Quantum Feature Encoding using VGG16 and Amplitude Embedding

Koruturu Rithvik

April 2, 2025

1 Introduction

This document presents a hybrid approach for encoding image features into quantum states using **VGG16** for feature extraction and **Amplitude Encoding** in PennyLane. The primary goal is to leverage deep learning for feature extraction and quantum computing for efficient data representation.

2 Dataset and Preprocessing

We use the **PetImages** dataset, containing two categories: *Cats* and *Dogs*. The dataset is loaded and processed using OpenCV and PyTorch.

Dataset Path

```
dataset_path = "D:/datasets/PetImages"
categories = ["Cat", "Dog"]
```

To ensure compatibility with **VGG16**, images are resized to **224×224** pixels:

Listing 1: Image Preprocessing

```
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

3 Feature Extraction with VGG16

We use a **pretrained VGG16** model and remove the fully connected (FC) layers to extract convolutional features:

Listing 2: Loading Pretrained VGG16

```
vgg16 = models.vgg16(pretrained=True)
model = nn.Sequential(*list(vgg16.children())[:-1])  # Remove final classifier
model.eval()
```

Given an image of shape $(3, 224, 224)$, the extracted features are reshaped to a **25088-dimensional** vector:

$$\text{feature\_vector} = 512 \times 7 \times 7 = 25088$$

4 Quantum Encoding Using Amplitude Embedding

To encode extracted features into a quantum state, we use **Amplitude Embedding**, which maps a classical vector x to a quantum state:

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

Since quantum systems require power-of-two states, we find the nearest power of 2 for feature size 25088:

$$n = \lceil \log_2 25088 \rceil = 15 \tag{2}$$

Thus, we pad the feature vector to size $2^{15} = 32768$ before quantum embedding:

Listing 3: Quantum Feature Encoding

```
def amplitude_encoding(features):
    feature_len = len(features[0])
    num_qubits = int(np.ceil(np.log2(feature_len)))

    # Pad feature vector to match 2^num_qubits
    padded_len = 2 ** num_qubits
    padded_features = np.zeros((len(features), padded_len))
    padded_features[:, :feature_len] = features

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

    @qml.qnode(dev)
    def encode_feature_vector(feature_vector):
        qml.templates.AmplitudeEmbedding(feature_vector,
            wires=range(num_qubits), normalize=True)
        return [qml.expval(qml.PauliZ(i)) for i in range(num_qubits)]

    return np.array([encode_feature_vector(f) for f in padded_features])
```

5 Processing and Saving Encoded Data

We process and encode images for both categories:

Listing 4: Processing Dataset

```
encoded_data = {}
for category in categories:
    images = load_images(category)
    features = extract_features(images)
    qfeatures = amplitude_encoding(features)
    encoded_data[category] = qfeatures
```

Finally, we save the quantum-encoded data using **Pickle**:

Listing 5: Saving Encoded Data

```
with open("quantum_encoded_data.pkl", "wb") as f:
    pickle.dump(encoded_data, f)
```

6 Conclusion

This project successfully integrates deep learning with quantum encoding by:

- Extracting high-dimensional features using VGG16.
- Mapping these features into quantum states using Amplitude Embedding.
- Preparing the data for future quantum machine learning applications.