# Quantum Image Encoding with PennyLane

Rithvik Koruturu

Apr 2025

## 1 Introduction

This document explains a Python script for encoding grayscale pet images into quantum states using **Amplitude Embedding** in PennyLane. The dataset consists of images of cats and dogs, and each image is processed to a fixed size, normalized, padded, and finally mapped into a quantum state.

## 2 Importing Required Libraries

```
import pennylane as qml
from pennylane import numpy as np
import cv2
import os
import pickle
from tqdm import tqdm
```

- **PennyLane**: Used for quantum computing. - **NumPy (from PennyLane)**: Used for mathematical operations. - **OpenCV (cv2)**: Used for image preprocessing. - **OS**: Handles file path operations. - **Pickle**: Saves quantum-encoded images. - **tqdm**: Displays progress bars.

## 3 Dataset Configuration

```
dataset_path = "D:/datasets/PetImages"
categories = ["Cat", "Dog"]
num_images = 1000  # Encode only 1000 images per category
```

- The dataset is stored in `"D:/datasets/PetImages"`. - The script processes only **1000 images per category**.

## 4 Image Processing and Preprocessing

```
# Resize Parameters
image_size = (255, 255)
```

- Each image is resized to $255 \times 255$ pixels.

### 4.1 Finding the Nearest Power of 2

```
def nearest_power_of_2(n):
    return 2 ** int(np.ceil(np.log2(n)))
```

This function ensures that the image data length is compatible with quantum state representation by finding the smallest power of 2 greater than or equal to $n$.

$$2^m \geq N$$

where $m = \lceil \log_2(N) \rceil$.

## 5 Quantum Encoding Pipeline

### 5.1 Processing Images

```
quantum_encoded_images = []

for category in categories:
    folder_path = os.path.join(dataset_path, category)
    images = os.listdir(folder_path)[:num_images]

    for img_file in tqdm(images, desc=f"Encoding {category} images"):
        try:
            img_path = os.path.join(folder_path, img_file)
            image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            image = cv2.resize(image, image_size)
            image = image.flatten() / 255.0
```

- Each image is loaded, resized, flattened, and normalized to range $[0, 1]$.

### 5.2 Calculating the Required Number of Qubits

```
target_size = nearest_power_of_2(len(image))
num_qubits = int(np.log2(target_size))  # log2(65536) = 16
```

$$\text{num\_qubits} = \log_2(\text{target\_size})$$

### 5.3 Defining a Quantum Device

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

- Creates a quantum simulator with *num_qubits* wires.

### 5.4 Quantum Encoding Function

```
@qml.qnode(dev)
def quantum_image_encoding(image):
    qml.AmplitudeEmbedding(features=image, wires=range(num_qubits), normalize=Tr
    return qml.state()
```

- Uses **Amplitude Embedding** to encode pixel intensities into quantum states.
- The quantum state is given by:

$$\psi = \sum_{i=0}^{N-1} x_i i$$

where $x_i$ are pixel values mapped to quantum amplitudes.

### 5.5 Padding Image Data

```
padded_image = np.pad(image, (0, target_size - len(image)), 'constant')
```

- Pads the image vector to the nearest power of 2.

### 5.6 Encoding the Image into a Quantum State

```
quantum_state = quantum_image_encoding(padded_image)
quantum_encoded_images.append((category, img_file, quantum_state))
```

## 6 Saving the Encoded Data

```
with open("quantum_encoded_pet_images.pkl", "wb") as f:
    pickle.dump(quantum_encoded_images, f)

print("Encoding Successful!")
```

- Saves quantum-encoded images using `pickle`.

## 7 Conclusion

This script successfully converts pet images into quantum representations using amplitude embedding. The resulting quantum states can be further used for quantum classification or quantum-enhanced image analysis.