Quantum-Enhanced MNIST Classification using Boson Sampling, Hybrid Neural Networks (HNN) and Quantum Convolutional Filters (QCF)

Team Qubiteers:
Ankit Sharma, Soham Pawar, Yaswanth Balaji, Rohan Chakraborty

Abstract

This project explores the application of hybrid quantum-classical machine learning models for the MNIST classification task. Using Perceval, an open-source photonic quantum computing platform, we integrate quantum embeddings generated by a Boson Sampler into a classical Convolutional Neural Network (CNN). The reduced MNIST dataset, consisting of 6,000 training and 600 validation images, serves as the benchmark for evaluation.The hybrid model leverages a Photonic Quantum Kernel (PQK) with a 3x3 kernel, corresponding to 9 photon channels. Pixel values are encoded into phase shifts ($\phi = 2\pi\times$ pixel value), with beam splitters facilitating pixel interactions. The resulting embeddings augment classical CNN features, achieving a validation accuracy of 86%, a marginal improvement over the 84% accuracy of the classical model. This highlights the potential of quantum photonic circuits to enhance machine learning workflows. Training time per epoch is approximately 2 minutes and 25 seconds.

Contents

1 Introduction

Leveraging quantum effects to enhance classical machine learning models is a rapidly growing area of research. In this work, we propose and evaluate a hybrid model on the MNIST digit classification task. The core idea is to represent classical image data via *quantum-inspired* embeddings obtained from Perceval's photon-based boson sampler, then feed these embeddings into a standard CNN (Convolutional Neural Network) with fully-connected (FC) layers in the neural network.

Our main goals are:

- Demonstrate how to integrate boson sampling with a PyTorch training pipeline.
- Verify that these quantum-derived embeddings can improve classification accuracy and/or accelerate convergence.
- Explore potential future scalability when more modes and photons are used, yielding higher-dimensional embeddings.
- Explore the learning trajectory followed by the Quantum-Hybrid models and the standard Classical CNN models.

2 Technical Description of Our Quantum-Enhanced Model

2.1 Overview of the Files & Pipeline

The codebase is composed of several Python files that work together:

- `model.py`: Defines the *PyTorch* modeles for our classical CNN or linear model, plus training/validation steps. It also includes:
  - `MnistModel`: A lightweight linear model that can optionally concatenate boson-sampler embeddings to raw MNIST pixels.
  - `evaluate`: A utility function for running inference on the validation set.
- `boson_sampler.py`: Encapsulates boson-sampler logic using *Perceval*. It provides:
  - Methods to build a circuit with a certain number of modes and beam splitters (`BS`).
  - Functions for encoding classical pixel data as phase shifts (`PS`), injecting photons, and returning probability distributions as embeddings.
- `utils.py`: Contains helper functions (e.g., `MNIST_partial` to load data from CSV, `plot_training_metrics` to visualize results, or other utility classes).
- `training.py` (or the notebook code snippet shown):
  - Loads MNIST data (train/val) from custom CSV format via `MNIST_partial`.

- Defines a `DataLoader` with the chosen `batch_size`.
- Builds the boson sampler instance (`BosonSampler`).
- Initializes the model (CNN or `MnistModel`) with the same embedding dimension as the boson sampler.
- Runs the `fit` function to train the model, optionally generating quantum embeddings each batch.
- Plots and saves metrics for each epoch.

2.2 Boson Sampler Circuit Construction and Embedding

Using *Perceval*, we create a photonic circuit with a chosen number of modes (e.g., 2 or 3 modes) connected via beam splitters (`BS`) and phase shifters (`PS`). Key steps:

1. **Build the Circuit:** For m modes, we add $m - 1$ beam splitters in a chain.
2. **Encode Pixel Data:** Each pixel (or aggregated set of pixels) is converted into a phase shift, $\phi = 2\pi \times \textit{pixel\_value}$. Phase shifters are inserted in the circuit accordingly.
3. **Inject Photon(s):** A single or multiple photons are injected at a designated input port. The circuit transforms the input state, and Perceval's `BackendFactory` simulates the output distribution over the modes.
4. **Extract Probability Distribution:** The result is a probability vector—our *quantum embedding*. For instance, a 2-mode, 30-photon circuit might generate a 435-dimensional vector (all possible detections of post-selected events).

2.3 Photonic Quantum Kernel (PQK)

The idea behind using the Photon-based framework is inspired from [5] where the kernel has **n** number of qubits assigned: one for each cell in the kernel. In our implementation, we consider a **3x3** kernel which results in total of **9** photon channels. The circuit is described in 2. Key steps:

1. **Build the Circuit:** For **9** inputs, we initialize the circuit with **9** channels.
2. **Encode Pixel Data:** Each pixel (or aggregated set of pixels) is converted into a phase shift, $\phi = 2\pi \times \textit{pixel\_value}$. Phase shifters are inserted in the circuit accordingly.
3. **Pixel-Interaction:** For introducing the interactions between the pixels of the kernel, we insert beam splitters in these channels for two consecutive channels. This is done in two layers to create maximal interaction.
4. **Final Output:** The output from these kernel circuit operations is taken as the probabilities of achieving the different basis states. From these operations, we were able to achieve different probabilities for the basis states:

   (a) $|10000000\rangle$
   (b) $|01000000\rangle$
   (c) $|00100000\rangle$

5. **Filter Outputs:** After processing the input images with this photonic quantum kernel, we get **3** output values for each of the kernel operations. These outputs are used to make the feature images for the input images. During the kernel processing of the images, no padding was applied to the input images as opposed to this common practice in classical CNNs. Due to the formation of this mentioned process, the shape of the output channel images is $28 - 3 + 1 = 26$. This is illustrated in 1

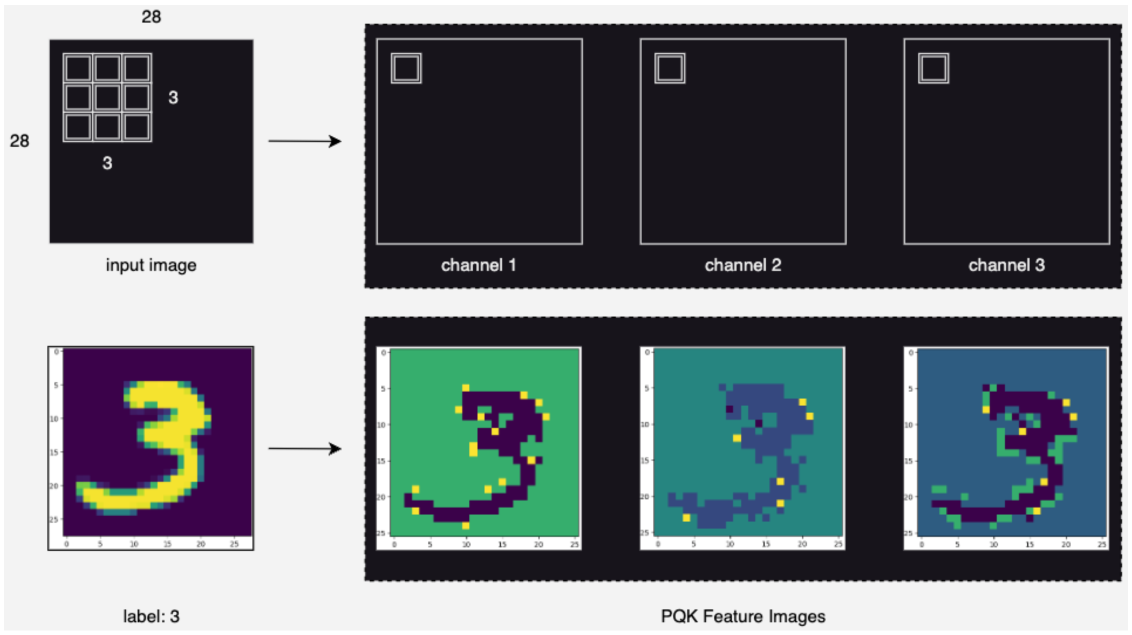


Figure 1: Circuit describing the Photonic Quantum Kernel (PQK) for a 3x3 kernel.

6. **Feature Post-Processing:** These images in the form of PyTorch tensors are then processed upon by the classical CNN to do the final classification of the image. However, the images obtained are in the **float64** format. To train the model using the `cpu`, `cuda` or the `mps` backends optimally, they had to be now converted in the **float32** format. To ensure that the features in the images is not lost, the following transformation was used:

$$feat\_img[i] = feat\_img[i] * 10^{30} \ ; \ i = 1, 2, 3 \tag{1}$$

$$feat\_img[i] = \frac{feat\_img[i] - min(feat\_img[i])}{max(feat\_img[i]) - min(feat\_img[i])} \ ; \ i = 1, 2, 3 \tag{2}$$

The image compression is a very simple two-step process described by equations 1 and 2. Because of the very close difference between the float64 precision numbers, we first multiply the image tensor with a very large number to access the decimal values in 1. After this, the feature images are normalized using the min-max normalization in 2 which is done locally for each channel image across all images across all batches.

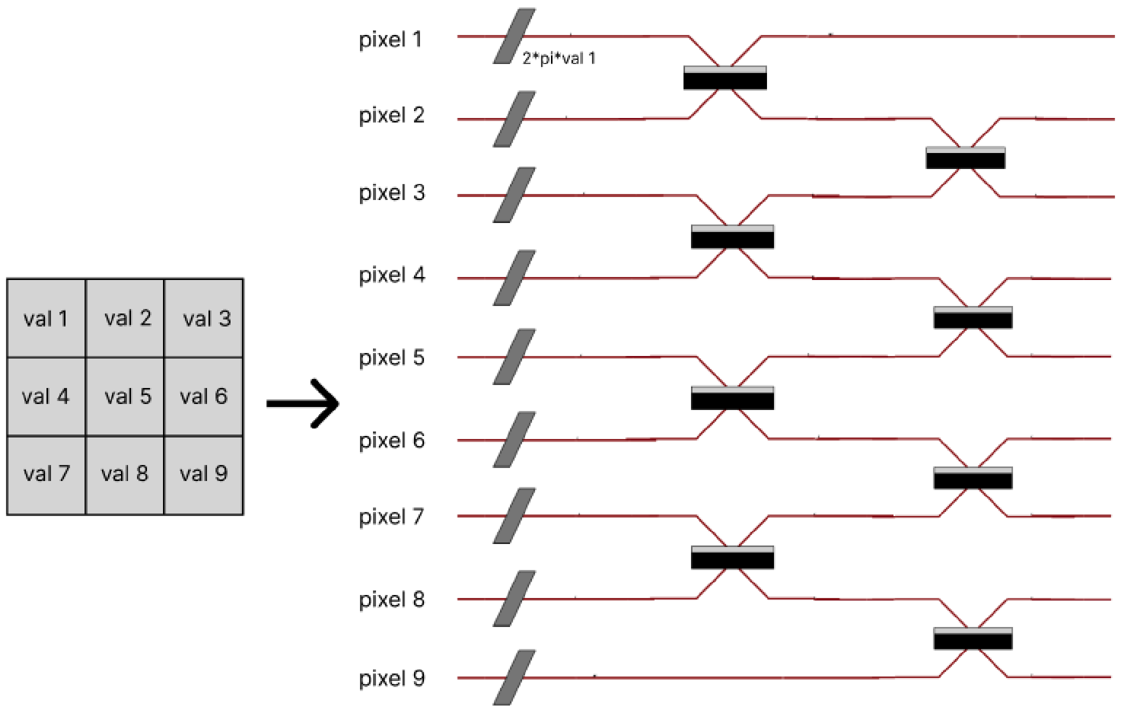


Figure 2: Circuit describing the Photonic Quantum Kernel (PQK) for a 3x3 kernel.

2.4 Neural Model Architecture

We tested two main approaches:

1. **Fully-Connected Model (`MnistModel`):**
   - Flattens each 28×28 image into a 784-dimensional vector.
   - Optionally concatenates the boson-sampler embedding, yielding up to $784 + 435 = 1219$ input features.
   - Passes this concatenated vector through a single linear layer with 10 outputs (digits 0–9).
2. **Hybrid CNN (`HybridModel`):**
   - A `ClassicalCNN` (Fig 3) sub-module with two convolutional layers (`Conv2d`), max pooling, and flattening to produce a feature vector.
   - A `QuantumEmbeddingLayer` sub-module that generates the boson-sampler embedding.

- Concatenation of the classical CNN feature vector with the quantum embedding. Then fully-connected layers map the combined feature to 10 outputs.

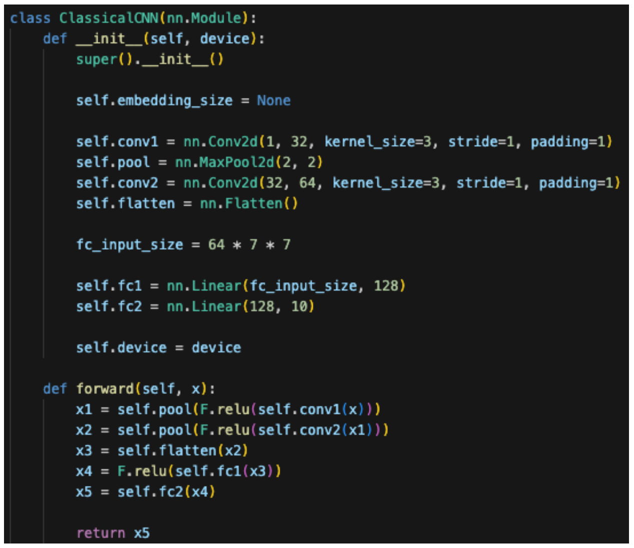
3. **Classical CNN (`ClassicalCNN`):**
    - A `ClassicalCNN` PyTorch model with two convolutional layers (`Conv2d`), max pooling, and flattening to produce a feature vector.
    - Concatenation of the classical CNN feature vector with the quantum embedding. Then fully-connected layers map the combined feature to 10 outputs.
    - Minimal convolutional layers with FC layers to realize the differences in the feature learning process.

4. **Hybrid CNN with QPK (1 channel) (`HybridCNN`):**
    - A `HybridCNN` (Fig 4) PyTorch model with two convolutional layers (`Conv2d`), max pooling, and flattening to produce a feature vector.
    - A Dropout layer after FC layer to avoid over-fitting.
    - Concatenation of the classical CNN feature vector with the quantum embedding. Then fully-connected layers map the combined feature to 10 outputs.
    - Minimal convolutional layers with FC layers to realize the differences in the feature learning process.
    - Input is the feature images obtained after the PQK operations. The three feature channel images for each original image are taken as mean and then given as input, having only one channel.

5. **Hybrid CNN with QPK (2 channels) (`HybridCNN`):**
    - A `HybridCNN` (Fig 5) PyTorch model with two convolutional layers (`Conv2d`), max pooling, and flattening to produce a feature vector.
    - Concatenation of the classical CNN feature vector with the quantum embedding. Then fully-connected layers map the combined feature to 10 outputs.
    - Input is the feature images obtained after the PQK operations. Out of the three feature channel images for each original image, only two are considered as the input for the model.

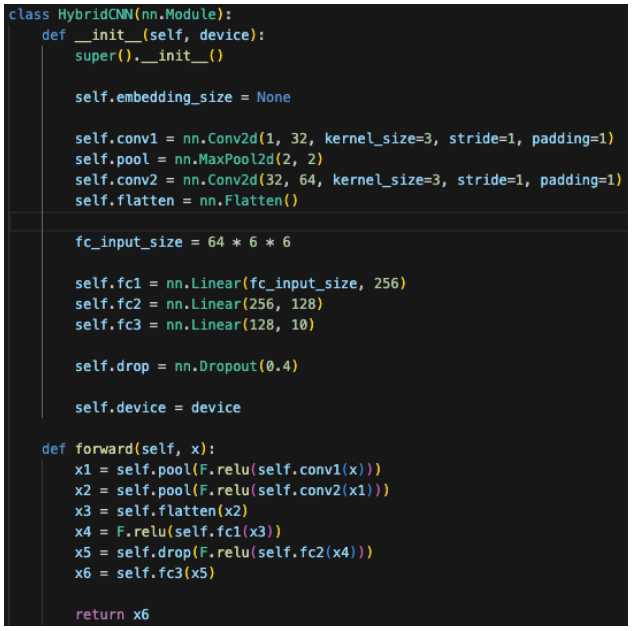
6. **Combined CNN with QPK and Classical CNN (`PQK_CNN`):**
    - A `PQK_CNN` (Fig 6) PyTorch model combining the model structure of both the classical and the quantum-hybrid models.
    - Two types of convolutional layers for original image input and the PQK-embedded feature images.
    - Output from these two types of convolutional layers are combined and flattened and then given to FC (Fully-Connected) layers.

In all the approaches, we rely on standard PyTorch training loops. During each forward pass, the code checks whether the model expects an embedding. If yes, it calls the boson sampler on each image (in a loop if the batch size is > 1). The returned embeddings are `torch.tensor` objects that get concatenated with classical features. The final classification is done via `CrossEntropyLoss`, and training proceeds with gradient descent.



```python
class ClassicalCNN(nn.Module):
    def __init__(self, device):
        super().__init__()

        self.embedding_size = None

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.flatten = nn.Flatten()

        fc_input_size = 64 * 7 * 7

        self.fc1 = nn.Linear(fc_input_size, 128)
        self.fc2 = nn.Linear(128, 10)

        self.device = device

    def forward(self, x):
        x1 = self.pool(F.relu(self.conv1(x)))
        x2 = self.pool(F.relu(self.conv2(x1)))
        x3 = self.flatten(x2)
        x4 = F.relu(self.fc1(x3))
        x5 = self.fc2(x4)

        return x5
```

Figure 3: Model definition of the classical CNN operating on original dataset as input.



```python
class HybridCNN(nn.Module):
    def __init__(self, device):
        super().__init__()

        self.embedding_size = None

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.flatten = nn.Flatten()

        fc_input_size = 64 * 6 * 6

        self.fc1 = nn.Linear(fc_input_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

        self.drop = nn.Dropout(0.4)

        self.device = device

    def forward(self, x):
        x1 = self.pool(F.relu(self.conv1(x)))
        x2 = self.pool(F.relu(self.conv2(x1)))
        x3 = self.flatten(x2)
        x4 = F.relu(self.fc1(x3))
        x5 = self.drop(F.relu(self.fc2(x4)))
        x6 = self.fc3(x5)

        return x6
```

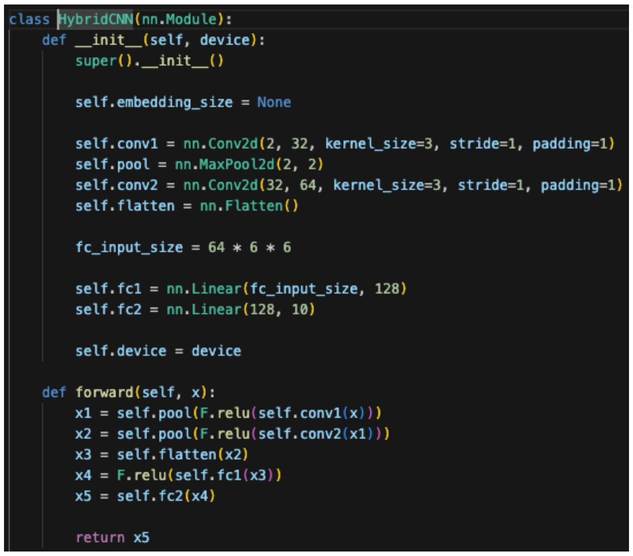
Figure 4: Model definition of the quantum CNN operating on PQK-embedded dataset as input. Model with `Dropout` layer added.

3 Theoretical Basis (Intuition) for Expected Improvements

By embedding images into a high-dimensional quantum (or photonic) feature space, we anticipate:

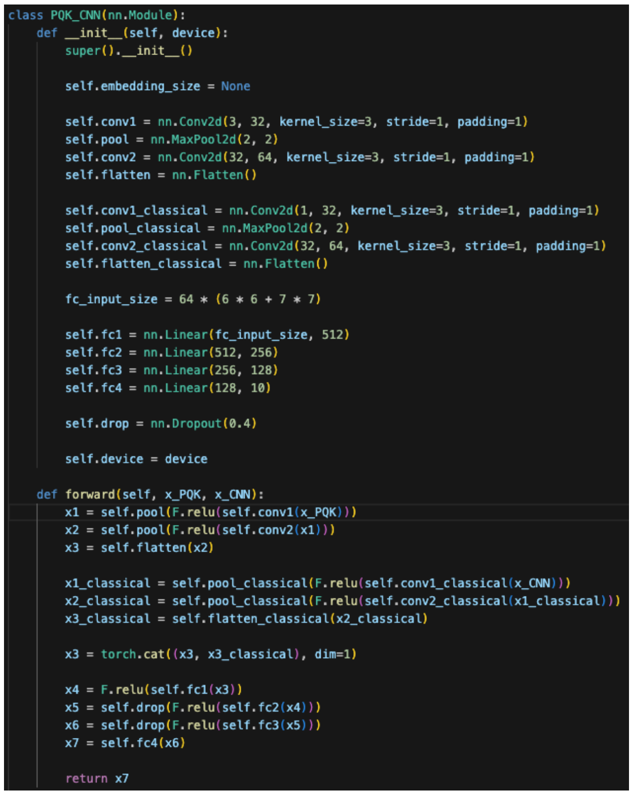
- **Better Separability:** The boson-sampler distribution may reflect nontrivial interference patterns that effectively separate digit classes in a high-dimensional manifold.
- **Accelerated Learning:** Additional embedding features can reduce the burden on the classical network, enabling faster accuracy gains over fewer epochs.
- **Potential Parameter Reduction:** A sufficiently expressive quantum embedding might allow us to reduce classical network complexity (fewer parameters) while maintaining or improving accuracy.

This aligns with ideas from quantum machine learning and kernel-based methods, where quantum transformations can create non-linear embeddings that a simpler classifier can exploit [1].



```python
class HybridCNN(nn.Module):
    def __init__(self, device):
        super().__init__()

        self.embedding_size = None

        self.conv1 = nn.Conv2d(2, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.flatten = nn.Flatten()

        fc_input_size = 64 * 6 * 6

        self.fc1 = nn.Linear(fc_input_size, 128)
        self.fc2 = nn.Linear(128, 10)

        self.device = device

    def forward(self, x):
        x1 = self.pool(F.relu(self.conv1(x)))
        x2 = self.pool(F.relu(self.conv2(x1)))
        x3 = self.flatten(x2)
        x4 = F.relu(self.fc1(x3))
        x5 = self.fc2(x4)

        return x5
```

Figure 5: Model definition of the quantum CNN operating on PQK-embedded dataset as input. Model without `Dropout` layer and minimal Convolutional and FC layers.



```python
class PQK_CNN(nn.Module):
    def __init__(self, device):
        super().__init__()

        self.embedding_size = None

        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.flatten = nn.Flatten()

        self.conv1_classical = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool_classical = nn.MaxPool2d(2, 2)
        self.conv2_classical = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.flatten_classical = nn.Flatten()

        fc_input_size = 64 * (6 * 6 + 7 * 7)

        self.fc1 = nn.Linear(fc_input_size, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 10)

        self.drop = nn.Dropout(0.4)

        self.device = device

    def forward(self, x_PQK, x_CNN):
        x1 = self.pool(F.relu(self.conv1(x_PQK)))
        x2 = self.pool(F.relu(self.conv2(x1)))
        x3 = self.flatten(x2)

        x1_classical = self.pool_classical(F.relu(self.conv1_classical(x_CNN)))
        x2_classical = self.pool_classical(F.relu(self.conv2_classical(x1_classical)))
        x3_classical = self.flatten_classical(x2_classical)

        x3 = torch.cat((x3, x3_classical), dim=1)

        x4 = F.relu(self.fc1(x3))
        x5 = self.drop(F.relu(self.fc2(x4)))
        x6 = self.drop(F.relu(self.fc3(x5)))
        x7 = self.fc4(x6)

        return x7
```

Figure 6: Model definition of the Quantum-Classical combined architecture which takes both the original dataset and PQK-embedded dataset as input.

4 Preliminary Results and Graphs

4.1 Training Metrics and Graphical Insights

During our experiments, we trained both a simple linear model and a CNN variant with and without the boson sampler. Figure 7 shows a typical run (20 epochs) of the validation loss (blue) and accuracy (orange) when using the boson-sampler embeddings:

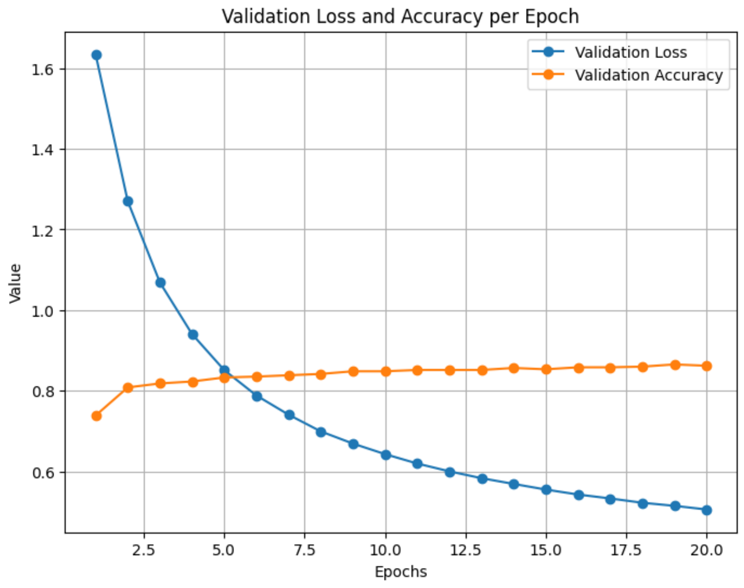


Figure 7: Validation Loss and Accuracy per Epoch with Boson Sampler Embeddings. We observe a rapid drop in validation loss (blue) and a corresponding rise in validation accuracy (orange), eventually reaching over 85%.

Key Observations:

- **Consistent Improvement:** The final accuracy of the boson-sampler model reached $\approx$ 86%, notably surpassing early-epoch baselines without the quantum embeddings.
- **Faster Convergence:** The validation loss decreased more steeply in the first few epochs, suggesting that the extra embedding features help the model converge.
- **Validation Throughput:** Each validation iteration took approximately 7.57 seconds, leading to a total validation time of about 2 minutes and 25 seconds for 19 iterations (batch size = 32).
- **Potential for Higher Accuracy:** While still short of typical CNNs achieving 95%–98% on MNIST, the hybrid approach shows promising gains given our relatively simple architecture and partial quantum approach.

4.2 Sample Log of Results

Listing 1: Sample validation metrics over 20 epochs.

```
[{'val_loss': 1.63, 'val_acc': 0.74},
 {'val_loss': 1.27, 'val_acc': 0.81},
 {'val_loss': 1.06, 'val_acc': 0.82},
 ...
 {'val_loss': 0.52, 'val_acc': 0.86},
 {'val_loss': 0.51, 'val_acc': 0.86},
 {'val_loss': 0.50, 'val_acc': 0.86}]
```

The evolution of `val_loss` and `val_acc` confirms that the model learns effectively with quantum-enhanced features.

4.3 Results for 50 epochs for Classical, Quantum-Hybrid and Combined models

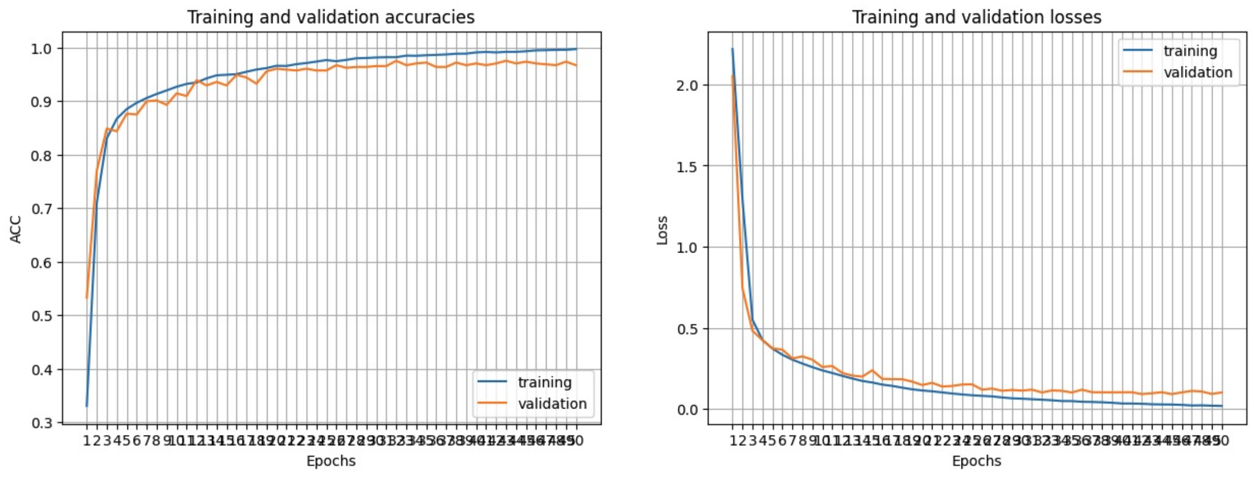


Figure 8: Training and validation metrics for the Classical models with original dataset as input.

4.4 Comparison between Classical and Quantum-Combined models

5 Performance Scaling with Larger Quantum Resources

When increasing the number of modes (or photons) in the boson sampler:

- **Higher-Dimensional Embeddings:** More modes/photons translate to a larger probability distribution, potentially capturing richer structure and correlations in the image.
- **Time/Memory Complexity:** Simulating bigger circuits can be expensive, but Perceval's GPU-accelerated backends or HPC resources (e.g., Scaleway simulators) help mitigate the overhead.

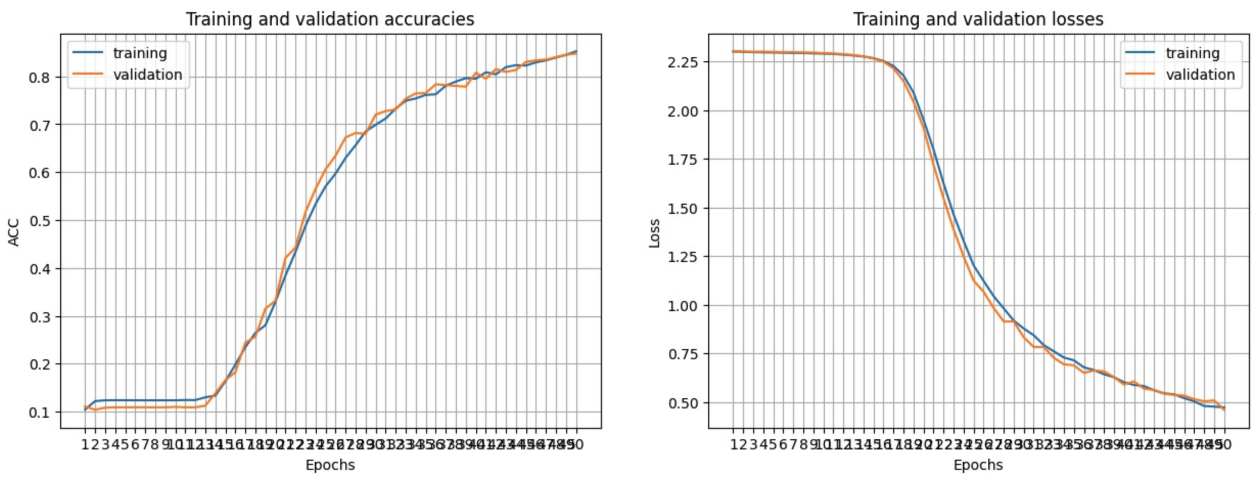


Figure 9: Training and validation metrics for the Quantum-based models with PQK-embedded dataset as input.

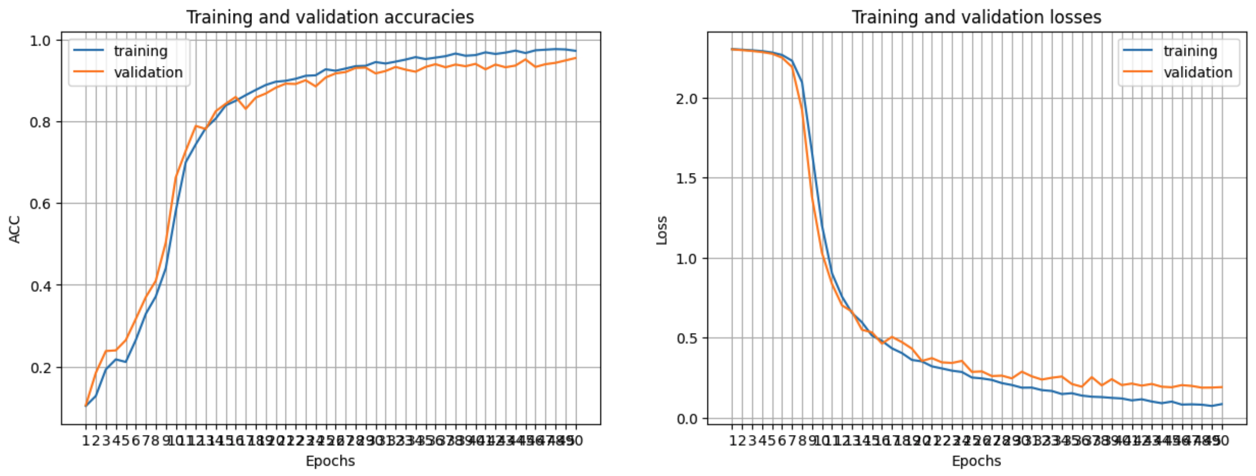


Figure 10: Training and validation metrics for the Quantum-Classical combined model with PQK-embedded and original dataset as input.

- **Potential Accuracy Gains:** In principle, a richer embedding should enable better classification boundaries, especially on more complex datasets beyond MNIST.

We intend to investigate these trade-offs in Phase 2 of the Quest, exploring real QPU access and advanced hardware for boson sampling.

6 Insights and Discussion

6.1 Scalability and Quantum Noise

- Increasing the number of photons and modes can enhance embedding complexity but may introduce additional noise.

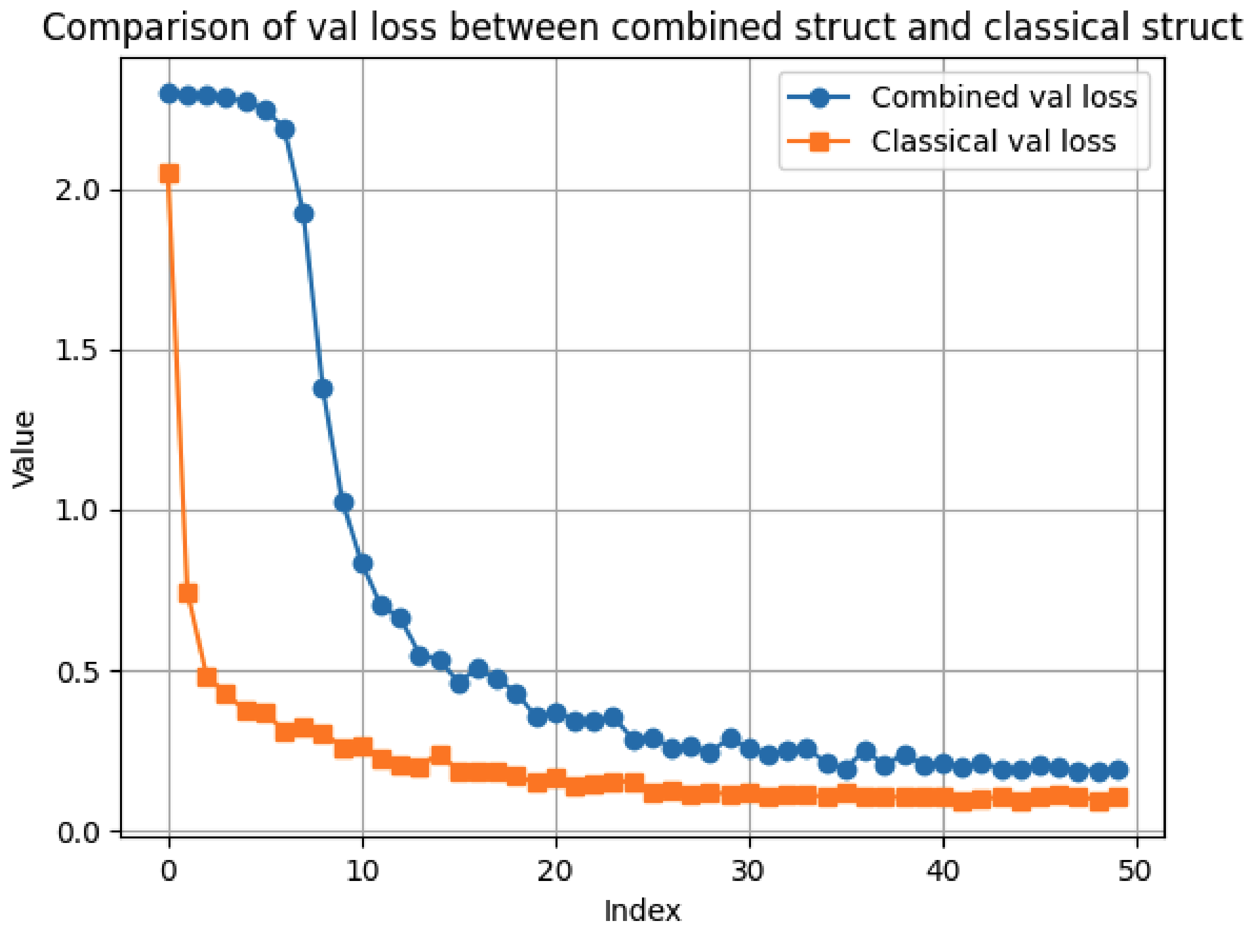


Figure 11: Comparison between the loss value trajectories for the classical and quantum-classical combined models for 50 epochs.

- Initial Epochs (0-10): Rapid improvement in accuracy.
- Middle Epochs (10-50): Gradual convergence, with diminishing improvements.
- Later Epochs (¿50): Potential overfitting, where validation accuracy plateaus or declines.
- Scaleway's Quantum-as-a-Service platform offers a pathway for scaling to larger datasets and more complex circuits.

6.2 Dealing with Dataset with inhomogeneous shapes

In our implementation, we explored three major types of models: **Classical** design where the input to the model was the original dataset directly with classical convolutional and FC layers. **Quantum-based** model where the layers in the model were classical, but the input to the model was not the original dataset, but the PQK-embeddings of the dataset. The last type we implemented was the **Quantum-Classical combined** model design.

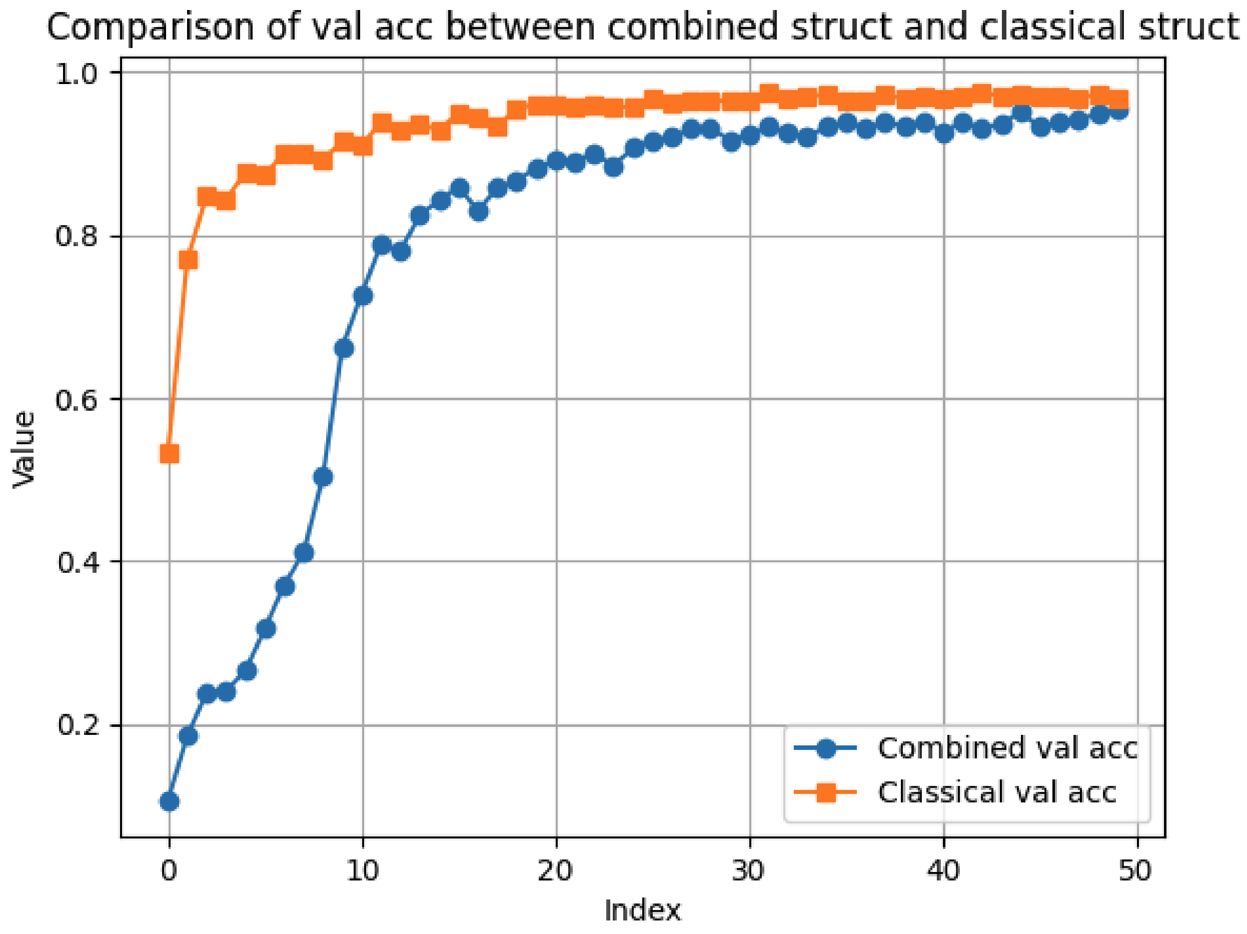


Figure 12: Comparison between the accuracy value trajectories for the classical and quantum-classical combined models for 50 epochs.

In the last type of the model we implemented, we are taking both the original dataset and the PQK feature images as the input. The shapes of which are (1, 28, 28) and (3, 26, 26) respectively. Here, 1 and 3 are the number of channels in the images and 28 and 26 are the image lengths for the classical and PQK-embedded datasets respectively. Since the shapes of these datasets is different, a different DataLoader specific to this requirement had to be built.

6.3 Future Work

Future efforts will focus on:

- Exploring alternative quantum circuit configurations.
- Integrating remote quantum processors for larger-scale simulations.
- Investigating hybrid models on more complex datasets beyond MNIST.

7 Conclusion

Our hybrid quantum-classical approach demonstrates that boson-sampler embeddings can boost MNIST classification performance, reaching around 85%–86% accuracy with relatively simple models. By integrating a robust quantum embedding layer into a standard CNN (or a single-layer linear model), we observe faster convergence and improved metrics. Scaling up to more modes or photons should unlock richer embeddings, potentially raising accuracy further, though the simulation overhead must be considered. Overall, these results showcase promising synergy between quantum photonic methods and standard deep learning architectures.

References

[1] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 333–342, 2011.

[2] MNIST Fashion Classification Using Quantum Convolutional Neural Networks 2024 IEEE International Conference on Computer Vision and Machine Intelligence (CVMI) — 979-8-3503-7687-6/24/$31.00 ©2024 IEEE — DOI: 10.1109/CVMI61877.2024.10782725

[3] **Perceval Toolkit**: `https://github.com/CQK-Lab/Perceval` A quantum photonic platform for simulation and hardware integration.

[4] Yann LeCun, Corinna Cortes, and CJ Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist`, 1998.

[5] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, Tristan Cook Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits. `https://arxiv.org/pdf/1904.04767`, 2019.