Quantum Image Processing with PennyLane

Kavya

April 2025

1 Introduction

This document presents the implementation of quantum image processing using the PennyLane framework. The process includes loading images, encoding them into quantum states, applying Quantum Fourier Transform (QFT), and performing classical classification using K-Nearest Neighbors (KNN) and Logistic Regression.

2 Python Code

2.1 Importing Libraries

```
import numpy as np
import pennylane as qml
import cv2
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
from PIL import Image
from sklearn.linear_model import LogisticRegression
```

Listing 1: Importing required libraries

2.2 Loading and Preprocessing Images

```
# Define file paths for the images
image_paths = [
    "/content/cat_10.jpeg", "/content/cat_11.jpeg", "/content/dog_2
    .jpeg",
    "/content/dog_3.jpeg", "/content/dog_4.jpeg", "/content/dog_5.
    jpeg", "/content/dog_6.jpeg",
    "/content/cat_7.jpeg", "/content/cat_8.jpeg", "/content/cat_9.
    jpeg"
]

def load_and_preprocess_images(image_paths):
    images = []
```

```
    for image_path in image_paths:
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        image_resized = cv2.resize(image, (224, 224))
        image_normalized = image_resized / 255.0
        images.append(image_normalized)
    return np.array(images)

images = load_and_preprocess_images(image_paths)
```

Listing 2: Load and preprocess images

2.3 Quantum Basis Encoding

```
num_qubits = 8
dev = qml.device("default.qubit", wires=num_qubits)

def basis_encoding(image_features):
    binary_features = (image_features.flatten() > 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()
```

Listing 3: Encoding images into quantum states

2.4 Quantum Fourier Transform (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])
```

Listing 4: Applying QFT on quantum states

2.5 Classical Classification with KNN and Logistic Regression

```
true_labels = np.array([1,1, 0,0,0,0,0,1,1,1])
qft_flattened = np.abs(qft_transformed_states).reshape(len(qft_transformed_states), -1)
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(qft_flattened, true_labels)
predicted_labels = clf.predict(qft_flattened)

```

```
clf_logistic = LogisticRegression(max_iter=1000)
clf_logistic.fit(qft_flattened, true_labels)
predicted_logistic_labels = clf_logistic.predict(qft_flattened)
```

Listing 5: Training classical classifiers

3 Conclusion

This document presented an approach to quantum image processing using PennyLane, including quantum state preparation, QFT transformation, and classical classification. Future work may explore variational quantum classifiers and hybrid models for enhanced performance.