Quantum Image Classification on Amazon Braket

Abstract

This paper embarks on a groundbreaking exploration of quantum image classification, harnessing the capabilities of Amazon Braket, a comprehensive quantum computing service. We present a novel approach that integrates Quantum Machine Learning (QML) with Variational Quantum Circuits (VQCs), focusing on the binary classification of images from the MNIST dataset. Our methodology encompasses edge detection, Principal Component Analysis (PCA) for dimensionality reduction, and Amplitude Encoding for mapping classical data into quantum states. The implementation involves a bespoke quantum circuit with rotation and entangling gates, optimized through gradient-based learning techniques. This study not only demonstrates the practical application of QML in image classification but also advances our understanding of quantum computing's potential in complex data processing tasks. The outcomes of this research are poised to significantly impact fields reliant on image processing, offering insights into the future trajectory of quantum computing in real-world applications.

1. Introduction

In recent years, quantum computing has emerged as a revolutionary technology. Quantum computing offers unprecedented capabilities that are poised to revolutionize various fields, including image processing and classification. This paper specifically focuses on leveraging quantum computing for image classification, a crucial task within the domains of computer vision and artificial intelligence. Utilizing Amazon Braket, a robust quantum computing platform, this study delves into the practical application of quantum image processing (QIP) techniques in classifying images from the widely recognized Modified National Institute of Standards and Technology (MNIST) dataset [13].

Quantum computing differs fundamentally from classical computing in its ability to process and store information. Unlike classical computing (which relies on classical bits) quantum bits or qubits, the fundamental units of quantum information can exist in superpositions. This capability enables parallel processing of vast amounts of data, a feature that is immensely beneficial in complex tasks like image classification [1, 2]. This quantum advantage is particularly potent in the field of image classification, where the need to process large volumes of data efficiently is paramount. QIP represents a novel approach that leverages quantum mechanical properties to enhance image processing techniques. The field has witnessed rapid evolution and exploration of various quantum image representations [2, 4]. Among these, the Flexible Representation of Quantum Images (FRQI) and its variations have shown considerable promise in improving the efficiency of image storage and processing operations, including compression and watermarking [4, 5, 6]. The Novel Enhanced Quantum Representation (NEQR) of digital images, for instance, provides an enhanced approach for representing grayscale images, facilitating more effective image processing operations [7]. Extensions of these methods, such as the Quantum Realization of the Nearest Neighbor Value Interpolation method for INEQR, further demonstrate the versatility of quantum approaches in image processing. In addition, techniques like block-based quantum image scrambling have been explored for secure image transmission, showcasing the potential of quantum computing in enhancing image security [9].

A significant aspect of QIP is its potential in image classification, which forms the core of our project. Image classification, the process of categorizing images into predefined classes, is integral to various applications ranging from medical diagnostics to autonomous vehicles. The implementation of quantum circuits for image classification on platforms like Amazon Braket could mark a paradigm shift in how we process and analyze images. Our project aims to implement a quantum circuit for image classification, adapting methodologies from existing research while tailoring it to operate within Amazon Braket's framework. This approach allows us to investigate not only the theoretical aspects of quantum image processing but also its practical applications [3, 10]. Figure 1 exemplifies this process, showcasing the stages of transforming classical image data into a quantum-ready format. This figure provides a visual understanding of the methodological transition, beginning with the initial image data and moving through various stages of processing, including feature extraction through edge detection and Principal Component Analysis (PCA), followed by the encoding of these features into

quantum states via Amplitude Encoding [4, 11]. In implementing a quantum circuit for image classification on Amazon Braket, we illustrate the practical application of these theoretical advances. Amazon Braket provides a versatile platform that bridges the gap between quantum theory and practical, real-world applications. By conducting this project on Amazon Braket, we demonstrate the feasibility and effectiveness of QIP in a realistic setting, which is invaluable for both academic research and industry application.

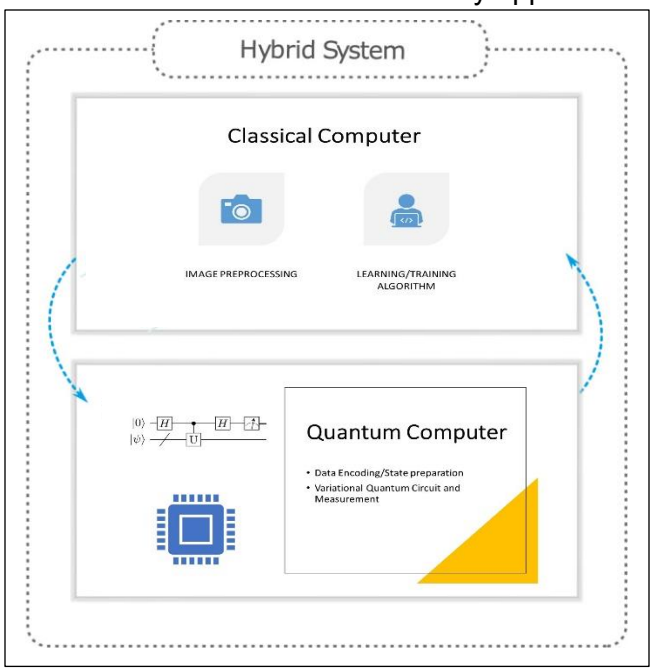

Figure 1: Classical-Quantum Hybrid System.

As we embark on this exploration of quantum image classification, we are positioned at the cusp of a new era in image processing. It holds the promise of not only advancing our understanding of quantum mechanics in the realm of computer vision but also paves the way for future innovations that could redefine how we process and analyze images. By harnessing the power of quantum mechanics, we aim to uncover novel methodologies and insights that could significantly enhance the field of image classification, paving the way for future advancements in quantum computing and its applications in real-world scenarios.

2. Significance of Research

The significance of exploring quantum image classification, particularly in the context of quantum computing platforms like Amazon Braket, cannot be overstated. Quantum computing's potential to revolutionize various technological sectors, especially in the realms of data processing and analysis, is immense. This project's focus on quantum image classification taps into this potential, exploring how quantum mechanics can provide novel solutions to challenges in image processing.

The field of image classification, a crucial aspect of computer vision, has traditionally relied on classical computing methods. However, these methods often encounter limitations when dealing with large datasets, high-dimensional data, and the need for real-time processing. Quantum image processing (QIP), on the other hand, offers a paradigm shift in dealing with such challenges. By leveraging the principles of quantum mechanics, such as superposition and entanglement, QIP can process complex computations more efficiently than classical algorithms [2]. This efficiency is crucial in image classification, where processing large volumes of image data quickly and accurately is paramount.

This project's exploration of quantum circuits for image classification on Amazon Braket showcases a potential pathway to harness these advantages for practical applications. It not only underpins the theoretical feasibility but also presents a concrete instance of quantum advantage in processing complex data, potentially revolutionizing fields reliant on image analysis. This project stands at the forefront of exploring the intersection of quantum computing and image classification. The outcomes of this research have the potential to impact

various sectors, including healthcare, security, autonomous vehicles, and more, where image classification plays a crucial role.

3. Review of Literature

The exploration of quantum image processing (QIP) has evolved significantly over the years, with numerous contributions enhancing our understanding and capabilities in this field. Central to this advancement is the development of various quantum image representations, each offering unique advantages and applications.

One of the foundational works in QIP is the Flexible Representation of Quantum Images (FRQI) proposed by Le, Dong, and Hirota in 2011. This representation marked a significant step forward in efficiently storing and processing images on quantum computers. FRQI utilizes the principles of superposition and entanglement to represent images, enabling operations like polynomial preparation, image compression, and processing operations, which are more efficient than their classical counterparts [4]. Building on the foundation laid by FRQI, advancements like the Novel Enhanced Quantum Representation (NEQR) introduced by Zhang et al. in 2013 have further optimized the process of representing grayscale images. NEQR allows for a more precise and efficient representation of digital images, facilitating enhanced image processing operations [7]. The field of QIP has also seen innovations in image security and watermarking. Li, Xiao, and Li, in 2016, presented a quantum representation and watermark strategy for color images. Their approach uses controlled rotation of qubits, demonstrating how quantum mechanics can be leveraged for securing and watermarking digital images [5]. Rabia Amin Khan in 2019 introduced an improved version of the flexible representation of quantum images, focusing on optimizing the existing methodologies and enhancing their efficiency [6]. Similarly, advancements in quantum image scrambling, as explored by Hai-Sheng Li et al. in 2019, showcase the potential of quantum computing in securing image data, a critical aspect in the digital age [9]. In addition to these, the field has witnessed the development of methods for interpolating images in quantum computing. For instance, the quantum realization of the nearest neighbor value interpolation method for INEQR by RiGui Zhou et al. in 2018 demonstrates the practical applicability of quantum computing in enhancing image processing techniques [8].

Quantum machine learning (QML) circuits in image classification underscores a burgeoning field where quantum principles augment computational capabilities, potentially outpacing classical algorithms. Early explorations into QML, as outlined by Venegas-Andraca and Bose [3], laid the groundwork for using quantum states for information processing. Subsequent advancements, such as the NEQR representation by Zhang et al. [7], further refined quantum image processing techniques. The significance of embedding strategies, particularly amplitude encoding, has been highlighted in the works of Schuld et al. [11], illustrating how classical information can be efficiently transposed into the quantum realm. The potential of variational circuits in machine learning models, showcased by Benedetti et al. [15], has opened avenues for adaptive quantum algorithms that learn from data. These studies collectively form a narrative arc from theoretical underpinnings to practical implementations, evidencing the growing synergy between quantum computing and machine learning in handling complex data-driven tasks like image classification.

These developments reflect a broader trend in the quantum computing community, where the focus is increasingly on applying quantum mechanics to solve complex, real-world problems. The literature in this field not only provides a rich source of theoretical knowledge but also serves as a springboard for practical applications, such as the project being undertaken on Amazon Braket.

4. Methodology

Quantum Machine Learning (QML) represents a groundbreaking approach in computational science, blending quantum computing principles with machine learning techniques. This methodology section explores the design and implementation of a QML circuit for classifying images from the MNIST dataset, focusing on binary classification due to hardware and time constraints.

In quantum computing, the state of a quantum system with n qubits is described by a vector in a 2^n-dimensional Hilbert space. Each qubit added to the system doubles the size of this Hilbert space, allowing for an exponential growth in the representational capacity of the system. This exponential expansion enables quantum systems to encode and process complex datasets more efficiently than classical systems. However, this remarkable

increase in dimensionality presents unique challenges in QML tasks. A larger Hilbert space can lead to a landscape in the optimization process that is riddled with numerous local minima [14]. This complexity can be problematic, as it makes the task of finding the global minimum of the loss or cost function more arduous and computationally intensive. This phenomenon is particularly significant in the context of variational quantum algorithms, where navigating through the complex optimization landscape is crucial for the effective training and performance of the QML models.

4.1 Data or Feature Encoding into Quantum States

Initially, each image from the MNIST dataset undergoes edge detection using a Sobel filter, highlighting crucial features. Subsequently, Principal Component Analysis (PCA) reduces the dimensionality of these features to reduced number of components (number of qubits=2 for our case), making them suitable for quantum processing [4].

The data from the reduced dataset is then embedded into quantum states using Amplitude Encoding. Amplitude Encoding is a crucial technique in QML, particularly in the context of quantum neural networks and quantum state preparation. This process involves encoding classical data into quantum states, utilizing the amplitudes of a quantum state to represent data or feature values [11]. Amplitude Encoding leverages this property by mapping a classical data vector, typically a normalized version of the original data, into the amplitudes of a quantum state. For a classical data vector $x \in \mathbb{R}^N$ with $N = 2^n$ elements, the quantum state $|\psi\rangle$ can be represented as:

$$|\psi\rangle = \sum_{i=0}^{N-1} x_i |i\rangle \tag{1}$$

where $|i\rangle$ are computational basis states, and x_i are the components of the normalized data vector x. The data vector x must be normalized since the sum of the squares of the amplitudes in a quantum state must equal 1, reflecting the probability interpretation in quantum mechanics:

$$\sum_{i=0}^{N-1} |x_i|^2 = 1 \tag{2}$$

Amplitude encoding is particularly powerful in QML due to its efficient use of quantum resources. It allows for the encoding of a classical vector of size 2^n using only n qubits. This exponential efficiency is a cornerstone in the potential advantage of quantum algorithms over classical counterparts for specific machine learning tasks.

4.2 Variational Quantum Circuit for Binary Classification

Variational Quantum Circuits (VQCs) or Ansatz have emerged as a promising approach in QML. The circuit design is variational, containing parameterized gates whose settings can be optimized through training. These circuits are characterized by their parametric nature and adaptability, making them suitable for tackling complex computational problems [11, 15]. In our study, the designed quantum circuit integrates rotation gates RX, RY, and RZ, each parameterized by angles that are analogous to the weights of the quantum neural network. For a single layer within our variational circuit, the unitary transformation can be mathematically represented as follows:

$$U(\theta) = \left(\bigotimes_{i=1}^{n-1} \mathrm{CNOT}_{i,i+1}\right) \bigotimes_{k=1}^{n} (RZ(\theta_{k,3}) RY(\theta_{k,2}) RX(\theta_{k,1})) \tag{3}$$

where $\theta_{k,1}, \theta_{k,2}, \theta_{k,3}$ are variational parameters for the k^{th} qubit, and the Controlled-NOT(CNOT) gates are utilized to entangle the qubits sequentially, effectively capturing the correlations between them. $U(\theta)$, the sequence of quantum gate operations, constitutes one layer of our quantum circuit. The entanglement can be spread across all qubits to create a highly entangled state. VQCs often use a layered structure, with each layer

consisting of a set of rotation gates followed by an entanglement scheme. This structure allows for deep and expressive models [15]. Figure 2 illustrates the quantum circuit that was used for this study.

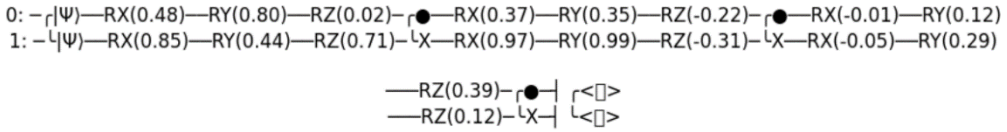

Figure 2: Quantum Circuit

Our investigational framework utilizes a quantum circuit composed of three such layers, analogous to the multi-layered topology of classical neural networks. This tri-layered configuration allows for a profound and intricate representation of the computational space, enhancing the circuit's capacity for binary classification tasks. The empirical application of this architecture has been primarily demonstrated with the MNIST dataset, where the circuit has been tasked to discern between two distinct classes. Through rigorous training, the multilayered quantum circuit learns a mapping from input features to binary outputs, with the final prediction derived from the expected value of an observable by applying *Sigmoid* activation function on it [17]. The *Sigmoid* function given the expectation value $\langle A \rangle$ is as follows:

$$\sigma(\langle A \rangle) = \frac{1}{1 + e^{-\langle A \rangle}} = \frac{e^{\langle A \rangle}}{1 + e^{\langle A \rangle}} = 1 - \sigma(-\langle A \rangle) \tag{4}$$

The paradigm of quantum measurement in the regime of QML diverges from traditional quantum computing. Classical information is retrieved from a quantum system through the act of measurement, a process fundamentally rooted in the projection postulate of quantum mechanics. In the pursuit of comprehensive state characterization, we calculate an observable and take the expected value of it. This observable is meticulously constructed by summing the Pauli X, Y, and Z operators for each qubit, formulated as:

$$\hat{A} = \sum_{i=1}^{n} (X_i + Y_i + Z_i) \tag{5}$$

where X_i, Y_i, and Z_i correspond to the Pauli operators acting on the i^{th} qubit. This summing of operators for the construction of $\hat{A}$ is a deliberate strategy to yield a composite observable that encapsulates the multifaceted nature of quantum information processed by the circuit. The expectation value of an observable $\hat{A}$ given a quantum state $| \psi \rangle$ is mathematically defined as:

$$\langle A \rangle = \langle \psi \mid \hat{A} \mid \psi \rangle \tag{6}$$

This formulation encapsulates the core principle that the expectation value is the statistical mean of all possible measurements of the observable $\hat{A}$ should the quantum state $| \psi \rangle$ be prepared identically numerous times and measured. Thus, it provides a deterministic scalar indicative of the system's average behavior, rather than a single probabilistic outcome. For a quantum state that is a superposition of eigenstates of $\hat{A}$, this value can be seen as the weighted average of the eigenvalues of $\hat{A}$, where the weights are the probabilities of the corresponding eigenstates in the superposition [17].

In the context of PennyLane, a specialized QML framework, and other quantum computing frameworks that utilize quantum circuits for QML, the observable $\hat{A}$ is often a function of the Pauli matrices $\hat{\sigma}_x$, $\hat{\sigma}_y$, and $\hat{\sigma}_z$, which represent measurements in the X, Y, and Z bases, respectively. The Pauli matrices are defined as:

$$\hat{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \hat{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \hat{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{7}$$

The expectation value for a qubit in state $|\psi\rangle$ with respect to a Pauli matrix can then be calculated using the state vector or the density matrix of the system.

4.3 Cost and Gradient-Based Learning

The training of the circuit involves optimizing the parameters by minimizing a cost function that reflects the classification accuracy. The cost function embodies a quantitative measure of the discrepancy between the predicted outcomes of the quantum circuit and the actual labels of the data. It serves as a pivotal guide in the training process, steering the parameter adjustments within the quantum circuit to enhance its predictive accuracy. For binary classification tasks, such as distinguishing between two classes within the MNIST dataset, a commonly employed cost function is the Binary Cross-Entropy (BCE) Loss. This function is specifically tailored to scenarios where the model predictions are probabilities, lying within the interval [0, 1]. The BCE Loss for a single prediction can be mathematically formulated as:

$$J(\Theta) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{8}$$

where y is the true label (0 or 1) and $\hat{y}$ is the predicted probability that the sample belongs to class 1. The BCE Loss is a compelling choice due to its probabilistic interpretation. It penalizes predictions that are confident and wrong with a higher loss, while providing a gentler penalty for uncertain predictions.

In the context of gradient-based learning, the objective is to find the set of parameters Θ that minimizes the cost function $J(\Theta)$. The optimization process involves computing the gradient of $J(\Theta)$ with respect to Θ, denoted as $\nabla_\Theta J(\Theta)$, and iteratively updating Θ in the opposite direction of this gradient:

$$\Theta_{\text{new}} = \Theta_{\text{old}} - \eta\nabla_\Theta J(\Theta) \tag{9}$$

where η is the learning rate, a hyperparameter that determines the step size of the updates. The gradient $\nabla_\Theta J(\Theta)$ is efficiently computed through backpropagation, which in the quantum setting is implemented via the parameter shift rule, as elucidated in Figure 3. This process employs gradient descent or its variants like Adam optimizer [1].

Figure 3 depicts a schematic for the calculation of gradients in a quantum circuit, an essential process for the optimization of variational quantum circuits. The diagram illustrates two separate quantum circuits that are part of a technique known as the parameter shift rule. This rule is a method to compute the gradient of a quantum circuit with respect to its parameters and is widely used due to its simplicity and effectiveness [16]. The circuits are almost identical except for a small shift in the parameter θ of the unitary gate U, which is applied to an initial state $|0\rangle$. In the first circuit, a positive shift $+x$ is applied, resulting in the output state $\hat{y}_{\theta+x}$, and in the second circuit, a negative shift $-x$ results in the output state $\hat{y}_{\theta-x}$.

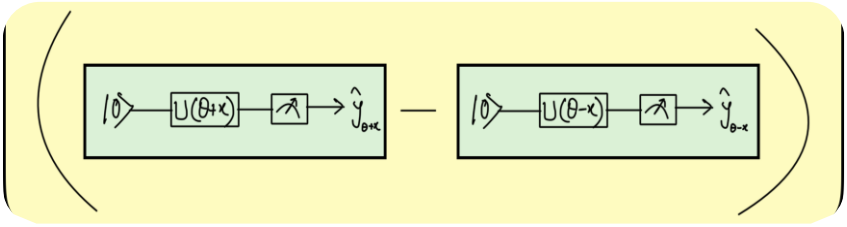

Figure 3: Quantum Parameters Gradient Calculation.

Mathematically, the gradient of the expectation value of an observable $\langle A\rangle$, as exemplified by the output of our Variational Quantum Circuit (VQC) described in Section 4.2, with respect to the parameter θ, can be computed using the outputs of these two circuits as follows:

$$\frac{\partial\langle A\rangle}{\partial\theta} = \frac{y_{\theta+x} - y_{\theta-x}}{2sin(x)} \tag{10}$$

Here, x is a small, non-zero shift value, and the difference in the expectation values obtained from the two circuits is divided by twice the sin of x, giving the gradient of the expectation value with respect to the parameter θ. This value is then used in classical optimization algorithms to update the parameter θ in the direction that reduces the cost function. The parameter shift rule is especially powerful in quantum circuits because it provides a way to compute gradients without needing to rely on finite differences, which can be error-prone and inefficient. It leverages the inherent properties of quantum gates and their parameterized rotations, allowing for more accurate and efficient optimization of quantum circuits [16].

The efficacy of the BCE Loss function, coupled with the parameter shift rule for gradient computation, provides a robust framework for training variational quantum circuits. It allows for the systematic tuning of parameters to achieve convergence on datasets such as MNIST, even within the constraints of limited quantum hardware resources and computational time.

5. Evaluation and Results

The performance of our quantum machine learning (QML) model is depicted through the trends observed in Figures 4 and 5. These figures chronicle the evolution of the model's loss and accuracy metrics through successive training epochs, providing a comprehensive view of the model's learning trajectory and performance stability.

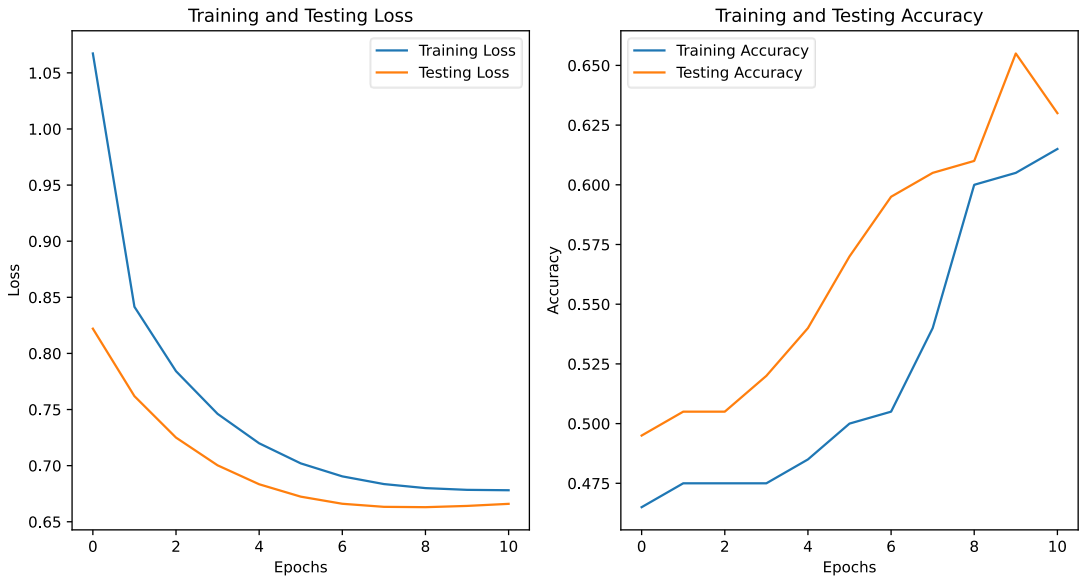

Figure 4: Train/Test Loss and Accuracy curves for a learning rate of 0.003

Figure 4 captures the initial phase of model training, characterized by a steep descent in both training and testing loss, which indicates that the model is effectively learning from the dataset. The sharp rise in accuracy during the same early epochs underscores the model's swift adaptation to the features distinguishing the binary classes of the MNIST dataset.

Expanding the evaluation over an extended training period, Figure 5 displays a more nuanced learning behavior. The training loss exhibits a decelerating descent, eventually stabilizing and suggesting the model's convergence to an optimization plateau. The testing loss, while largely mirroring the training loss's trend, also presents sporadic fluctuations indicative of the model's efforts to generalize its learned parameters to new, unseen data—a hallmark of robust machine learning models. The testing accuracy in Figure 5 illustrates the model's consistent performance on the test dataset with a smaller learning rate, a noteworthy feat considering the complex nature of quantum circuit-based classification tasks. This sustained accuracy level, persisting through the latter epochs, speaks to the model's capability to maintain a stable predictive prowess without succumbing to overfitting—a common pitfall where models exhibit high performance on training data at the expense of their ability to generalize.

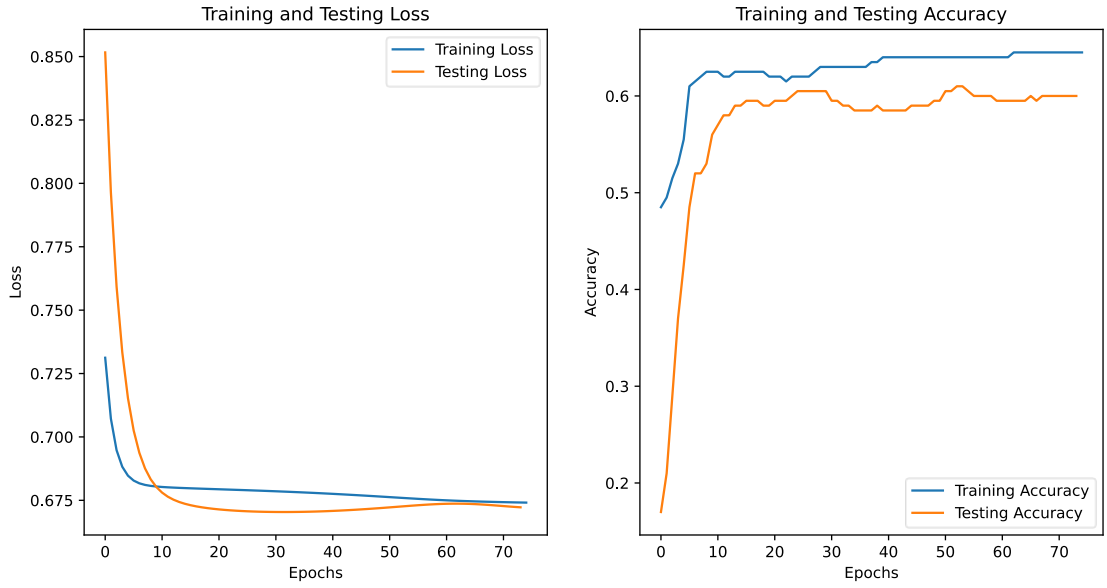

Figure 5: Train/Test Loss and Accuracy curves for a learning rate of 0.001

The accuracy scores, while not comparable to state-of-the-art classical models, are significant in the context of quantum computing. Achieving these results with a mere 2 qubits underscores the potential of quantum models to process and classify information with substantially fewer resources than classical counterparts. It hints at the possibility of exponential efficiency gains as quantum hardware evolves and more qubits become available for computational tasks.

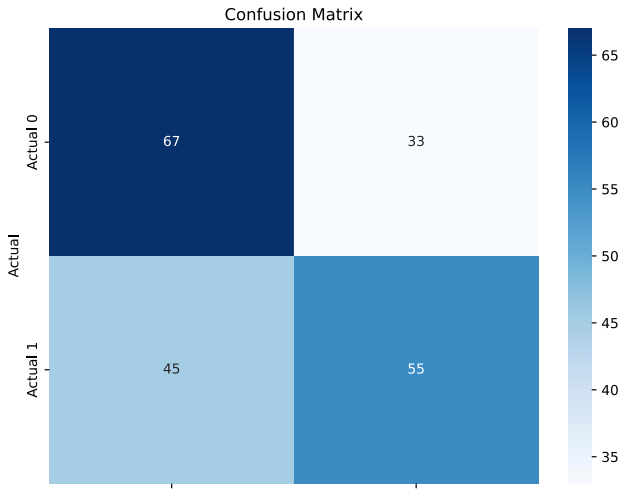

Figure 6: Confusion Matrix generated for Test Data.

This detailed evaluation serves not only as a testament to the model's current classification capabilities but also as a basis for future enhancements. It's clear that the model, in its current form, establishes a foundation for effective binary classification. Yet, the results also invite further exploration into the scalability of such QML models, particularly in handling multi-class classification scenarios and larger, more diverse datasets. The

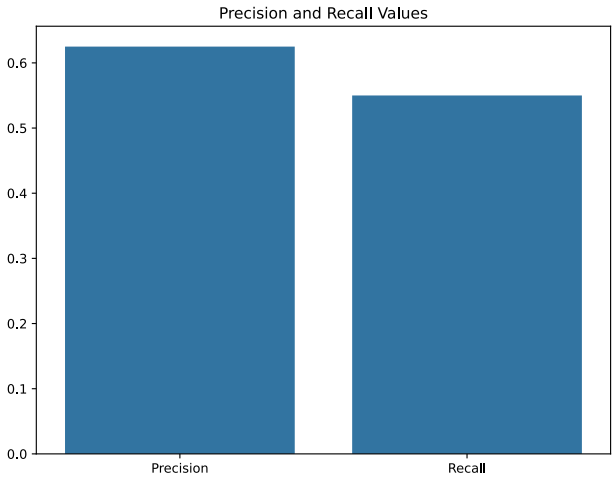

Figure 7: Precision and Recall on Test Data.

model's learning dynamics and performance indicators, as encapsulated in Figures 4 and 5, demonstrate a successful convergence of QML in image classification. They also open avenues for future research to push the boundaries of quantum computing in machine learning, exploring the full extent of its potential in various complex classification tasks. Despite the nascent stage of quantum hardware, the QML model demonstrates a capability that exceeds traditional binary classification methods by leveraging the quantum system's inherent parallelism. The model's modest accuracy scores, while not yet surpassing classical benchmarks, reflect the initial exploratory steps into the realm where quantum advantage begins to materialize.

Figures 6 and 7 detail the confusion matrix and precision-recall curve respectively. The confusion matrix (Figure 6) visualizes the model's classification accuracy by delineating the true positives and negatives against the instances of false positives and negatives. This depiction not only assesses the model's predictive precision but also highlights its sensitivity and specificity in distinguishing the classes. The precision-recall curve (Figure 7) extends this analysis, mapping the trade-offs between precision—the proportion of true positive results among all positive predictions—and recall—the proportion of true positive results across all actual positives. It accentuates the model's response at varying threshold levels, offering insights into the balance it maintains between inclusivity and exclusivity of the class predictions.

The evaluation concludes that even with a mere two-qubit system, the project exhibits a significant stride in quantum computational applications. It posits a future where expanded quantum systems could potentially handle complex, high-dimensional datasets with speed and efficiency currently unattainable by classical means. As the field matures, we anticipate quantum algorithms to become increasingly integral to solving intricate computational problems, with this project serving as an early indicator of such a revolutionary trajectory.

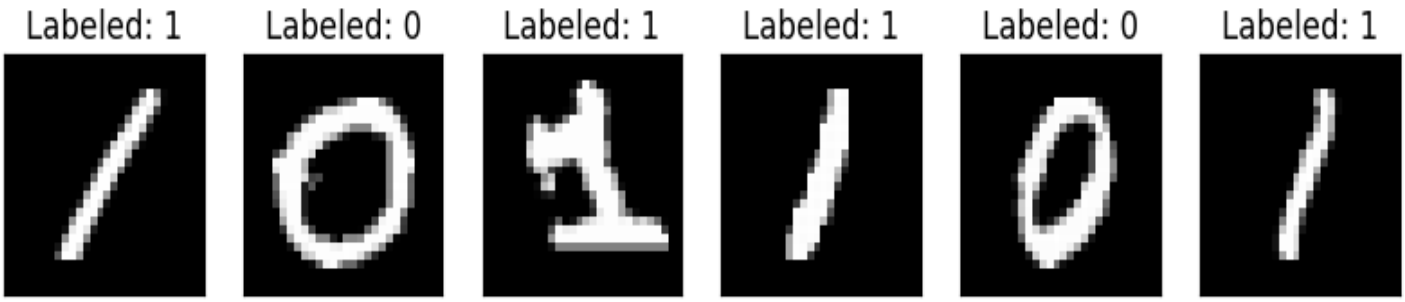

Figure 8: A sample of test data.

6. Conclusion

This project focused on the intersection of quantum computing and image classification, affirms the potential for quantum approaches to enhance computational abilities in machine learning. Through the meticulous construction and optimization of a quantum machine learning circuit, we have demonstrated a fundamental capability for binary image classification using a quantum system. The modest yet promising results reinforce the potential for quantum algorithms to address complex computational problems beyond the reach of current classical methods.

This project, although preliminary, sets a precedent for future research, especially as quantum hardware evolves and algorithms become more sophisticated. The promise shown by Quantum Machine Learning in this study ignites a vision for its broader application in artificial intelligence, potentially revolutionizing the field with quantum advantages. This project not only underscores the current capabilities but also the immense future potential of quantum computing in machine learning.

As quantum hardware and algorithms continue to evolve, the anticipation for a quantum advantage in various domains grows. This project serves as a testament to the progress made and a beacon for future endeavors that will undoubtedly push the boundaries of quantum machine learning further. The knowledge gained here lays a groundwork for continued exploration, ultimately leading to breakthroughs that may revolutionize data processing and analytics in the quantum era.

Bibliography

1. Ruan, Yue & Xue, Xiling & Shen, Yuanxia. (2021). Quantum Image Processing: Opportunities and Challenges. Mathematical Problems in Engineering. 2021. 1-8. 10.1155/2021/6671613.

2. Yan, F., Iliyasu, A.M. & Venegas-Andraca, S.E. A survey of quantum image representations. Quantum Inf Process 15, 1–35 (2016). https://doi.org/10.1007/s11128-015-1195-6.

3. Salvador E Venegas-Andraca and Sougato Bose. Storing, processing, and retrieving an image using quantum mechanics. In Eric Donkor, Andrew R. Pirich, and Howard E. Brandt, editors, Quantum Information and Computation, volume 5105, pages 137 – 147. International Society for Optics and Photonics, SPIE, 2003.

4. Phuc Q Le, Fangyan Dong, and Kaoru Hirota. A flexible representation of quantum images for polynomial preparation, image compression, and processing operations. Quantum Information Processing, 10(1):63–84, 2011.

5. Li, P., Xiao, H. & Li, B. Quantum representation and watermark strategy for color images based on the controlled rotation of qubits. Quantum Inf Process 15, 4415–4440 (2016). https://doi.org/10.1007/s11128-016-1413-x.

6. Rabia Amin Khan. An improved flexible representation of quantum images. Quantum Information Processing, 18(7):1–19, 2019.

7. Yi Zhang, Kai Lu, Yinghui Gao, and Mo Wang. NEQR: a novel enhanced quantum representation of digital images. Quantum information processing, 12(8):2833–2860, 2013.

8. RiGui Zhou, WenWen Hu, GaoFeng Luo, XingAo Liu, and Ping Fan. Quantum realization of the nearest neighbor value interpolation method for INEQR. Quantum Information Processing, 17(7):1–37, 2018.

9. Hai-Sheng Li, Xiao Chen, Shuxiang Song, Zhixian Liao, and Jianying Fang. A block-based quantum image scrambling for GNEQR. IEEE Access, 7:138233–138243, 2019.

10. Caraiman, S., Manta, V. (2013). Image Representation and Processing Using Ternary Quantum Computing. In: Tomassini, M., Antonioni, A., Daolio, F., Buesser, P. (eds) Adaptive and Natural Computing

Algorithms. ICANNGA 2013. Lecture Notes in Computer Science, vol 7824. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37213-1_38.

11. Cerezo, M., et al. “Variational Quantum Algorithms.” Nature Reviews Physics, vol. 3, no. 9, Aug. 2021, pp. 625–44. https://doi.org/10.1038/s42254-021-00348-9.

12. Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. 2022. Data Encoding Patterns for Quantum Computing. In Proceedings of the 27th Conference on Pattern Languages of Programs (Virtual Event) (PLoP ’20). The Hillside Group, USA, Article 2, 11 pages. https://doi.org/10.5555/3511065.3511068.

13. L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.

14. Sim, Sukin, et al. “Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms.” Advanced Quantum Technologies, vol. 2, no. 12, Oct. 2019. https://doi.org/10.1002/qute.201900070.

15. Benedetti, Marcello, et al. “Parameterized Quantum Circuits as Machine Learning Models.” Quantum Science and Technology, vol. 4, no. 4, Nov. 2019, p. 043001. https://doi.org/10.1088/2058-9565/ab4eb5.

16. Schuld, Maria, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. "Evaluating analytic gradients on quantum hardware." Physical Review A 99, no. 3 (2019): 032331. https://doi.org/10.1103/PhysRevA.99.032331.

17. Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco (eds.). From Natural to Artificial Neural Computation. Lecture Notes in Computer Science. Vol. 930. pp. 195–201. doi:10.1007/3-540-59497-3_175.

18. Nielsen, M., & Chuang, I. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511976667.

Appendix

Code used in the project can be found at https://github.com/akmmuhitulislam/quantum-computing-courseproject. The notebook “QCproject.ipynb” contains all code.

Run this code to install all the required dependencies:
!pip install scikit-learn scikit-image scipy torch torchvision seaborn

```
import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from braket.aws import AwsDevice

from braket.circuits import Circuit

from scipy.ndimage import sobel
```

```
from scipy.ndimage import zoom
from collections import Counter
from sklearn.decomposition import PCA
import pennylane as qml
import pickle
import torch
from torch.nn import BCELoss
from torch.utils.data import DataLoader, SubsetRandomSampler, TensorDataset
from torchvision import datasets, transforms

def extract_features(image):
    """
    Extract edge features from the image using the Sobel filter.

    :param image: 2D numpy array representing the image.
    :return: 2D numpy array representing the edge features.
    """
    sx = sobel(image, axis=0, mode='constant')  # Sobel filter on x-axis
    sy = sobel(image, axis=1, mode='constant')  # Sobel filter on y-axis
    sobel_edges = np.hypot(sx, sy)  # Combine the two gradients
    return sobel_edges


def apply_pca(features, num_components=10):
    """
    Apply PCA to reduce the dimensionality of the image features.

    :param features: Flattened array of image features.
    :param num_components: Number of principal components to keep.
```

```
    :return: Reduced feature vector.
    """
    pca = PCA(n_components=num_components)
    reduced_features = pca.fit_transform(features.reshape(1, -1))
    return reduced_features.flatten()



def apply_sobel_filter_to_dataset(dataset):
    """
    Apply the Sobel filter to each image in the dataset.

    :param dataset: 3D numpy array where each 2D array represents an image.
    :return: 3D numpy array of images after applying the Sobel filter.
    """
    edge_dataset = np.array([extract_features(image) for image in dataset])
    return edge_dataset

def apply_pca_to_dataset(dataset, num_components=10):
    """
    Apply PCA to reduce the dimensionality of a dataset with edge features.

    :param dataset: 3D numpy array where each 2D array represents an edge-featured image.
    :param num_components: Number of principal components to keep.
    :return: 2D numpy array with reduced dimensionality, PCA model.
    """
    # Flatten each image and stack them into a 2D array
    flattened_images = np.array([image.flatten() for image in dataset])

    # Apply PCA
```

```
    pca = PCA(n_components=num_components)
    reduced_dataset = pca.fit_transform(flattened_images)
    return reduced_dataset, pca

# The number of qubits
num_qubits = 2
# Load MNIST dataset
batch_size = 16
n_samples = 100  # We will concentrate on the first 100 samples
# Use pre-defined torchvision function to load MNIST train data
X_train = datasets.MNIST(
    root="./data", train=True, download=True, transform=transforms.Compose([transforms.ToTensor()])
)
# Filter out labels (originally 0-9), leaving only labels 0 and 1
idx = np.append(
    np.where(X_train.targets == 0)[0][:n_samples],
    np.where(X_train.targets == 1)[0][:n_samples],
)
X_train.data = X_train.data[idx]
X_train.targets = X_train.targets[idx]

# Use pre-defined torchvision function to load MNIST test data
X_test = datasets.MNIST(
    root="./data", train=False, download=True, transform=transforms.Compose([transforms.ToTensor()])
)

# Filter out labels (originally 0-9), leaving only labels 0 and 1
idx = np.append(
    np.where(X_test.targets == 0)[0][:n_samples],
```

```
    np.where(X_test.targets == 1)[0][:n_samples],
)
X_test.data = X_test.data[idx]
X_test.targets = X_test.targets[idx]

x_train, y_train, x_test, y_test = X_train.data.numpy(), X_train.targets.numpy(), X_test.data.numpy(), X_test.targets.numpy()

# Apply Sobel filter to both training and test datasets
edge_x_train = apply_sobel_filter_to_dataset(x_train)
edge_x_test = apply_sobel_filter_to_dataset(x_test)

# Apply PCA to the edge feature datasets
reduced_edge_x_train, pca_model = apply_pca_to_dataset(edge_x_train, num_components=num_qubits)
reduced_edge_x_test = pca_model.transform(np.array([image.flatten() for image in edge_x_test]))

# Convert the numpy arrays to PyTorch tensors
reduced_edge_x_train_torch = torch.tensor(reduced_edge_x_train, dtype=torch.float32, requires_grad=False)
reduced_edge_x_test_torch = torch.tensor(reduced_edge_x_test, dtype=torch.float32, requires_grad=False)

# Convert the labels to PyTorch tensors as well
y_train_torch = torch.tensor(y_train, dtype=torch.double)
y_test_torch = torch.tensor(y_test, dtype=torch.double)

# Define the size of the test subset
test_subset_size = 20

# Generate shuffled indices for the test set
indices = np.arange(len(reduced_edge_x_test_torch))
np.random.shuffle(indices)
```

```
indices = indices[:test_subset_size]  # Take only 20 indices for the test subset

# Create TensorDataset objects for training and a subset of testing
train_dataset = TensorDataset(reduced_edge_x_train_torch, y_train_torch)
test_dataset = TensorDataset(reduced_edge_x_test_torch, y_test_torch)

# Create a SubsetRandomSampler for the test subset
test_sampler = SubsetRandomSampler(indices)

# Create DataLoader for batched training
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)

# Create DataLoader for the test subset using the sampler
test_loader = DataLoader(dataset=test_dataset, batch_size=test_subset_size, sampler=test_sampler)

# Set the number of layers for the VQC
num_layers = 3

# Braket device
dev = qml.device("braket.local.qubit", wires=num_qubits)

# Define a custom observable
def custom_observable(num_qubits):
    # Sum of PauliX, PauliY, and PauliZ for each qubit
    observable = sum([qml.PauliX(i) + qml.PauliY(i) + qml.PauliZ(i) for i in range(num_qubits)])
    return observable

@qml.qnode(dev, interface='torch')
def quantum_circuit(params, features):
```

```
    # Amplitude Encoding
    qml.AmplitudeEmbedding(features, wires=range(num_qubits), pad_with=0., normalize=True)

    for l in range(num_layers):
        # Parameterized circuit/Ansatz for classification
        for i in range(num_qubits):
            qml.RX(params[l, i, 0], wires=i)
            qml.RY(params[l, i, 1], wires=i)
            qml.RZ(params[l, i, 2], wires=i)

        # Entangling layer
        for i in range(num_qubits - 1):
            qml.CNOT(wires=[i, i + 1])

    # Measurement of the custom observable
    return qml.expval(custom_observable(num_qubits))



# # Cost Function
def cost(params, feature, label):
    #print((torch.abs(quantum_circuit(params, feature)) >= 0.5).int(),torch.abs(quantum_circuit(params, feature)), label)
    return criterion(torch.sigmoid(quantum_circuit(params, feature)), label)

# Getting both cost and accuracy for illustration purposes
def compute_accuracy_and_loss(circuit, params, feature, label):
    correct = 0
    total_loss = 0
    for f, l in zip(feature, label):
        prediction = torch.sigmoid(circuit(params, f))
```

```
        #print(prediction)
        pred_label = (prediction >= 0.5).int()
        loss = criterion(prediction, l)
        total_loss += loss
        #print(prediction,pred_label,l,loss,total_loss)
        if pred_label == l:
            correct += 1

    return correct, total_loss



# # Initialize parameters
#params = torch.tensor(np.random.rand(3 * num_qubits), requires_grad=True)
params = torch.tensor(np.random.rand(num_layers, num_qubits, 3), requires_grad=True)
# opt = qml.GradientDescentOptimizer(stepsize=0.5)

# print(cost_function(params,reduced_edge_x_train[:10],y_train[:10]))

opt = torch.optim.Adam([params], lr = 0.003)
criterion = BCELoss()
epochs = 200



# Initialize variables to store metrics
train_losses = []
train_accuracies = []
test_accuracies = []
test_losses = []
```

```
best_accuracy = 0
best_params = None

# Training loop
for epoch in range(epochs):
    total_loss = 0
    correct_train = 0

    # Training
    for data, label in train_loader:
        opt.zero_grad()
        correct, loss = compute_accuracy_and_loss(quantum_circuit, params, data, label)
        correct_train += correct
        total_loss += loss.item()
        # Calculate the gradients
        loss.backward()
        # Do the gradient update
        opt.step()

    avg_train_loss = total_loss / len(train_loader)
    train_losses.append(avg_train_loss)
    train_accuracy = correct_train / len(reduced_edge_x_train_torch)
    train_accuracies.append(train_accuracy)

    # Testing
    total_test_loss = 0
    correct_test = 0
    with torch.no_grad():
        for data, label in test_loader:
```

```
            correct, loss = compute_accuracy_and_loss(quantum_circuit, params, data, label)
            total_test_loss += loss.item()
            correct_test += correct

    avg_test_loss = total_test_loss / len(test_loader)
    test_accuracy = correct_test / test_subset_size
    test_losses.append(avg_test_loss)
    test_accuracies.append(test_accuracy)

    # Save the best model
    if test_accuracy > best_accuracy:
        best_accuracy = test_accuracy
        best_params = params.clone()  # Clone the best parameters
        data_to_save = {
            "best_params": best_params,
            "train_accuracies": train_accuracies,
            "training_losses": train_losses,
            "test_accuracies": test_accuracies,
            "test_losses": test_losses
        }

        # Save the data to a file
        filename = 'quantum_model_torch_training_data_001_multi_2.pkl'
        with open(filename, 'wb') as file:
            pickle.dump(data_to_save, file)
        print(f"Best model saved to {filename}")

    print(f"Epoch {epoch+1}/{epochs}, Train Loss: {avg_train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Test Loss: {avg_test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

# Plotting
```

```
plt.figure(figsize=(12, 6))

# Plot Training and Testing Loss
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.plot(test_losses, label='Testing Loss')
plt.title('Training and Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot Training and Testing Accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(test_accuracies, label='Testing Accuracy')
plt.title('Training and Testing Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Save the plots as SVG files
plt.savefig('quantum_model_torch_training_data_001_multiemb_training_testing_plots_002.svg', format='svg')

plt.show()

## Testing and evaluating pretrained model
file_path = 'quantum_model_torch_training_data_001_multiemb.pkl'
with open(file_path, 'rb') as file:
    data = pickle.load(file)
```

```
best_params = data['best_params']

train_accuracies = data['train_accuracies']

train_losses = data['training_losses']

test_accuracies = data['test_accuracies']

test_losses = data['test_losses']

from sklearn.metrics import confusion_matrix, precision_recall_fscore_support


# Assuming test_labels are your actual labels for the test dataset

confusion_mat = confusion_matrix(y_test_torch, test_predictions)

precision, recall, _, _ = precision_recall_fscore_support(y_test_torch, test_predictions, average='binary')


import seaborn as sns

from sklearn.metrics import confusion_matrix


# Calculate the confusion matrix

conf_matrix = confusion_matrix(y_test_torch, test_predictions)


# Plot

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.savefig('confusion_matrix.svg', format='svg')


from sklearn.metrics import precision_recall_fscore_support


# Calculate precision and recall
```

```
precision, recall, _, _ = precision_recall_fscore_support(y_test_torch, test_predictions, average='binary')


# Plot
plt.figure(figsize=(8, 6))
sns.barplot(x=['Precision', 'Recall'], y=[precision, recall])
plt.title('Precision and Recall Values')
plt.savefig('precision_recall.svg', format='svg')
```