UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

COMPARATIVE ANALYSIS OF QUANTUM ENCODING METHODS IN QUANTUM NEURAL NETWORKS FOR IMAGE CLASSIFICATION ON THE MNIST DATASET

ABUGA LINCOLN SIMBA

P52/11207/2018

SUPERVISOR: DR. STEPHEN NGANGA MBURU

A project report submitted in partial fulfillment of the requirement for the award of the degree of Master of Science in Computational Intelligence

September, 2023

DECLARATION

I affirm that I am the original author of this report. The material included in this document is entirely my own creation, unless explicitly indicated otherwise within the text. To the best of my understanding, this piece of work has not been presented for any other academic or professional credential, except as explicitly mentioned.

Name: Abuga Lincoln Simba **Registration Number:** P52/11207/2018

Signature:  **Date:** 29th August, 2023

This project report has been presented as a partial fulfillment of the prerequisites for conferring the degree of Master of Science in Computational Intelligence by the University of Nairobi. I, as the faculty supervisor, grant my approval for its submission.

Name: Dr. Stephen Nganga Mburu

Signature:  **Date:** 04/09/2023

ACKNOWLEDGEMENT

I express my heartfelt appreciation to God for His abundant blessings, unwavering strength, and invaluable guidance that have accompanied me on this academic journey.

I also extend my heartfelt appreciation to my family for their unending love, encouragement, and sacrifices. Their belief in me and unwavering support have been instrumental in my accomplishments, and I am truly blessed to have them by my side.

I wish to extend my genuine gratitude to the University of Nairobi for providing me with a conducive and stimulating environment to pursue my academic aspirations. The university's resources and commitment to excellence have played a significant role in shaping my academic journey.

Special gratitude goes to the School of Computing and Informatics at the University of Nairobi. The exceptional faculty and staff have enriched my knowledge and provided me with invaluable insights that have contributed to the successful completion of this research.

I am profoundly grateful to my supervisor, Dr. Stephen Nganga Mburu, for his exceptional guidance, expertise, and unwavering encouragement. His consistent support, invaluable insights, and mentorship have played a pivotal role in shaping the trajectory of my research and enhancing the caliber of this research paper.

Lastly, I wish to thank all my friends and colleagues who have been a source of inspiration and motivation throughout this endeavor.

ABSTRACT

Quantum computing, a paradigm that utilizes the concepts of quantum mechanics, has emerged as a transformative force in various fields, including machine learning. Within this setting, Quantum Neural Networks (QNNs) have gained attention for their potential to revolutionize image classification tasks.

Encoding plays a critical role in bridging the gap between classical data and quantum computation, allowing QNNs to utilize quantum resources thereby processing and manipulating classical data in a quantum-mechanical fashion.

In this research paper, we present a study of QNNs applied to image classification using the well-known MNIST dataset, with a specific focus on investigating the impact of four quantum encoding methods - Basis, Amplitude, Histogram and Pauli Basis Encoding - on the performance of these QNNs.

The research revealed that Basis Encoding and Pauli Basis Encoding outperformed Amplitude Encoding and Histogram Encoding across all the metrics tested. Overally, Pauli Basis Encoding exhibited the best results in enhancing the accuracy and validation loss of the QNNs

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1. BACKGROUND

Quantum computing, a paradigm based on the principles of quantum mechanics, has made significant advancements in recent times, promising unprecedented computational power and the potential to transform various scientific and technological domains (Liu and Huang 2021). Quantum computation harnesses quantum bits, or qubits, which can exist in superposition and entanglement, enabling the parallel processing of vast amounts of information and offering a remarkable advantage over classical computing.

QNNs represent a convergence of quantum computing methods and classical machine learning techniques, offering the possibility for significant computational advantages (Abbas et al., 2021). One of the key components of QNNs is the utilization of parameterized quantum circuits (PQCs), which are akin to classical neural networks(CNNs) and provide enhanced control and flexibility over quantum circuits (Benedetti, Lloyd, Sack, & Fiorentini, 2019). In QNNs, popular classical optimization routines, such as gradient-based methods, are employed to optimize the parameters of PQCs (Benedetti et al., 2019; Chen & Yoo, 2021). This hybrid approach has shown success in different applications (Benedetti et al., 2019; Chen & Yoo, 2021).

Recent research has demonstrated the remarkable capabilities of QNNs compared to CNNs, reaffirming their superiority in specific tasks (Abbas et al., 2021). A study showcased the impressive power of QNNs, revealing their potential to outperform comparable CNNs across various applications (Abbas et al., 2021).

Central to the success of QNNs is the process of quantum encoding, where classical data is represented in quantum states to be processed by quantum circuits. Quantum encoding acts as a vital link connecting classical information with quantum computing, enabling the seamless incorporation of classical data into quantum algorithms and Quantum Neural Networks (QNNs).

Several quantum encoding methods have been explored, each with its unique approach to encode classical information in quantum states. Notable quantum encoding techniques include Basis, Amplitude, Histogram and Pauli Basis Encoding. Basis Encoding maps classical information in quantum basis states, while Amplitude Encoding encodes classical information using quantum state amplitudes. Pauli Basis Encoding employs Pauli matrices to represent classical information in quantum states while Histogram encoding involves creating histograms of the pixel values in the classical image and using the histogram bins to represent the image data as quantum states.

In the architecture of QNNs, encoding plays a crucial role in preparing the input data for quantum computation. Quantum-encoded data is processed through a series of quantum gates, often implementing variational quantum circuits that adapt their parameters to optimize specific learning tasks. QNNs offer potential advantages in tackling complex problems, particularly in image classification and optimization tasks.

The choice of quantum encoding in QNNs significantly impacts the network's performance, convergence speed, and generalization capabilities. Identifying the most suitable encoding method for specific learning tasks is crucial for achieving the desired quantum advantage in quantum machine learning.

1.2. PROBLEM STATEMENT

Despite the promising potential of QNNs in revolutionizing image classification tasks through quantum computing, the optimal encoding method to use in representing classical image data in quantum states remains an open question. The performance of QNNs heavily relies on the choice of quantum encoding, which can significantly impact network performance, convergence, and generalization capabilities. With various quantum encoding methods available, there is a need to comprehensively compare and analyze their respective effects on QNNs for image classification tasks.

The existing studies (Nguyen et al., 2022; Sierra-Sosa et al., 2020) have made significant contributions by evaluating specific quantum encoding methods, but they do not address the comprehensive comparison of all relevant encoding techniques. (Nguyen et al. 2022) examined Qubit encoding and HE encoding on different quantum circuits, while Sierra-Sosa et al. (2020) focused on the benefits of Amplitude Encoding, leaving out other potential encoding methods. The gaps in the literature lie in understanding the full spectrum of quantum encoding techniques and their implications on QNNs' performance in image classification tasks.

This research aims at conducting a more comprehensive comparative analysis of multiple encoding methods, including Basis, Amplitude, Pauli Basis and Histogram Encoding in the context of QNNs for image classification on the MNIST dataset.

1.3. RESEARCH OBJECTIVES

1.3.1. Principal Objective

To compare and analyze the performance of different quantum encoding methods in QNNs for image classification.

1.3.2. Other Objectives

1. To identify encoding methods that can be used in QNNs for image classification.
2. To evaluate the training loss and accuracy of QNNs under each quantum encoding method identified.
3. To identify the encoding method that results in the best performance of the QNNs.

1.4. RESEARCH OUTCOME AND SIGNIFICANCE TO KEY AUDIENCES

This research will provide a comprehensive comparative analysis of various quantum encoding methods, including Basis, Amplitude, Pauli Basis, Histogram Encoding, and potentially other relevant techniques, in QNNs for image classification using the MNIST dataset. The study will provide a detailed evaluation of each encoding method's impact on QNNs' performance, including classification loss and accuracy.

Additionally, the research will identify the optimal quantum encoding method that yields the highest image classification performance in QNNs. It will also shed light on the strengths and limitations of each encoding technique, providing valuable insights into their practical implementation in quantum-based machine learning systems.

This research endeavor will make a substantial contribution to the progression of the quantum machine learning and quantum computing domains. It aims to foster a more profound comprehension of the influence of quantum encoding on the effectiveness and potential of quantum neural networks.

This research will provide valuable guidance for researchers in selecting the most suitable quantum encoding technique for image classification tasks. It will inspire future research efforts to optimize quantum neural networks by leveraging the strengths of different encoding methods.

In addition, it will contribute to the growing body of knowledge on quantum computing and its applications in machine learning. It will aid researchers and practitioners in acquiring a more comprehensive comprehension of quantum neural networks and their potential application in tasks related to image classification.

2. LITERATURE REVIEW

2.1. Related Work

This study intends to conduct a comprehensive comparative analysis of different quantum encoding techniques in QNNs for image classification, focusing on the MNIST dataset. The research investigates Basis Encoding, Amplitude Encoding, Histogram Encoding and Pauli Basis Encoding techniques on the performance of QNNs. The outcomes of this investigation yield invaluable insights into the correlations between encoding methodologies and the performance of Quantum Neural Networks (QNNs). This progress in understanding significantly contributes to the broader field of quantum machine learning.

Quantum machine learning has garnered substantial interest due to its potential to revolutionize various computational tasks (Biamonte et al., 2017). Early studies, such as (Farhi and Neven 2018), utilized random quantum circuits with qubit encoding to classify images. Similarly, (Grant et al. 2018) employed hierarchical circuits and principal component analysis (PCA) features, followed by qubit encoding. However, these approaches raised concerns about the preservation of image details in complex datasets like CT scan images (Sengupta & Srivastava, 2021).

A notable approach in quantum circuit design is the hardware-efficient (HE) heuristic developed by Kandala et al. (2017), forming the foundation for the proposed quantum deep neural network (QDNN) architecture (Zhao & Gao, 2021). The Quantum Deep Neural Network (QDNN) architecture comprises three distinct quantum components within each layer: the encoder, the transformation, and the output. The encoder leverages qubit entanglement to incorporate classical data into quantum states. The transformation, based on the HE multi-layer structure, applies linear operations to the prepared quantum state. The output part employs quantum measurements to obtain classical data outputs, mirroring the classical neural network layers (Zhao & Gao, 2021). This QDNN architecture allows for an in-depth study of HE encoding's impact on classification performance.

Some studies have explored the construction of quantum artificial neurons as basic building blocks for highly complex QNN architectures (Cao et al., 2017; Tacchino et al., 2019; Kristensen et al., 2021). Nevertheless, training such quantum artificial neurons remains a significant challenge (Benedetti et al., 2019). Consequently, the current evaluation focuses on hardware-efficient QNNs, enabling a thorough analysis of their performance on data in multi-dimensional space.

Past assessments of QNNs have predominantly focused on gauging their capacity for expressiveness and their capability to establish entanglement. (Sim et al., 2019), comparisons of different circuit complexities (Hubregtsen et al., 2021), and the influence of quantum state preparation on effectiveness (Sierra-Sosa et al., 2020). While some studies explored replacing the convolution operation in classical neural networks with quantum circuits (Henderson et al., 2020; Liu et al., 2021; Kerenidis et al., 2019). (Nguyen et al. 2022) examined Qubit encoding and HE encoding on different quantum circuits.

This evaluation uniquely delves into QNNs, conducting a comprehensive comparative analysis of multiple encoding methods in image classification.

2.2. Quantum Encoding Methods

2.2.1. Basis Encoding

It is among the fundamental quantum encoding methods and plays a significant role in QNNs for image classification tasks.

The main idea behind basis encoding is to map classical data to specific quantum basis states. For binary data, 0s and 1s are respectively mapped to quantum basis states $|0\rangle$ and $|1\rangle$.
Basis encoding allows for straightforward representation of classical information as quantum states, making it relatively easy to implement in quantum circuits (Schuld, 2021).

In basis encoding, the mathematical representation involves associating classical data with quantum basis states. For example, if we have a classical binary string "1010," it can be encoded as follows:

$$|1010\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle$$

where $|0\rangle$ and $|1\rangle$ are the quantum basis states for 0 and 1, respectively.

The computation for basis encoding is simple and involves mapping classical data to the corresponding quantum basis states. This is typically done using quantum gates that manipulate qubits to represent the classical information.

In image classification tasks, basis encoding allows classical pixel values of an image to be represented as quantum states, which can exploit quantum parallelism and entanglement for more efficient and powerful computations.

2.2.2. Amplitude Encoding

It is another technique used to encode classical data, such as image pixels, as quantum states. Amplitude encoding pertains to using the amplitude of quantum states to represent classical data. The quantum state's amplitude corresponds to the probability amplitude linked with that state, and it has the capacity to convey information derived from classical data. (Schuld, 2021). Compared to basis encoding, amplitude encoding can encode more complex classical data patterns and continuous values as quantum states, providing a more expressive representation. Amplitude encoding encodes a vector x of length N into the amplitudes of an n-qubit quantum state with $n = \lceil log2(N) \rceil$.

The process of amplitude encoding necessitates skillful manipulation of the probability amplitudes of quantum states to achieve an efficient encoding of classical data. This can involve the application of quantum gates to control the amplitude values. Amplitude encoding is a powerful technique in QNNs, as it allows for a richer and more continuous representation of

classical data as quantum states, enhancing the expressive power of the network. In image classification tasks, amplitude encoding offers advantages in representing continuous-valued features and complex patterns in image data.

2.2.3. Pauli Basis Encoding:

The main idea behind Pauli basis encoding is to map classical data to quantum states represented by Pauli matrices, which are fundamental operators in quantum mechanics.

The Pauli matrices used in the encoding process, namely the Pauli-X, Pauli-Y, and Pauli-Z matrices, form a building block for the quantum state. These matrices provide a set of orthogonal and Hermitian operators that can represent quantum states.

The mathematical representation of Pauli basis encoding involves mapping classical data x to a quantum state $|\psi(x)\rangle$ using Pauli matrices X, Y, and Z. The process of encoding can be mathematically expressed as:

$$|\psi(x)\rangle = exp(-iaxXx)exp(-ibxYx)exp(-icxZx)|\psi 0\rangle$$

Here, x represents the classical data, and a, b, and c are parameters that determine the encoding. Xx, Yx, and Zx are the Pauli-X, Pauli-Y, and Pauli-Z matrices acting on the x-th qubit, respectively. $|\psi 0\rangle$ denotes the initial quantum state before encoding.

2.2.4. Histogram Encoding

Is a technique used in representing classical data, such as image pixels, as quantum states by constructing histograms of the pixel values. A histogram represents the frequency distribution of pixel values, capturing statistical information about the image. Each bin in the histogram

corresponds to a range of pixel values, and the number of occurrences of each pixel value range is represented as the amplitude of the respective quantum state.

This is achieved by associating the histogram bins with quantum states and using their amplitudes to represent the frequency of pixel value occurrences. The computation for histogram encoding requires processing the classical image data to construct the histogram and then mapping the histogram bins' frequency information to the corresponding quantum states' amplitudes.

Histogram encoding is beneficial for scenarios where statistical distribution information is crucial for classification.

3. METHODOLOGY

3.1. The Research Design

3.1.1. The Research Methodology

Owing to the nature of the research, both qualitative and quantitative research methods were used. A detailed review and analysis of scholarly publications on comparative analysis of different quantum encoding techniques was done.

The methodology of experimental design was employed to conduct an all-encompassing comparative analysis of multiple quantum encoding methods, including Basis, Amplitude, Pauli Basis and Histogram Encoding in the context of QNNs for image classification on the MNIST dataset.

3.1.2. Data Collection

The Modified National Institute of Standards and Technology (MNIST) dataset was used. This dataset contains an extensive compilation of handwritten digits. It acts as a widely employed resource for training diverse image processing systems. Renowned for its prominence in the field of image classification, the MNIST dataset is accessible through numerous sources. Notably, both TensorFlow and Keras provide a convenient means to import the MNIST dataset directly and also download it from their API. Hence, commencing the process, TensorFlow and the MNIST dataset using the Keras API was imported.

The expansive MNIST database encompasses a total of 60,000 training images and an additional set of 10,000 images designated for testing. These images were sourced from individuals associated with the American Census Bureau, as well as American high school students. (Deng, L. 2012)

3.2. Evaluation

In the assessment of the model's applicability, the following metrics were incorporated: training loss, validation loss, training accuracy and validation accuracy.

3.3. Implementation

3.3.1. Architecture

QNNs concept was introduced by (Zhao and Gao, 2021). These QNNs consist of QNN layers (QNNLs), each comprising three integral components: the encoder, the transform and the output. Input data if fed into the encoder $x \rightarrow \in \mathbb{R}n$ and utilizes a parameterized quantum circuit (PQC)

$U(x\rightarrow)$ to modify the initial quantum state $|\psi 0\rangle$ into $|\psi(x\rightarrow)\rangle$. This encoding process is mathematically represented as:

$|\psi(x)\rangle = exp(-ix2X)|\psi 0\rangle$, where X represents the Pauli matrix X.

The input quantum state receives a linear transformation from the transformation part $|\psi 0\rangle$ using a PQC $V(W\rightarrow)$ with parameters $W\rightarrow$. Subsequently, the output is computed by evaluating m Hamiltonians Hj,j=1,…,m and generating output $y\rightarrow$. To resemble classical deep neural networks, each output component yj is augmented with a bias term bj, expressed as

$yj = \langle\psi(x\rightarrow)|V\dagger(W\rightarrow)HjV(W\rightarrow)|\psi(x\rightarrow)\rangle + bj$ (Zhao & Gao, 2021).

The value of m corresponds to the hidden dimensions' number in the model.

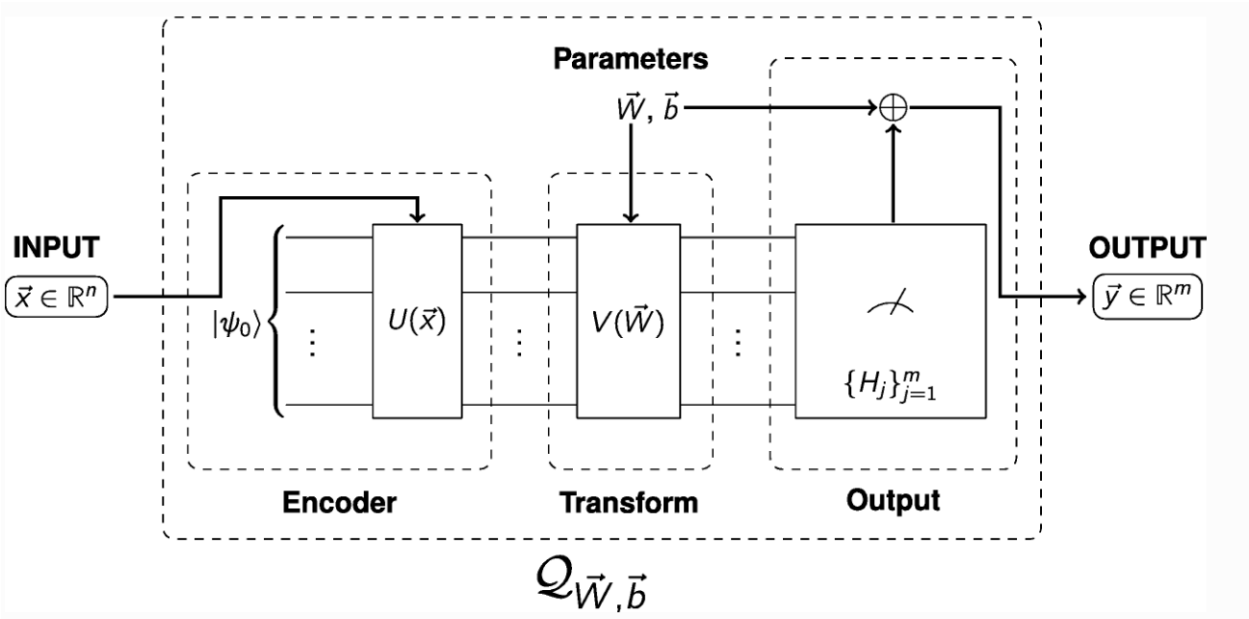

Figure 1: Architecture of QNN. Obtained from (Zhao and Gao, 2021)

The encoder and transform parts consist of multiple element layers, following the structure proposed by (Kandala et al., 2017), involving entanglement blocks and single-qubit rotation operations. This element layer configuration is adopted in the QNNL architecture proposed by

(Zhao and Gao, 2021). A QNN may comprise multiple QNNLs, each housing a significant number of element layers (Zhao & Gao, 2021). A conversion between a quantum state and a classical state takes place between two QNNLs.

Furthermore, (Zhao and Gao, 2021) described a specific encoding approach for input data. In this approach, an 8 × 8 image is encoded using only eight input qubits, and eight 8-dimensional vectors representing the image are embedded through the utilization of two hardware-efficient (HE) element layers.

3.3.2. Implementation Steps

3.3.2.1. Load the data

We begin by loading the MNIST dataset from the Keras library, which contains a collection of hand-written digit images. To prepare the dataset for quantum processing, we convert the label, y, into a boolean format, enabling us to represent the class labels as binary values.

Since the original image size is 28x28, which may exceed the capabilities of current quantum computers, we need to downscale the images to a manageable size. As a result, we resize the images down to 4x4, sacrificing some fine details but ensuring compatibility with the quantum circuitry.

Another crucial preprocessing step involves filtering the dataset to remove any contradictory examples. In some cases, an image might be ambiguously labeled, showing characteristics of multiple classes. Such contradictory instances can lead to confusion in the classification process and may negatively impact the QNN model's performance. Therefore, we carefully filter the dataset to eliminate images that are labeled as belonging to more than one class.

By performing these preprocessing steps, we aim to optimize the dataset for training with quantum circuits and improve the accuracy and efficiency of our image classification task.

3.3.2.2. Encode the data

In their work on image processing with quantum computers, (Farhi, Goldstone, and Gutmann 2014) suggested a method wherein each pixel of the image is represented using a qubit, with the qubit's state depending on the corresponding pixel's value. The initial stage of this approach involves converting the pixel values into a binary encoding to facilitate quantum processing.

Here we apply the various encoding techniques to the data:

3.3.2.2.1. Basis Encoding

This is performed by applying the X gate to set qubits to the $|1\rangle$ state for those pixels in the image that have a value of 1. The classical image data is first flattened and converted to binary form, and then the X gate is applied to corresponding qubits.

3.3.2.2.2. Amplitude Encoding

We set the quantum states' amplitudes based on the normalized values of the classical image data. The values array is normalized to lie between 0 and 1, and then the amplitude of each quantum state is set using the corresponding normalized value.

3.3.2.2.3. Pauli Basis Encoding

We replace the X gate with appropriate Pauli gates based on the classical image data. Pauli Basis Encoding maps the classical data to quantum states using different Pauli gates (X, Y, or Z) depending on the value of the classical data at each pixel. In this implementation we can use the following mappings:

- Classical data value 0: No gate applied (corresponding to the $|0\rangle$ state).
- Classical data value 1: Apply X gate (corresponding to the $|1\rangle$ state).
- Classical data value 2: Apply Y gate.
- Classical data value 3: Apply Z gate.

3.3.2.2.4. Histogram Encoding

We compute the histogram of the image and then use it to encode the data into a quantum circuit. Each pixel value in the image will correspond to a bin in the histogram, and the value of each bin will determine the number of X gates applied to the corresponding qubit in the quantum circuit.

3.3.2.3. Build the QNN

The QNN construction, as proposed by (Farhi et al. 2018), involved a layered approach for classification based on the readout qubit's expectation. To achieve this, they employed two-qubit gates, consistently acting on the readout qubit. Each layer was composed of n instances of the same gate, and the data qubits interacted with the readout qubit. A two-layered model was developed, specifically designed to align with the dimensions of the data circuit. This model encompasses both preparation and readout operations.

In order to implement the quantum circuit within a deep learning framework, a Keras model was wrapped. This model received quantum data and employed a Parametrized Quantum Circuit (PQC) layer to train the model circuit using the quantum data.

3.3.2.4. Train the QNN

The model was trained and evaluated.

3.4. Design Tools

3.4.1. Python

Is an extensively used programming language recognized for its simplicity, readability, and adaptability. Python offers powerful libraries and frameworks that simplify and accelerate machine learning tasks. Python's extensive set of data analysis libraries and tools enables efficient data preprocessing, feature engineering, and exploratory data analysis. These

capabilities are crucial for effectively preparing data for machine learning models. Also Python's concise and expressive syntax allows for quick prototyping and experimentation. Machine learning models can be rapidly developed and tested, facilitating iterative development cycles and accelerating the research and development process.

3.4.2. Jupyter Notebook

Is an open-source web application that enables interactive and exploratory computational tasks. It offers an interactive computing environment in which users can generate and distribute documents encompassing live code, equations, visualizations, and explanatory text. Jupyter Notebooks enable the integration of code, visualizations, and descriptive text within a single document. This makes it convenient to document your machine learning process, including data analysis, model development, and evaluation. Further, Jupyter Notebooks enable code reusability by providing a platform to write modular and reusable code.

3.4.3. Tensorflow

TensorFlow, an open-source machine learning framework, was developed by Google. Its purpose is to streamline the creation and deployment of machine learning models, with a particular emphasis on deep learning models. TensorFlow offers an extensive ecosystem comprising tools, libraries, and resources that cater to diverse tasks within machine learning, encompassing data preprocessing, model construction, training, and deployment. TensorFlow supports both low-level and high-level APIs, providing flexibility for developers to work at different levels of abstraction which was important for this project. In addition, TensorFlow integrates seamlessly with other popular deep learning and machine learning libraries, such as NumPy, Pandas, and Matplotlib. This integration enables easy data manipulation, preprocessing, and visualization within TensorFlow workflows.

3.4.4. Tensorflow Quantum

TensorFlow Quantum (TFQ) is an extension of TensorFlow, tailored for quantum machine learning (QML). It seamlessly integrates quantum circuits with classical neural networks, enabling hybrid models that leverage the power of both classical and quantum computing paradigms. TFQ offers tools to preprocess and process complex quantum data, allowing researchers to harness quantum computation in their machine learning tasks.

One of its key features is hybrid quantum-classical training, where classical neural network layers and quantum circuits are jointly trained using gradient-based optimization. This empowers quantum components to learn from data and adapt during training, enhancing the expressiveness and adaptability of QNNs.

TFQ (TensorFlow Quantum) offers access to quantum simulators and quantum hardware backends, empowering researchers to execute quantum circuits on actual quantum processors or validate models using simulators. Moreover, it offers a library of quantum gates and circuits, simplifying the construction and experimentation of QNNs.

3.4.5. Numpy

NumPy (Numerical Python) is an open-source library designed for numerical computing in Python. It equips users with robust tools and functions to manipulate arrays and matrices, allowing for streamlined mathematical and logical operations on substantial datasets. NumPy serves as a fundamental building block for numerous Python libraries and frameworks employed in scientific computing and data analysis. It encompasses an extensive assortment of mathematical functions and operations, suitable for element-wise application to arrays. NumPy's optimized implementation ensures rapid and efficient computation of these operations.

3.4.6. Cirq

Cirq is an open-source quantum computing framework created by Google. Its purpose is to simplify tasks involving Noisy Intermediate-Scale Quantum (NISQ) devices. In QNNs, Cirq serves as a library for constructing and simulating quantum circuits necessary for tasks like data encoding, parameterized quantum circuits, and measurements. This is achieved by defining the quantum circuit architecture by selecting quantum gates, encoding techniques and measurement strategies. Cirq provides functions to create, manipulate, and simulate quantum circuits, allowing analysis of QNN performance. It enables classical simulations to estimate output probabilities and measure fidelity. For real quantum hardware experiments, Cirq interfaces with various quantum processors, facilitating quantum circuit execution on NISQ devices.

4. RESULTS AND DISCUSSIONS

4.1. Accuracy

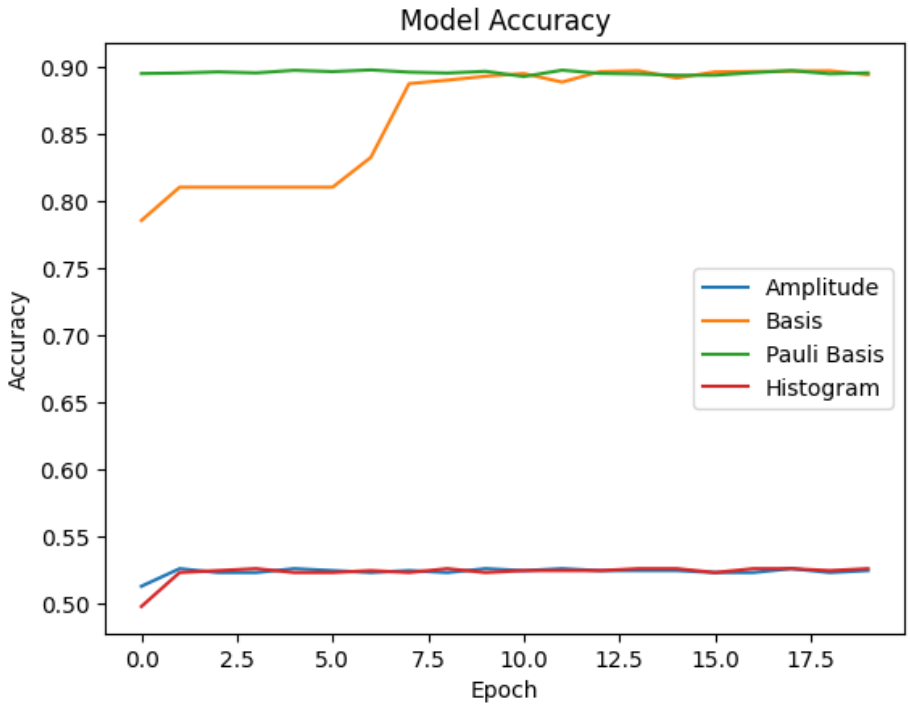

Figure 2: Learning Curves in the Training Accuracy of QNNs with different Encoding Methods

It's a metric that calculates the ratio of correctly predicted instances to the total instances present in the dataset. This metric is often used to assess the accuracy of classification models.

Amplitude Encoding

The accuracy values for Amplitude Encoding range from approximately 0.513 to 0.526. The accuracy starts relatively low and shows slight fluctuations during training. The model using Amplitude Encoding is not performing well, with a relatively low accuracy rate.

Basis Encoding

The accuracy values for Basis Encoding are notably higher, ranging from approximately 0.785 to 0.896. The accuracy improves significantly with each epoch, indicating effective learning and generalization on the training data. The model using Basis Encoding achieves relatively high accuracy and shows consistent improvement.

Pauli Basis Encoding

The accuracy values for Pauli Basis Encoding range from approximately 0.894 to 0.897. The accuracy starts relatively high and remains stable throughout training. The model using Pauli Basis Encoding achieves consistently high accuracy and performs well on the training data.

Histogram Encoding

The accuracy values for Histogram Encoding range from approximately 0.498 to 0.526. The accuracy fluctuates around a relatively low value, and there is no significant improvement during training. The model using Histogram Encoding is not performing well, with a relatively low and inconsistent accuracy rate.

Comparison:

Basis Encoding and Pauli Basis Encoding show the best performance among all the encodings, with accuracy consistently above 0.89 and approaching 0.90. These models effectively classify the data and demonstrate strong generalization capabilities.

Amplitude Encoding and Histogram Encoding perform relatively poorly compared to Basis and Pauli Basis Encoding, with accuracy rates below 0.53. The models with these encodings are not effective in classifying the data and might require further optimization or adjustments to improve their performance.

4.2. Validation Accuracy

Validation accuracy is a metric utilized to assess a machine learning model's performance on new and unseen data. It quantifies the ratio of accurately predicted instances to the total instances within the validation dataset.

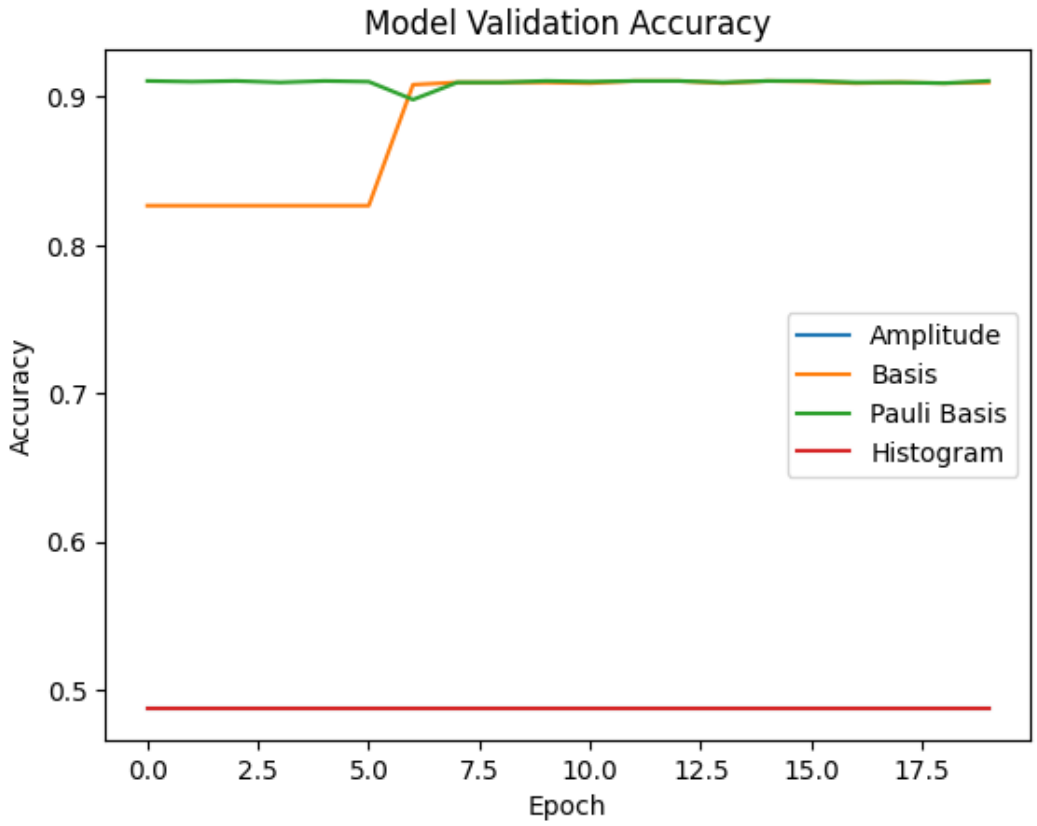

Figure 3: Learning Curves in the Validation Accuracy of QNNs with different Encoding Methods

Amplitude Encoding

The validation accuracy values for Amplitude Encoding are consistently low, remaining around 0.487 for all epochs. This indicates that the model is not performing well on unseen data, and there is no significant improvement during training. The model using Amplitude Encoding shows poor generalization capabilities.

Basis Encoding

The validation accuracy values for Basis Encoding start at approximately 0.826 and gradually increase to around 0.911. The accuracy improves with each epoch, suggesting effective learning

and generalization on the validation data. The model using Basis Encoding achieves relatively high validation accuracy and demonstrates consistent improvement.

Pauli Basis Encoding

The validation accuracy values for Pauli Basis Encoding range from approximately 0.898 to 0.911. The accuracy starts high and remains stable throughout training. The model using Pauli Basis Encoding achieves consistently high validation accuracy and performs well on unseen data.

Histogram Encoding

The validation accuracy values for Histogram Encoding are consistently low, remaining around 0.487 for all epochs. Similar to Amplitude Encoding, this indicates that the model using Histogram Encoding is not performing well on unseen data and lacks the ability to generalize effectively.

Comparison

Basis Encoding and Pauli Basis Encoding show the best performance among all the encodings, with validation accuracy consistently above 0.89 and approaching 0.91. These models effectively generalize to unseen data and demonstrate strong performance.

Amplitude Encoding and Histogram Encoding perform poorly compared to Basis and Pauli Basis Encoding, with validation accuracy rates around 0.487. These models lack the ability to generalize to new data, resulting in low validation accuracy.

4.3. Loss

Loss values indicate the disparity between the anticipated output generated by the model and the true target output throughout the training procedure.

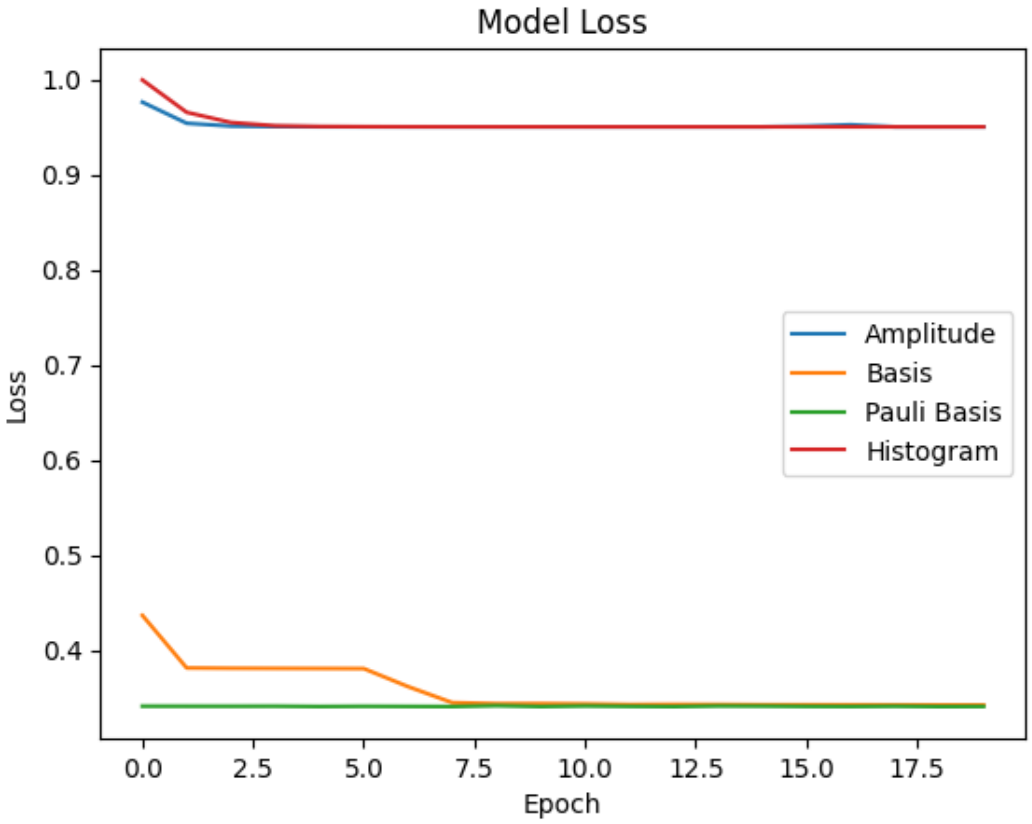

Figure 4: Learning Curves in the Training Loss of QNNs with different Encoding Methods

Amplitude Encoding

The loss values for Amplitude Encoding start relatively high at Epoch 0 (0.9762599468) and steadily decrease as the training progresses. By Epoch 15, the loss reaches 0.9509592652, and after Epoch 15, it starts to fluctuate slightly but remains close to 0.9502805471. This indicates that the model is learning and improving its predictions, but there might be some overfitting as the loss fluctuates afterward.

Basis Encoding

The loss values for Basis Encoding start significantly higher than the other encodings at Epoch 0 (0.4365152121). However, it quickly decreases within the first few epochs. By Epoch 20, the loss reaches 0.3419340253. This rapid decrease suggests that the model is effectively learning and fitting the data well.

Pauli Basis Encoding

The loss values for Pauli Basis Encoding start at 0.3410262465 in Epoch 0 and continue to decrease gradually. By Epoch 15, the loss is 0.3408143222, and afterward, it remains relatively stable around 0.3407. This indicates that the model is consistently learning without significant fluctuations.

Histogram Encoding

The loss values for Histogram Encoding start at 0.9995663166 in Epoch 0, which is the highest among all encodings. Similar to Amplitude Encoding, the loss steadily decreases over epochs, reaching 0.9502807856 at Epoch 15. Afterward, it slightly fluctuates but remains close to 0.9502807856. The model shows some improvements, but the loss is relatively higher compared to other encodings.

Comparison

Basis Encoding and Pauli Basis Encoding show the most significant reduction in loss, indicating that these encodings might be more effective in representing the data and capturing its features. Amplitude Encoding and Histogram Encoding also show improvements, but their loss values are relatively higher compared to Basis and Pauli Basis Encoding, suggesting that they may not be as efficient in representing the data for an image classification problem.

4.4. Validation Loss

Validation loss is a metric employed to appraise the efficacy of a machine learning model using a distinct validation dataset. It measures the level of dissimilarity between the model's predicted outputs and the actual target outputs during validation.

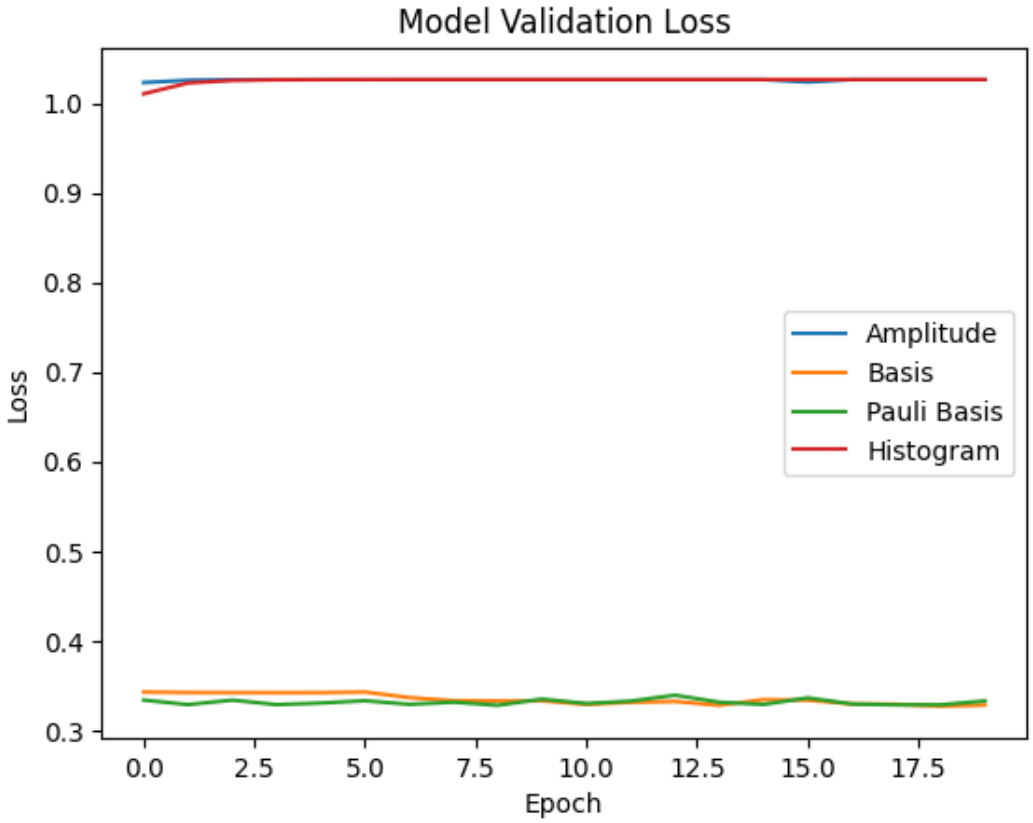

Figure 5: Learning Curves in the Validation Loss of QNNs with different Encoding Methods

Amplitude Encoding

The validation loss values for Amplitude Encoding start high at Epoch 0 (1.023080349) and slightly fluctuate around 1.026 throughout the training. This indicates that the model is not performing well on the unseen validation data. The lack of significant reduction in validation loss suggests that the model might not generalize effectively, potentially overfitting the training data.

Basis Encoding

Similar to Amplitude Encoding, the validation loss values for Basis Encoding also start high at Epoch 0 (1.023080349) and remain consistently around 1.026 throughout the training process. Like Amplitude Encoding, the model using Basis Encoding might suffer from overfitting, as it does not improve its performance on the validation data.

Pauli Basis Encoding

The validation loss values for Pauli Basis Encoding start at 0.3337479532 in Epoch 0 and continue to decrease significantly throughout the training process. By Epoch 8, the validation loss reaches 0.3279569149, indicating that the model is effectively generalizing to new data. The consistently low validation loss suggests that the model using Pauli Basis Encoding is performing well on unseen data and not overfitting.

Histogram Encoding

The validation loss values for Histogram Encoding start at 1.0104937553 in Epoch 0 and remain relatively high throughout the training, hovering around 1.026. Similar to Amplitude and Basis Encoding, the model using Histogram Encoding might be overfitting the training data, as it does not improve its performance on the validation data.

Comparison

Pauli Basis Encoding demonstrates the best performance among all the encodings, as it achieves the lowest validation loss, indicating effective generalization to new data.

Amplitude, Basis, and Histogram Encoding show similar behavior, with their validation losses remaining consistently high and not showing significant improvement. These models might suffer from overfitting.

4.5. Overall Comparison

Basis Encoding and Pauli Basis Encoding outperform Amplitude Encoding and Histogram Encoding in all metrics. They achieve higher accuracy, validation accuracy, and lower loss and validation loss, indicating superior performance and generalization capabilities. Amplitude Encoding and Histogram Encoding models show similar trends in all metrics, with moderate improvement in accuracy and slightly higher loss values. However, both models struggle to generalize to unseen data, as indicated by consistently low validation accuracy and validation loss.

Hence, Basis Encoding and Pauli Basis Encoding are the most effective encodings for constructing Quantum Neural Network models in this analysis. They demonstrate better learning, superior generalization to unseen data, and overall better performance compared to Amplitude Encoding and Histogram Encoding. The choice of encoding significantly impacts the model's performance, and selecting appropriate encoding techniques is essential for successful Quantum Neural Network implementations.

5. LIMITATIONS OF THE RESEARCH

5.1. Limited Dataset and Problem Domain

The research encountered a limitation due to its reliance on a restricted dataset and its narrow focus on a specific problem domain. The study draws upon the widely-used MNIST database, renowned for its application in image classification tasks. However, this dataset solely comprises handwritten digits, specifically limited to the numbers 3 and 6. While the MNIST dataset serves as a valuable benchmark for evaluating image classification algorithms, it may not encompass the full complexity and diversity found in real-world data.

5.2. Limited Availability of Quantum Computing Resources

A critical constraint was in the restricted accessibility of quantum computing resources. There are challenges in obtaining and utilizing quantum computing environments, where quantum hardware remains scarce and inaccessible for most researchers. Consequently, the study resorts to simulating quantum processing on classical computers. However, this approach may not fully capture the genuine potential and performance of QNNs.

5.3. Choice of Quantum Encoding Methods

The research focused on four specific quantum encoding methods: Amplitude Encoding, Basis Encoding, Pauli Basis Encoding, and Histogram Encoding. While these are commonly used techniques, other encoding methods could exist, and their exclusion might limit the breadth of the study.

6. CONCLUSION

In this research paper, we conducted a comparative analysis of different quantum encoding methods in QNNs for image classification on the MNIST dataset. Our objectives were to compare and analyze the performance of four quantum encoding methods, namely Amplitude Encoding, Basis Encoding, Pauli Basis Encoding, and Histogram Encoding, and to identify effective encoding methods for QNNs in image classification tasks. Additionally, we evaluated the training loss and training performance of QNNs under each identified quantum encoding method.

Through our experiments Basis Encoding and Pauli Basis Encoding for QNNs provided the best results in image classification. These two encodings consistently outperformed Amplitude Encoding and Histogram Encoding across all metrics, including accuracy, validation accuracy, loss, and validation loss. They showcased remarkable learning capabilities, achieved higher accuracy, and demonstrated superior generalization to unseen data, making them suitable choices for image classification tasks in QNNs.

Amplitude Encoding and Histogram Encoding, while showing moderate improvements in accuracy and loss during training, struggled to generalize to new data. Their validation accuracy and validation loss remained consistently low, indicating limited generalization capabilities.

In conclusion, Basis Encoding and Pauli Basis Encoding stand out as promising quantum encoding methods in QNNs for image classification on the MNIST dataset. Further research could explore their effectiveness in other image datasets and extend the investigation to more complex quantum algorithms and architectures.

7. SUGGESTIONS FOR FUTURE WORK

Exploration of Other Quantum Datasets

While this study focused on the MNIST dataset, future research could investigate the performance of different quantum encoding methods on other image datasets with varying complexities and characteristics. Evaluating the encoding methods on diverse datasets can provide a more comprehensive understanding of their generalizability and effectiveness across different image classification tasks.

Exploration of other quantum encoding techniques

While the impact of Basis Encoding, Amplitude Encoding, Histogram Encoding, and Pauli Basis Encoding on QNN performance was investigated, there are other quantum encoding methods that could be explored. Investigating these alternative encoding methods could provide valuable insights into their suitability for specific image classification tasks and their potential advantages over the ones already examined.

Exploration of other QNN Architectures

Our study utilized the QNN architecture obtained from (Zhao and Gao, 2021). However, there exist various other architectures that may offer unique advantages for image classification tasks. The performance of different encoding methods on QNN architectures like Quantum Convolutional Neural Networks (QCNNs), Quantum Variational Autoencoders (QVAEs), or Quantum Recurrent Neural Networks (QRNNs) should be explored.

8. REFERENCES

1. Liu, J., Lim, K. H., Wood, K. L., Huang, W., Guo, C., & Huang, H. L. (2021). Hybrid quantum-classical convolutional neural networks. Science China Physics, Mechanics & Astronomy, 64(9), 290311.
2. Abbas, A., Sutter, D., Zoufal, C., Lucchi, A., Figalli, A., & Woerner, S. (2021). The power of QNNs. Nature Computational Science, 1, 403–409.
3. Benedetti, M., Lloyd, E., Sack, S., & Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. Quantum Science and Technology, 4(4), 043001.
4. Chen, S. Y. C., & Yoo, S. (2021). Federated quantum machine learning. Entropy, 23(4), 460.
5. Schuld, Maria. "Quantum machine learning models are kernel methods." arXiv:2101.11020 (2021).
6. Nguyen, T.; Paik, I.; Watanobe, Y.; Thang, T.C. An Evaluation of Hardware-Efficient Quantum Neural Networks for Image Data Classification. Electronics 2022, 11, 437.
7. Sierra-Sosa, D.; Telahun, M.; Elmaghraby, A. Tensorflow quantum: Impacts of quantum state preparation on quantum machine learning performance. IEEE Access 2020, 8, 215246–215255.
8. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. Nature, 549, 195–202.
9. Cao, Y., Guerreschi, G.G., & Aspuru-Guzik, A. (2017). Quantum neuron: An elementary building block for machine learning on quantum computers. arXiv preprint arXiv:1711.11240.
10. Farhi, E., & Neven, H. (2018). Classification with quantum neural networks on near term processors. arXiv preprint arXiv:1802.06002.
11. Grant, E., Benedetti, M., Cao, S., Hallam, A., Lockhart, J., Stojevic, V., ... Severini, S. (2018). Hierarchical quantum classifiers. NPJ Quantum Information, 4, 1–8.
12. Henderson, M., Shakya, S., Pradhan, S., & Cook, T. (2020). Quanvolutional neural networks: Powering image recognition with quantum circuits. Quantum Machine Intelligence, 2, 1–9.

13. Hubregtsen, T., Pichlmeier, J., Stecher, P., & Bertels, K. (2021). Evaluation of parameterized quantum circuits: On the relation between classification accuracy, expressibility, and entangling capability. Quantum Machine Intelligence, 3, 1–19.
14. Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J.M., & Gambetta, J.M. (2017). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature, 549, 242–246.
15. Kristensen, L.B., Degroote, M., Wittek, P., Aspuru-Guzik, A., & Zinner, N.T. (2021). An artificial spiking quantum neuron. NPJ Quantum Information, 7, 1–7.
16. Liu, J., Lim, K.H., Wood, K.L., Huang, W., Guo, C., & Huang, H.L. (2021). Hybrid quantum-classical convolutional neural networks. Science China Physics, Mechanics & Astronomy, 64, 1–8.
17. Sierra-Sosa, D., Telahun, M., & Elmaghraby, A. (2020). TensorFlow Quantum: Impacts of quantum state preparation on quantum machine learning performance. IEEE Access, 8, 215246–215255.
18. Sim, S., Johnson, P.D., & Aspuru-Guzik, A. (2019). Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Advanced Quantum Technologies, 2, 1900070.
19. Tacchino, F., Macchiavello, C., Gerace, D., & Bajoni, D. (2019). An artificial neuron implemented on an actual quantum processor. NPJ Quantum Information, 5, 1–8.
20. Zhao, C., & Gao, X.S. (2021). QDNN: Deep neural networks with quantum layers. Quantum Machine Intelligence,
21. Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.
22. Zhao, C., & Gao, X. S. (2021). QDNN: Deep neural networks with quantum layers. Quantum Machine Intelligence, 3(1), 1-9.
23. Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J. M., & Gambetta, J. M. (2017). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature, 549(7671), 242-246.
24. Lalkhen, A. G., & McCluskey, A. (2008). *Clinical tests: sensitivity and specificity. Continuing Education in Anaesthesia Critical Care & Pain, 8*(6), 221-223.

9. APPENDIX

9.1. Import Packages, Load dataset and Preprocess Images Code

```
import tensorflow as tf
import tensorflow_quantum as tfq

import cirq
import sympy
import numpy as np
import seaborn as sns
import collections

# visualization tools
%matplotlib inline
import matplotlib.pyplot as plt
from cirq.contrib.svg import SVGCircuit


(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train[..., np.newaxis]/255.0, x_test[..., np.newaxis]/255.0

print("Number of original training examples:", len(x_train))
print("Number of original test examples:", len(x_test))
def filter_36(x, y):
    keep = (y == 3) | (y == 6)
    x, y = x[keep], y[keep]
    y = y == 3
    return x,y

x_train, y_train = filter_36(x_train, y_train)
x_test, y_test = filter_36(x_test, y_test)

print("Number of filtered training examples:", len(x_train))
print("Number of filtered test examples:", len(x_test))
print(y_train[0])

plt.imshow(x_train[0, :, :, 0])
plt.colorbar()
```

```
x_train_small = tf.image.resize(x_train, (4,4)).numpy()
x_test_small = tf.image.resize(x_test, (4,4)).numpy()
print(y_train[0])

plt.imshow(x_train_small[0,:,:,0], vmin=0, vmax=1)
plt.colorbar()

def remove_contradicting(xs, ys):
    mapping = collections.defaultdict(set)
    orig_x = {}
    # Determine the set of labels for each unique image:
    for x,y in zip(xs,ys):
       orig_x[tuple(x.flatten())] = x
       mapping[tuple(x.flatten())].add(y)

    new_x = []
    new_y = []
    for flatten_x in mapping:
      x = orig_x[flatten_x]
      labels = mapping[flatten_x]
      if len(labels) == 1:
          new_x.append(x)
          new_y.append(next(iter(labels)))
      else:
          # Throw out images that match more than one label.
          pass

    num_uniq_3 = sum(1 for value in mapping.values() if len(value) == 1 and True in value)
    num_uniq_6 = sum(1 for value in mapping.values() if len(value) == 1 and False in value)
    num_uniq_both = sum(1 for value in mapping.values() if len(value) == 2)

    print("Number of unique images:", len(mapping.values()))
    print("Number of unique 3s: ", num_uniq_3)
    print("Number of unique 6s: ", num_uniq_6)
    print("Number of unique contradicting labels (both 3 and 6): ", num_uniq_both)
    print()
    print("Initial number of images: ", len(xs))
    print("Remaining non-contradicting unique images: ", len(new_x))

    return np.array(new_x), np.array(new_y)
```

```
x_train_nocon, y_train_nocon = remove_contradicting(x_train_small, y_train)

THRESHOLD = 0.5

x_train_bin = np.array(x_train_nocon > THRESHOLD, dtype=np.float32)
x_test_bin = np.array(x_test_small > THRESHOLD, dtype=np.float32)
```

9.2. Different Encoding Functions Code and application on different datasets

```
# Basis
def b_convert_to_circuit(image):
    """Encode truncated classical image into quantum datapoint."""
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
    circuit = cirq.Circuit()
    for i, value in enumerate(values):
        if value:
            circuit.append(cirq.X(qubits[i]))
    return circuit

# Amplitude
def a_convert_to_circuit(image):
    """Encode truncated classical image into quantum datapoint using Amplitude Encoding."""
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
    circuit = cirq.Circuit()

    # Normalize the pixel values to be between -1 and 1
    normalized_values = (2 * values) - 1

    # Set the amplitudes of the quantum state based on the normalized pixel values
    for i, value in enumerate(normalized_values):
        circuit.append(cirq.X(qubits[i]) ** value)

    return circuit

# Pauli Basis Encoding
```

```
def pb_convert_to_circuit(image):
    """Encode truncated classical image into quantum datapoint using Pauli Basis Encoding."""
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
    circuit = cirq.Circuit()
    for i, value in enumerate(values):
        if value == 1:  # For classical data value 1, apply X gate
            circuit.append(cirq.X(qubits[i]))
        elif value == 2:  # For classical data value 2, apply Y gate
            circuit.append(cirq.Y(qubits[i]))
        elif value == 3:  # For classical data value 3, apply Z gate
            circuit.append(cirq.Z(qubits[i]))
    return circuit

# Histogram Encoding
def h_convert_to_circuit(image):
    """Encode truncated classical image into quantum datapoint using Histogram Encoding."""
    num_bins = 4  # Number of bins for histogram encoding
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
    circuit = cirq.Circuit()

    # Compute histogram of the image
    hist, bin_edges = np.histogram(values, bins=num_bins, range=(0, 1))

    # Apply X gates based on the histogram values
    for i in range(num_bins):
        for j in range(hist[i]):
            circuit.append(cirq.X(qubits[i]))

    return circuit


a_x_train_circ = [a_convert_to_circuit(x) for x in x_train_bin]
a_x_test_circ = [a_convert_to_circuit(x) for x in x_test_bin]

b_x_train_circ = [b_convert_to_circuit(x) for x in x_train_bin]
b_x_test_circ = [b_convert_to_circuit(x) for x in x_test_bin]

pb_x_train_circ = [pb_convert_to_circuit(x) for x in x_train_bin]
```

```
pb_x_test_circ = [pb_convert_to_circuit(x) for x in x_test_bin]

h_x_train_circ = [h_convert_to_circuit(x) for x in x_train_bin]
h_x_test_circ = [h_convert_to_circuit(x) for x in x_test_bin]

a_x_train_tfcirc = tfq.convert_to_tensor(a_x_train_circ)
a_x_test_tfcirc = tfq.convert_to_tensor(a_x_test_circ)

b_x_train_tfcirc = tfq.convert_to_tensor(b_x_train_circ)
b_x_test_tfcirc = tfq.convert_to_tensor(b_x_test_circ)

pb_x_train_tfcirc = tfq.convert_to_tensor(pb_x_train_circ)
pb_x_test_tfcirc = tfq.convert_to_tensor(pb_x_test_circ)

h_x_train_tfcirc = tfq.convert_to_tensor(h_x_train_circ)
h_x_test_tfcirc = tfq.convert_to_tensor(h_x_test_circ)
```

9.3. Build Model and Its Accuracy Evaluation Code

```
class CircuitLayerBuilder():
    def __init__(self, data_qubits, readout):
        self.data_qubits = data_qubits
        self.readout = readout

    def add_layer(self, circuit, gate, prefix):
        for i, qubit in enumerate(self.data_qubits):
            symbol = sympy.Symbol(prefix + '-' + str(i))
            circuit.append(gate(qubit, self.readout)**symbol)


demo_builder = CircuitLayerBuilder(data_qubits = cirq.GridQubit.rect(4,1),
                                   readout=cirq.GridQubit(-1,-1))

circuit = cirq.Circuit()
demo_builder.add_layer(circuit, gate = cirq.XX, prefix='xx')
SVGCircuit(circuit)
def create_quantum_model():
    """Create a QNN model circuit and readout operation to go along with it."""
    data_qubits = cirq.GridQubit.rect(4, 4)  # a 4x4 grid.
```

```
    readout = cirq.GridQubit(-1, -1)        # a single qubit at [-1,-1]
    circuit = cirq.Circuit()

    # Prepare the readout qubit.
    circuit.append(cirq.X(readout))
    circuit.append(cirq.H(readout))

    builder = CircuitLayerBuilder(
        data_qubits = data_qubits,
        readout=readout)

    # Then add layers (experiment by adding more).
    builder.add_layer(circuit, cirq.XX, "xx1")
    builder.add_layer(circuit, cirq.ZZ, "zz1")

    # Finally, prepare the readout qubit.
    circuit.append(cirq.H(readout))

    return circuit, cirq.Z(readout)

model_circuit, model_readout = create_quantum_model()

# Build the Keras model.
model = tf.keras.Sequential([
    # The input is the data-circuit, encoded as a tf.string
    tf.keras.layers.Input(shape=(), dtype=tf.string),
    # The PQC layer returns the expected value of the readout gate, range [-1,1].
    tfq.layers.PQC(model_circuit, model_readout),
])
print('6')
y_train_hinge = 2.0*y_train_nocon-1.0
y_test_hinge = 2.0*y_test-1.0
def hinge_accuracy(y_true, y_pred):
    y_true = tf.squeeze(y_true) > 0.0
    y_pred = tf.squeeze(y_pred) > 0.0
    result = tf.cast(y_true == y_pred, tf.float32)

    return tf.reduce_mean(result)

model.compile(
```

```
    loss=tf.keras.losses.Hinge(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=[hinge_accuracy])


EPOCHS = 3
BATCH_SIZE = 32
print('7')
NUM_EXAMPLES = len(a_x_train_tfcirc)
# NUM_EXAMPLES = 500

a_x_train_tfcirc_sub = a_x_train_tfcirc[:NUM_EXAMPLES]
b_x_train_tfcirc_sub = b_x_train_tfcirc[:NUM_EXAMPLES]
pb_x_train_tfcirc_sub = pb_x_train_tfcirc[:NUM_EXAMPLES]
h_x_train_tfcirc_sub = h_x_train_tfcirc[:NUM_EXAMPLES]
y_train_hinge_sub = y_train_hinge[:NUM_EXAMPLES]
```

9.4. Model Training Code

```
a_qnn_history = model.fit(
      a_x_train_tfcirc_sub, y_train_hinge_sub,
      batch_size=32,
      epochs=EPOCHS,
      verbose=2,
      validation_data=(a_x_test_tfcirc, y_test_hinge))
b_qnn_history = model.fit(
      b_x_train_tfcirc_sub, y_train_hinge_sub,
      batch_size=32,
      epochs=EPOCHS,
      verbose=2,
      validation_data=(b_x_test_tfcirc, y_test_hinge))
pb_qnn_history = model.fit(
      pb_x_train_tfcirc_sub, y_train_hinge_sub,
      batch_size=32,
      epochs=EPOCHS,
      verbose=2,
      validation_data=(pb_x_test_tfcirc, y_test_hinge))
h_qnn_history = model.fit(
      h_x_train_tfcirc_sub, y_train_hinge_sub,
```

```
    batch_size=32,
    epochs=EPOCHS,
    verbose=2,
    validation_data=(h_x_test_tfcirc, y_test_hinge))
```

9.5. Model Training Graphs Code

```
plt.plot(a_qnn_history.history['hinge_accuracy'][:20])
plt.plot(b_qnn_history.history['hinge_accuracy'][:20])
plt.plot(pb_qnn_history.history['hinge_accuracy'][:20])
plt.plot(h_qnn_history.history['hinge_accuracy'][:20])
plt.legend(['Amplitude', 'Basis', 'Pauli Basis', 'Histogram'], loc='right')

plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()

plt.plot(a_qnn_history.history['loss'][:20])
plt.plot(b_qnn_history.history['loss'][:20])
plt.plot(pb_qnn_history.history['loss'][:20])
plt.plot(h_qnn_history.history['loss'][:20])
plt.legend(['Amplitude', 'Basis', 'Pauli Basis', 'Histogram'], loc='right')

plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()

plt.plot(a_qnn_history.history['val_hinge_accuracy'][:20])
plt.plot(b_qnn_history.history['val_hinge_accuracy'][:20])
plt.plot(pb_qnn_history.history['val_hinge_accuracy'][:20])
plt.plot(h_qnn_history.history['val_hinge_accuracy'][:20])
plt.legend(['Amplitude', 'Basis', 'Pauli Basis', 'Histogram'], loc='right')

plt.title('Model Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.show()
```

```
plt.plot(a_qnn_history.history['val_loss'][:20])
plt.plot(b_qnn_history.history['val_loss'][:20])
plt.plot(pb_qnn_history.history['val_loss'][:20])
plt.plot(h_qnn_history.history['val_loss'][:20])
plt.legend(['Amplitude', 'Basis', 'Pauli Basis', 'Histogram'], loc='right')

plt.title('Model Validation Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
```