Offline Handwritten Character Recognition using Quantum Convolutional Neural Networks

EC61409
Neural Networks and Applications

19 November 2021

Joel Antony Thomas
17EC10023

ABSTRACT

Despite the availability of a lot of new technological tools to write, most people still prefer taking their notes traditionally, with pen and paper. Adoption of a Handwriting Text Recognition software which converts handwritten text into a digital format helps in ease of management, easy access, analyzing and sharing notes. So, the objective of this project is to build a Offline Handwritten Character Recognition using Quantum Convolutional Neural Networks and compare it's performance with a Classic CNN model trained from the EMNIST dataset which describes characters contained in scanned text images into digital form. The QML model gives 78%, 83% and 85% accuracy with 2000, 5000 and 10000 training data whereas Classic CNN model gives 69%, 78% and 83% accuracy respectively.

Chapter 1

Introduction

Despite the availability of a lot of new technological tools to write, most people still prefer taking their notes traditionally, with pen and paper, be it in meeting, or classes. However, there are a lot of drawbacks to handwritten notes as the storage, access and processing of physical copies of documents in an efficient manner is very difficult. Important records in such documents have to be manually updated to computers, and extra labor work is required to properly organise this data. It is also cumbersome to search topic-wise through physical documents for keywords, or even organising them and sharing them efficiently with others. A lot of important information gets lost in these physical notes and never gets reviewed only because it was never converted to a digital format.

Adoption of a Handwritten Character Recognition software is a practical solution to all these problems, additionally providing more security since the data will now be stored in digital mode, thus helping people

who still prefer traditional note-making techniques despite the availability of modern technology, by converting handwritten text into a digital format which further helps in ease of management, easy access, analyzing and sharing notes.

Optical character recognition (OCR) is the electronic or mechanical translation of images of typed, handwritten, or printed text into machine-encoded text, whether from a scanned document, a picture of a document, a scene photo (for example, text on signs and billboards in a landscape photo), or subtitle text superimposed on an image. The task of changing a language represented in its spatial form of graphical marks into its symbolic representation is known as handwriting recognition. OCR systems have various applications where is becomes necessary to process huge volumes of handwritten text such as recognition of addresses and postal codes from letters, interpretation of amounts on bank checks, verification of signatures, license plate interpretation etc. OCR systems can be classified mainly into two types - Online and Offline Handwritten Text Recognition systems.

In Online Handwriting Text Recognition systems, a special electronic pen is used to sense and interpret the handwritten input characters from a computer surface. Here, the recognition is performed in real-time. The features which are extracted are based on a dynamic information which is used as input. In Offline Handwriting Character Recognition systems, the information which is being fed as input does not undergo and dynamic changes, whereas the most challenging aspect for these systems are the variability in the style of writing, as different people have their own style of

writing. This project is focused on Offline Handwriting character recognition, mainly taking written English characters as input, in cursive or block writing and interpreting into a digital letter. This project can be combined along with algorithms which segment the characters from words, or segment the words from a line of text image, which can be further combined with algorithms that segment a whole handwritten page into lines of text, thus taking the form of a fully functional deliverable which can be used by the end user.

The rest of the report is organized as follows : Chapter 2 presents the various techniques and algorithms explored along with the Literature Survey, Chapter 3 presents the Proposed Architecture of the OCR system, Chapter 4 presents the Experimental Results and Chapter 5 presents Conclusion and Future Scope.

Chapter 2

Literature Survey

2.1 EMNIST Dataset

This project uses the Extended-MNIST Dataset [2]. The EMNIST dataset which is derived from the NIST Special Database 19 has a group of handwritten English characters and digits which is transfomed to a 28x28 pixel image format, which is in the same format as the famous MNIST dataset.

2.1.1 ByClass and ByMerge Dataset

The ByClass and ByMerge splits has the the complete available data of the NIST Special Database 19, even though the classes are unbalanced. The only difference between the datasets is the number of images in each of the classes, all other image configurations remain the same. Both datasets have an unequal number of images per class, with digits outnumbering letters. The number of letters in the datasets roughly corresponds to

the actual frequency in which they are used in the English language.

- train: 697,932
- test: 116,323
- total: 814,255
- classes: ByClass 62 (unbalanced) / ByMerge 47 (unbalanced)

2.1.2 Balanced Dataset

The balance issues in the ByClass and ByMerge datasets is addressed by the EMNIST Balanced dataset. It is derived from the ByMerge dataset and has an equal number of samples per class to minimise misclassification errors caused by capital and lower case letters. This dataset is intended to be the most applicable. In this project, we use a subset of the Balanced dataset with 47 classes.

- train: 112,800
- test: 18,800
- total: 131,600
- classes: 47 (balanced)

2.1.3 Letters Dataset

The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task.

- train: 88,800
- test: 14,800
- total: 103,600
- classes: 37 (balanced)

The figure given below shows some examples from the dataset.

Figure 2.1: Samples of images in the EMNSIT Dataset

The figure give below shows the structure, breakdown and the various divisions of the datasets visually. Handwritten digits, handwritten letters, or a combination of the two can be found in each dataset. For each dataset, the number of samples in each class is shown, highlighting the wide variance in the number of samples in unbalanced datasets. A vertical arrow is used to

denote the class into which the lowercase letter is merged in datasets that contain merged classes.

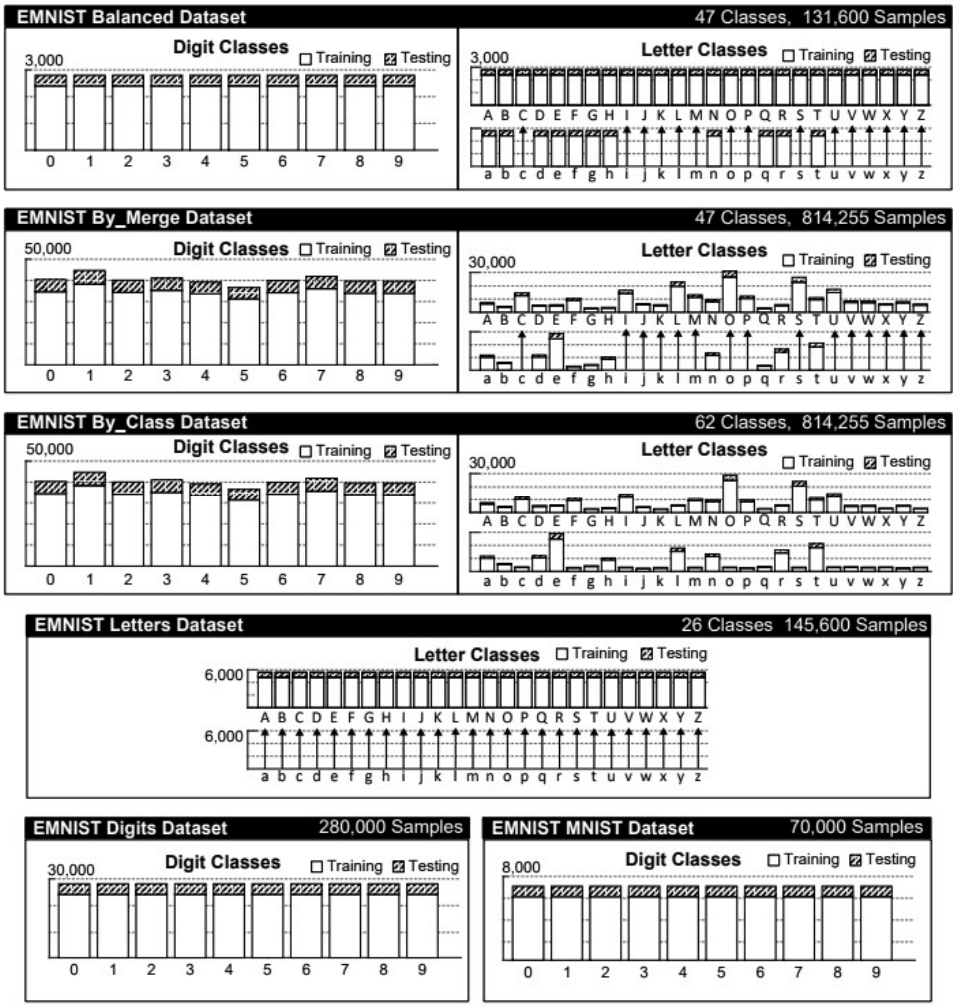

Figure 2.2: Visual Representation of the EMNIST Datasets

2.2 Principles of Quantum Computing

In quantum computing, the qubit is the fundamental unit of data. Unlike a traditional bit, which can only have a value of 0 or 1, superposition allows a qubit to exist in both states. In general, a qubit is represented as:

$$|\psi\rangle \;=\; \alpha\,|0\rangle \;+\; \beta\,|1\rangle \tag{2.1}$$

$|0\rangle$ and $|1\rangle$ correspond to each of the two computational basis states, α and β are complex amplitudes of the qbit satisfying the equation $|\alpha|^2 + |\beta|^2 = 1$. When you observe a qubit, it collapses into one of the basis states. Each state's probability of being observed is proportional to the square of its coefficient's amplitude ie, the probabilities of observing $|0\rangle$ and $|1\rangle$ are $|\alpha|^2$ and $|\beta|^2$ respectively. A qubit can be realised physically as a basic quantum device, with the two basis states corresponding to the horizontal and vertical polarisation of a photon, for example. Due to parallel computations on probabilistic combinations of states, quantum computing systems can theoretically achieve exponential speedups over their classical counterparts because of superposition.

The process of entanglement occurs when qubits show correlation with one another. A superposition of 2^n basis states exists in general for a set of n entangled qubits. When one or more qubits are observed, their states collapse, and the original superposition is changed to account for the observed values of the qubits.

Two fundamental logic gates (AND and OR) perform irreversible computations in classical computing. Quantum gates (which work with

qubits) must be reversible, and they must operate on the input state to produce an output of the same dimension. Usually, unitary matrices, which are square matrices whose inverse is their complex conjugate, are used to represent quantum gates.

A superposition of 2^n basis states occurs in an n-qubit scheme. A 2^n-dimensional vector containing the coefficients corresponding to each basis state can be used to define its state. As a result, a $2^n x 2^n$ unitary matrix acting on the state vector is represented by an n-qubit quantum gate H. The Hadamard and CNOT gates are two common quantum gates. The Hadamard gate maps the basis states $|0\rangle$ and $|1\rangle$ as $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ respectively. The CNOT gate acts on 2-qubits and maps $|a, b\rangle$ to$|a\, a, \oplus b\rangle$. In other words, if the first bit is 1, the first bit is copied and the second bit is flipped.

The Pauli matrices ($\{\sigma_x, \sigma_y, \sigma_z\}$) are a group of three 2 x 2 complex matrices that together with the 2 x 2 identity matrix form the real vector space of 2 x 2 Hermitian matrices.

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.2}$$

The density operator ρ represents a mixed state in a d dimensional function space and is defined as:

$$\rho \;=\; \sum_{i=0}^{2^d} p_i \, |\psi_i\rangle \, \langle\psi_i| \tag{2.3}$$

where $\{\psi_i\}$ denotes the $\mathcal{H}^{2^n}$ Hilbert space's computational bases, p_i denotes nonnegative probabilities that add up to 1, and $|\psi\rangle\langle\psi|$ denotes an outer

product.

The density operator can be used to calculate the estimated value of a measurement X using the following formula:

$$\langle X \rangle \;=\; \sum_i p_i \, tr(|\psi_i\rangle \, \langle \psi_i | X) \;=\; tr(\rho X) \tag{2.4}$$

where tr denotes the trace of the matrix.

2.3 Variational Circuits

Quantum algorithms that depend on free parameters are known as variational or parametrized quantum circuits [9]. They have the same three components as normal quantum circuits:

1. Creating a predetermined initial condition (e.g., the vacuum state or the zero state).

2. A quantum circuit $U(\theta)$, parameterized by a set of free parameters θ.

3. At the output, an observable $\hat{B}$ is measured. This observable could be made up of local observables for each wire in the circuit, or just a subset.

Typically, the expectation values $f(\theta) = \langle 0 \mid U^\dagger(\theta) \, \hat{B} \, U(\theta) \mid 0 \rangle$ one or more such circuits describe a scalar cost for a given task — probably with some classical post-processing. The circuits free parameters $\theta = (\theta 1, \theta 2, ...)$ are tuned to optimize this cost function.

A classical optimization algorithm that allows queries to the quantum computer trains variational circuits. With each stage, the optimization is generally an iterative process that seeks out better candidates for the parameters θ.

As a way of thinking about quantum algorithms for near-term quantum devices, variational circuits have become common. Due to the lack of fault tolerance, such devices can only run short gate sequences, as each gate increases the error in the output. A quantum algorithm is typically broken down into a series of regular elementary operations, which are then implemented by quantum hardware.

The fascinating concept of a variational circuit for near-term devices is to combine this two-step technique into a single step by "learning" the circuit on a noisy computer for a specific purpose. This way, the device's "natural" tunable gates can be used to formulate the algorithm, rather than having to go through a fixed elementary gate sequence. Furthermore, systemic errors can be corrected automatically during optmization.

2.4 Hybrid Computation

The term hybrid refers to the technique of combining classical and quantum computations in quantum computing. This is at the core of optimising variational circuits, which involves using a classical co-processor to optimise a quantum algorithm. Quantum devices are typically used to approximate averages of measurement results (i.e., quantum measurable expectations), which are then combined in a single classical cost function that

defines how "strong" the quantum circuits are.

We can estimate the gradients of variational quantum circuits using hybrid computations, which are compatible with techniques like the common backpropagation algorithm (also known as reverse-mode automatic differentiation), the workhorse algorithm for training deep learning models. This means that using hybrid quantum-classical computations, we can differentiate end-to-end. As a result, quantum algorithms [4] can be trained in the same way that traditional deep learning models can.

2.5 Quantum Differentiable Programming

In quantum computing, the derivatives of variational circuits with respect to their input parameters can be computed automatically. Quantum differentiable programming is a programming paradigm that takes advantage of this to make quantum algorithms differentiable and trainable [10].

Classical differentiable programming is a programming style that computes the derivatives of functions with respect to programme inputs using automatic differentiation. Differentiable programming is similar to deep learning, although there are some conceptual distinctions between the two. The structure of models like neural networks, such as the number of nodes and hidden layers, is described statically in many early deep learning frameworks. This is called as the "define-and-run" strategy. Although the network can be differentiated and educated using automated differentiation and backpropagation, the program's fundamental structure remains unchanged.

More recent methods, on the other hand, use a "define-by-run"

strategy, in which no fundamental assumptions about the model's structure are made. The models are made up of parameterized function blocks with a dynamic structure that changes depending on the input data but remains trainable and differentiable. This is especially important because it holds true even when traditional control flow is present, such as for loops and if statements. These concepts enable us to use diffentiable programming in hybrid classical-quantum computations, allowing us to make the entire programme differentiable end-to-end

2.6 Convolutional Neural Networks

A Convolutional Neural Network (CNN/ConvNet) is a Deep Learning algorithm which takes an image as an input and assigns learnable parameters like weights and biases to the neurons in the neural network to detect various features of the image which makes these images differentiable from each other. Each specific neuron receives a lot of inputs, over which it takes a weighted sum, and returns an output based on an activation function. The lower layers capture simple aspects of the image like edges and bright spots, and the higher layers capture the more sophisticated parts of an image, such as shapes and patterns. Thus, a CNN captures the Spatial and Temporal dependencies in a picture through the appliance of relevant filters. The hidden layers of a CNN architecture typically consist of convolutional layers, pooling layers and fully connected layers.

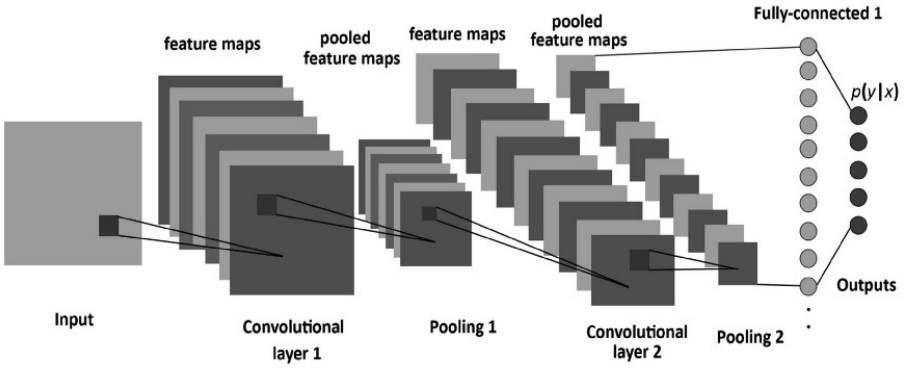

Figure 2.3: CNN Layers

2.6.1 Convolution Layer - The Kernel

Convolutions have filter operations which are used to learn the needed features from the image, without mentioning them explicitly resulting into a feature map.

2.6.2 Pooling Layers

Pooling layers are used to down sample a representation of an image. This helps for dimensionality reduction in data thus helping in decreasing the computational power required and capturing the dominant features.

2.6.3 Fully Connected Layers

FCC layer is used to learn non-linear combinations from the high-level features gotten as output after the convolutions.

2.7 Quanvolutional Neural Networks

A quantum neural network (QNN) [1] is a machine learning model or algorithm that incorporates quantum computing and artificial neural networks concepts.

The term has been used to define a wide range of concepts in recent decades, from quantum computers that mimic the exact computations of neural nets to general trainable quantum circuits that bear no resemblance to the multi-layer perceptron structure.

Variational or parameterized quantum circuits are increasingly referred to as "quantum neural networks." While the comparison isn't quite the same as the inner workings of neural networks, it does illustrate the "modular" existence of quantum gates in a circuit, as well as the widespread use of tricks from training neural networks in quantum algorithm optimization.

By using some powerful aspects of quantum computation, QNNs expand the capabilities of CNNs. The quantum convolutional (or quanvolutional) layer is a new type of transformational layer added to the traditional CNN architecture by QNNs. Quanvolutional layers [7] are made up of a group of N quantum filters that produce feature maps by locally transforming input data, much like their classical convolutional layer counterparts. The main difference is that quanvolutional filters extract features from input data by using quantum circuits to convert spatially local subsections of data [5].

Purely quantum data or the transformation of classical data into quantum states can be fed into QNNs. When it comes to quantum data, $|\psi_{1,....d}\rangle$ can be a superposition of the 2^d computational basis in the d-dimensional Hilbert space $\mathcal{H}^{2^d} = \mathcal{H}^2 \bigotimes \ ..... \ \bigotimes \ \mathcal{H}^2$, where $\mathcal{H}^2$ denotes the 2-dimensional

Hilbert space with basis $\{|0\rangle , |1\rangle\}$.

The features created by quanvolutional layers can improve the classification accuracy of machine learning models. Near-term quantum computers, or noisy (not error corrected), intermediate-scale (50 - 100 qubits) quantum (NISQ) computers [3], will thus be a powerful application from QNNs for three main reasons :

1. Since quantum bits (qubits) with shallow gate depths are only applied to local subsections of input data, quanvolutional filters can work with a small number of qubits.

2. Quanvolutions are error-resistant; as long as the quantum circuit's error model is consistent, it can be thought of as another variable in the random quantum circuit.

3. Since determining the performance of random quantum circuits cannot be done classically at scale, quantum devices will be needed for efficient computation.

In the current era of Noisy Intermediate-Scale Quantum (NISQ) devices [8], QNN particularly appealing since it opens the possibility to classically pre-process large input samples (e.g., high resolution images) with any state-of-the-art deep neural network and to successively manipulate few but highly informative features with a variational quantum circuit. This scheme is quite convenient since it makes use of the power of quantum computers, combined with the successful and well-tested methods of classical machine learning.

QNNs can efficiently access kernel functions in high dimensional Hilbert spaces, which if useful for machine learning purposes could provide a pathway to quantum advantage [11].

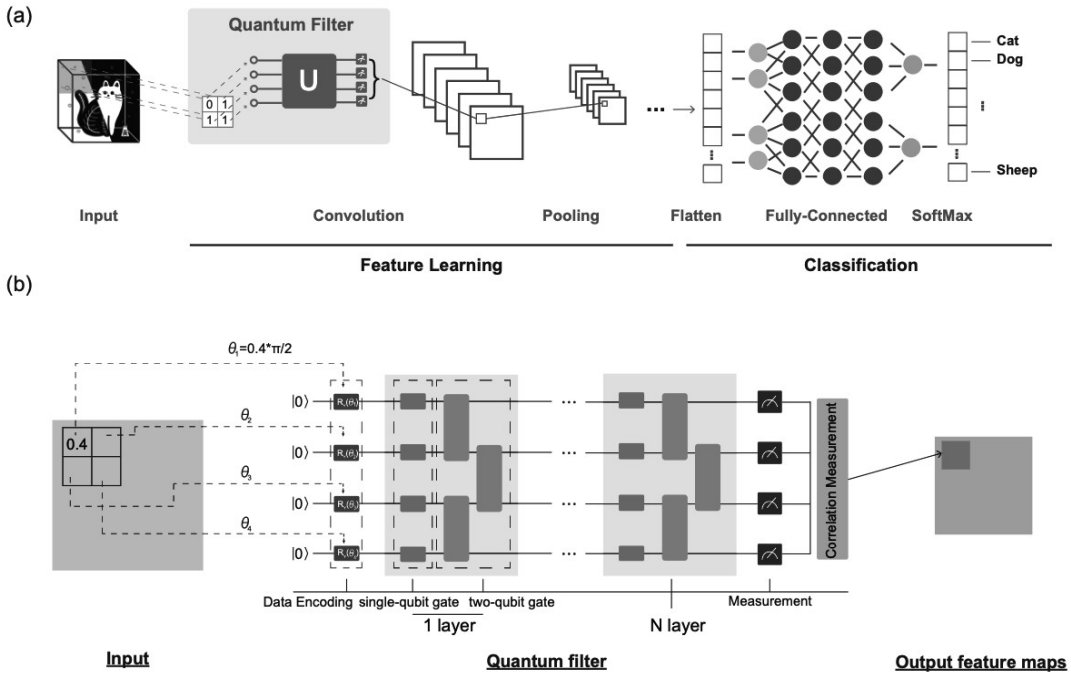

Figure 2.4: QCNN Model Example

Chapter 3

Proposed Architecture

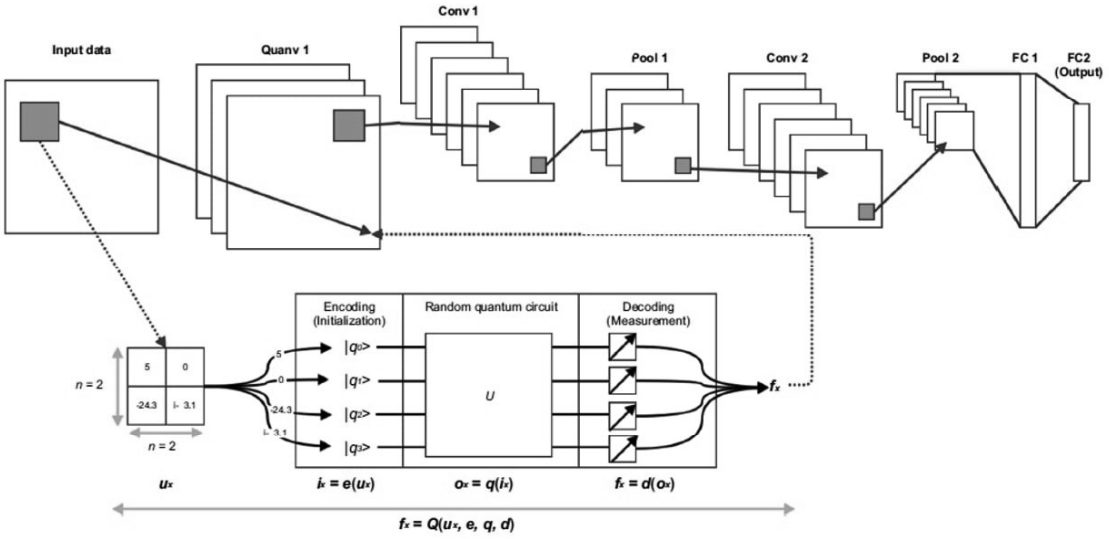

Figure 3.1: Model Architecture

3.1 Quanvolutional Filter

When applied to an input tensor, quanvolutional filters each generate a feature map by transforming spatially local subsections of the input tensor using the quanvolutional filter. Unlike a classical convolutional filter, which uses a simple element-wise matrix multiplication operation, a quanvolutional filter transforms input data using a quantum circuit, which can be structured or random. We use randomly generated quantum circuits for the quanvolutional filters in this work instead of circuits with a built structure for simplicity and to create a baseline.

Quanvolutional filters use the 2D matrix of scalars output by a universal quantum computing (UQC) circuit to transform an input tensor into an output scalar. This method for transforming classical data using quanvolutional filters can be formalised as follows:

1. Consider the case of a single quanvolutional filter. This quanvolutional filter employs a random quantum circuit q that accepts spatially local subsections of images from dataset u as input. For our objectives, we'll call each of these inputs u_x, and each u_x will be an n-by-n 2D matrix with $n > 1$.

2. Although there are several ways to encode u_x as an initialised state of q, we choose one encoding function e for each quanvolutional filter, and we define the encoded initialization state i_x as $i_x = e(u_x)$.

3. The consequence of the quantum computation will be an output quantum state o_x, with the relationship $o_x = q(i_x) = q(e(u_x)$ after the quantum circuit is applied to the initialised state i_x.

4. Although there are several ways to decode the information about o_x obtained through a finite number of measurements, we describe the final decoded state as $f_x = d(o_x) = d(q(e(u_x))$ where d is our decoding function and f_x is a scalar value to ensure that the quanvolutional filter output is compatible with similar output from a standard classical convolution.

5. From now on, we'll refer to the complete transformation of $d(q(e(u_x))$ as the "quanvolutional filter transformation" Q of u_x, aka $f_x = Q(u_x, e, q, d)$.

6. The number of calculations needed when applying a classical convolution filter to input from dataset u is simply $\mathscr{O}(n^2)$, putting the computational complexity squarely in the P range.

As a result, even though the quantum circuit may involve a quantum computation, $q()$ is simply a black-box similar to a classical deep network when viewed from a global perspective.

3.2 Quanvolutional Layer

The following steps are followed to process the image using a Quanvolutional Layer :

1. Read the image and convert into a grey-scale image
2. Normalize the image pixel values by dividing it with 255.
3. The image is divided into squares of 2×2 pixels and embedded into a quantum circuit.
4. A quantum computation, associated to a unitary U, is performed on the system. The unitary could be generated by a variational quantum circuit or, more simply, by a random circuit which is used in this project.
5. The 4 expectation values are mapped into 4 different channels of a single output pixel.
6. By repeating the process over various areas, the entire input image can be scanned, resulting in an output object that is structured as a multi-channel image.

The resolution of the input image is halved as a result of this procedure. This corresponds to a convolution with a 2x2 kernel and a stride of 2. The key difference between a quantum circuit and a classical convolution is that a quantum circuit may produce highly complex kernels whose computation may be classically intractable.

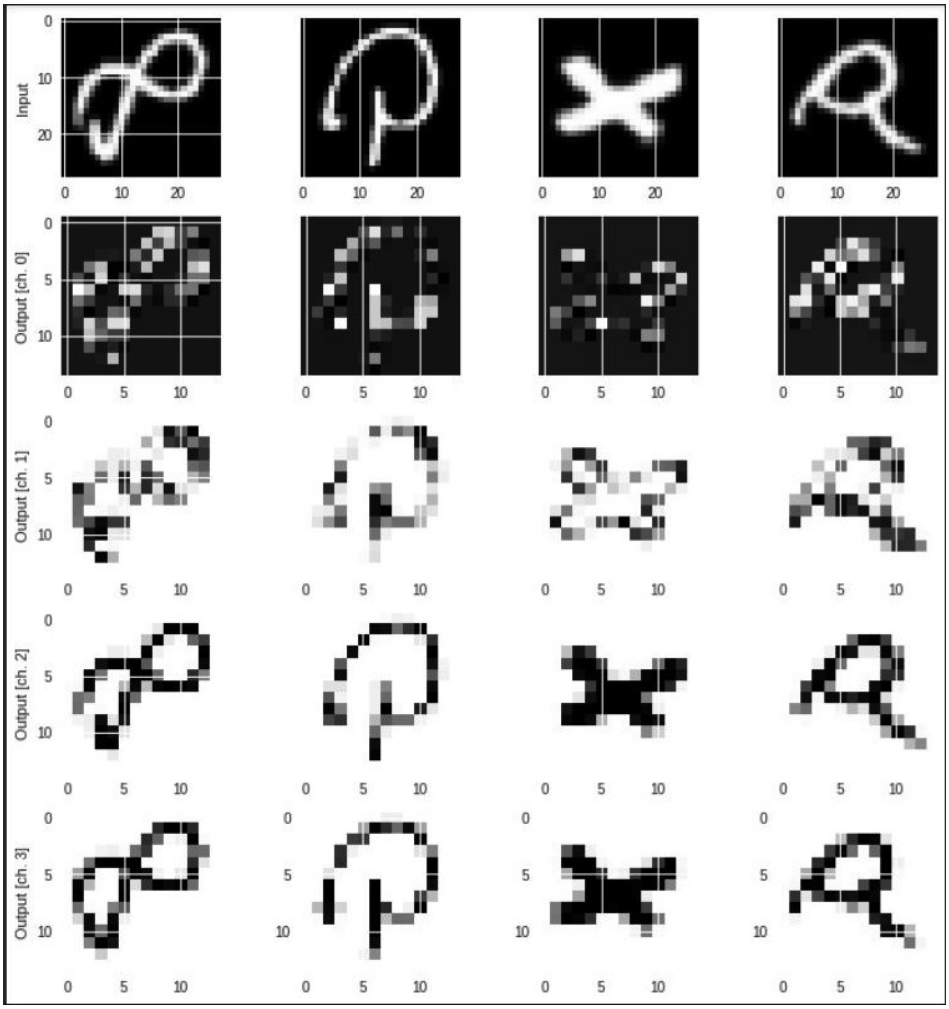

Figure 3.2: Sample Image after Quanvolution

The pre-processed images of the dataset done by Quanvolutional layer then are fed into the classical CNN layers.

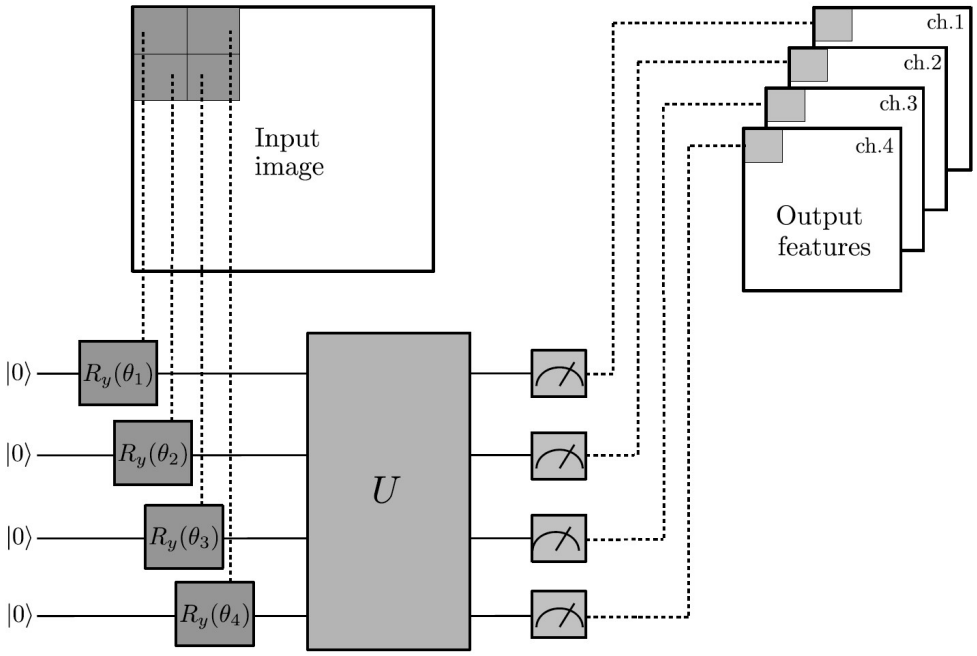

Figure 3.3: Quanvolutional Layer Architecture

3.3 Convolutional Layers

The input images are then passed through some convolution layers which help in extracting important features from the input image.

We train two models classical Convolutional model and another Quanvolutional model to compare the performance of both.

- CNN MODEL - A strictly classical convolutional neural network with the following network structure: CONV1- POOL1 - CONV2 - POOL2. The input image size is (28, 28, 1)Each convolutional layer used ReLU

activation with 5-by-5 and 3-by-3 filters respectively, with 128 and 64 filters in the first and second convolutional layers. The dropout layer has a 0.2 drop, while the softmax layer has 47 hidden units and is fully connected (1 for each target variable label).

- QNN MODEL - A CNN network with a single quanvolutional layer is the most basic quanvolutional neural network. The first transformation in the stack is the single quanvolutional layer, and the remaining architecture on top is identical to the CNN MODEL - QUANV1 - CONV1 - POOL1 - CONV2 - POOL2. The input image size is (14, 14, 4) after quanvolution. Each convolutional layer used ReLU activation with 5-by-5 and 3-by-3 filters respectively, with 128 and 64 filters in the first and second convolutional layers. The dropout layer has a 0.2 drop, while the softmax layer has 47 hidden units and is fully connected (1 for each target variable label).

3.4 Training

The models are trained for 20 epochs, with 2000, 5000 and 10000 images with a Batch size of 100 using Adam Optimizer and Sparse Categorical Cross Entropy Loss.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 128)       3328
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 64)        73792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 47)                147439
=================================================================
Total params: 224,559
Trainable params: 224,559
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.4: Classic CNN Layers

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 14, 14, 128)       12928
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 7, 7, 64)          73792
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 576)               0
_________________________________________________________________
dropout_17 (Dropout)         (None, 576)               0
_________________________________________________________________
dense_11 (Dense)             (None, 47)                27119
=================================================================
Total params: 113,839
Trainable params: 113,839
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.5: Quanvolution with CNN Layers

Chapter 4

Experimental Results

Experiments performed on Google Colab. By default, the platform offers Linux OS, with 12GB RAM and Nvidia Tesla T4 GPU 16GB memory.

Metric	Classic CNN Model	QML Model
Loss	1.56	0.92
Accuracy	0.69	0.78

Table 4.1: Accuracy & Loss with 2000 Training Images

Metric	Classic CNN Model	QML Model
Loss	0.97	0.63
Accuracy	0.78	0.83

Table 4.2: Accuracy & Loss with 5000 Training Images

Metric	Classic CNN Model	QML Model
Loss	0.74	0.51
Accuracy	0.83	0.85

Table 4.3: Accuracy & Loss with 10000 Training Images

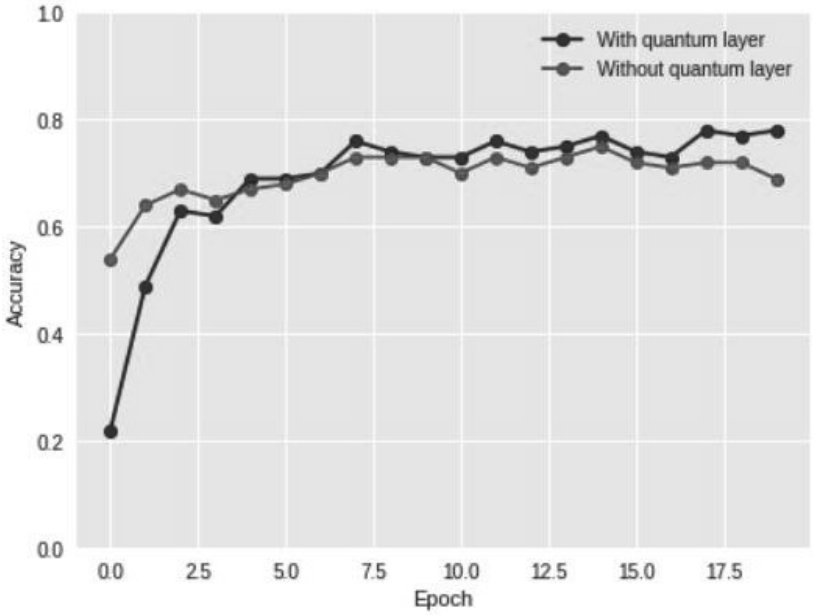

Figure 4.1: Accuracy with 2000 Training Images

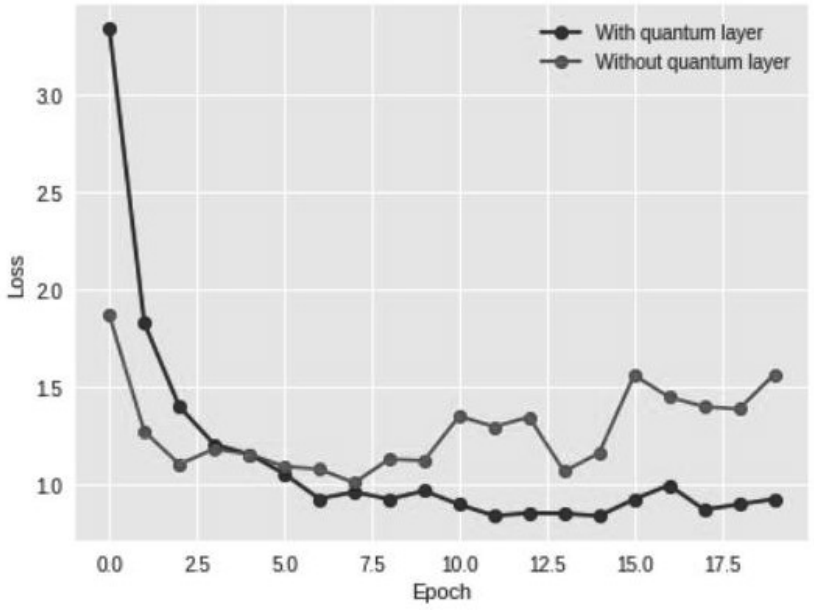

Figure 4.2: Loss with 2000 Training Images

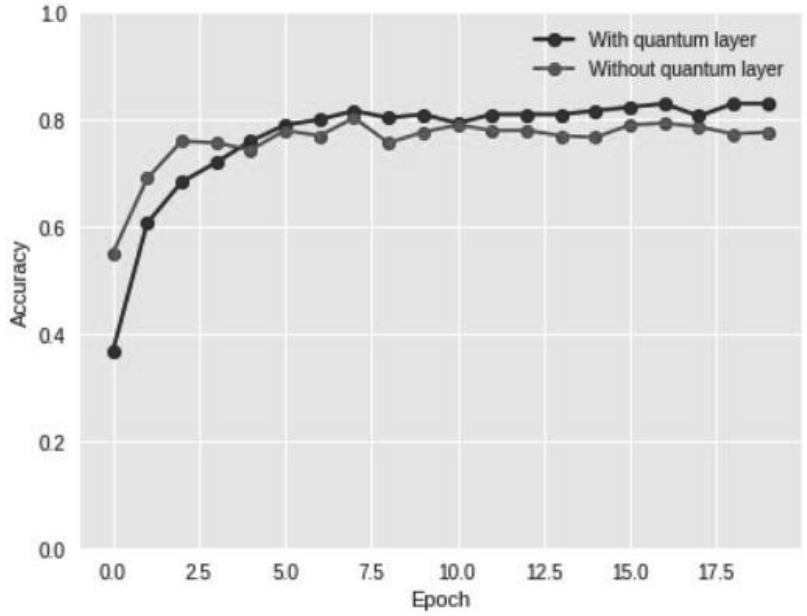

Figure 4.3: Accuracy with 5000 Training Images

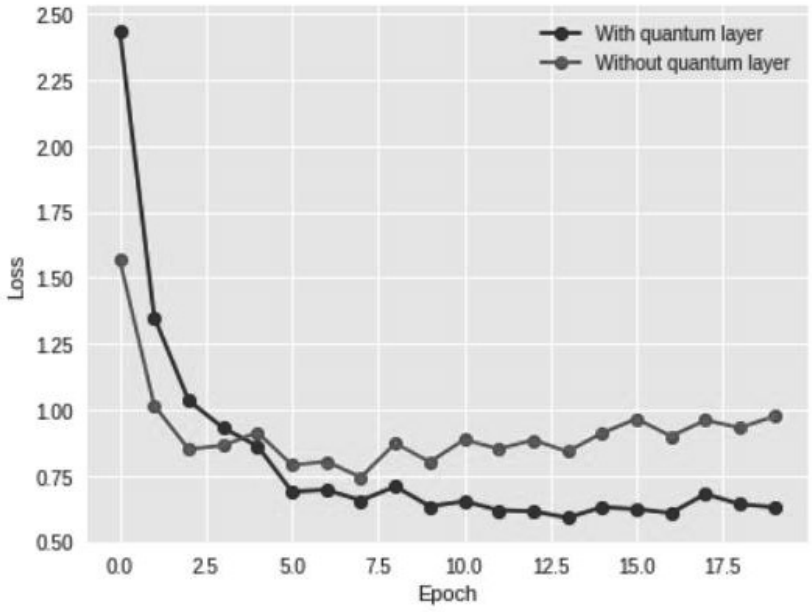

Figure 4.4: Loss with 5000 Training Images

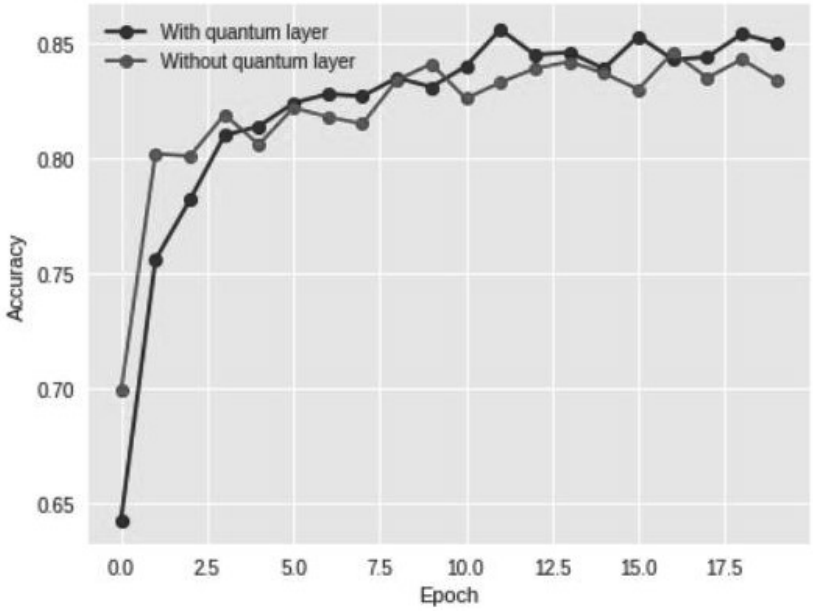

Figure 4.5: Accuracy with 10000 Training Images

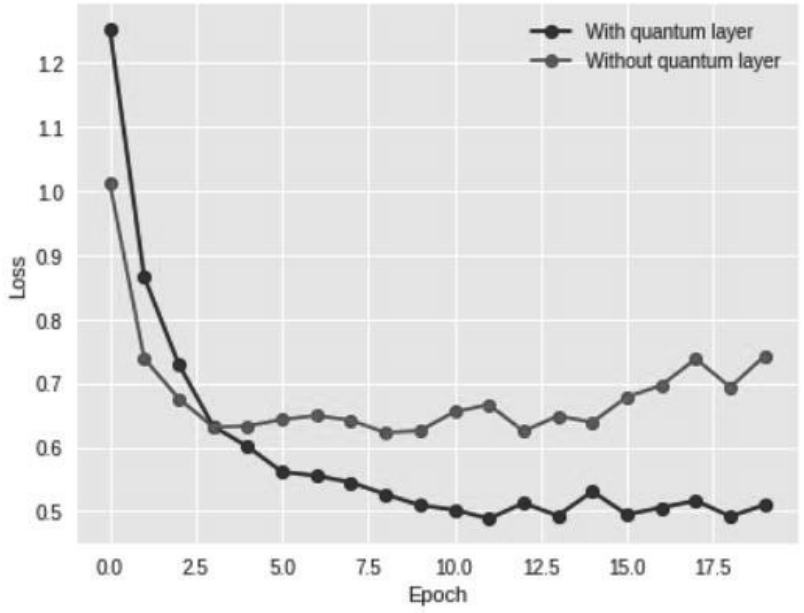

Figure 4.6: Loss with 10000 Training Images

Chapter 5

Conclusion and Future Scope

In this project a Offline handwriting character recognition module was built for converting English character images to digital text, with an accuracy of 85% using the Quantum Machine Learning model, as compared to the Classic CNN model which achieved only 83% accuracy using 10000 training images. Thus, we can see that the QML model clearly outperforms the Classic CNN Model. We can also see that as the number of training images increases QML model outperforms CNN even more, with a significant reduction in the Loss function. This project can further be expanded by adding character segmentation module for words, line segmentation module for pages, word segmentation module for lines to develop a fully functional end deliverable for a user. We can further experiment and explore with different CNN architectures working alongside a Quanvolutional layer to see which architectures gives a better accuracy.

Bibliography

[1] Bu-Qing Chen and Xu-Feng Niu. A novel neural network based on quantum computing. *International Journal of Theoretical Physics*, 59, 07 2020.

[2] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926, 2017.

[3] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. 02 2018.

[4] Siddhant Garg and Goutham Ramakrishnan. Advances in quantum deep learning: An overview. 05 2020.

[5] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. Quanvolutional neural networks: powering image recognition with quantum circuits. *Quantum Machine Intelligence*, 2:1–9, 02 2020.

[6] Zhaokai Li, Liu Xiaomei, Xu Nanyang, and Du jiangfeng. Experimental realization of a quantum support vector machine. *Physical Review Letters*, 114, 10 2014.

[7] Ian MacCormack, Conor Delaney, Alexey Galda, Nidhi Aggarwal, and Prineha Narang. Branching quantum convolutional neural networks. 12 2020.

[8] Andrea Mari, Thomas Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. 12 2019.

[9] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98, 03 2018.

[10] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99, 03 2019.

[11] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical Review Letters*, 122, 03 2018.