Google Summer of Code Project Proposal

Quantum Convolutional Neural Networks for High Energy Physics Analysis at the LHC

Eraraya Ricardo Muten
erarayaricardo.m@students.itb.ac.id
+62-812-1224-2740

Bandung Institute of Technology, Bandung
Bachelor of Engineering in Engineering Physics
Indonesia/GMT +7
Twitter GitHub LinkedIn Gitter: @eraraya-ricardo
To give feedback, please contact the email address.

Contents

1 Overview

1.1 Project Abstract

One of the challenges in High-Energy Physics (HEP) is events classification, which is to predict whether an image of particle jets belongs to events being sought after or just background signals. Classical Convolutional Neural Network (CNN) has been proven a powerful algorithm in image classification, including jets image. As quantum computers promise many advantages over classical computing, comes a question on whether quantum machine learning (QML) can give any improvement in solving the problem.

This project aims to demonstrate quantum machine learning's potential, specifically Quantum Convolutional Neural Network (QCNN), in HEP events classification from image data. Although many previous works have tried to classify images with QCNN, none of them is fully quantum. They were still incorporating classical fully-connected layers after variational circuits. This project will be one of the first to try classifying images with a fully quantum implementation of QCNN and probably the first one to do so with particle jets images.

1.2 Benefits to Community

QML is still in its infancy, and it needs more research before we can conclude any concrete advantage compared to classical methods. Every little step counts. The open-source community will benefit from both the study conducted and the source codes produced during the project. The community can use the source codes as a starting template for further investigation. And since the documentation will also be written in tutorial-like form, this project acts as learning resources for people interested in the field.

I also believe that a good contribution to the open-source community is the one that uses existing open-source projects. Other than the usual suspects (TensorFlow, Keras, TensorFlow Quantum, Cirq, Scikit-Learn, etc.), this project will also utilize Quple from last year's QML-HEP GSoC project to boost Quple's exposure and to maintain the continuity between QML-HEP GSoC projects every year.

2 Goals and Deliverables

2.1 Goals

Based on the project description page, this project aims to explore and demonstrate that quantum computing can be the new paradigm (Proof of Principle) for HEP. In addition, the project also would like to develop a common QML interface for HEP which can support different quantum frameworks such as TensorFlow Quantum (TFQ).

After asking the mentors about the expected result from this project, Dr. Sergei Gleyzer mentioned that the project's focus would be more towards scientific research rather than software or framework development.

2.2 Deliverables

Considering the goals and the time available, I divide the deliverables into two categories: required and optional. Required means the deliverables must be finished during GSoC and are the project's main aims, while optional means the deliverables will be completed if time allowed.

Required

1. Python code(s) that contains:

   - Code to train a QCNN model (including the fine-tuning) with several different architectures (encoding methods and variational ansatzes).
   - Code to train several selected classical machine learning models.
   - Code to compare the performance of the QCNN models to the classical machine learning models.

   The quantum programming frameworks that will be used for all the codes are Cirq[1] and TFQ[2]. The Quple framework will be used whenever it supports circuit templates that are needed.

2. Train QCNN models (including the fine-tuning) using tutorial-like Jupyter Notebook(s) that used the Python codes above and compare them to the classical machine learning models. This notebook doubles as the documentation for the code.

3. A list of bugs (if any) that are found in the Quple framework during the course of the project. The list will be reported to the Quple's maintainer at the end of the project.
4. A white paper contains a summary of previous related works, an explanation of the QCNN model implementation and how it is trained, and an analysis of their performance compared to the classical machine learning models. This white paper also doubles as GSoC final report.

Optional

1. If the project's final results are considered published-worthy by the mentors, the white paper will be written into a research paper and will be submitted to a conference, poster, talk/workshop, or journal.
2. Contribute to Quple, by integrating some of the best performing QCNN architectures into the Quple framework.

3 Related Works and Recommendations

3.1 Related Works

This project has two essential parts: the dataset (including preprocessing) and the circuit architecture. The circuit architecture itself consists of classical data encoding circuit and variational circuit. This section will discuss some related works and my past experience (including the experience from working on the evaluation test).

3.1.1 Dataset and Preprocessing

Dataset

Ideally, we want to do experiments on a well-known dataset, so there would be many papers that we can use as both inspirations and comparisons to our models' performance. Since the project description page doesn't specify what dataset(s) should be used, I assumed that the applicants are free to propose and discuss ideas with mentors during the project. But looking at the evaluation test, the project most probably will use a similar dataset to the Task III of the evaluation test, images of particle jets.

Chen et al. have tried to use their version of QCNN to classify similar dataset, images of simulated particle trajectories measured by the LArTPC detector[3] (we will call this dataset as LArTPC dataset). Some previous works also used similar datasets but with classical CNN[4, 5, 6, 7]. DeepLearnPhysics has created public datasets of this kind of image.

CERN also has provided some public LHC datasets. There are two datasets there that I believe would be appropriate for this project. The first dataset is the 25x25 simulated jet images that contain one of the two classes, either signal (i.e., W boson) or background (i.e., QCD)[8]. This dataset was used to train a GANs model to reproduce the images[9]. Figure 1 shows an example of images from the first dataset. The second dataset is the simulated electromagnetic jet images with different sizes of images for each calorimeter layer[10].

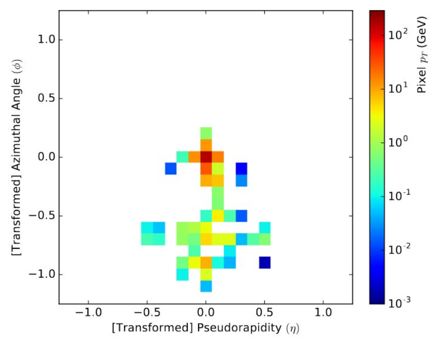


Figure 1: An example of images from [8].

As far as this preliminary research goes, I believe images of particle jets (or dataset that similar to it) is the best candidate for this project.

Dimensionality Reduction

Considering the number of qubits that we can use is limited, dimensionality reduction becomes very crucial. An ideal method is to reduce the total number of features (pixels) to about 10-20 features but still maintaining the dataset variance as much as possible. Cropping the image's edge may be one of the easiest ways, but this is very dataset-dependent since there are still probably significant pixels value at the edge of the image (as happened with the Task III dataset). Also, even after cropping, we usually end up with images that are still too large and need further processing. In the TensorFlow MNIST Classification tutorial (and [11]), bilinear interpolation from TensorFlow is used to reduce the image size. But as we can see in those works, a significant amount of the images from different labels became pretty much the same (as also happened with the Task III dataset).

Skolik et al.[12] and Mardirosian[13] have tried to use Principal Component Analysis (PCA) to reduce the dimension of the MNIST dataset[14] before classifying it with a quantum classifier. From my experience working on the undergraduate thesis (further explanation in section 5.1), I found that PCA is indeed very effective. We can easily determine how much variance percentage left in the reduced dataset by calculating the cumulative sum of all used principal components' eigenvalues. There will be a trade-off between the number of features left and variance in the dataset. A low number of features would be ideal, but the variance left in the reduced dataset might be too low to be easily separable by the classifier. This needs to be fine-tuned during model development.

One weakness of PCA is that standardization (z-score), a critical preprocessing step before PCA, won't work if the number of samples is too small since the standard deviation will be closed to zero. This is what happened with the Task III dataset, 100 data samples are too small. The distribution of training and testing samples might also differ and will render standardization and PCA useless for evaluating testing samples. But I believe this won't be a problem in the actual project since the number of samples used for training (and testing) will definitely much more than what was used in Task III of the evaluation test.

Normalization

The values of the dataset's features must also be normalized to a certain range based on the data encoding method that will be used. Angle encoding is one of the most widely used in the QML community, where the feature value will be treated as rotation angle for a rotation gate. There are no strict rules on normalization. Some research normalized the value to [-pi, pi], some others to [-1, 1] or [0,1]. From my experience, either of them is good and doesn't affect the performance too much. The one thing for certain is that the normalization range must not surpass [-pi, pi] since it is a rotation angle. Another way to normalize the value is by making the feature vector of every sample have its length equal to 1. This is especially necessary for the amplitude encoding method[15]. Amplitude encoding is very effective when dealing with high-dimensional dataset. It only needs $\log_2(N)$ qubits where N is the total number of features. Considering that PCA or convolution will be used, the total number of features should not be too big, and amplitude encoding might not be needed. It is not trivial to determine which normalization will give the best performance since it depends on the dataset itself and the rest of the architecture. This also needs to be fine-tuned during model development.

Convolution

Another thing to note is that we may not need dimensionality reduction if we do convolution-like operations to the images with quantum circuits. To be fair, using PCA to reduce the image's dimension is actually not in the spirit of QCNN, where 'convolution' should be the one that reduces the image's dimension. Many previous research papers attempt to do convolution-like operations with quantum circuits, which will be discussed in the section 3.1.2.

3.1.2 Circuit Architecture

Two parts of the circuit will be trained in this project: the convolution part (including the encoding method) and the classification part. Before we discuss them, I have a short note on Cong et al. implementation of QCNN[16].

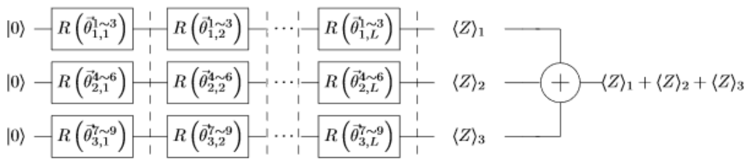
Note on Cong et al. Implementation

Even though the Task III of the evaluation test seems like a hint that what mentors mean by "QCNN" is the QCNN from Cong et al. implementation, I am honestly a little bit skeptical that this implementation could work well on classical images. In the original paper itself, the authors used the algorithm for quantum phase recognition, not classical images. The encoding method that we use to convert classical images into quantum states may not necessarily produce the 'correct' kind of quantum states that this QCNN implementation can efficiently recognize. In the 'Outlook' section of the original paper, the authors also didn't mention anything regarding the possibility of extending this implementation for classical images. I also can't find papers that tried to classify classical images using this implementation.

Convolution Part

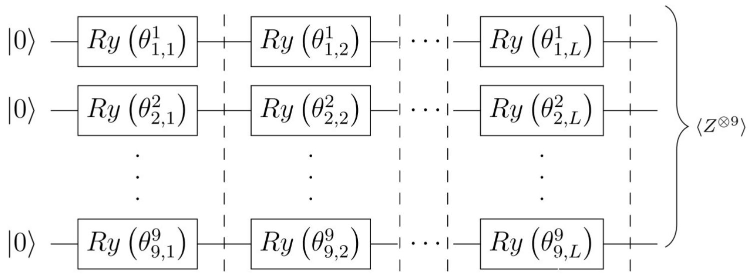
In my opinion, the better implementation of QCNN for classical images is the one that does intermediate measurements[17] and treats the quantum circuits as convolutional filters similar to classical CNN instead of continuing the circuit from start to finish like Cong et al. implementation. For example, if we choose to use a $2x2$ filter to convolve an image with this circuit, we take all pixels covered by the filter (4 pixels in this case) and feed the value to an encoding circuit. A trainable variational circuit then follows that encoding circuit. Measurement is done at the end of the circuit, giving a single value of output. This output is the result of convolution operation on those $2x2$ pixels (just like classical CNN). It can then be either feed to the subsequent convolution layer, pooling layer, or classification part. Since we only feed four values (or F^2 for FxF filter size) to the quantum circuit, there is no concern about the limited number of qubit available. If we use angle encoding, we only need four qubits for this circuit. Also, the convolution operations will reduce the image's dimension along the way before it enters the classification part, eliminating the need for dimensionality reduction. After every convolution layer, we could do a pooling operation to reduce the dimension further and reduce the computational load.

This kind of implementation has been done many times, with different circuit ansatz. In LArTPC dataset classification, Chen et al.[3] used angle encoding followed by entangling layer with CNOT gate and single-qubit unitary parameterized rotations layer as the convolution circuit. A similar but simpler implementation has been done by Liu et al.[18] to classify a custom Tetris dataset. In MNIST dataset classification, Henderson et al.[19] used binary encoding (pixels with a higher value than a certain threshold are considered as $|1\rangle$, $|0\rangle$ otherwise) followed by untrainable (fix parameters) randomly generated two-qubit and single-qubit gate as the convolution circuit. In Oh et al.[20] implementation, the convolution circuit is an angle encoding followed by trainable controlled-rotation gates. In my undergraduate thesis, I tested three different ansatzes as convolution circuits (Figure 2). Equation 1 explains the figure's notation, where n, q, and l are the index for the data sample's feature, qubit, and layer, respectively. w and b are weight and bias (trainable parameters), while x is the data sample. This style of parametrization was inspired by a recent paper from Pérez-Salinas et al.[21]. All works mentioned here reported respectable results compared to classical CNN, if not better.



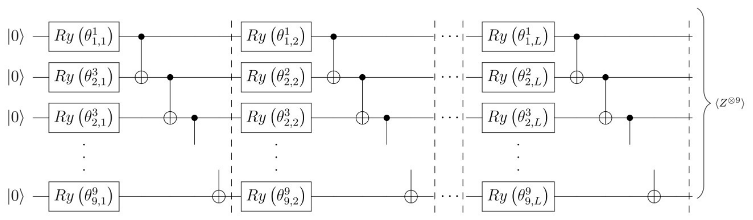
$|0\rangle$
$R\left(\vec{\theta}^{1\sim3}_{1,1}\right)$
$R\left(\vec{\theta}^{1\sim3}_{1,2}\right)$
$R\left(\vec{\theta}^{1\sim3}_{1,L}\right)$
$R\left(\vec{\theta}^{4\sim6}_{2,1}\right)$
$R\left(\vec{\theta}^{4\sim6}_{2,2}\right)$
$R\left(\vec{\theta}^{4\sim6}_{2,L}\right)$
$R\left(\vec{\theta}^{7\sim9}_{3,1}\right)$
$R\left(\vec{\theta}^{7\sim9}_{3,2}\right)$
$R\left(\vec{\theta}^{7\sim9}_{3,L}\right)$
$\langle Z\rangle_1$
$\langle Z\rangle_2$
$\langle Z\rangle_3$
$\langle Z\rangle_1+\langle Z\rangle_2+\langle Z\rangle_3$

(a)



$|0\rangle$
$Ry\left(\theta^1_{1,1}\right)$
$Ry\left(\theta^1_{1,2}\right)$
$Ry\left(\theta^1_{1,L}\right)$
$Ry\left(\theta^2_{2,1}\right)$
$Ry\left(\theta^2_{2,2}\right)$
$Ry\left(\theta^2_{2,L}\right)$
$Ry\left(\theta^9_{9,1}\right)$
$Ry\left(\theta^9_{9,2}\right)$
$Ry\left(\theta^9_{9,L}\right)$
$\langle Z^{\otimes 9}\rangle$

(b)



$|0\rangle$
$Ry\left(\theta^1_{1,1}\right)$
$Ry\left(\theta^3_{2,1}\right)$
$Ry\left(\theta^3_{2,1}\right)$
$Ry\left(\theta^9_{9,1}\right)$
$Ry\left(\theta^1_{1,2}\right)$
$Ry\left(\theta^2_{2,2}\right)$
$Ry\left(\theta^3_{2,2}\right)$
$Ry\left(\theta^9_{9,2}\right)$
$Ry\left(\theta^1_{1,L}\right)$
$Ry\left(\theta^2_{2,L}\right)$
$Ry\left(\theta^3_{2,L}\right)$
$Ry\left(\theta^9_{9,L}\right)$
$\langle Z^{\otimes 9}\rangle$

(c)

Figure 2: Circuit schematics of three ansatzes that I tested as convolution circuits for my undergraduate thesis. The notations in the schematics are explained in Equation 1.

$$\begin{aligned} \vec{\theta}_{\mathrm{q,l}}^{\mathrm{n\sim n+2}} &= \left(\theta_{q,l}^{n}, \theta_{q,l}^{n+1}, \theta_{q,l}^{n+2}\right) \\ \theta_{q,l}^{n} &= w_{q,l}^{n} x^{n} + b_{q,l}^{n} \end{aligned} \tag{1}$$

Note that I assumed we wouldn't consider QCNN implementation that uses QRAM[22, 23] for this project as it takes a lot of resources and hard to implement on NISQ devices.

Classification Part

All works mentioned in the **Convolution Part** (except my undergraduate thesis) used classical fully-connected layers as the follow-up to the convolution circuits to give the prediction results. While this implementation is indeed working, I believe using a variational circuit also for the classification part (fully quantum without any classical neural networks) is more in the spirit of this project's goals. **By using a fully quantum model, this project offers a new research contribution that has never been done before in QCNN-based HEP image data analysis**.

There are many different implementations, and different circuit ansatzes have been proposed as a classifier, a lot more than convolution circuit proposals discussed in the previous section. One implementation that works really well from my experience is the Data Re-uploading Classifier (DRC) by Pérez-Salinas et al.[21]. This implementation, in theory, allows us to classify data with any number of features and any number of classes with just a single qubit. In the NISQ era, this is a very favorable trait. In practice, we may need to increase the number of qubits if there are more classes to be classified to maintain good classification accuracy. But since the number of classes in a HEP dataset is usually small (in the range of $2-5$), $2-3$ qubits will be enough to obtain good accuracy from my experience. My experience with DRC is explained further in section 5.1.

Optimizer

Optimizer is an integral part of a QML model. Barren plateau is a known issue in QML, and choosing the right optimizer may lead to a better model. For example, Quantum Natural Gradient has been found to converge faster and more stable than regular gradient descent or Adam in optimizing VQEs[24, 25, 26]. As interesting as it is, I believe the time given for this project may not be enough to explore the optimizer. Testing different ansatzes and hyperparameter tuning already need a lot of time.

The papers discuss in the "Convolution Part" used starndard optimizer like Adam[27] or RMSprop[28], and it worked. So, in my opinion, using Adam or RMSprop for this project is acceptable. The optimizer exploration for QML may become an interesting topic for the next cohort of GSoC.

3.2 Recommendations

Considering all things mentioned in section 3.1, I suggest trying both these approaches for this project:

1. First approach: Try to see whether PCA works well for the dataset chosen. If yes, then we directly feed the PCA-reduced data to DRC to obtain the prediction. This may sound weird as there is no "convolution" aspect with this approach. But considering it takes less than 5 hours to implement (from experience), I believe it is worth the try. If the performance is good, this may also give us a hint that the DRC works well for the dataset.

2. Second approach: Implement convolution-like operation with trainable variational circuits (intermediate measurements), then classify the resulting transformed features with DRC to obtain the prediction. This implementation is fully quantum, in the sense that we don't use any classical methods/layers in the model. The exact architectural details of the ansatzes for the convolution circuit that will be tested in the project are subject to further discussion with mentors. I believe it is ideal not to deviate too much from some already proposed ansatzes in section 3.1, considering the time is limited and exploring too many different ansatzes usually takes much time.

3. The classical machine learning methods that will be compared to the QCNN are the ones that were used in the last year's GSoC QML-HEP project, which are BDT, SVM, and classical neural network, plus classical CNN. TensorFlow[29] and Scikit-learn[30] will be used for the training.

4. The model will be trained to classify images of particle jets. The dataset classes will be limited to two (binary classification) to shorten the training time.

If there are no changes made after further discussion with mentors during the project, the project will use this setup:

- Dataset: the 25x25 simulated jet images from [8].
- Two approaches will be tested. The first one is PCA followed by DRC, and the second one is QCNN.
- The QCNN approach that will be implemented is the one that does intermediate measurements as explained in the "Convolution Part" of section 3.1. Ansatzes from my undergraduate thesis (Figure 2) and ansatz from Chen et al. implementation[3] will be tested for the convolution circuit. The pooling layer may be added and tested during model development.
- The classification part of QCNN will use the DRC[21]. No classical fully-connected or classical convolution layer will be used.
- The optimizers that will be tested are Adam and RMSprop. If both fail (unlikely, but just in case), other optimizers available in TensorFlow will be used.
- The performance metric of the models that will be compared includes the training and testing loss curve, the training and testing accuracy curve, and the final models' ROC curve and AUC score. This project's main numerical result is the accuracy and AUC score average from the best model of QCNN and all classical machine learning methods tested.
- The exact details of hyperparameters, e.g., number of epochs, batch size, number of convolution and classification layer, learning rate, will vary as they will be fine-tuned during model development.

I am recommending these because I have tried them myself. When I worked on my thesis, I tried many different approaches to classify MNIST dataset, and these approaches were the ones that worked. In my opinion, images of particle jets are very similar to MNIST in the sense that there are only slight variations between images with the same label and the images are sparse (most pixels have zero value).

4 Timeline and Research Plan

The timeline written here is based on the timeline listed on the ML4SCI organization page, not the general timeline by GSoC.

4.1 Application Review Period (14 April - 17 May)

Try and get used to the Quple framework. Implement the DRC in TFQ. I have implemented the DRC in PennyLane and Qiskit. I actually have already planned to implement it in TFQ. So regardless of the discussions with mentors later, the code to implement DRC will be ready if we decide to use it.

4.2 Community Bonding (17 May - 7 June)

The works that need to be done during this period are categorized into three. The timeline may overlap between categories as the works between categories are closely related.

Week 1-2: Planning (17 May – 31 May)

Get feedback from mentors about the proposal. Discuss with mentors the proposal's high-level decisions (goals, priority and deliverables, meeting schedule) that need to be changed and modify them accordingly. After that, discuss with mentors the more low-level decisions (dataset and preprocessing, the whole pipeline including optimizers, ansatzes, coding training procedures) to the minute details and modify the timeline and research plan accordingly.

Week 2-3: Early Works (24 May – 7 June)

Prepare the working environment and repository. Obtain and clean the dataset. Do all necessary preprocessing (normalization and PCA) in Jupyter Notebook.

Do the first approach: if PCA works, train a DRC model by feeding the PCA-transformed dataset directly. Plot and analyze the performance. This will be done in Jupyter Notebook. Make a short report on the performance, and discuss it with mentors at the first meeting in the "Coding" period. This short report will be reused as the final report's material. The first approach finishes here. The "Coding" period will be fully spare for the second approach.

Week 3: Bonding (31 May – 7 June)

I would also like to know more about mentors' and organization's works other than GSoC. My dream is to become a researcher in quantum computing, and this is a precious opportunity to hear from people who are more experienced in the field.

4.3 Coding (7 June - 16 August)

Week 1 (7 June – 14 June)

Code all convolution circuit ansatzes that will be used in Jupyter Notebook or as a separate Python module that can be imported. Suppose it is decided that the project will use the ansatzes that I used for my thesis, I already have the PennyLane implementations of all the convolution circuit ansatzes, so it should not take a lot of time to code them.

Code the QCNN training pipeline and test it by training a model with a small number of training samples in Jupyter Notebook.

Week 2 (14 June – 21 June)

Code and train classical machine learning methods listed in section 3.2 in Jupyter Notebook. Do the training several times, save the performance statistics and the models. These performance statistics will be used as a benchmark when developing the QCNN model.

Make any changes to the first approach report and notebook as per feedback.

Week 3 – 7 (21 June – 26 July) & Evaluations (13 July – 17 July)

This is the central part of the project, developing the QCNN model. I spent about a month coding and developing good-performing models for my undergraduate thesis. Considering that the training pipeline code is already done in week 2, I believe a month should also be enough for this project.

The strategy is to focus on one type of convolution circuit ansatz per week. Every week, a combination of one convolution circuit ansatz followed by DRC will be trained and fine-tuned. The fine-tuning here includes hyperparameters (number of epochs, batch size, learning rate), normalization, optimizer, number of DRC and convolution layers, pooling layers, etc.

At the end of the week, if the model shows good performance with a potential to be better, we will spend another week fine-tuning the model. If it doesn't show good performance, we will start next week with another convolution circuit ansatz. The decision is made at the weekly meeting with mentors.

We will stick to this strategy and training procedures discussed during Community Bonding, but it is hard to plan for certain every step in the model development since there are many moving parts in it, and every decision depends on the numerical results of the model.

Note that after every training, the configurations, the trained model, and that model's statistical performance will be saved.

Write a short evaluation for mentors during the "Evaluations" period.

Week 8 (26 July – 2 August)

After we obtained the best configuration of the QCNN model, we will focus on that model only and compare it to the performances of the classical models. Some of the classical models may need to be re-trained as the number of trainable parameters of that model may be too small or too large compared to the best QCNN model. We want to compare models with about the same number of trainable parameters to ensure fairness.

Visualize the comparison, e.g., the training and testing loss curve, the training and testing accuracy curve, and the final models' ROC curve and AUC score. Tabulate the average accuracy and AUC score of the best model from both QCNN and classical methods.

Write performance comparison analysis from the visualizations, e.g., which model performs better, how the number of qubits and layers affect QCNN performance, what could have been done to potentially make the QCNN model better. This analysis will be reused as the final report's material.

Write the bugs list in the Quple framework found during the code and model development, report it to Quple maintainer.

Week 9 (2 August – 9 August)

Clean up the Jupyter Notebook for QCNN training. Write comments and markdowns explaining everything done in the notebook. This notebook will serve as documentation and tutorial for people interested in trying QCNN, similar to what I did for the evaluation test but in more detail. PennyLane's research demos are excellent examples of what I have in mind.

Make any changes to the performance analysis made in week 8 as per feedback.

Week 10 (9 August – 16 August)

Write the final report/white paper. This report will contain:

- Short Introduction (project summary, motivation, related works, dataset). Contents in the project proposal can be reused.
- Explain the best QCNN model's implementation that is found during model development and how it was trained in minute details. Most of the contents are actually already written in the tutorial-like documentation made in week 9, so this is just a matter of moving and reformatting them.
- Performance comparison analysis made in week 8 (no reformatting as it can be directly reused).
- If the first approach gives good results, then the short report that was made will be integrated here as well.
- Conclusions and future work recommendations.

Make any changes to the tutorial-like docs made in week 9 as per feedback.

Discuss with mentors whether the white paper will be written as research paper and published or not. Because the date of acceptance of publishers may not in the range of GSoC timeline, I am willing to spend time outside GSoC to write the research paper and make the submission.

4.4 Students Submit Code and Final Evaluations (16 August - 23 August)

Make any changes to the tutorial-like docs made in week 9 and the final report made in week 10 as per feedback.

Write final evaluation for mentors.

If there is still time left, contribute to the Quple framework by integrating successful QCNN architecture into the framework.

This is a spare week in case of some works in the timeline need more time, emergency events, etc.

5 About Me

I recently graduated from the engineering physics department at Bandung Institute of Technology. The department's multi-disciplinary nature has allowed me to obtain knowledge and skillsets ranging from computational methods, digital signal & image processing, data science & machine learning, and quantum mechanics. I am confident with my programming skills. I use Python daily and have experience in MATLAB, C++, C, and LaTeX.

I am the kind of person who always tries my best and puts 100% of my focus into things that I am passionate about. During my undergraduate studies, machine learning and quantum computing are the two fields that excite me the most. In the next section, I explained briefly how I got into those fields and my experience in them.

5.1 Quantum Computing and Quantum Machine Learning

Since I was in junior high, I have always been fascinated by physics, especially quantum physics. In high school, this excitement of learning physics had led me to the national camp for Asian Physics Olympiad, where the top 15 students in physics across the country were invited. I discovered my passion for programming and computing during my undergraduate studies. It is then only natural that I am inclined towards quantum computing when I heard about it. I enjoyed it a lot and decided to take the field seriously.

The first time I heard about quantum computing, I was immediately interested in QML and variational quantum algorithms. I started learning it by reading e-books and participating in online courses, conferences, and summer schools. I picked up **Qiskit**, **PennyLane**, and **Cirq** on the way. I felt the responsibility to give back to the open-source community after getting many free learning resources. Seeing that quantum computing in my country isn't well developed yet, I founded the Indonesian Qiskit docs localization team. We translated Qiskit docs to Indonesian, hoping that it will become more accessible to many more Indonesians. We have translated more than 90% of it (the number is subject to change since Qiskit keeps adding new docs from time to time). It is now undergoing review and to be published soon.

I recently defended my undergraduate thesis in QML with Prof. Andriyan and Dr. Nugraha as advisors. I proposed to use ansatz similar to the Data Re-uploading Classifier (DRC)[21] for convolution and classification of the MNIST dataset. The proposed method obtained a much better binary classification testing accuracy than the previous related works[12, 11, 31, 13], up to 99.7% with PCA reduction (without convolution circuit) and 98.9% with convolution circuit (without PCA, fully quantum). After defending the thesis, I also tried to train the exact same model to classify the Fashion MNIST dataset[32] to see whether the model generalizes well. It obtained 93.7% of testing accuracy with the convolution circuit (without PCA, fully quantum). I also proposed to use the DRC with layerwise learning[12] to classify events in HEP for the Xanadu's QHack Quantum Machine Learning Open Hackathon 2021. The proposal obtained a better AUC score than previous works[33], and my team won second place in the event.

I am now working on another open-source project, making a code tutorial for the **Quantum Graph Recurrent Neural Networks**[34] with Qiskit as part of the Qiskit Advocate Mentorship Program. The codes will be put as entries for Qiskit Textbook. The project will be finished around mid-May.

5.2 Classical Machine Learning

My journey in machine learning started at the end of my second year of undergraduate. I have always been fascinated by the idea of artificial intelligence. I was curious, so I began to learn by myself from the internet. Not long after that, I was accepted as a research intern at Gentiane Venture's Lab, Tokyo Univ. of Agriculture and Technology. I researched the usage of **Convolutional Neural Networks** to classify several types of human touch interaction by learning the data pattern from a force sensor. Real-time testing of the model gave 88% of accuracy in predicting touch interactions given by humans. This internship gave me an excitement to keep working on machine learning projects.

I spent both the summer and winter holidays that came after that doing machine learning projects. In the summer of 2019, I came back to Japan as an intern at IHI Corporation, where I developed microservices to help the company employees train, test, and deploy reinforcement learning agents fast and easily. Later, I participated in my university's research lab and conducted research in utilizing **Error-state Kalman Filter** as the state-estimator and **Diagonal Recurrent Neural Network** (Diagonal RNN) & **LSTM** to make the localization of an autonomous car more reliable. Our proposal reduced 70% of localization errors. This work was then published in IEEE, and I presented the paper at the 6th International Conference on Electric Vehicular Technology. All these deep learning experiences have taught me how to use **TensorFlow**, **Keras**, and **Scikit-Learn**.

In the winter holiday, I interned as an AI engineer at Nodeflux Inc. I developed a prototype of real-time face tracking and blemish removal algorithm to create a filter application for the webcam. I learned how to train a **YOLO** architecture-based model to track and recognize human faces using the **PyTorch** framework. I also designed and coded algorithms for blemish removal using **OpenCV**.

In conjunction with HEP, I have taken a Coursera course in applying machine learning techniques for LHC problems. I learned to use **deep learning & gradient boosting for particle identification** and **Bayesian optimization for tracking system optimization**. I have also taken deep learning courses in **GANs**, **computer vision**, and **sequence models** both in university and online.

6 Other Commitments

- I will have an online summer project with CERN from early July until September. The schedule is quite flexible. All code development part and classical machine learning training will be finished before the

CERN's project started. Model development is demanding, but most of the time will be spent waiting for the training to finish, so it needs regular attention but only for a short amount of time. I believe it won't disrupt the timeline flow, and I will ensure that this will not affect the required deliverables. The CERN's project will also be in the field of QML. I believe this will make me stay focus since both projects are on the same topic.

- I already graduated, and I have nothing else to do this summer. I used to work 10 hours per day when I was still a university student.
- Before CERN's project started, I will devote 8 – 10 hours every day. This would sum to about 60 – 65 hours per week. When the CERN's project started, I will devote 2 – 3 hours every weekday and 7 – 8 hours every weekend. This would sum to about 25 - 30 hours per week. However, I am willing to put in more effort if the work requires.

References

[1] Cirq Developers. *Cirq*. Version v0.10.0. See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors. Mar. 2021. DOI: 10.5281/zenodo.4586899. URL: https://doi.org/10.5281/zenodo.4586899.

[2] Michael Broughton et al. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning*. 2020. arXiv: 2003.02989 [quant-ph].

[3] Samuel Yen-Chi Chen et al. *Quantum Convolutional Neural Networks for High Energy Physics Data Analysis*. 2020. arXiv: 2012.12177 [cs.LG].

[4] R. Acciarri et al. "Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber". In: *Journal of Instrumentation* 12.3 (Mar. 2017), P03011. ISSN: 17480221. DOI: 10.1088/1748-0221/12/03/P03011. arXiv: 1611.05531. URL: https://iopscience.iop.org/article/10.1088/1748-0221/12/03/P03011%20https://iopscience.iop.org/article/10.1088/1748-0221/12/03/P03011/meta.

[5] Laura Dominé and Kazuhiro Terao. "Scalable deep convolutional neural networks for sparse, locally dense liquid argon time projection chamber data". In: *Phys. Rev. D* 102 (1 July 2020), p. 012005. DOI: 10.1103/PhysRevD.102.012005. URL: https://link.aps.org/doi/10.1103/PhysRevD.102.012005.

[6] C. Adams et al. "Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber". In: *Phys. Rev. D* 99 (9 May 2019), p. 092001. DOI: 10.1103/PhysRevD.99.092001. URL: https://link.aps.org/doi/10.1103/PhysRevD.99.092001.

[7] F. Psihas et al. "Context-enriched identification of particles with a convolutional network for neutrino events". In: *Phys. Rev. D* 100 (7 Oct. 2019), p. 073005. DOI: 10.1103/PhysRevD.100.073005. URL: https://link.aps.org/doi/10.1103/PhysRevD.100.073005.

[8] Benjamin Nachman, Luke de Oliveira, and Michela Paganini. *Pythia Generated Jet Images for Location Aware Generative Adversarial Network Training*. Zenodo, Feb. 2017. DOI: 10.17632/4r4v785rgx.1. URL: https://doi.org/10.17632/4r4v785rgx.1.

[9] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. "Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis". In: *Computing and Software for Big Science* 1.1 (Sept. 2017). ISSN: 2510-2044. DOI: 10.1007/s41781-017-0004-6. URL: http://dx.doi.org/10.1007/s41781-017-0004-6.

[10] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. *Electromagnetic Calorimeter Shower Images with Variable Incidence Angle and Position*. Zenodo, Aug. 2017. DOI: 10.17632/5fnxs6b557. URL: https://doi.org/10.17632/5fnxs6b557.

[11] Edward Farhi and Hartmut Neven. "Classification with Quantum Neural Networks on Near Term Processors". In: (2018), pp. 1–21. arXiv: 1802.06002. URL: http://arxiv.org/abs/1802.06002.

[12] Andrea Skolik et al. "Layerwise learning for quantum neural networks". In: *Quantum Machine Intelligence* 3.1 (June 2021), p. 5. ISSN: 2524-4906. DOI: 10.1007/s42484-020-00036-4. URL: http://link.springer.com/10.1007/s42484-020-00036-4.

[13] Sevak Mardirosian. "Quantum-enhanced Supervised Learning with Variational Quantum Circuits". PhD thesis. Leiden University, 2019.

[14] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[15] Maria Schuld and Francesco Petruccione. *Supervised Learning with Quantum Computers*. Quantum Science and Technology. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-96423-2. DOI: 10.1007/978-3-319-96424-9. URL: http://link.springer.com/10.1007/978-3-319-96424-9.

[16] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. "Quantum convolutional neural networks". In: *Nature Physics* 15.12 (Dec. 2019), pp. 1273–1278. ISSN: 17452481. DOI: 10.1038/s41567-019-0648-8. arXiv: 1810.03787. URL: https://www.nature.com/articles/s41567-019-0648-8.

[17] Lukas Franken and Bogdan Georgiev. *Explorations in Quantum Neural Networks with Intermediate Measurements*. ISBN: 9782875870742. URL: http://www.i6doc.com/en/..

[18] Junhua Liu et al. *Hybrid Quantum-Classical Convolutional Neural Networks*. 2019. arXiv: 1911.02998 [quant-ph].

[19] Maxwell Henderson et al. "Quanvolutional neural networks: powering image recognition with quantum circuits". In: *Quantum Machine Intelligence* 2.1 (June 2020), pp. 1–9. ISSN: 2524-4906. DOI: 10.1007/s42484-020-00012-y. arXiv: 1904.04767.

[20] Seunghyeok Oh, Jaeho Choi, and Joongheon Kim. *A Tutorial on Quantum Convolutional Neural Networks (QCNN)*. 2020. arXiv: 2009.09423 [quant-ph].

[21] Adrián Pérez-Salinas et al. "Data re-uploading for a universal quantum classifier". In: *Quantum* 4 (Feb. 2020), p. 226. ISSN: 2521-327X. DOI: 10.22331/q-2020-02-06-226. URL: https://doi.org/10.22331/q-2020-02-06-226.

[22] Yaochong Li et al. "A quantum deep convolutional neural network for image recognition". In: *Quantum Science and Technology* 5.4 (Oct. 2020), p. 044003. ISSN: 20589565. DOI: 10.1088/2058-9565/ab9f93. URL: https://iopscience.iop.org/article/10.1088/2058-9565/ab9f93%20https://iopscience.iop.org/article/10.1088/2058-9565/ab9f93/meta.

[23] Iordanis Kerenidis, Jonas Landman, and Anupam Prakash. *Quantum Algorithms for Deep Convolutional Neural Network*. Tech. rep. Sept. 2019.

[24] James Stokes et al. "Quantum Natural Gradient". In: *Quantum* 4 (May 2020), p. 269. ISSN: 2521327X. DOI: 10.22331/Q-2020-05-25-269. arXiv: 1909.02108. URL: https://quantum-journal.org/papers/q-2020-05-25-269/.

[25] David Wierichs, Christian Gogolin, and Michael Kastoryano. "Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer". In: *Phys. Rev. Research* 2 (4 Nov. 2020), p. 043246. DOI: 10.1103/PhysRevResearch.2.043246. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.2.043246.

[26] Naoki Yamamoto. *On the natural gradient for variational quantum eigensolver*. 2019. arXiv: 1909.05074 [quant-ph].

[27] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[28] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[29] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] *MNIST classification — TensorFlow Quantum*. URL: https://www.tensorflow.org/quantum/tutorials/mnist (visited on 02/06/2021).

[32] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[33] Koji Terashi et al. "Event Classification with Quantum Machine Learning in High-Energy Physics". In: *Computing and Software for Big Science* 5.1 (Dec. 2021), p. 2. ISSN: 2510-2036. DOI: 10.1007/s41781-020-00047-7. arXiv: 2002.09935. URL: https://doi.org/10.1007/s41781-020-00047-7%20http://link.springer.com/10.1007/s41781-020-00047-7.

[34] Guillaume Verdon et al. "Quantum Graph Neural Networks". In: *arXiv* (Sept. 2019). arXiv: 1909.12264. URL: http://arxiv.org/abs/1909.12264.