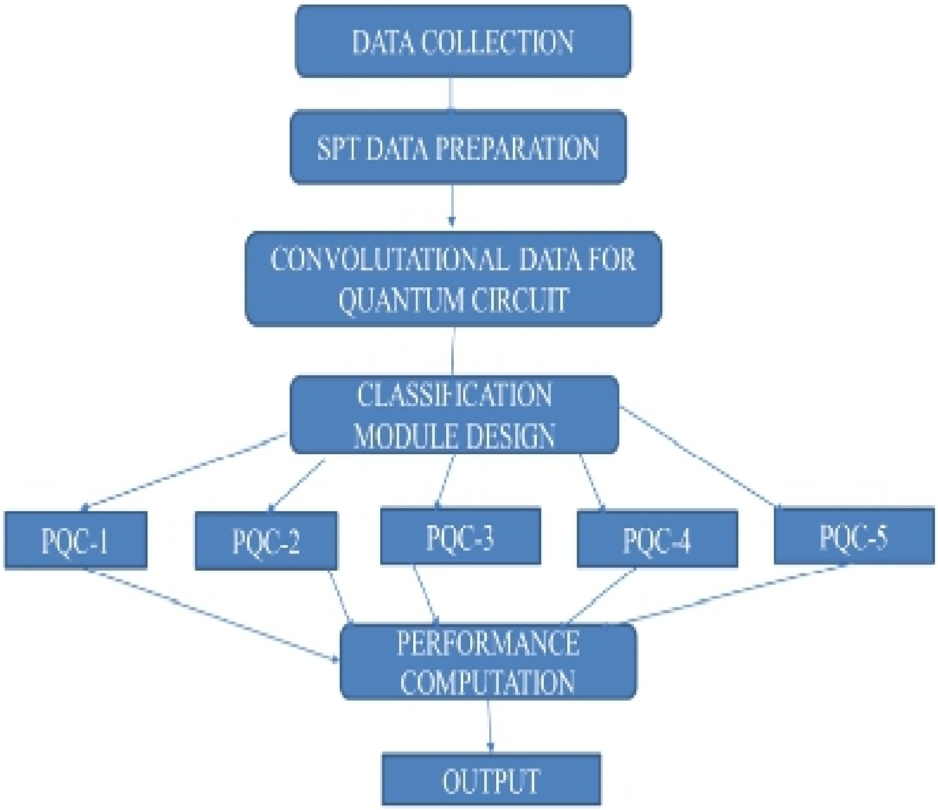


Figure 4.2: Data Flow Diagram

Once the data was collected and the data will go to Pre-processing model for the SPT data preparation. Then the data will pass through linear regressor and random cnvolutional neural network and the train and testing data will be achieved and the output will represent as graphical representation.

4.2.2 Use Case Diagram

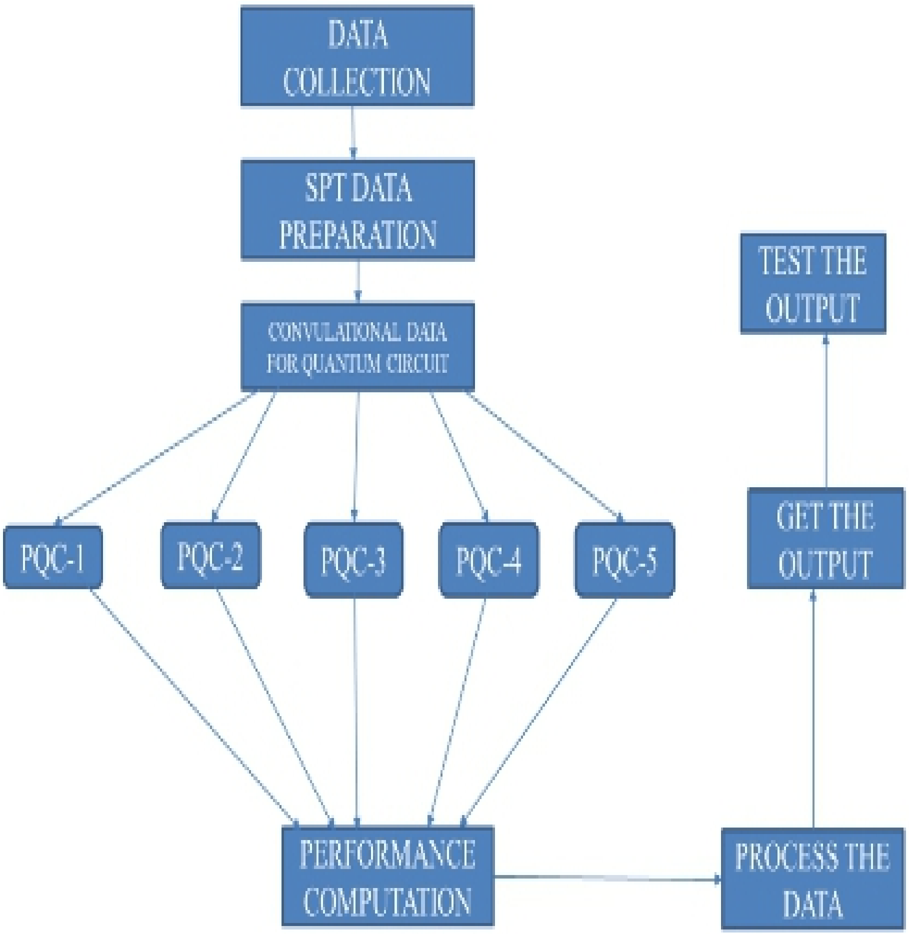


Figure 4.3: Use Case Diagram

All the models like SPT data preparation,Convolutional data for quantum circuit, PQC, Performance computation, process the data and result analysis are to be get the output and to be passed to the test the output.

4.2.3 Sequence Diagram

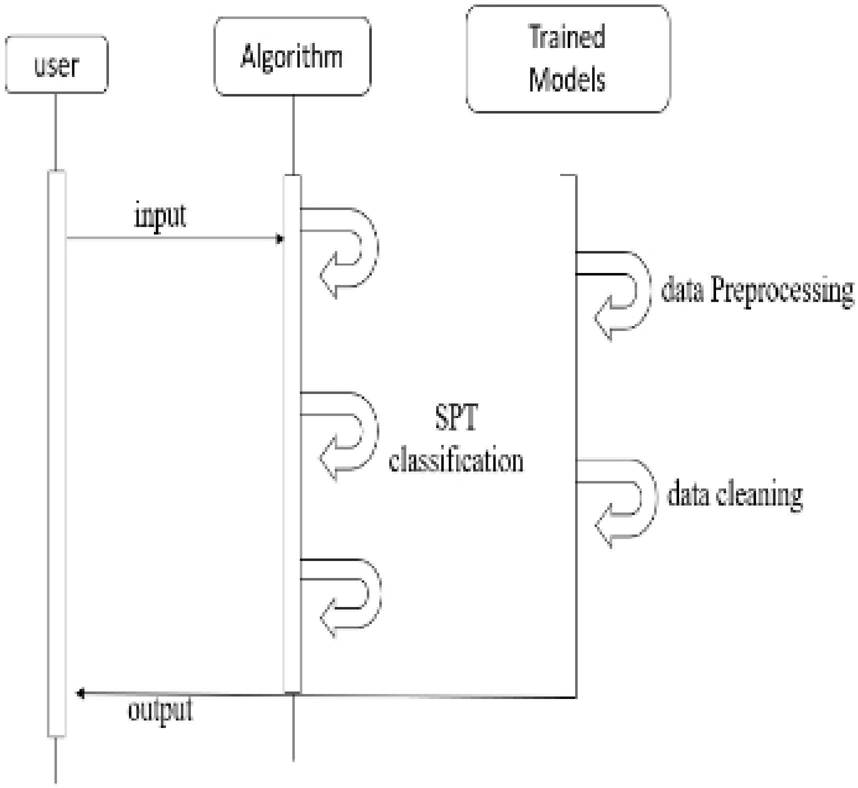


Figure 4.4: Sequence Diagram

The sequence diagram is a Unified Modeling Language diagram that illustrates the sequence of messages between objects in an interaction .The modules like data pre-processing,data cleaning, training, testing, predicting are to be done from user to system and further processing the system need to produce analyze the output and generate an output in graphical representation.

4.2.4 Collaboration diagram

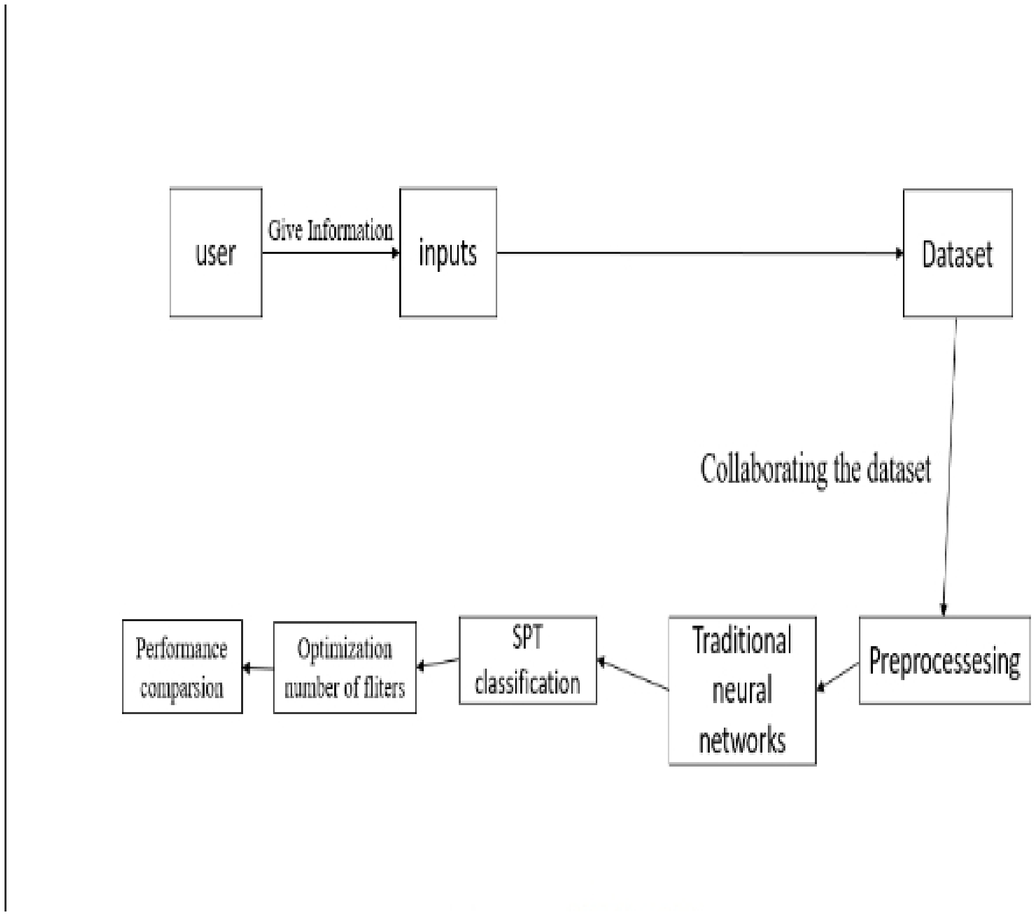


Figure 4.5: Collaboration diagram

The sequence diagram is a Unified Modeling Language diagram that illustrates the sequence of messages between objects in an interaction .The modules like data pre-processing,data cleaning, training, testing, predicting are to be done from user to system and further processing the system need to produce analyze the output and generate an output in graphical representation.

4.3 Algorithm & Pseudo Code

4.3.1 Algorithm

Step 1: Start the process. Step 2: Setting up tensor flow quantum and circuits.
Step 3: Generate Data using a Produces n rounds* n qubits datapoints.
Step 4: Training a Quantum CNN to Detect Excited Cluster States.
Step 5: Make a model circuit with less quantum pool and conv operations.

Step 6: Design quantum classifier with multiple PQC filters.
Step 7: comparison of proposed HQCNN with various number of filters.
Step 8: Repeat the above process upto the end of the dataset.
Step 9: Output generation process.
Step 10:End.

4.3.2 Pseudo Code

```
def create_model_circuit(qubits):
    model_circuit = cirq.Circuit()
    symbols = sympy.symbols('qconv0:63')
    # Cirq uses sympy.Symbols to map learnable variables. TensorFlow Quantum
    # scans incoming circuits and replaces these with TensorFlow variables.
    model_circuit += quantum_conv_circuit(qubits, symbols[0:15])
    model_circuit += quantum_pool_circuit(qubits[:4], qubits[4:],
                                          symbols[15:21])
    model_circuit += quantum_conv_circuit(qubits[4:], symbols[21:36])
    model_circuit += quantum_pool_circuit(qubits[4:6], qubits[6:],
                                          symbols[36:42])
    model_circuit += quantum_conv_circuit(qubits[6:], symbols[42:57])
    model_circuit += quantum_pool_circuit([qubits[6]], [qubits[7]],
                                          symbols[57:63])
    return model_circuit


cluster_state_bits = cirq.GridQubit.rect(1, 8)
readout_operators = cirq.Z(cluster_state_bits[-1])

excitation_input = tf.keras.Input(shape=(), dtype=tf.dtypes.string)
cluster_state = tfq.layers.AddCircuit()(
    excitation_input, prepend=cluster_state_circuit(cluster_state_bits))

quantum_model = tfq.layers.PQC(create_model_circuit(cluster_state_bits),
                               readout_operators)(cluster_state)

qcnn_model = tf.keras.Model(inputs=[excitation_input], outputs=[quantum_model])

tf.keras.utils.plot_model(qcnn_model,
                          show_shapes=True,
                          show_layer_names=False,
                          dpi=70)
```

4.4 Module Description

4.4.1 Design Quantum SVM Model

Quantum computing is a computing paradigm based on the laws of quantum mechanics, enabling a breakthrough in computing power. By carefully exploiting quantum effects such as interference or entanglement, quantum computers aim to efficiently solve particularly difficult problems that would be unsolvable for classical machines, with quantum advantages such as exponential acceleration. On the other hand, Quantum Machine Learning (QML) brings somewhat different research elements from the intersection with classical Machine Learning (ML) while using the computational advantage of quantum computing. There are many aspects and algorithms of QML, such as solving systems of linear equations, principal component analysis (QPCA) and support vector machines. In this article, we focus specifically on the Support Vector Machine (QSVM) model. Similar to support vector machines, the Quantum SVM algorithm (QSVM) is applied to classification problems that require a mapping of functions implicitly specified by a kernel (i.e., a function that is the inner product in the space of the functions being mapped represents). In particular, some previous papers analyze cases where the kernel computation is not classically efficient since it would scale exponentially with the size of the problem (i.e., large number of functions). In addition to speeding up kernel computation, other potential benefits of QSVM could include improved analysis performance (e.g., higher model accuracy), speedup of model training, and data protection.

4.4.2 Design Quantum Neural Network

Machine learning (ML), particularly applied to deep neural networks through the backpropagation algorithm, has enabled a wide range of revolutionary applications, ranging from social to scientific1,2. Achievements include the, which now provides daily handwriting and speech recognition for applications at the frontier of scientific research2-4. Despite rapid theoretical and practical advances, ML training algorithms are computationally intensive, and now that Moore's law is failing, we must look to a future with a slower progress rate5. However, exciting new possibilities are opening up due to the impending advent of quantum computing devices that directly exploit the laws of quantum mechanics to circumvent the technological and thermodynamic limitations of classical computing.

4.5 Steps to execute/run/implement the project

4.5.1 Connections and Importing

• Establish the connection between System with COLAB IDE for executing the modules.
• After establishing the connection import all the required data from dataset.

4.5.2 Graphical representation of data

• Now do all the required operations like generating graphical representations for the data.
• Generate the output as testcases for training and testing modules by using QCNN Analysis. Describe steps with title and mention steps in bullet points

4.5.3 Predicting the Training and Testing dataset Output

• Test all the training and testing modules for future enhancement and generate individual output for every testcase.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design

At first, a novel HQCNN model with one quantum filter was proposed for the STP phase data classification. This model combines a quantum filter with the traditional artificial neural network. After the quantum filter operation, the qubits are transferred to the fully connected neural network. The fully connected neural network contains two dense layers. The second dense layer was used to classify the data to respective labels. The layered architecture of the proposed HQCNN1QF was illustrated in fig.5.1.The training and validation set of the dataset was used to train the model. The HQCNN-1QF model was trained to 300 epochs on a GPU environment.

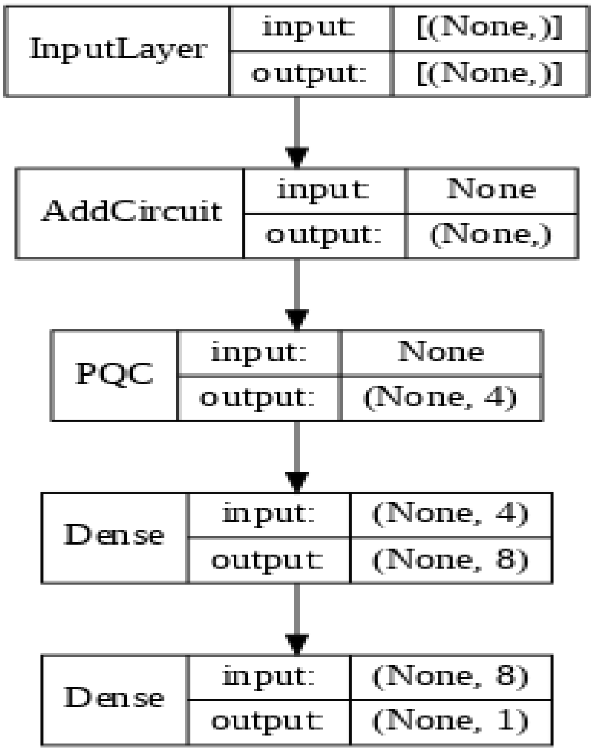


Figure 5.1: The layered architecture of the HQCNN-1QF model

5.1.2 Output Design

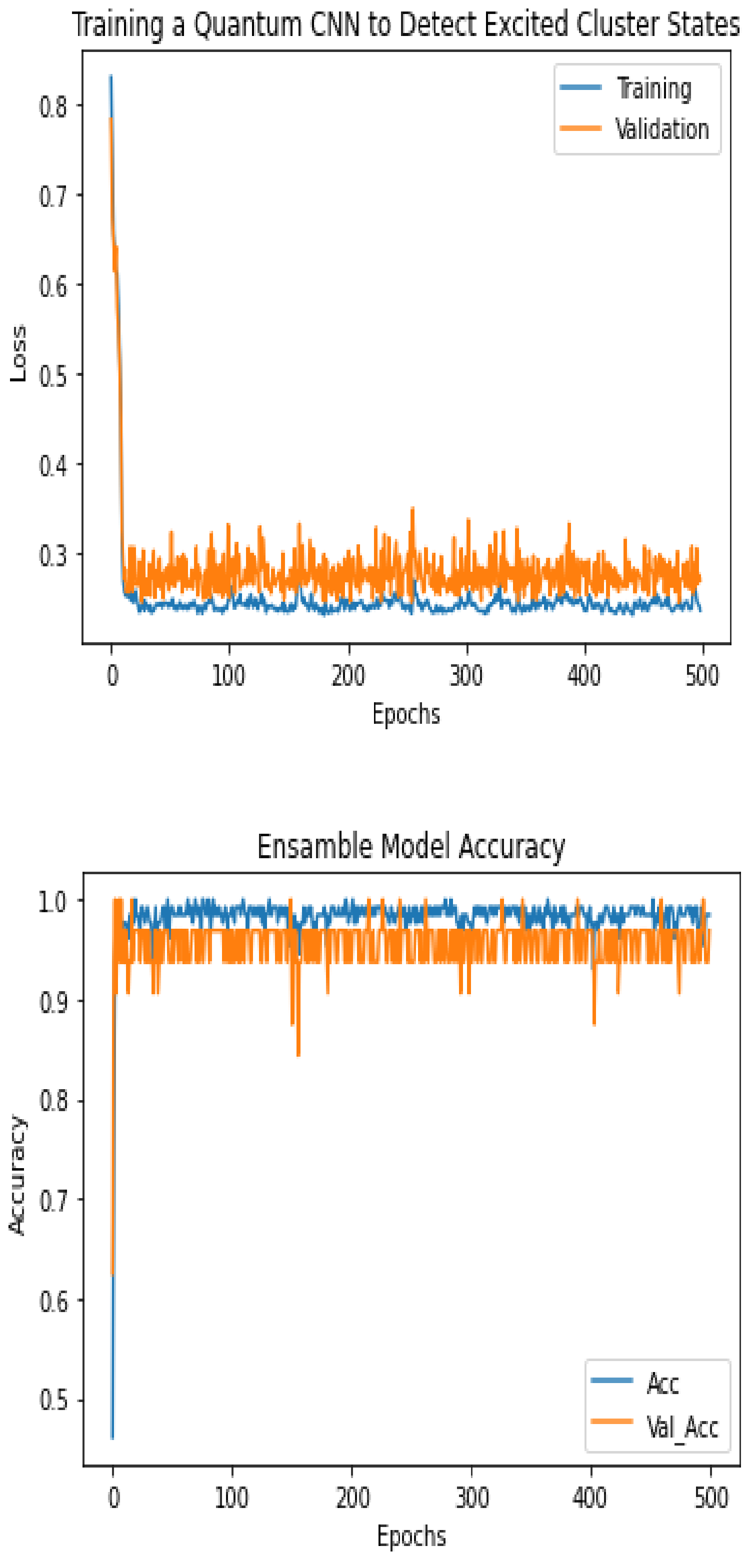


Figure 5.2: The layered architecture of the HQCNN-1QF model

5.2 Testing

Testing is that the method of evaluating a system is that the intent to seek out whether or not it satisfies the desired needs or not.The purpose of testing is to urge errors. Testing is that the strategy of making an endeavor to urge every conceivable fault or weakness in a very work product.It is the strategy of effort software with the intent of guaranteeing that the code meets its requirements associate degreed user expectations and doesn't fail in an unacceptable manner.

5.3 Types of Testing

5.3.1 Unit testing

Unit Testing is done for every model in the project. So, the models are data importing, data processing, data executing with the help of linear regressor, data analysis using PQC, analysing part with graphical notation. It was achieved by testing each module and need to check for correctness of model components.

Input

```
from collections import Counter

import matplotlib.pyplot as plt
import numpy as np
import pennylane as qml
import sklearn.datasets
import sklearn.decomposition
import torch
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
```

5.3.2 Integration testing

Input

```
def plot_points(x_train, y_train, x_test, y_test):
    c_train = []
    c_test = []

    for y in y_train:
```

```
        c_train.append(colours[y])

    for y in y_test:
        c_test.append(colours[y])

    plt.scatter(x_train[:, 0], x_train[:, 1], c=c_train)
    plt.scatter(x_test[:, 0], x_test[:, 1], c=c_test, marker="x")

    plt.xlabel("Feature 1", fontsize=16)
    plt.ylabel("Feature 2", fontsize=16)

    ax = plt.gca()
    ax.set_aspect(1)

    c_transparent = "#00000000"

    custom_lines = [
        Patch(facecolor=colours[0], edgecolor=c_transparent, label="Class 0"),
        Patch(facecolor=colours[1], edgecolor=c_transparent, label="Class 1"),
        Patch(facecolor=colours[2], edgecolor=c_transparent, label="Class 2"),
        Line2D([0], [0], marker="o", color=c_transparent, label="Train",
               markerfacecolor="black", markersize=10),
        Line2D([0], [0], marker="x", color=c_transparent, label="Test",
               markerfacecolor="black", markersize=10),
    ]

    ax.legend(handles=custom_lines, bbox_to_anchor=(1.0, 0.75))


plot_points(x_train, y_train, x_test, y_test)
plt.show()
```

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The quantum filters of the proposed HQCNN model were extended to three and it is named HQCNN-3QF in this section. The quantum filters are extracting features from the output of the AddCircuit layer. And combined using concatenate layer in the model. Each quantum filter produced 4 qubits of extracted information from the data. The layered architecture of the HQCNN-3QF was illustrated.

6.2 Comparison of Existing and Proposed System

SL.No	EXISTING SYSTEM	PROPOSED SYSTEM
1	It not based on quantum theory.	It is based on quantum theory.
2	Sends digital signals using bits.	Sends data through the use of particles or photons.
3	Operates is not extreme cold environments.	Operates in extreme cold environments.
4	Encryption is not based on quantum properties.	Encryption is based on quantum properties.

Table 6.1: Comparison of Existing and Proposed System

6.3 Sample Code

```
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import pennylane as qml
import sklearn.datasets
import sklearn.decomposition
import torch
from matplotlib.lines import Line2D
from matplotlib.patches import Patch
n_features = 2
n_classes = 3
n_samples = 150
```

```
data = sklearn.datasets.load_iris()
x = data["data"]
y = data["target"]
np.random.seed(1967)
x, y = zip(*np.random.permutation(list(zip(x, y))))

pca = sklearn.decomposition.PCA(n_components=n_features)
pca.fit(x)
x = pca.transform(x)
x_min = np.min(x, axis=0)
x_max = np.max(x, axis=0)

x = 2 * np.pi * (x - x_min) / (x_max - x_min) - np.pisplit = 125

x_train = x[:split]
x_test = x[split:]
y_train = y[:split]
y_test = y[split:]


colours = ["#ec6f86", "#4573e7", "#ad61ed"]


def plot_points(x_train, y_train, x_test, y_test):
    c_train = []
    c_test = []

    for y in y_train:
        c_train.append(colours[y])

    for y in y_test:
        c_test.append(colours[y])

    plt.scatter(x_train[:, 0], x_train[:, 1], c=c_train)
    plt.scatter(x_test[:, 0], x_test[:, 1], c=c_test, marker="x")

    plt.xlabel("Feature 1", fontsize=16)
    plt.ylabel("Feature 2", fontsize=16)

    ax = plt.gca()
    ax.set_aspect(1)

    c_transparent = "#00000000"

    custom_lines = [
        Patch(facecolor=colours[0], edgecolor=c_transparent, label="Class 0"),
        Patch(facecolor=colours[1], edgecolor=c_transparent, label="Class 1"),
        Patch(facecolor=colours[2], edgecolor=c_transparent, label="Class 2"),
        Line2D([0], [0], marker="o", color=c_transparent, label="Train",
```

```
                  markerfacecolor="black", markersize=10),
        Line2D([0], [0], marker="x", color=c_transparent, label="Test",
                  markerfacecolor="black", markersize=10),
    ]

    ax.legend(handles=custom_lines, bbox_to_anchor=(1.0, 0.75))


plot_points(x_train, y_train, x_test, y_test)
plt.show()
```

Output

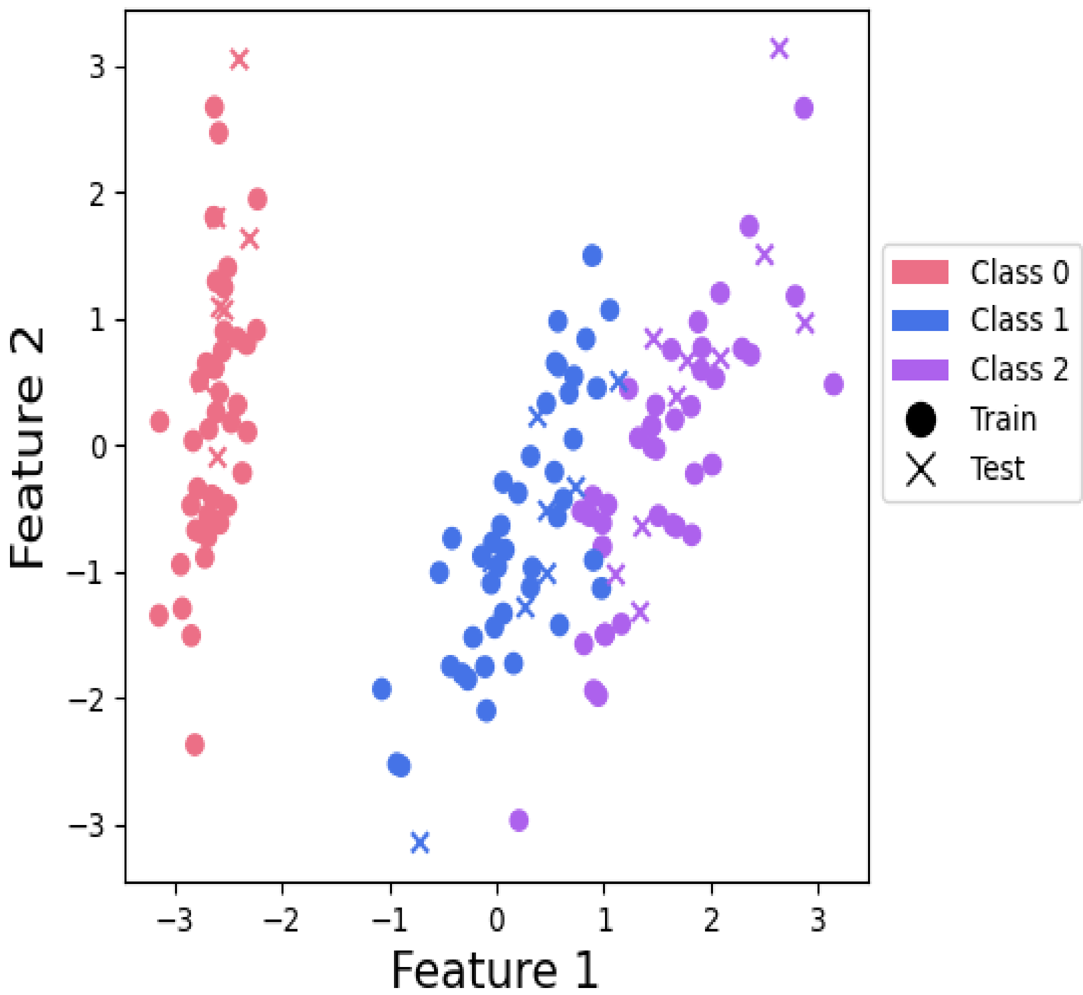


Figure 6.1: Ensemble classification with Forest and Qiskit devices

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

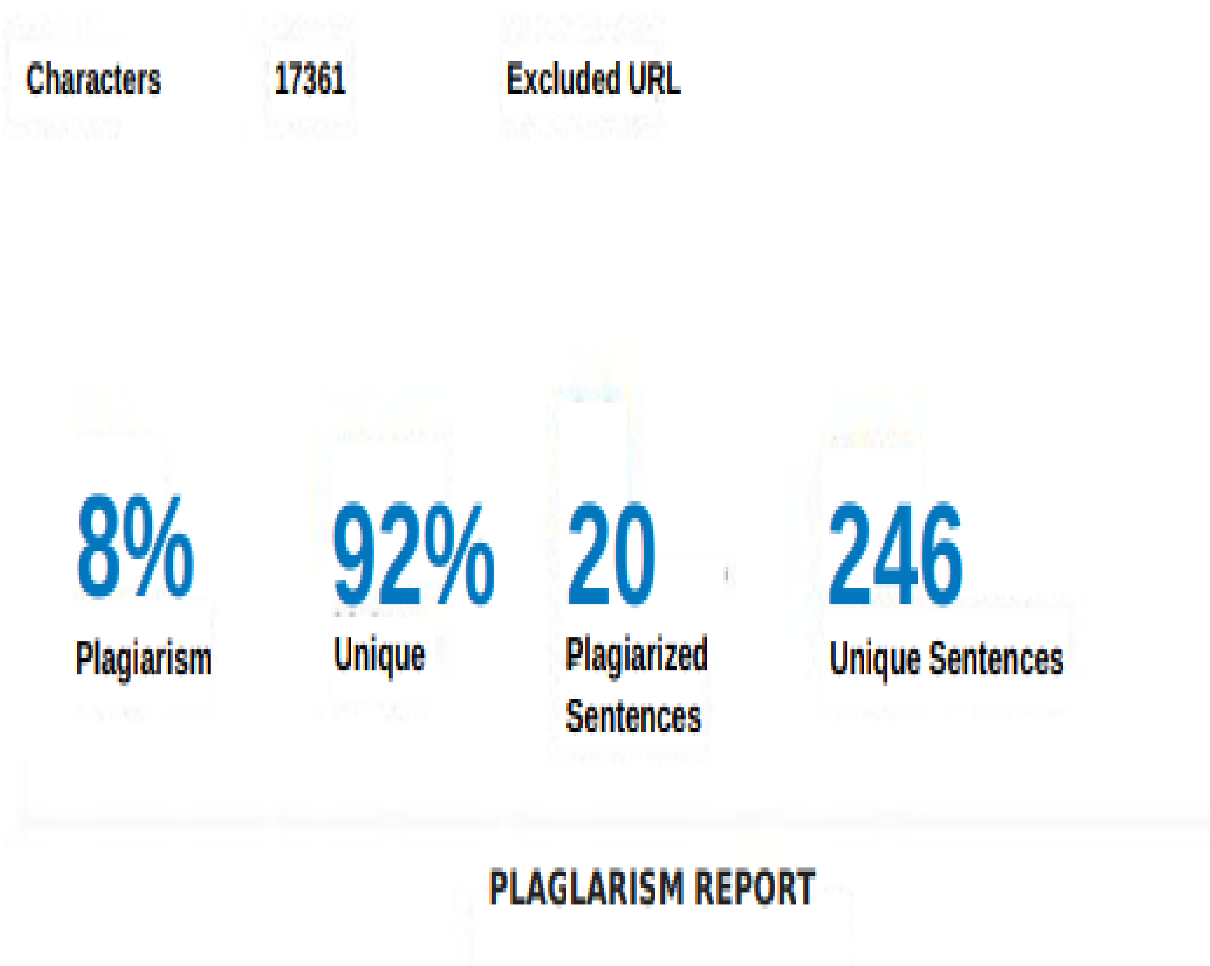
A Deep Convolutional Neural Network is a part of deep neural networks commonly used to classify multimedia data such as images and audios. The CNN requires high computation power to get trained on sample data for classification tasks. In addition, the high computational power requirement consumes more energy. A quantum computing technique provides high computing power with low energy consumption; it is the best alternative to traditional computing techniques to perform complex tasks.The proposed HQCNN combines a quantum filter with the traditional fully connected network for speeding up the training process. The quantum filter consists of QConv and QPool layers for feature extraction from data. The fully connected neural network was introduced after the quantum filter process. The number of quantum filters was optimized using their validation performance. In the future, the research will focus on the optimization of the number of filters and layers in the traditional neural network part of the HQCNN model. Also, the HQCNN model will extend to some other classification datasets

7.2 Future Enhancements

To Develop a quantum data using Projected Quantum Kernel features form image dataset.To Design a novel hybrid classification technique using quantum circuits and convolutional neural network for image classification.To Analysis the performance of the hybrid QCNN network and classical CNN networks on image classification.Developing a novel hybrid classification technique using quantum circuits and convolutional neural network for image classification on quantum data.Identify the importance of the quantum filter numbers on classification model design.

Chapter 8

PLAGIARISM REPORT



Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
!pip install tensorflow==2.7.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow==2.7.0
  Downloading https://us-python.pkg.dev/colab-wheels/public/tensorflow/tensorflow-2.7.0%2
      Bzzzcolab20220506150900-cp37-cp37m-linux_x86_64.whl
     \ 665.5 MB 8.2 MB/s
Successfully installed gast-0.4.0 keras-2.7.0 tensorflow-2.7.0+zzzcolab20220506150900 tensorflow-
    estimator-2.7.0
!pip install tensorflow-quantum
      Successfully uninstalled google-api-core-1.31.6
ERROR: pip's dependency resolver does not currently take into account all the packages that are
    installed. This behaviour is the source of the following dependency conflicts.
pydata-google-auth 1.4.0 requires google-auth<3.0dev,>=1.25.0; python_version >= "3.6", but you have
     google-auth 1.18.0 which is incompatible.
google-cloud-bigquery-storage 1.1.1 requires google-api-core[grpc
    ]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5, but you have google-api-core 1.21.0 which
    is incompatible.
Successfully installed backports.cached-property-1.0.1 cirq-core-0.14.1 cirq-google-0.14.1 duet
    -0.2.6 google-api-core-1.21.0 google-auth-1.18.0 googleapis-common-protos-1.52.0 sympy-1.8
    tensorflow-quantum-0.6.1 typing-extensions-3.10.0.0
import importlib, pkg_resources
importlib.reload(pkg_resources)
<module 'pkg_resources' from '/usr/local/lib/python3.7/dist-packages/pkg_resources/__init__.py'>
import tensorflow as tf
import tensorflow_quantum as tfq

import cirq
import sympy
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
from cirq.contrib.svg import SVGCircuit
qubit = cirq.GridQubit(0, 0)

```

```
circuit1 = cirq.Circuit(cirq.X(qubit))
circuit2 = cirq.Circuit(cirq.H(qubit))

input_circuit_tensor = tfq.convert_to_tensor([circuit1, circuit2])

y_circuit = cirq.Circuit(cirq.Y(qubit))

y_appender = tfq.layers.AddCircuit()

output_circuit_tensor = y_appender(input_circuit_tensor, append=y_circuit)
print(tfq.from_tensor(input_circuit_tensor))
[cirq.Circuit([
     cirq.Moment(
         cirq.X(cirq.GridQubit(0, 0)),
     ),
 ])
 cirq.Circuit([
     cirq.Moment(
         cirq.H(cirq.GridQubit(0, 0)),
     ),
 ])                                    ]

print(tfq.from_tensor(output_circuit_tensor))
[cirq.Circuit([
     cirq.Moment(
         cirq.X(cirq.GridQubit(0, 0)),
     ),
     cirq.Moment(
         cirq.Y(cirq.GridQubit(0, 0)),
     ),
 ])
 cirq.Circuit([
     cirq.Moment(
         cirq.H(cirq.GridQubit(0, 0)),
     ),
     cirq.Moment(
         cirq.Y(cirq.GridQubit(0, 0)),
     ),
 ])                                    ]
 def generate_data(qubits):
    n_rounds = 20  # Produces n_rounds * n_qubits datapoints.
    excitations = []
    labels = []
    for n in range(n_rounds):
        for bit in qubits:
            rng = np.random.uniform(-np.pi, np.pi)
            excitations.append(cirq.Circuit(cirq.rx(rng)(bit)))
            labels.append(1 if (-np.pi / 2) <= rng <= (np.pi / 2) else -1)

    split_ind = int(len(excitations) * 0.8)
```

```
    train_excitations = excitations[:split_ind]
    test_excitations = excitations[split_ind:]

    train_labels = labels[:split_ind]
    test_labels = labels[split_ind:]

    return tfq.convert_to_tensor(train_excitations), np.array(train_labels), \
        tfq.convert_to_tensor(test_excitations), np.array(test_labels)
        sample_points, sample_labels, _, __ = generate_data(cirq.GridQubit.rect(1, 4))
print('Input:', tfq.from_tensor(sample_points)[0], 'Output:', sample_labels[0])
print('Input:', tfq.from_tensor(sample_points)[1], 'Output:', sample_labels[1])
Input: (0, 0):     X     ^-0.124          Output: 1
Input: (0, 1):     X     ^0.394          Output: 1
def cluster_state_circuit(bits):
    circuit = cirq.Circuit()
    circuit.append(cirq.H.on_each(bits))
    for this_bit, next_bit in zip(bits, bits[1:] + [bits[0]]):
        circuit.append(cirq.CZ(this_bit, next_bit))
    return circuit
SVGCircuit(cluster_state_circuit(cirq.GridQubit.rect(1, 4)))
def one_qubit_unitary(bit, symbols):
    return cirq.Circuit(
        cirq.X(bit)**symbols[0],
        cirq.Y(bit)**symbols[1],
        cirq.Z(bit)**symbols[2])


def two_qubit_unitary(bits, symbols):
    circuit = cirq.Circuit()
    circuit += one_qubit_unitary(bits[0], symbols[0:3])
    circuit += one_qubit_unitary(bits[1], symbols[3:6])
    circuit += [cirq.ZZ(*bits)**symbols[6]]
    circuit += [cirq.YY(*bits)**symbols[7]]
    circuit += [cirq.XX(*bits)**symbols[8]]
    circuit += one_qubit_unitary(bits[0], symbols[9:12])
    circuit += one_qubit_unitary(bits[1], symbols[12:])
    return circuit


def two_qubit_pool(source_qubit, sink_qubit, symbols):
    pool_circuit = cirq.Circuit()
    sink_basis_selector = one_qubit_unitary(sink_qubit, symbols[0:3])
    source_basis_selector = one_qubit_unitary(source_qubit, symbols[3:6])
    pool_circuit.append(sink_basis_selector)
    pool_circuit.append(source_basis_selector)
    pool_circuit.append(cirq.CNOT(control=source_qubit, target=sink_qubit))
    pool_circuit.append(sink_basis_selector**-1)
    return pool_circuit
SVGCircuit(one_qubit_unitary(cirq.GridQubit(0, 0), sympy.symbols('x0:3')))
SVGCircuit(two_qubit_unitary(cirq.GridQubit.rect(1, 2), sympy.symbols('x0:15')))
```

```
SVGCircuit(two_qubit_pool(*cirq.GridQubit.rect(1, 2), sympy.symbols('x0:6')))
def quantum_conv_circuit(bits, symbols):
    circuit = cirq.Circuit()
    for first, second in zip(bits[0::2], bits[1::2]):
        circuit += two_qubit_unitary([first, second], symbols)
    for first, second in zip(bits[1::2], bits[2::2] + [bits[0]]):
        circuit += two_qubit_unitary([first, second], symbols)
    return circuit
SVGCircuit(
    quantum_conv_circuit(cirq.GridQubit.rect(1, 8), sympy.symbols('x0:15')))
def quantum_pool_circuit(source_bits, sink_bits, symbols):
    circuit = cirq.Circuit()
    for source, sink in zip(source_bits, sink_bits):
        circuit += two_qubit_pool(source, sink, symbols)
    return circuit
test_bits = cirq.GridQubit.rect(1, 8)

SVGCircuit(
    quantum_pool_circuit(test_bits[:4], test_bits[4:], sympy.symbols('x0:6')))
def create_model_circuit(qubits):
    model_circuit = cirq.Circuit()
    symbols = sympy.symbols('qconv0:63')
    # Cirq uses sympy.Symbols to map learnable variables. TensorFlow Quantum
    # scans incoming circuits and replaces these with TensorFlow variables.
    model_circuit += quantum_conv_circuit(qubits, symbols[0:15])
    model_circuit += quantum_pool_circuit(qubits[:4], qubits[4:],
                                          symbols[15:21])
    model_circuit += quantum_conv_circuit(qubits[4:], symbols[21:36])
    model_circuit += quantum_pool_circuit(qubits[4:6], qubits[6:],
                                          symbols[36:42])
    model_circuit += quantum_conv_circuit(qubits[6:], symbols[42:57])
    model_circuit += quantum_pool_circuit([qubits[6]], [qubits[7]],
                                          symbols[57:63])
    return model_circuit


cluster_state_bits = cirq.GridQubit.rect(1, 8)
readout_operators = cirq.Z(cluster_state_bits[-1])

excitation_input = tf.keras.Input(shape=(), dtype=tf.dtypes.string)
cluster_state = tfq.layers.AddCircuit()(
    excitation_input, prepend=cluster_state_circuit(cluster_state_bits))

quantum_model = tfq.layers.PQC(create_model_circuit(cluster_state_bits),
                               readout_operators)(cluster_state)

qcnn_model = tf.keras.Model(inputs=[excitation_input], outputs=[quantum_model])

tf.keras.utils.plot_model(qcnn_model,
                          show_shapes=True,
```

```
                          show_layer_names=False,
                          dpi=70)
train_excitations, train_labels, test_excitations, test_labels = generate_data(
    cluster_state_bits)


@tf.function
def custom_accuracy(y_true, y_pred):
    y_true = tf.squeeze(y_true)
    y_pred = tf.map_fn(lambda x: 1.0 if x >= 0 else -1.0, y_pred)
    return tf.keras.backend.mean(tf.keras.backend.equal(y_true, y_pred))


qcnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.02),
                   loss=tf.losses.mse,
                   metrics=[custom_accuracy])

history = qcnn_model.fit(x=train_excitations,
                         y=train_labels,
                         batch_size=16,
                         epochs=500,
                         verbose=1,
                         validation_data=(test_excitations, test_labels))
readouts = [cirq.Z(bit) for bit in cluster_state_bits[4:]]


def multi_readout_model_circuit(qubits):
    """Make a model circuit with less quantum pool and conv operations."""
    model_circuit = cirq.Circuit()
    symbols = sympy.symbols('qconv0:21')
    model_circuit += quantum_conv_circuit(qubits, symbols[0:15])
    model_circuit += quantum_pool_circuit(qubits[:4], qubits[4:],
                                          symbols[15:21])
    return model_circuit


excitation_input_dual = tf.keras.Input(shape=(), dtype=tf.dtypes.string)

cluster_state_dual = tfq.layers.AddCircuit()(
    excitation_input_dual, prepend=cluster_state_circuit(cluster_state_bits))

quantum_model_dual = tfq.layers.PQC(
    multi_readout_model_circuit(cluster_state_bits),
    readouts)(cluster_state_dual)

d1_dual = tf.keras.layers.Dense(8)(quantum_model_dual)

d2_dual = tf.keras.layers.Dense(1)(d1_dual)

hybrid_model = tf.keras.Model(inputs=[excitation_input_dual], outputs=[d2_dual])
```

```

tf.keras.utils.plot_model(hybrid_model,
                          show_shapes=True,
                          show_layer_names=False,
                          dpi=70)
hybrid_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.02),
                     loss=tf.losses.mse,
                     metrics=[custom_accuracy])

hybrid_history = hybrid_model.fit(x=train_excitations,
                                  y=train_labels,
                                  batch_size=16,
                                  epochs=500,
                                  verbose=1,
                                  validation_data=(test_excitations,
                                                   test_labels))
plt.plot(hybrid_history.history['custom_accuracy'], label='Acc')
plt.plot(hybrid_history.history['val_custom_accuracy'], label='Val_Acc')
plt.title('Ensamble Model Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.ylabel('Accuracy')
plt.show()
plt.plot(hybrid_history.history['loss'], label='Loss')
plt.plot(hybrid_history.history['val_loss'], label='Val_Loss')
plt.title('Ensamble Model Loss')
plt.xlabel('Epochs')
plt.legend()
plt.ylabel('Loss')
plt.show()
```

9.2 Poster Presentation

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

PROJECT TITLE

Department of Computer Science & Engineering
School of Computing
1156CS601 – MINOR PROJECT
WINTER SEMESTER 21-22

ABSTRACT

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the classification performance. We use the forest.qvm device to simulate one QPU and the qiskit.aer device to simulate another. Each QPU makes an independent prediction, and an ensemble model is formed by choosing the prediction of the most confident QPU. The iris dataset is used in this project, consisting of three classes of iris flower.The TensorFlow quantum package was used to add the outcomes of the quantum circuits to dense layers for efficient classification.

TEAM MEMBER DETAILS

vtu11836(Ayush Raj)
vtu12952(Anshuman Raj)
vtu13664(Rahul kumar Thakur)
<+91 6299552593>
<+91 6204158450>
<+91 9127579584>
vtu11836@veltech.edu.in
vtu12952@veltech.edu.in
vtu13664@veltech.edu.in

INTRODUCTION

Quantum machine learning is a research area that explores the interplay of ideas from quantum computing and machine learning. Quantum machine learning extends the pool of hardware for machine learning by an entirely new type of computing device — the quantum computer. Information processing with quantum computers relies on substantially different laws of physics known as quantum theory We use the forest.qvm device to simulate one QPU and the qiskit.aer device to simulate another. Each QPU makes an independent prediction, and an ensemble model is formed by choosing the prediction of the most confident QPU. The iris dataset is used in this consisting of three classes of iris flower. Using a pre-trained model and the PyTorch interface, we'll see that ensembling allows the QPUs to specialize towards different classes.

In the modern viewpoint, quantum computers can be used and trained as like neural networks. We can systematically adapt the physical control parameters, such as an electromagnetic field strength or a laser pulse frequency, to solve a problem. For example, a trained circuit can be used to classify the content of images, and by encoding the image into the physical state of the device and taking measurements

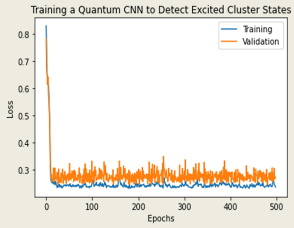
METHODOLOGIES

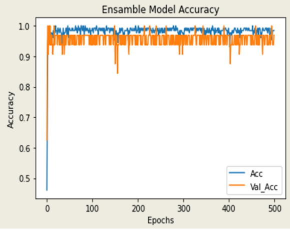
Quantum computing is a computing paradigm based on the laws of quantum mechanics, enabling a breakthrough in computing power. By carefully exploiting quantum effects such as interference or entanglement, quantum computers aim to efficiently solve particularly difficult problems that would be unsolvable for classical machines, with quantum advantages such as exponential acceleration. On the other hand, Quantum Machine Learning (QML) brings somewhat different research elements from the intersection with classical Machine Learning (ML) while using the computational advantage of quantum computing. There are many aspects and algorithms of QML, such as solving systems of linear equations, principal component analysis (QPCA) and support vector machines. In this article, we focus specifically on the Support Vector Machine (QSVM) model. Similar to support vector machines, the Quantum SVM algorithm (QSVM) is applied to classification problems that require a mapping of functions implicitly specified by a kernel (i.e., a function that is the inner product in the space of the functions being mapped represents). In particular, some previous papers analyze cases where the kernel computation is not classically efficient since it would scale exponentially with the size of the problem (i.e., large number of functions). In addition to speeding up kernel computation, other potential benefits of QSVM could include improved analysis performance (e.g., higher model accuracy), speedup of model training, and data protection

RESULTS

- The quantum filters of the proposed HQCNN model were extended to three and it is named HQCNN-3QF in this section.
- The quantum filters are extracting features from the output of the AddCircuit layer. And combined using concatenate layer in the model.
- Each quantum filter produced 4 qubits of extracted information from the data. The layered architecture of the HQCNN-3QF was illustrated

Table 1. The layered architecture of the HQCNN-1QF model





STANDARDS AND POLICIES

- Anaconda Prompt

Anaconda prompt is a type of command line interface which explicitly deals with the ML(MachineLearning) modules.And navigator is available in all the Windows,Linux and MacOS.The anaconda prompt has many number of IDE's which make the coding easier. The UI can also be implemented in python. Standard Used: ISO/IEC 10918-1:1994

- Jupyter

It's like an open source web application that allows us to share and create the documents which contains the live code, equations, visualizations and narrative text. It can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning. Standard Used: ISO/IEC WD TR 24772-4

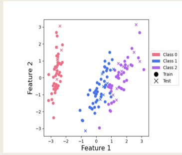


Figure 2 Ensemble classification with Forest and Qiskit devices

CONCLUSIONS

- A Deep Convolutional Neural Network is a part of deep neural networks commonly used to classify multimedia data such as images and audios.
- A quantum computing technique provides high computing power with low energy consumption; it is the best alternative to traditional computing techniques to perform complex tasks.
- The proposed HQCNN combines a quantum filter with the traditional fully connected network for speeding up the training process.
- The quantum filter consists of QConv and QPool layers for feature extraction from data. The fully connected neural network was introduced after the quantum filter process

ACKNOWLEDGEMENT

Dr. Arun Pandian J , M.E. Ph.D.,/ASSOCIATE PROFESSOR

+91 88078 13091

jarunpandian@veltech.edu.in

References

[1] Friis, N., Marty, O., Maier, C., Hempel, C., Holzäpfel, M., Jurcevic, P., Plenio, M.B., Huber, M., Roos, C., Blatt, R. and Lanyon, B., 2018. Observation of entangled states of a fully controlled 20-qubit system. Physical Review X, 8(2), p.021012.

[2] Gahi, Y., El Alaoui, I. and Guennoun, M., 2020. An End to End Cloud Computing Privacy Framework Using Blind Processing. International Journal of Smart Security Technologies (IJSST), 7(1), pp.1-20.

[3] A. W. Harrow, Small quantum computers and large classical data sets. arXiv preprint arXiv:2004.00026. (2020)

[4] Gustiani, C. and Bandung, I., 2020. Blind Oracular Quantum Computation: from Concept to Physical Implementation (Doctoral dissertation, Universitätsbibliothek der RWTH Aachen).

[5]] Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks," Nat. Phys., vol. 15, no. 12, pp. 1273–1278, 2019.

[6] Chen, F. Miao, and X. Shen, "Hyperspectral Remote Sensing Image Classification with CNN Based on Quantum Genetic-Optimized Sparse Representation," IEEE Access, vol. 8, pp. 99900–99909, 2020.

[7] H.-L. Huang et al., "Experimental Quantum Generative Adversarial Networks for Image Generation," Phys. Rev. Appl., vol. 16, no. 2, 2021.

[8]] J. Liu, K. H. Lim, K. L. Wood, W. Huang, C. Guo, and H.-L. Huang, "Hybrid quantum-classical convolutional neural networks," Sci. China Physics, Mech. Astron., vol. 64, no. 9, 2021.

[9] M. Henderson, S. Shakya, S. Pradhan, and T. Cook, “Quanvolutional neural networks: powering image recognition with quantum circuits,” Quantum Mach. Intell., vol. 2, no. 1, 2020.
=

[10] M. Ahmadi, A. Sharifi, S. Hassantabar, and S. Enayati, “QAIS-DSNN: Tumor Area Segmentation of MRI Image with Optimized Quantum Matched-Filter Technique and Deep Spiking Neural Network,” Biomed Res. Int., vol. 2021, 2021.

[11] N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, “Continuousvariable quantum neural networks,” Phys. Rev. Res., vol. 1, no. 3, 2019.

[12] T. E. Potok et al., “A study of complex deep learning networks on high performance, neuromorphic, and quantum computers,” in Proceedings of MLHPC 2016: Machine Learning in HPC Environments - Held in conjunction with SC 2016: The International Conference for High Performance Computing, Networking, Storage and Analysis, 2017, pp. 47–55.

[13] Y. Zhang, Q. Li, D. Song, P. Zhang, and P. Wang, “Quantum-inspired interactive networks for conversational sentiment analysis,” in IJCAI International Joint Conference on Artificial Intelligence, 2019, vol. 2019-Augus, pp. 5436–5442

[14]] Y. Li, R.-G. Zhou, R. Xu, J. Luo, and W. Hu, “A quantum deep convolutional neural network for image recognition,” Quantum Sci. Technol., vol. 5, no. 4, 2020.