Quantum Image Processing with PennyLane

Kavya

April 2025

1 Introduction

This document presents a quantum approach to image processing using the PennyLane framework. The workflow includes feature mapping, quantum encoding, and Fourier transformation.

2 Feature Mapping and Quantum Encoding

We process a set of image files and encode their features into quantum states using basis encoding.

2.1 Loading and Processing Image Data

```
import numpy as np
import pennylane as qml

npy_files = ["/content/npy_files/cat_10.npy", "/content/npy_files/cat_11.npy", ...]
num_qubits = 8  # Number of qubits for encoding

dev = qml.device("default.qubit", wires=num_qubits)

# Basis Encoding Function
def basis_encoding(image_features):
    binary_features = (image_features > 0.5).astype(int)

    @qml.qnode(dev)
    def circuit():
        for i in range(num_qubits):
            if binary_features[i]:
                qml.PauliX(wires=i)
        return qml.state()
    return circuit()

quantum_states = []
```

```
for file in npy_files:
    image_features = np.load(file)[:num_qubits]
    quantum_state = basis_encoding(image_features)
    quantum_states.append(quantum_state)

np.save("/content/quantum_encoded_images.npy", quantum_states)
```

3 Quantum Fourier Transform (QFT)

Quantum Fourier Transform (QFT) is applied to the encoded quantum states.

3.1 Implementation of QFT

```
def qft(wires):
    n = len(wires)
    for i in range(n):
        qml.Hadamard(wires=i)
        for j in range(i+1, n):
            qml.ControlledPhaseShift(np.pi / 2**(j - i), wires=[j, i])
    for i in range(n // 2):
        qml.SWAP(wires=[i, n - i - 1])

@qml.qnode(dev)
def apply_qft(state):
    qml.StatePrep(state, wires=range(num_qubits))
    qft(wires=range(num_qubits))
    return qml.state()

quantum_encoded_images = np.load("/content/quantum_encoded_images.npy")
qft_transformed_states = [apply_qft(state) for state in quantum_encoded_images]
np.save("/content/qft_transformed_images.npy", qft_transformed_states)
```

4 Quantum Circuit Visualization

Figure 1 shows the quantum circuit for image processing.

```
import matplotlib.pyplot as plt
fig, ax = qml.draw_mpl(circuit)(dummy_image)
plt.title("Quantum Circuit for Image Processing")
plt.savefig("/content/quantum_circuit.png")
```

5 Data Splitting for Training and Testing

The dataset is split into training and testing sets.

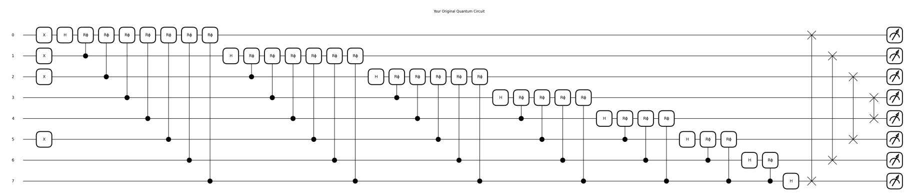

Figure 1: Quantum Circuit for Image Processing

```
from sklearn.model_selection import train_test_split

labels = [0 if 'cat' in file else 1 for file in npy_files]
data = [np.load(file) for file in npy_files]

train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=42, stratify=labels)

np.save("/content/train_data.npy", train_data)
np.save("/content/test_data.npy", test_data)
np.save("/content/train_labels.npy", train_labels)
np.save("/content/test_labels.npy", test_labels)
```

6 Conclusion

This document presents an end-to-end quantum image processing pipeline. It leverages quantum encoding and QFT to extract meaningful transformations.