QUANTUM CONVOLUTIONAL NEURAL NETWORK

1. Loads the dependencies

In [ ]:
```
import cirq
import sympy
import collections
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow_quantum as tfq
from cirq.contrib.svg import SVGCircuit
```

2. Data preprocessing

2.1 Load the raw data

Load the MNIST dataset distributed with Keras.

In [ ]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

2.2 Normalizing

Rescale the images from [0,255] to the [0.0,1.0] range.

In [ ]:
```
x_train, x_test = x_train[..., np.newaxis]/255.0, x_test[..., np.newaxis]/255.0

print("Number of original training examples:", len(x_train))
print("Number of original test examples:", len(x_test))
```

2.3 Filtering

Filter the dataset to keep just the 3s and 6s, remove the other classes. At the same time convert the label, `y`, to boolean: `True` for `3` and `False` for 6.

In [ ]:
```
def filter_36(x, y):
    keep = (y == 3) | (y == 6)
    x, y = x[keep], y[keep]
    y = y == 3
    return x,y
```

In [ ]:
```
x_train, y_train = filter_36(x_train, y_train)
x_test, y_test = filter_36(x_test, y_test)

print("Number of filtered training examples:", len(x_train))
print("Number of filtered test examples:", len(x_test))
```

2.4 Resizing

An image size of 28x28 is much too large for current quantum computers. Resize the image down to 4x4:

In [ ]:
```
x_train_small = tf.image.resize(x_train, (4,4)).numpy()
x_test_small = tf.image.resize(x_test, (4,4)).numpy()
```

3. Quantum neural network

3.1 Encode the data as quantum circuits

To process images using a quantum computer, proposed representing each pixel with a qubit, with the state depending on the value of the pixel. The first step is to convert to a binary encoding.proposed

In [ ]:
```
THRESHOLD = 0.5

x_train_bin = np.array(x_train_small > THRESHOLD, dtype=np.float32)
x_test_bin = np.array(x_test_small > THRESHOLD, dtype=np.float32)
```

3.2 Convert the quantum circuits to tensors

In [ ]:
```
def convert_to_circuit(image):
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
    circuit = cirq.Circuit()
    for i, value in enumerate(values):
        if value:
            circuit.append(cirq.X(qubits[i]))
    return circuit
```

In [ ]:
```
x_train_circ = [convert_to_circuit(x) for x in x_train_bin]
x_test_circ = [convert_to_circuit(x) for x in x_test_bin]
```

In [ ]:
```
x_train_tfcirc = tfq.convert_to_tensor(x_train_circ)
x_test_tfcirc = tfq.convert_to_tensor(x_test_circ)
```

3.3 Build the model

In [ ]:
```
class CircuitLayerBuilder():
    def __init__(self, data_qubits, readout):
        self.data_qubits = data_qubits
        self.readout = readout

    def add_layer(self, circuit, gate, prefix):
        for i, qubit in enumerate(self.data_qubits):
            symbol = sympy.Symbol(prefix + '-' + str(i))
            circuit.append(gate(qubit, self.readout)**symbol)
```

In [ ]:
```
def create_quantum_model():
    data_qubits = cirq.GridQubit.rect(4, 4)
    readout = cirq.GridQubit(-1, -1)
    circuit = cirq.Circuit()
    circuit.append(cirq.X(readout))
    circuit.append(cirq.H(readout))
    builder = CircuitLayerBuilder(data_qubits = data_qubits,readout=readout)
    builder.add_layer(circuit, cirq.XX, "xx1")
    builder.add_layer(circuit, cirq.ZZ, "zz1")
    circuit.append(cirq.H(readout))

    return circuit, cirq.Z(readout)
```

In [ ]:
```
model_circuit, model_readout = create_quantum_model()
```

In [ ]:
```
model = tf.keras.Sequential([tf.keras.layers.Input(shape=(), dtype=tf.string),tfq.layers.PQC(model_circuit, model_readout),])
```

3.4 Compile the model

In [ ]:
```
y_train_hinge = 2.0*y_train-1.0
y_test_hinge = 2.0*y_test-1.0
```

In [ ]:
```
def hinge_accuracy(y_true, y_pred):
    y_true = tf.squeeze(y_true) > 0.0
    y_pred = tf.squeeze(y_pred) > 0.0
    result = tf.cast(y_true == y_pred, tf.float32)

    return tf.reduce_mean(result)
```

In [ ]:
```
model.compile(loss=tf.keras.losses.Hinge(),optimizer=tf.keras.optimizers.Adam(),metrics=[hinge_accuracy])
```

3.5 Train the model

Control panel

In [ ]:
```
EPOCHS = 2
NUM_EXAMPLES =  100 #len(x_train_tfcirc)
x_train_tfcirc_sub = x_train_tfcirc[:NUM_EXAMPLES]
y_train_hinge_sub = y_train_hinge[:NUM_EXAMPLES]
```

In [ ]:
```
qnn_history = model.fit(x_train_tfcirc_sub, y_train_hinge_sub,batch_size=32,epochs=EPOCHS,verbose=1,validation_data=(x_test_tfcirc, y_test_hinge))
```

3.6 Evaluate the model

In [ ]:
```
qnn_results = model.evaluate(x_test_tfcirc, y_test)
```

4. Classical neural network

4.1 Build the model

In [ ]:
```
def create_classical_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Conv2D(32, [3, 3], activation='relu', input_shape=(28,28,1)))
    model.add(tf.keras.layers.Conv2D(64, [3, 3], activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.25))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(128, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.5))
    model.add(tf.keras.layers.Dense(1))
    return model


model = create_classical_model()
```

4.2 Compile the model

In [ ]:
```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])
```

4.3 Train the model

In [ ]:
```
cnn_history = model.fit(x_train,y_train,batch_size=128,epochs=EPOCHS, verbose=1, validation_data=(x_test, y_test))
```

4.4 Evaluate the model

In [ ]:
```
cnn_results = model.evaluate(x_test, y_test)
```

5. Comparison

5.1 Quantum CNN performance

In [ ]:
```
qnn_history.history
qnn = pd.DataFrame()
qnn['accuracy']  = qnn_history.history['hinge_accuracy']
qnn['loss']      = qnn_history.history['loss']
qnn['val_accuracy'] = qnn_history.history['val_hinge_accuracy']
qnn['val_loss']  = qnn_history.history['val_loss']
print(" Quantum CNN DataFrame" )
print(qnn.head())
values = ['accuracy','loss']
for x in values:
    fig = plt.figure(figsize=(10,5))
    plt.subplot(1, 2, 1)
    plt.plot(qnn[x],'bo--', label = x )
    plt.plot(qnn['val_'+x], 'ro--', label ='val_'+x)
    plt.title("train_"+x + " vs val_"+x)
    plt.ylabel(x)
    plt.xlabel("epochs")
    plt.legend()
    plt.show()
```

5.2 Classical CNN performance

In [ ]:
```
cnn_history.history
cnn = pd.DataFrame()
cnn['accuracy']  = cnn_history.history['accuracy']
cnn['loss']      = cnn_history.history['loss']
cnn['val_accuracy'] = cnn_history.history['val_accuracy']
cnn['val_loss']  = cnn_history.history['val_loss']
print("Classical CNN DataFrame")
print(cnn)
values = ['accuracy','loss']
for x in values:
    fig = plt.figure(figsize=(10,5))
    plt.subplot(1, 2, 1)
    plt.plot(cnn[x],'bo--', label = x )
    plt.plot(cnn['val_'+x], 'ro--', label ='val_'+x)
    plt.title("train_"+x + " vs val_"+x)
    plt.ylabel(x)
    plt.xlabel("epochs")
    plt.legend()
    plt.show()
```

5.3 Quantum Vs Classical using Barplot

In [ ]:
```
qnn_accuracy = qnn_results[1]
cnn_accuracy = cnn_results[1]

sns.barplot(["Quantum", "Classical"],[qnn_accuracy, cnn_accuracy])
```