iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** IX **Month of publication:** September 2022

DOI: https://doi.org/10.22214/ijraset.2022.46914

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue IX Sep 2022- Available at www.ijraset.com

Quantum AI: Deep Learning optimization using Hybrid Quantum Filters

Kaushik Ganguly
MTECH- Data Science & Engineering, Birla Institute of Technology and Science, Pilani, India
Senior Data Scientist, Cognizant Technology Solutions India Pvt. Ltd.

Abstract: ***Deep learning algorithms have shown promising results for different image processing tasks, particularly in remote sensing & image recognition. Till now many studies have been carried out on image processing, which brings a new paradigm of innovative capabilities under the umbrella of intelligent remote sensing and computer vision. Accordingly, quantum processing algorithms have proved to efficiently solve some issues that are undetectable to classical algorithms and processors. Keeping that in mind, a Quantum Convolutional Neural Network (QCNN) architecture along with Hybrid Quantum filters would be utilized supported by cloud computing infrastructures and data centers to provide a broad range of complex AI services and high data availability.***
This research summaries the conventional techniques of Classical and Quantum Deep Learning and it's research progress on real-world problems in remote sensing image processing as a comparative demonstration. Last but not least, we evaluate our system by training on Street View House Numbers datasets in order to highlight the feasibility and effectiveness of using Quantum Deep Learning approach in image recognition and other similar applications. Upcoming challenges and future research areas on this spectrum are also discussed.
Keywords: ***Deep Learning, Quantum Convolutional Neural Network (QCNN), Quantum Computing, Hybrid Quantum-CNN***

I. INTRODUCTION / OBJECTIVE

Objective of the project is to:

1) Make use of Quantum Computing and Deep Learning - CNN to develop a more efficient and outperforming technique that can be applied to solve complex machine learning tasks like digit recognition from high resolution images taken from a distance to identify addresses of congested households too many at a time
2) Use superposition and entanglement theories, which are not seen in traditional computing environments, to achieve high performance through qubit parallelism. Here qubit is referred to as the quantum bit which is basically a unit of quantum information.
3) Extend the key features and structures of existing CNN to quantum systems defined in the many-body Hilbert space is transferred to a classical computing environment, the data size grows exponentially in proportion to the system size, making it unsuitable for efficient solutions. Because data in a quantum environment can be expressed using qubits, the problem can be avoided by applying a CNN structure to a quantum system

II. PROPOSED APPROACH

Quantum Convolutional Neural Network (QCNN) provides a fresh approach to the solution of the problem to solve with CNN using a quantum computing environment, and also an effort to improve the model performance in recognizing digits.

1) The first part of the solution to be introduced proposes a model to effectively solve the classification problem using classical Deep Learning by applying the structure of CNN to the classical convolution environment.
2) The second part introduces a method to check whether the algorithm could be performed by adding a layer using Quantum Computing to the CNN learning model used in the existing computer vision. Hybrid Quantum filters can be introduced to optimize model accuracy and loss.

This model can also be used in small quantum computers, and a hybrid learning model can be designed by adding a quantum convolution layer to the CNN model or replacing it with a convolution layer. If successful, this solution could be applied on UAV's or Quantum Drones (QD) by building an Internet of Quantum Drones (IoQD) network to effectively process high resolution images of house addresses to recognize digits from the later to enhance commercial activities like smart delivery hence making it unique.

III. POTENTIAL CHALLENGES & RISKS

A. Issues, knowledge gaps and futuristic aspects of the IoQDs and related architectures for which this solution could be actually implemented
B. Lack of enough opportunities and time for implementing this disruptive technology stack
C. Scientific knowledge lacking
D. Energy and Environmental challenge in testing this project
E. Tracking transmission delays and other challenges involved with space communication
F. Cyberattack protection
G. Achieving performance with compact resource-constrained devices
H. Security based on Quantum Computation

IV. BACKGROUND OF PREVIOUS WORK DONE

A. Quantum Convolutional Neural Networks (QCNN) Using Deep Learning for Computer Vision Applications
B. Dynamic fluorescence lifetime sensing with CMOS single-photon avalanche diode arrays and deep learning processors
C. 3D Object Detection and Tracking Methods using Deep Learning for Computer Vision Applications
D. Pneumonia classification using quaternion deep learning
E. Deep Learning based Object Detection Model for Autonomous Driving Research using CARLA Simulator
F. Quantum Neural Network-Based Deep Learning System to Classify Physical Timeline Acceleration Data of Agricultural Workers
G. Artificial Intelligence (AI) Enabled Vehicle Detection and counting Using Deep Learning

V. IMPLEMENTATION

A. Technology Stack Used – Resources

1) Code repository will contain implementation of Quantum Neural Network as well as Classical Convolutional Neural Network for Classification Task on Street View House Numbers dataset.
2) Python 3.9
3) For the Quantum Implementation we would need:
  - TensorFlow-Quantum
  - Cirq

4) For the Classical Implementation we would need:
  - TensorFLow 2.x
  - TensorBoard

5) The Jupyter Notebooks are developed in Google Colab.
6) The Repository contains weights for trained Classical and Quantum Models.
7) Deployment feasibility of the entire QuantumAI setup on GCP, Qiskit Runtime Simulation, etc.

B. Datasource And Loading

- *Dataset Chosen:* The Street View House Numbers (SVHN) dataset is one of the most popular benchmarks for object recognition tasks in academic papers. The images were obtained from house numbers in Google Street View images, are hosted by Stanford University and are very similar in philosophy with the MNIST dataset. However, the original purpose of this dataset is to solve a harder problem: that of recognizing digits and numbers in natural scene images.
- *Dataset Format:* The data of the Street View House Numbers dataset, which can originally be found here are originally in .mat, i.e. files which can be best processed with MATLAB; thus, some preprocessing is. It is important to note that the data are divided into two formats and in this particular kernel we are going to use Format 2.

1) Format 1: The original, variable-resolution colored house-number images with character level bounding boxes.
2) Format 2: The cropped digits (32x32 pixels) which follow the philosophy of the MNIST dataset more closely, but also contain some distracting digits to the sides of the digit of interest.

C. Workflow Operations

1) Classical Deep Learning with TensorFlow

a) Configurations/Libraries

- Keras
- Tensorflow

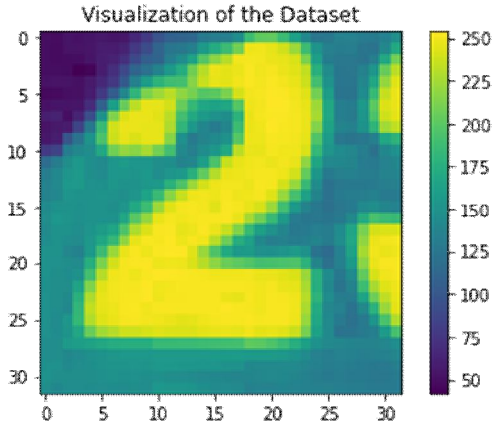
b) Preprocessing Functions

- FormatArray()
- fixLabel()
- rgb2gray()

The above three functions formats the array in proper shape, fix the label range and convert the pixels from RGB to GrayScale format.

2) Data Preprocessing

Let's have a look at the first image from our X_train and the corresponding label from y_train.



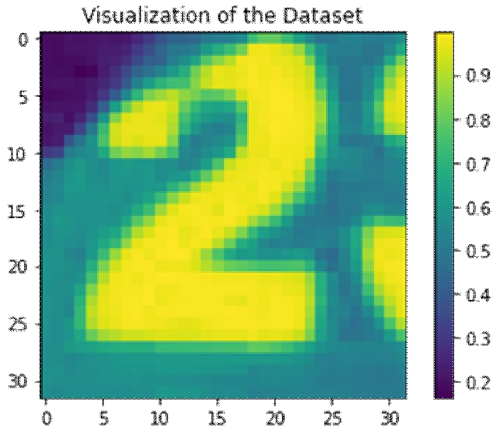
From the colorbar in the above visualization, we see that there are grayscale images in the dataset and hence their values range from 0 to 255. It's better to scale these pixel values in our dataset so that the values range from 0 to 1. This will help us to converge our CNN training faster.

3) Normalizing the Train and test Image Data

X_train = X_train/255.0

X_test = X_test/255.0

Post normalizing let's again have a look at the first image from our X_train.



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue IX Sep 2022- Available at www.ijraset.com

Before proceeding, we need to reshape our images in the dataset to include the grayscale parameter at the end.

```
X_train = X_train.reshape(X_train.shape[0], *(32,32,1))
X_valid = X_valid.reshape(X_valid.shape[0], *(32,32,1))
X_test = X_test.reshape(X_test.shape[0], *(32,32,1))
```

- The shape of the X_train is (58605, 32, 32, 1)
- The shape of the X_valid is (14652, 32, 32, 1)
- The shape of the X_test is (26032, 32, 32, 1)

4) Building the Deep Learning Model

Now that we are aware of the dataset, we can start building our Deep Learning model. We will use TensorFlow and specifically the tensorflow.keras API for building the model. TensorFlow is one of the leading Machine Learning libraries that is being used these days and can be used for constructing Neural networks. Building our network involves the following steps which together create Python code:

- Adding imports: we depend on other packages for building our Neural network. We have to import the specific components that we require first.
- Specifying the configuration options: configuring a Neural network involves specifying some configuration options.
- Creating the model skeleton: we then actually create the Neural network, or more specifically the model skeleton. We will then know what our model looks like, but it's not real yet.
- Compiling the model: when compiling the model, we make it real, by instantiating it and configuring it. It can now be used.
- Fitting data to the model: in other words, training the model.
- Evaluating the model: checking how well it works after it was trained.

5) Libraries to be imported to build CNN

The first thing we have to do is adding the imports.

- First of all, we'll be using the Extra Keras Datasets package for importing SVHN Dataset.
- We then import the Sequential Keras API, which is the foundation for our Keras model. Using this API, we can stack layers on top of each other, which jointly represent the Deep Learning model.
- We will also use a few layers: we'll use Convolutional ones (Conv2D) for 2D data (i.e., images), Densely-connected ones (for generating the actual predictions) and Flatten (Dense layers can't handle non-1D data, so we must flatten the outputs of our final Conv layers).
- For optimization, we use the Adam optimizer (tensorflow.keras.optimizers.Adam) and for loss we use categorical_crossentropy loss.
- Finally, because we use categorical crossentropy loss, we must one-hot encode our targets. Using the to_categorical util, we can achieve this.

6) Configuration Parameters

- batch_size = 250
- no_epochs = 150
- validation_split_size = 0.20
- verbosity = 1
- optimizer = Adam()
- loss_function = categorical_crossentropy
- additional_metrics = ['accuracy']

7) Creating the Model Skeleton

We can then create the model skeleton:

- We first initialize the Sequential API into the CNN_model variable.
- We then stack a few layers on top of each other and indirectly on top of the model foundation. Specifically, we use 3 Convolutional layers, then Flatten the feature maps generated by the last layer, and use Dense layers for the final prediction (using Softmax).

8) Compiling and Model Training

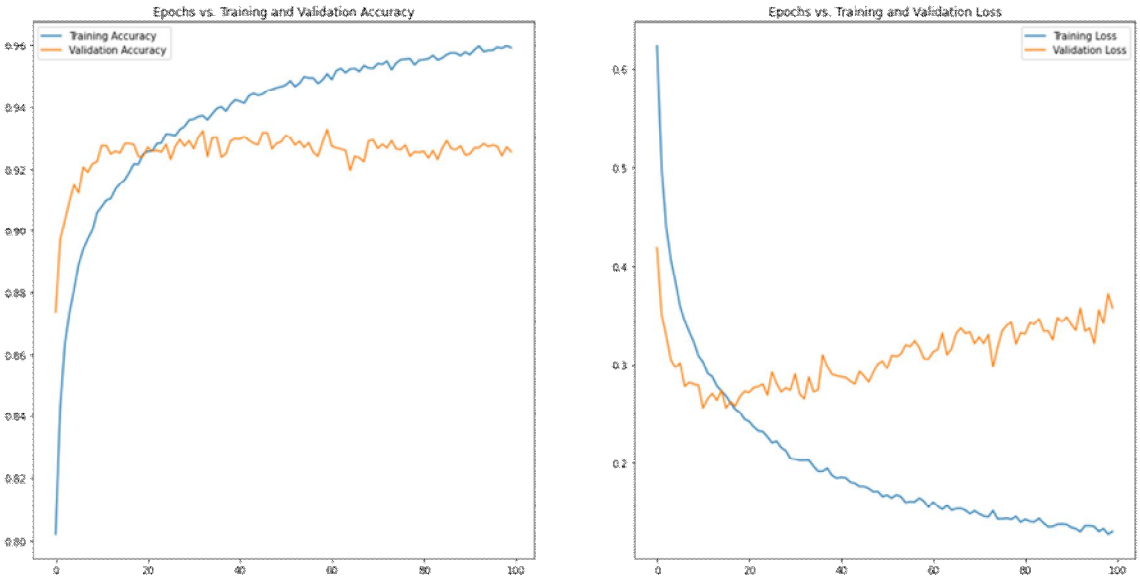
Model training at 100th Epoch:

Epoch 100/100

1832/1832 [==========] - 39s 21ms/step - loss: 0.1311 - accuracy: 0.9592 - val_loss: 0.3571 - val_accuracy: ***0.9256***

9) Model Evaluation

- The loss on the test set is 0.3858301639556885
- The accuracy on the test set is 0.922633707523346



10) Quantum Deep learning - QCNN

- About QML – Quantum Machine Learning: We shall perform QML on America's household dataset using TensorFLow Quantum and Cirq.
- TensorFlow-Quantum is a great place to start learning QML and get into this amazing field. TensorFlow Quantum (TFQ) is a quantum machine learning library for rapid prototyping of hybrid quantum-classical ML models. TensorFlow Quantum focuses on quantum data and building hybrid quantum-classical models. It integrates quantum computing algorithms and logic designed in Cirq, and provides quantum computing primitives compatible with existing TensorFlow APIs, along with high-performance quantum circuit simulators.
- Cirq is a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulators. Cirq provides useful abstractions for dealing with today's noisy intermediate-scale quantum computers, where details of the hardware are vital to achieving state-of-the-art results.

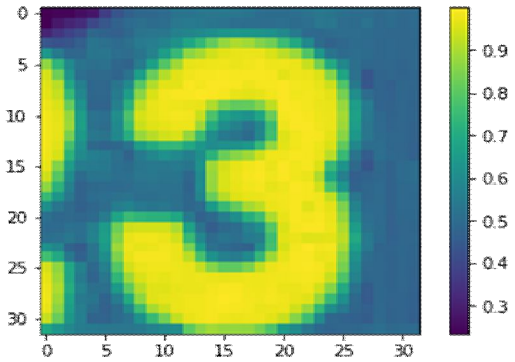
a) Configurations/Libraries

- Cirq 0.7
- Tensorflow 2.3.1
- Tensorflow-quantum 0.2

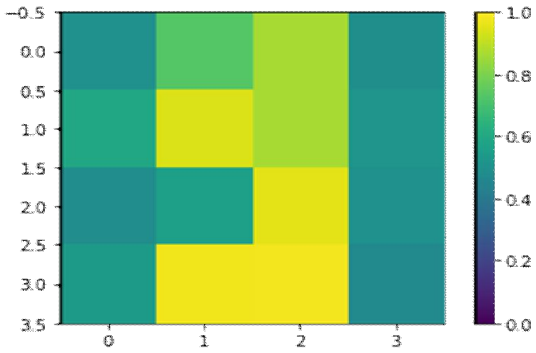
b) Preprocessing Functions (same as CNN)

- FormatArray()
- fixLabel()
- rgb2gray()

c) Image Plot



d) Filter data to 4 x 4 size:

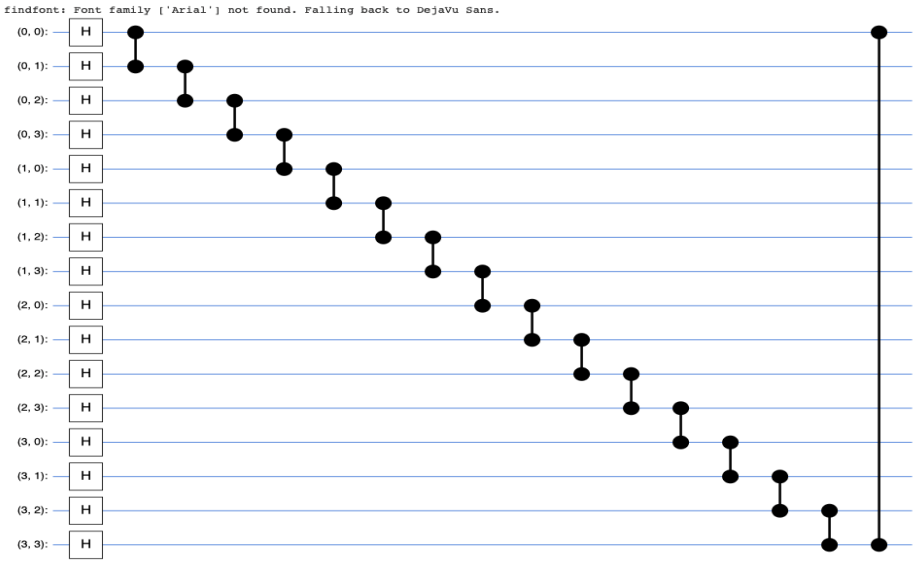


e) Data Preprocessing:

- Remove contradictions
- Determine the set of labels for each unique image
- Throw out images that match more than one label

f) Building Quantum Circuit:

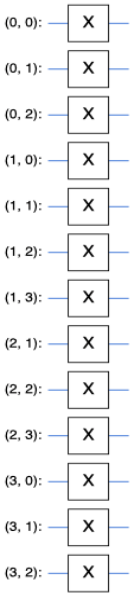
In quantum information and quantum computing, a cluster state is a type of highly entangled state of multiple qubits



g) Encode truncated classical image into quantum datapoint:

- Function convert_to_circuit(image)=> Take image and convert to circuit using gates
- convert data to circuit using the above convert_to_circuit() function
- convert data to tensor format – Apply convert_to_tensor function of tensorflow_quantum library to convert to tensorflow_quantum circuits

h) SVG Circuit Representation

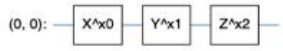


i) Build Model – QCNN Layers:

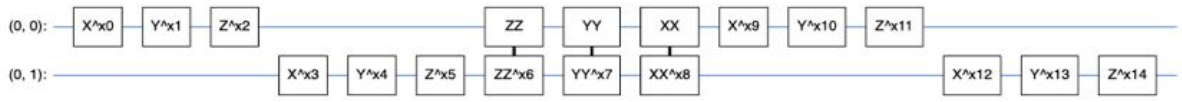
We then define model layer by using the *Cong and Lukin QCNN paper.*

Apply SVGCircuit for the above layer representations:

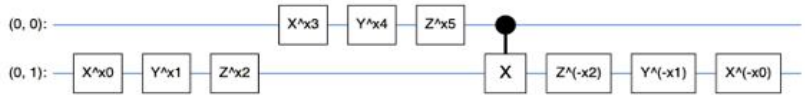
```
SVGCircuit(one_qubit_unitary(cirq.GridQubit(0, 0), sympy.symbols('x0:3')))
```




```
SVGCircuit(two_qubit_unitary(cirq.GridQubit.rect(1, 2), sympy.symbols('x0:15')))
```




```
SVGCircuit(two_qubit_pool(*cirq.GridQubit.rect(1, 2), sympy.symbols('x0:6')))
```



j) Create Model Rebuild QCNN + CNN:

- Define GridQubit in 4X4 form
- Take symbols – sympy.symbols to pass as parameters
- Add layers one by one to fulfill the grid
- Append to circuit -> apply Hadamard Gate -> GridQubit
- Define Cluster State and form the PQC – Parameterized Quantum Circuit
- Ultimately form the qcnn_model initialized with all parameters to use in training

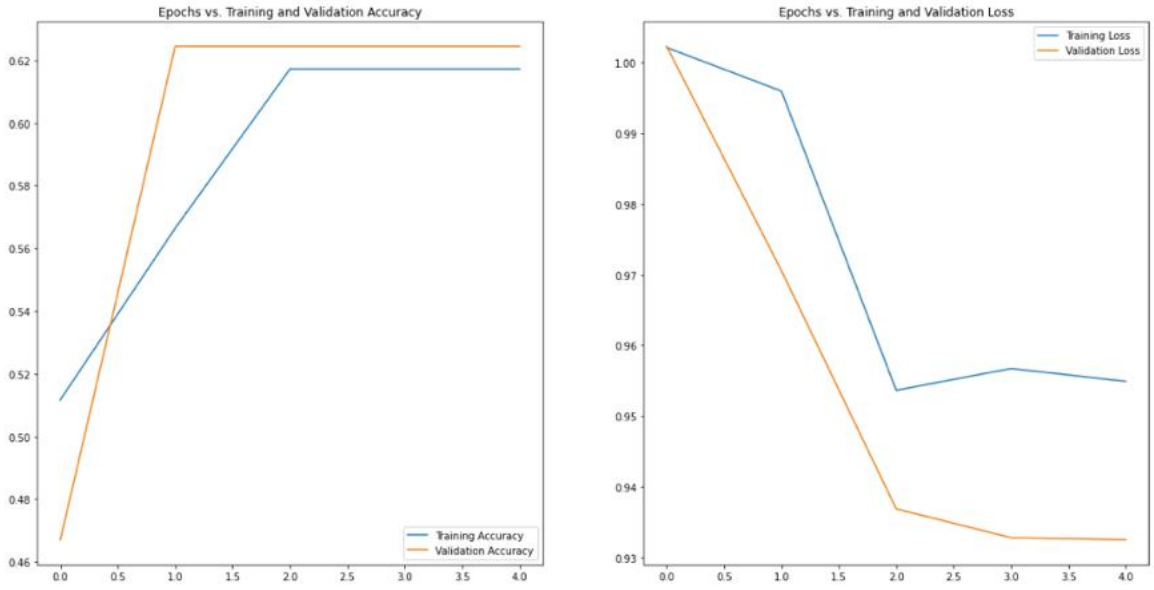
k) Train Model

- Define Hinge Accuracy
- Compile qcnn_model with learning_rate=.02, loss=mse & metrics=hinge_accuracy (defined above)
- Capture history of model training – qcnn_model.fit(..)

l) Training History Report

	accuracy	loss	val_accuracy	val_loss
0	0.511719	1.002065	0.467068	1.002244
1	0.566406	0.995914	0.624421	0.970553
2	0.617188	0.953655	0.624421	0.936903
3	0.617188	0.956728	0.624421	0.932841
4	0.617188	0.954910	0.624421	0.932542

m) Model Evaluation



D. Model Optimization/Improvement Plan

Of course, there is room for quite a bit of tuning in order to improve performance such as:

- Change the way the images are transformed in the augmentation process.
- Change the architecture of our model by modifying the embedded circuit layers etc.
- Train multiple CNNs and make ensemble predictions.
- Use some of the extra data which can be found along with the original dataset.

Referred from Cong and Lukin paper on Quantum Convolutional Neural Network (QCNN)

Implementing #2 and #3 of Performance Optimization/Improvement Plan:

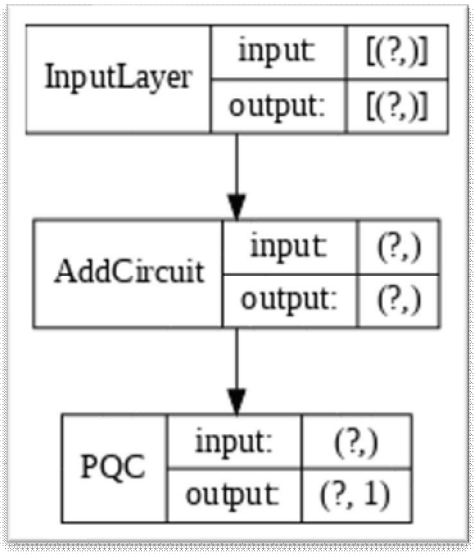
- Changing Model Definition
- Introducing Hybrid Models:
  - Hybrid model with a single quantum filter
  - Hybrid convolution with multiple quantum filters

1) Changing Model Definition

We will now construct a purely quantum CNN. We will start with 8 qubits then we pool down to1.

Below code snippets Referred from Cong and Lukin paper on Quantum Convolutional Neural Network (QCNN)

```
def create_model_circuit(qubits):
    """Create sequence of alternating convolution and pooling operators
    which gradually shrink over time."""
    model_circuit = cirq.Circuit()
    symbols = sympy.symbols('qconv0:63')
    # Cirq uses sympy.Symbols to map learnable variables. TensorFlow Quantum
    # scans incoming circuits and replaces these with TensorFlow variables.
    model_circuit += quantum_conv_circuit(qubits, symbols[0:15])
    model_circuit += quantum_pool_circuit(qubits[:4], qubits[4:],
                                          symbols[15:21])
    model_circuit += quantum_conv_circuit(qubits[4:], symbols[21:36])
    model_circuit += quantum_pool_circuit(qubits[4:6], qubits[6:],
                                          symbols[36:42])
    model_circuit += quantum_conv_circuit(qubits[6:], symbols[42:57])
    model_circuit += quantum_pool_circuit([qubits[6]], [qubits[7]],
                                          symbols[57:63])
    return model_circuit


# Create our qubits and readout operators in Cirq.
cluster_state_bits = cirq.GridQubit.rect(1, 8)
readout_operators = cirq.Z(cluster_state_bits[-1])

# Build a sequential model enacting the previously defined logic of this notebook.
# Here you are making the static cluster state prep as a part of the AddCircuit and the
# "quantum datapoints" are coming in the form of excitation
excitation_input = tf.keras.Input(shape=(), dtype=tf.dtypes.string)
cluster_state = tfq.layers.AddCircuit()(
    excitation_input, prepend=cluster_state_circuit(cluster_state_bits))

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

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



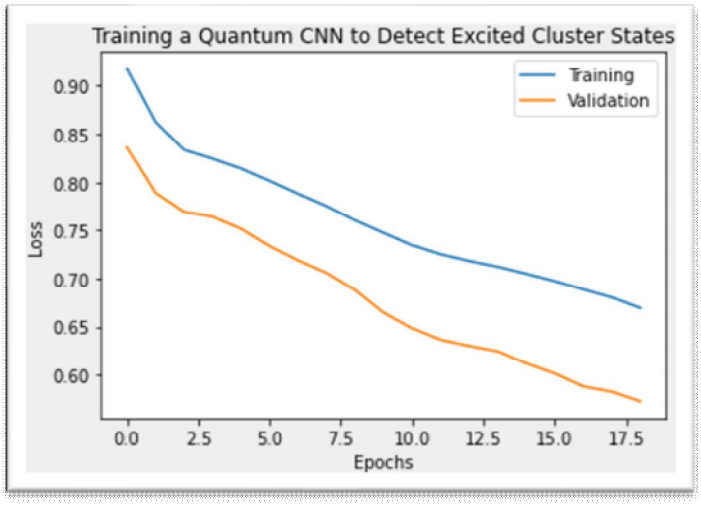
2) Model Training

```
# Generate some training data.
train_excitations, train_labels, test_excitations, test_labels = generate_data(
    cluster_state_bits)


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


qcnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.002),
                   loss=tf.losses.mse,
                   metrics=[custom_accuracy])

history = qcnn_model.fit(x=train_excitations,
                         y=train_labels,
                         batch_size=16,
                         epochs=20,
                         verbose=1,
                         validation_data=(test_excitations, test_labels))
```

```
Train on 112 samples, validate on 48 samples
```

3) Model Evaluation

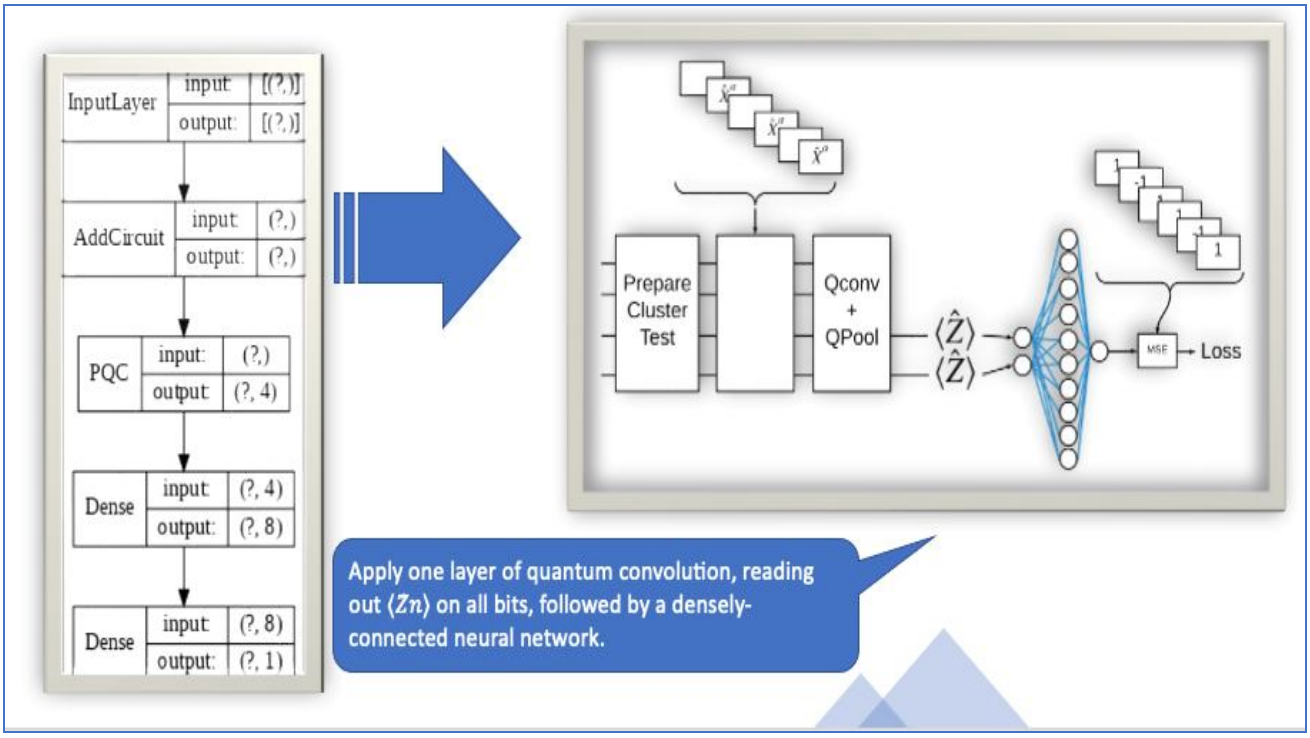
```
Epoch 16/20
112/112 [==============================] - 12s 103ms/sample - loss: 0.7049 - custom_accuracy: 0.8125 - val_loss: 0.6120 - val_custom_accuracy: 0.8750
Epoch 17/20
112/112 [==============================] - 12s 107ms/sample - loss: 0.6973 - custom_accuracy: 0.8125 - val_loss: 0.6015 - val_custom_accuracy: 0.8750
Epoch 18/20
112/112 [==============================] - 12s 106ms/sample - loss: 0.6887 - custom_accuracy: 0.8036 - val_loss: 0.5881 - val_custom_accuracy: 0.8750
Epoch 19/20
112/112 [==============================] - 12s 108ms/sample - loss: 0.6804 - custom_accuracy: 0.8125 - val_loss: 0.5826 - val_custom_accuracy: 0.8750
Epoch 20/20
112/112 [==============================] - 12s 107ms/sample - loss: 0.6695 - custom_accuracy: 0.8304 - val_loss: 0.5727 - val_custom_accuracy: 0.8750
```



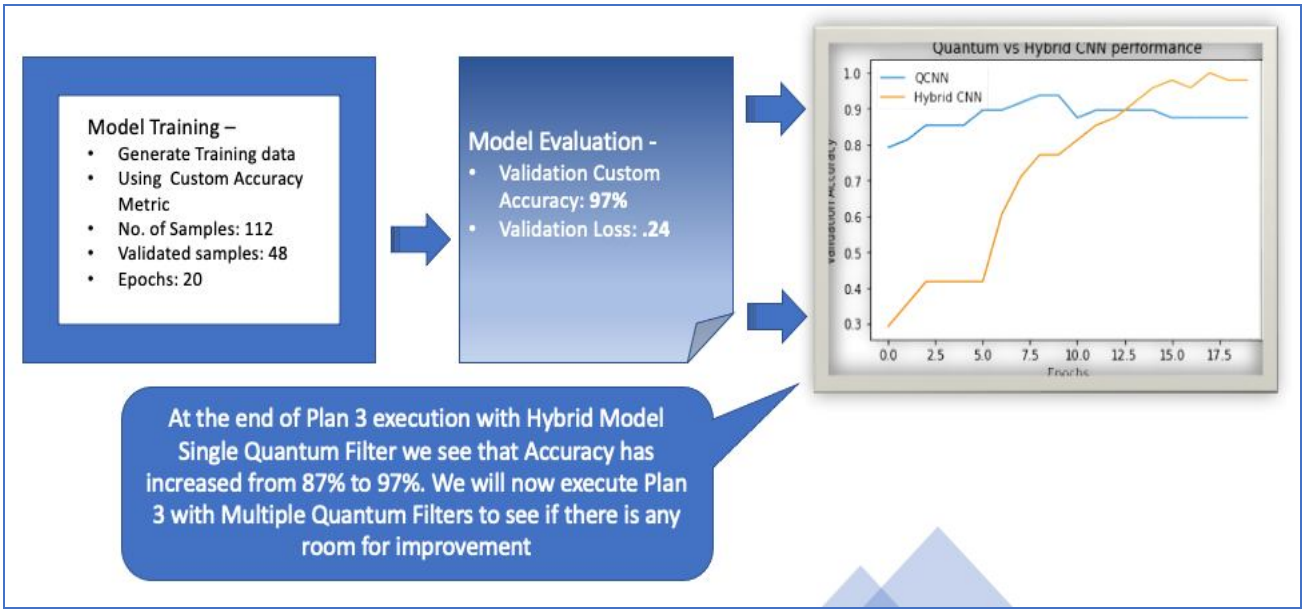
E. Introducing Hybrid Models

We perform 2 rounds of quantum convolution and feed the results into a classical neural network. This part explores classical-quantum hybrid models. (HQ-C)

1) Hybrid Model with a Single Quantum Filter



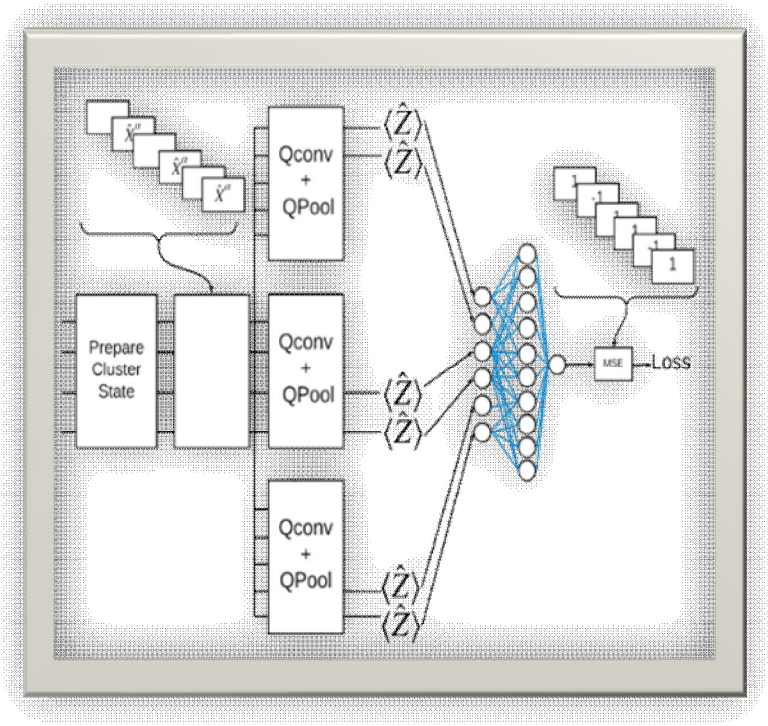
2) *Model Training & Model Evaluation:*



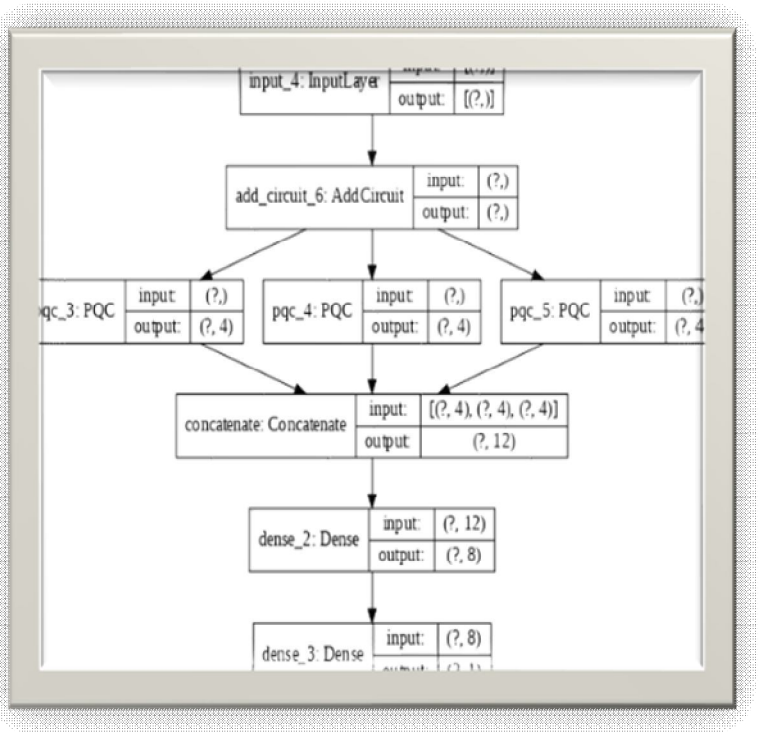
3) Hybrid model with multiple quantum filters:

We now try an architecture that consists of multiple quantum convolutions followed by a classical neural network for consolidation.

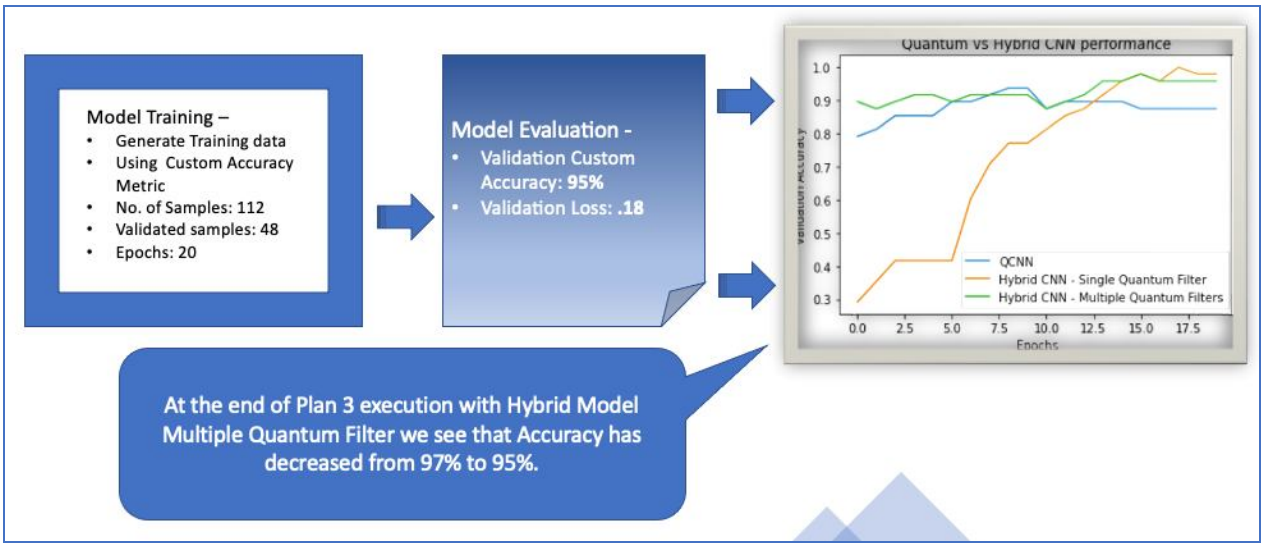
a) Multiple Quantum Convolution architecture with a classical Neural Network to combine them:



b) Model Architecture with Parameterized Quantum Circuit and Dense Layers:



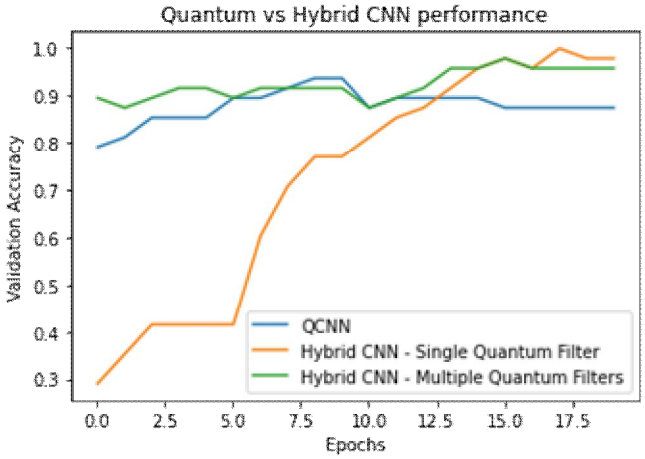
4) Model Training & Model Evaluation:



Result: Hybrid CNN with Single Quantum Filter gives the best result with an accuracy of ~97%

VI. CONCLUSION

Which is better between Classical and Quantum Deep Learning for high spectral image recognition use case?



The QCNN is much superior than it's classical Deep Learning competitor.

When we used earlier model we get an accuracy percentage of ~56-60%. Comparatively, when we modified model parameters described above we found the accuracy around 87%. As an optimization, plan we introduced Hybrid Models which gave us a maximum Accuracy achieved through out which is **97%**

A. Directions For Future Work

Some directions for future work or scope for improvements described here:

1) Transfer learning – model should be tuned, and algorithm should be improved using hyperparameter tuning to train on more diverse data

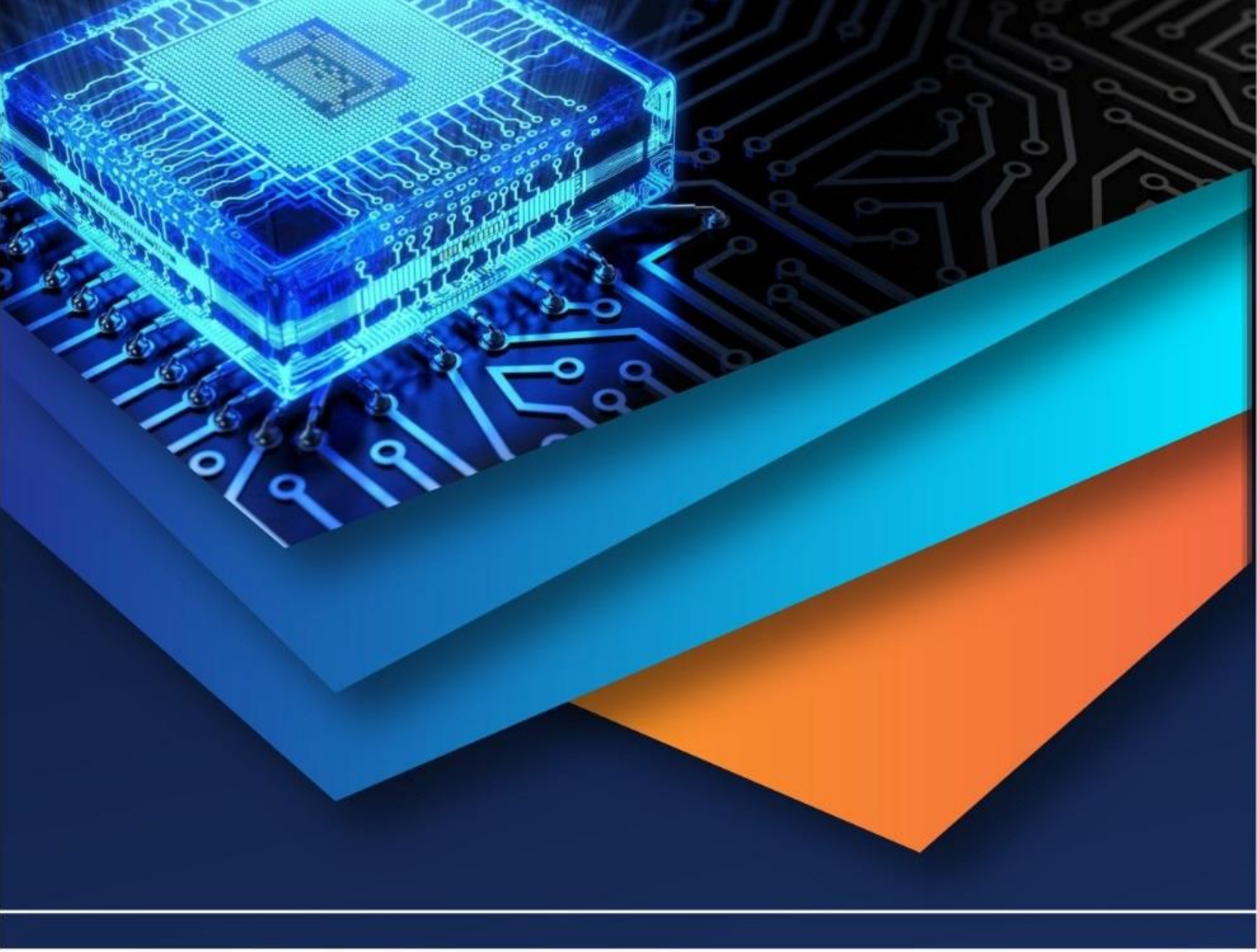
2) Train the model to classify data with highest possible accuracy

3) Batches used for training should be shuffled pretty well to reduce overfitting

4) Training should be continued even after achieving the best possible accuracy to increase likelihood and effectively use classification threshold values

REFERENCES

[1] "Adiabatic quantum computation." [Online]. Available: https://en.wikipedia.org/wiki/Adiabatic quantum computation
[2] "Calculus on Computational Graphs: Backpropagation { colah's blog." [Online]. Available: http://colah.github.io/posts/2015-08-Backprop/
[3] "Classification datasets results." [Online]. Available: http://rodrigob.github.io/are we there yet/build/classification datasets results.html
[4] "Continuous-variable quantum information." [Online]. Available: https://en.wikipedia.org/wiki/Continuous-variable quantum information
[5] "Convolution as matrix multiplication." [Online]. Available: https://en.wikipedia.org/wiki/Toeplitz matrix#Discrete convolution
[6] "Convolution theorem." [Online]. Available: https://en.wikipedia.org/wiki/Convolutiontheorem
[7] "Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation." [Online]. Available: http://deeplearning.net/tutorial/lenet.html
[8] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/classi cation/
[9] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/convolutional-networks/
[10] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/optimization-1/
[11] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/optimization-1/#numerical
[12] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/optimization-2/
[13] "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/optimization-2/#mat
[14] "Deep learning book. Optimization chapter." [Online]. Available: http://www.deeplearningbook.org/contents/optimization.html
[15] "Quantum logic gate." [Online]. Available: https://en.wikipedia.org/wiki/Quantum logic gate
[16] "Qubit." [Online]. Available: https://en.wikipedia.org/wiki/Qubit
[17] "TensorFlow clip by global norm." [Online]. Available: https://www.tensorflow.org/versions/r1.1/api docs/python/tf/clip by global norm
[18] S. Bai, J. Z. Kolter, and V. Koltun, \An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," mar 2018. [Online]. Available: http://arxiv.org/abs/1803.01271
[19] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. K egl, \Algorithms for Hyper-Parameter Optimization," in Advances in Neural Information Processing Systems, 2011, pp. 2546{2554. [Online]. Available: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization
[20] T. Bluche, H. Ney, and C. Kermorvant, \A Comparison of Sequence-Trained Deep Neural Networks and Recurrent Neural Networks Optical Modeling for Handwriting Recognition," pp. 199{210, oct 2014. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-11397-5 15
[21] https://www.academia.edu/43006431/Effective_Handwritten_Digit_Recognition_using_Deep_Convolution_Neural_Network
[22] https://www.researchgate.net/publication/332880911_Improved_Handwritten_Digit_Recognition_using_Quantum_K-Nearest_Neighbor_Algorithm
[23] https://www.researchgate.net/publication/341179765_Hand_Written_Digit_Recognition_Using_Quantum_Support_Vector_Machine#pf5
[24] https://www.tensorflow.org/quantum/tutorials/qcnn
[25] https://jovian.ai/imammuhajir992/quantum-convolutional-neural-network
[26] https://github.com/yh08037/quantum-neural-network
[27] https://www.nature.com/articles/s41467-020-14454-2
[28] https://devblogs.microsoft.com/qsharp/hybrid-quantum-classical-models/#:~:text=Naively%20speaking%2C%20%E2%80%9Chybrid%E2%80%9D%20means,Quantum%20Processing%20Unit%20(QPU).
[29] https://cupdf.com/document/masters-thesis-deep-learning-for-text-data-mining-.html
[30] https://www.nature.com/articles/s41567-019-0648-8
[31] https://arxiv.org/pdf/quant-ph/0504097.pdf



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY