

Team HackWizard

Planck'd 2025

Quantum Computing Hackathon organized by the Quantum Computing Club of
IIIT Bangalore, Qimaya.

Meet The Wizards



Quantum Machine Learning Track

Problem Statement-1: MNIST Classification

Dataset: MNIST Dataset

Classical Approach

We built a classical ML model to predict the input images into final class labels using CNN and SVM approach.

Sequential Convolutional Neural Network (CNN):

We constructed a Sequential Convolutional Neural Network (CNN) using **tensorflow.keras**.

The architecture of our model is as follows:

- **Convolutional Block 1:** Conv2D layer with 16 filters (3x3 kernel), followed by a LeakyReLU activation and MaxPooling2D.
- **Convolutional Block 2:** Conv2D layer with 32 filters (5x5 kernel), followed by an ELU activation and MaxPooling2D.
- **Convolutional Block 3:** Conv2D layer with 64 filters (3x3 kernel), followed by an ELU activation and MaxPooling2D.
- **Convolutional Block 4:** Conv2D layer with 128 filters (3x3 kernel) using swish activation, followed by MaxPooling2D.
- **Convolutional Block 5:** Conv2D layer with 256 filters (3x3 kernel) using gelu activation, followed by MaxPooling2D.
- **Fully Connected Layers:** A Flatten layer, followed by a Dense layer of 256 neurons (relu activation), a Dropout of 0.4, a Dense layer of 128 neurons (swish activation), and a Dropout of 0.3.
- **Output Layer:** A Dense layer with 10 neurons and **softmax** activation to produce class probabilities.

The model has a total of **500, 490 trainable parameters**.

Training Methodology

1. Data Preparation:

- The MNIST dataset was loaded.
- Pixel values were normalized to a [0, 1] range by dividing by 255.0.
- A channel dimension was added to the images using `np.expand_dims`.
- The training and test labels (`y_train`, `y_test`) were one-hot encoded using `to_categorical`.
- A data_augmentation pipeline (including rotation, translation, zoom, and contrast) was defined, though it was not used in the final `model.fit()` call .

2. Compilation:

- The model was compiled with the adam optimizer and `categorical_crossentropy` as the loss function.

3. Training:

- The model was trained using `model.fit()` for a target of 30 epochs, with a batch size of 64.
- An EarlyStopping callback was used to monitor `val_loss` with a patience of 3 epochs.
- Training stopped early after **Epoch 7**, as the validation loss did not improve sufficiently .

Testing Methodology

1. **Evaluation:** The model's final performance was evaluated on the unseen test set (`x_test`, `y_test`) using the `model.evaluate()` method, which provided the final test loss and test accuracy.
2. **Prediction:** The `model.predict()` function was called on `x_test` to get the raw probability predictions for each class.
3. **Label Conversion:** These probabilities were converted into final class labels (0-9) using `np.argmax`. The one-hot encoded true labels (`y_test`)

were also converted back to class indices using `np.argmax` for comparison.

Metrics

We used the following metrics to evaluate our CNN model:

- **Accuracy:** Calculated using both `model.evaluate()` and `accuracy_score` from `sklearn.metrics`.
- **Loss:** Training and validation loss were tracked during training and a final test loss was reported.
- **Classification Report:** A detailed report from `sklearn.metrics` was generated, showing **precision**, **recall**, and **f1-score** for each digit class.
- **Confusion Matrix:** A confusion matrix was generated and plotted using `seaborn.heatmap` to visualize correct and incorrect predictions for each class.
- **Training vs. Validation Plots:** We plotted the accuracy and loss curves over epochs for both training and validation sets.

Results

- **Final Test Accuracy: 0.9891** (or 98.91%).
- **Final Test Loss: 0.0493.**
- **Best Validation Accuracy: 0.9921** (achieved during training at Epoch 7).
- **Classification Report:** The final report showed excellent performance, with precision, recall, and f1-scores at or near 0.99 for almost all classes.
- **Confusion Matrix:** The heatmap on page 4 confirms the high accuracy, with very few misclassifications outside the main diagonal.

SVM (Support Vector Machine)

Our machine learning approach involved using a Support Vector Classifier (SVC) from the `sklearn.svm` library.

We experimented with the model by tuning the regularization parameter **C**, while keeping the kernel consistent. The specific models defined were:

- **Model 1:** SVC(kernel='rbf', C=1.0)
- **Model 2:** SVC(kernel='rbf', C=10)
- **Model 3:** SVC(kernel='rbf', C=50)

Training & Testing Methodology

- **Data Preparation:** The MNIST training (`x_train`) and test (`x_test`) datasets, which consist of 28x28 images, were flattened into 1-dimensional vectors of 784 features. This was done using the `.reshape(len(...), -1)` method.
- **Training:** Each of the three SVM models was trained on the flattened training data (`x_train_flat`) and its corresponding labels (`y_train`) using the `.fit()` method.
- **Testing:** After training, each model was used to generate predictions (`y_pred`, `y_pred_c`, `y_pred_c1`) on the flattened test data (`x_test_flat`) using the `.predict()` method.

Metrics

We used several metrics from `sklearn.metrics` to evaluate the performance of our models:

- **Accuracy Score:** `accuracy_score` was used to get the overall percentage of correct predictions.
- **Confusion Matrix:** `confusion_matrix` was used to visualize the performance of each model, showing correct and incorrect predictions for each digit class. This was plotted as a heatmap using seaborn.
- **Classification Report:** `classification_report` was generated for all three models to get a detailed breakdown of precision, recall, and f1-score for each class, along with the overall accuracy.

Results

The key results from our SVM model comparison were the test accuracies:

- **Model 1 (C=1.0):**
 - **Test Accuracy:** 0.9792
 - **Classification Report:** Showed an overall accuracy of 0.98 (rounded).
- **Model 2 (C=10):**
 - **Test Accuracy:** 0.9837
 - **Classification Report:** Showed an overall accuracy of 0.98 (rounded).
- **Model 3 (C=50):**
 - **Test Accuracy:** 0.9833
 - **Classification Report:** Showed an overall accuracy of 0.98 (rounded).

Based on these results, **the SVM model with C=10 yielded the highest test accuracy.**

Quantum Hybrid Model

We implemented a Hybrid Quantum-Classical (HQC) model that uses a classical convolutional neural network (CNN) to extract features, which are then processed by a quantum circuit. This was built using PyTorch and PennyLane.

The model, **HybridVQAModel**, has three main parts:

1. Classical Feature Extractor (CNN):

- A PyTorch nn.Sequential model with two Conv2d layers.
- This CNN processes the input 28x28 image and flattens it into a feature vector with 64 elements ($2^{\{N_QUBITS\}}$).

2. Quantum Layer (VQC):

- A PennyLane QNode (quantum_circuit) with 6 qubits and 3 variational layers was defined.
- **Encoding:** The 64 classical features are loaded into the quantum state using qml.AmplitudeEmbedding.
- **Variational Circuit:** The trainable part of the circuit uses qml.StronglyEntanglingLayers.
- **Measurement:** The circuit outputs a 6-element feature vector by measuring the PauliZ expectation value of each qubit.
- This QNode is wrapped in a QuantumLayer using qml.qnn.TorchLayer to make it compatible with PyTorch.

3. Classical Head:

- A final nn.Linear layer takes the 6 outputs from the quantum circuit and maps them to the 10 final classes (for digits 0-9).

Training Methodology

1. **Data Preparation:** Subsets of the MNIST dataset were used: 10,000 samples for training and 800 samples for testing/validation. DataLoaders were set up with a batch size of 32.
2. **Model Setup:** The HybridVQAModel was compiled with the optim.Adam optimizer (Learning Rate = 1e-3) and nn.CrossEntropyLoss as the loss function.

- 3. Training Loop:** The model was trained for a target of 50 epochs.
- 4. Best Model Checkpointing:** During training, the model's validation accuracy was checked after each epoch. If the current model had a better val_acc than the previous best, its state was saved to best_vqa_mnist.pth.
- 5. Early Stopping:** A custom EarlyStopping class was implemented to monitor the val_loss. Training would stop if the validation loss did not improve by 0.0001 for 5 consecutive epochs.

Testing Methodology

Our testing process had two phases:

- 1. Validation (During Training):** At the end of each epoch, the evaluate function was called to measure the model's loss and accuracy on the 800-sample test set. This was used to track progress and save the best model .
- 2. Final Evaluation (After Training):** Once the training loop finished (either by completing all epochs or via early stopping), the script was designed to:
 - Load the best performing model from the best_vqa_mnist.pth file.
 - Run the evaluate function one last time on the test loader to get the final predictions (final_preds) and true labels (final_labels).

Metrics

The script was set up to calculate and report the following metrics on the final "best" model's test results:

- Test Loss and Test Accuracy (accuracy_score).
- Macro-averaged Precision, Recall, and F1-Score.
- A detailed classification_report (per-class precision, recall, f1-score).
- A confusion_matrix, which was also plotted using ConfusionMatrixDisplay.
- Plots of Training vs. Validation Loss and Validation Accuracy over epochs.

Results

The execution log shows the model training and validating. The final summary metrics (from the "Final Evaluation" step) are not visible in the provided log, but the epoch-by-epoch validation results are:

- The model trained for at least 22 epochs.
- The **best validation accuracy seen in the log was 0.9712 (or 97.12%)**, which was achieved at **Epoch 20**.

Comparison

In our experiments, the **classical models performed better than the hybrid quantum models**.

Our best classical model (the complex CNN from the second file) achieved a test accuracy of 0.9891 (98.91%). Our best hybrid quantum model (the PyTorch-based VQA) achieved a peak validation accuracy of 0.9712 (97.12%).

Performance Comparison

Here is a breakdown of the best-performing model from each category:

- 1st Best Classical (CNN):
 - Model: The 5-block complex CNN
 - Test Accuracy: 0.9891.
 - Peak Validation Accuracy: 0.9921.
- 2nd Best Classical (SVM):
 - Model: The SVC with C=10 from iit_B_hackathon_code[1].pdf.
 - Test Accuracy: 0.9837.
- 3rd Hybrid Quantum (VQA):
 - Model: The PyTorch/PennyLane model
 - Peak Validation Accuracy: 0.9712.

Why the Classical Models Performed Better

Our results are a very common and expected outcome in current Quantum Machine Learning (QML) research. Here's why our classical models won:

1. **Maturity of Classical Models:** MNIST is a "solved" problem for classical AI. The complex CNN we built, with 5 convolutional blocks and over 500,000 parameters, is perfectly suited to this task and has been highly optimized over decades of research.
2. **The "Quantum Bottleneck":** Our best hybrid model (the VQA) used a classical CNN to extract features, but it had to compress all the information from the 28x28 image into a 64-element vector to fit it into 6 qubits. The quantum circuit then processed this and outputted only 6 values. This massive compression created an information bottleneck, where the 6-qubit quantum layer was the least powerful part of our entire model.
3. **Data and Parameter Scale:** The classical CNN had 500,490 parameters. The quantum part of our VQA model was limited by its 6-qubit design. A 6-qubit circuit, even with 3 layers of entanglement, simply cannot capture the same amount of complex patterns as a 500k-parameter classical network.

Reference:

Github Repo Link:

<https://github.com/deeptalon/Planckd2025-HackWizard.git>