Quantum Neural Networks

Eniola Sobimpe
University of Dallas, Irving TX

Abstract

This paper outlines the investigation, progression and perspectives of quantum neural networks – a fi
arranges classical neurocomputing with quantum computation. It is fought that the examination of qu
give us both new cognizance of brain work similarly as marvelous possibilities in making new structu
including dealing with classically ardent problems. At the end of this paper, I will be implementing qua
partial classical neural networks, and classical neural networks (CNN) to train simple mnist classifica
performs.

Learning Digit Recognition

Classical neural systems can order transcribed digits. The model originates from the MNIST informat
55,000 preparing tests that are 28 by 28 pixilated pictures of written by hand digits that have been ma
one of the ten digits from 0 to 9. Numerous early on classes in AI utilize this informational collection a
basic neural systems. So it appears to be normal for us to check whether quantum neural system can
information. There is no undeniable method to assault this logically so I resort to recreation. The impe
effectively deal with state 16 piece information utilizing a traditional test system of a 17 qubit quantu
utilize a downsampled form of the MNIST information which comprises of 4 by 4 pixilated pictures. W
ten digits so all things being equal I pick two digits, state 7 and 9, and lessen the informational collect
examples named as 7 or 9 and inquire as to whether the quantum system can recognize the informat
tests break into gatherings of approximately 5,500 examples for every digit. Yet, upon closer assessm
relating to state the digit 7, comprise of 797 unmistakable 16 piece strings while for the digit 9 there a
strings. The pictures are hazy and in actuality there are 197 unmistakable strings that are marked as b
qualification task I chose to lessen the Bayes mistake to 0 by evacuating the questionable strings. Re
for every digit and evacuating vague strings, leaves 3514 examples that are marked as 7's and 2517 t
these to make a preparation set of 6031 examples As a fundamental advance I present the named tes
Here I run a tensorflow classifier with one interior layer comprising of 10 neurons. Every neuron has 1
inclination weight so there are 170 parameters on the interior layer and 4 on the yield layer. The old st
difficulty discovering loads that give short of what one percent arrangement mistake on the preparatio
additionally takes a gander at the speculation mistake however to do so it picks an arbitrary 15 percen
for a test set. Since the information collection has rehashed events of a similar 16 piece strings, the te
concealed models. Still the speculation blunder is short of what one percent.

I presently go to the quantum classifier. Here I have little direction with respect to how to plan the qua
toolbox of unitaries to comprise of one and two qubit administrators of the structure. I take the one q
following up on any of the 17 qubits. For the two qubit unitaries I take Σ to be XY, Y Z, ZX, XX, Y and ZZ

qubits. The main thing I attempted was an arbitrary choice of 500 (or 1000) of these unitaries. The irr
9 door types are picked just as to which qubits the entryways are applied to. Beginning with an irregul
1000) edges, subsequent to introducing a couple hundred preparing tests, the all out blunder settled i
However, the example misfortune for singular strings was commonly just a piece beneath 1 which rel
likelihood of a little more than 50 percent for most stings. Here the pattern was the correct way yet I v

After some playing around I took a stab at confining my entryway set to ZX and XX with the second qu
readout qubit and the first qubit being one of the other 16. The inspiration here is that the related unit
qubit around the x course by a sum constrained by the information qubits. A full layer of ZX has 16 pa
XX. I attempted a variation of 3 layers of ZX with 3 layers of XX for a sum of 96 parameters. Here I fou
arbitrary arrangement of points I could accomplish two percent straight out mistake in the wake of se
example set. The achievement here is that I showed that a quantum neural system could figure out ho
information. In fact the informational index could without much of a stretch be grouped by an old styl
working at a fixed low number of bits blocks any conversation of scaling. Yet, my work is exploratory
have a quantum circuit that can arrange certifiable information. Presently the assignment is to refine
it performs better. Ideally I can discover a few standards (or just motivation) that manages the decisio

Implementation

```
import tensorflow as tf
import tensorflow_quantum as tfq

import cirq
import sympy
import numpy as np
import seaborn as sns
import collections

# visualization tools
%matplotlib inline
import matplotlib.pyplot as plt
from cirq.contrib.svg import SVGCircuit
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Rescale the images from [0,255] to the [0.0,1.0] range.
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_train))
```

```
Number of original training examples: 60000
Number of original test examples: 60000
```

Filter the dataset to keep just the 3s and 6s, remove the other classes. At the same time convert the l
and `False` for 9.

```
def filter_79(x, y):
    keep = (y == 7) | (y == 9)
    x, y = x[keep], y[keep]
    y = y == 7
    return x,y
```

```
x_train, y_train = filter_79(x_train, y_train)
x_test, y_test = filter_79(x_test, y_test)

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

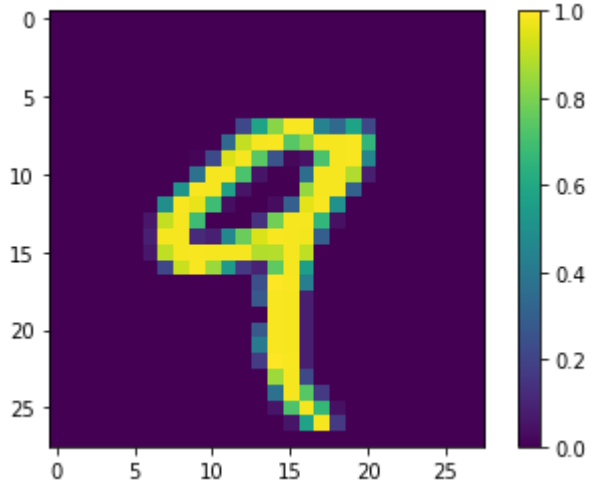
```
Number of filtered training examples: 12214
Number of filtered test examples: 2037
```

Show the first example:

```
print(y_train[0])

plt.imshow(x_train[0, :, :, 0])
plt.colorbar()
```

```
False
<matplotlib.colorbar.Colorbar at 0x7f6db7ab7390>
```



1.2 Downscale the images

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

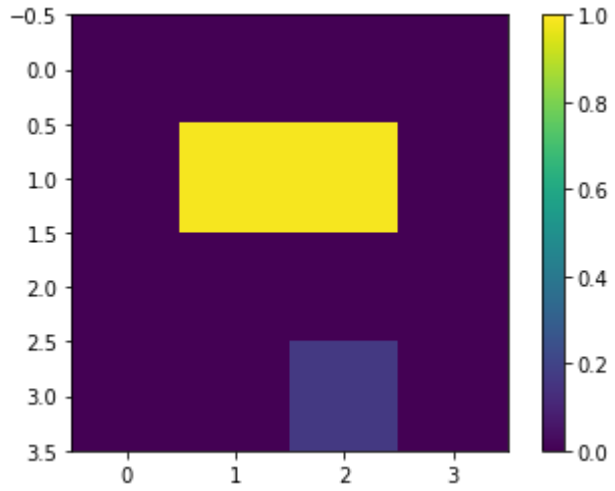
```
x_train_small = tf.image.resize(x_train, (4,4)).numpy()
x_test_small = tf.image.resize(x_test, (4,4)).numpy()
```

Again, display the first training example—after resize:

```
print(y_train[0])

plt.imshow(x_train_small[0,:,:,0], vmin=0, vmax=1)
plt.colorbar()
```

```
False
<matplotlib.colorbar.Colorbar at 0x7f6d5c131208>
```



1.3 Remove contradictory examples

Filter the dataset to remove images that are labeled as belonging to both classes.

```
def remove_contradicting(xs, ys):
    mapping = collections.defaultdict(set)
    # Determine the set of labels for each unique image:
    for x,y in zip(xs,ys):
       mapping[tuple(x.flatten())].add(y)

    new_x = []
    new_y = []
    for x,y in zip(xs, ys):
      labels = mapping[tuple(x.flatten())]
      if len(labels) == 1:
          new_x.append(x)
          new_y.append(list(labels)[0])
      else:
```

```
            # Throw out images that match more than one label.
            pass

    num_7 = sum(1 for value in mapping.values() if True in value)
    num_9 = sum(1 for value in mapping.values() if False in value)
    num_both = sum(1 for value in mapping.values() if len(value) == 2)

    print("Number of unique images:", len(mapping.values()))
    print("Number of 7s: ", num_7)
    print("Number of 9s: ", num_9)
    print("Number of contradictory images: ", num_both)
    print()
    print("Initial number of examples: ", len(xs))
    print("Remaining non-contradictory examples: ", len(new_x))

    return np.array(new_x), np.array(new_y)
```

The resulting counts do not closely match the reported values, but the exact procedure is not specifie

It is also worth noting here that applying filtering contradictory examples at this point does not totally receiving contradictory training examples: the next step binarizes the data which will cause more colli

```
x_train_nocon, y_train_nocon = remove_contradicting(x_train_small, y_train)
```

```
Number of unique images: 11077
Number of 7s:  5585
Number of 9s:  5659
Number of contradictory images:  167

Initial number of examples:  12214
Remaining non-contradictory examples:  11210
```

1.3 Encode the data as quantum circuits

To process images using a quantum computer, Farhi et al. proposed representing each pixel with a qu on the value of the pixel. The first step is to convert to a binary encoding.

```
THRESHOLD = 0.5

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

The qubits at pixel indices with values that exceed a threshold, are rotated through an X gate.

```
def convert_to_circuit(image):
    """Encode truncated classical image into quantum datapoint."""
    values = np.ndarray.flatten(image)
    qubits = cirq.GridQubit.rect(4, 4)
```

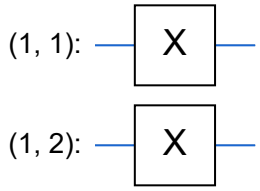
```
    circuit = cirq.Circuit()
    for i, value in enumerate(values):
        if value:
            circuit.append(cirq.X(qubits[i]))
    return circuit


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]
```

Here is the circuit created for the first example (circuit diagrams do not show qubits with zero gates):

```
SVGCircuit(x_train_circ[0])
```

```
findfont: Font family ['Arial'] not found. Falling back to DejaVu Sans.
```



Compare this circuit to the indices where the image value exceeds the threshold:

```
bin_img = x_train_bin[0,:,:,0]
indices = np.array(np.where(bin_img)).T
indices
```

```
array([[1, 1],
       [1, 2]])
```

Convert these `Cirq` circuits to tensors for `tfq`:

```
x_train_tfcirc = tfq.convert_to_tensor(x_train_circ)
x_test_tfcirc = tfq.convert_to_tensor(x_test_circ)
```

2. Quantum neural network

Since the classification is based on the expectation of the readout qubit, Farhi et al. propose using tw qubit always acted upon.

2.1 Build the model circuit

This following example shows this layered approach. Each layer uses *n* instances of the same gate, w acting on the readout qubit.

Start with a simple class that will add a layer of these gates to a circuit:

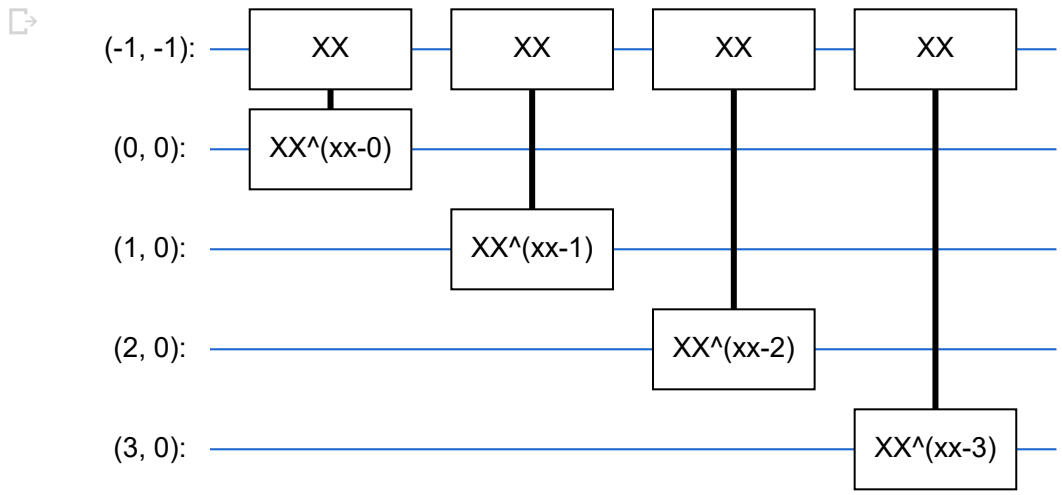
```
class CircuitLayerBuilder():
    def __init__(self, data_qubits, readout):
        self.data_qubits = data_qubits
        self.readout = readout

    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)
```

Build an example circuit layer to see how it looks:

```
demo_builder = CircuitLayerBuilder(data_qubits = cirq.GridQubit.rect(4,1),
                                   readout=cirq.GridQubit(-1,-1))

circuit = cirq.Circuit()
demo_builder.add_layer(circuit, gate = cirq.XX, prefix='xx')
SVGCircuit(circuit)
```



Now build a two-layered model, matching the data-circuit size, and include the preparation and readou

```
def create_quantum_model():
    """Create a QNN model circuit and readout operation to go along with it."""
    data_qubits = cirq.GridQubit.rect(4, 4)  # a 4x4 grid.
    readout = cirq.GridQubit(-1, -1)         # a single qubit at [-1,-1]
    circuit = cirq.Circuit()

    # Prepare the readout qubit.
    circuit.append(cirq.X(readout))
    circuit.append(cirq.H(readout))

    builder = CircuitLayerBuilder(
        data_qubits = data_qubits,
```

```
        readout=readout)

    # Then add layers
    builder.add_layer(circuit, cirq.XX, "xx1")
    builder.add_layer(circuit, cirq.ZZ, "zz1")

    # Finally, prepare the readout qubit.
    circuit.append(cirq.H(readout))

    return circuit, cirq.Z(readout)


model_circuit, model_readout = create_quantum_model()
```

2.2 Wrap the model-circuit in a tfq-keras model

Build the Keras model with the quantum components. This model is fed the "quantum data", from `x_t` classical data. It uses a *Parametrized Quantum Circuit* layer, `tfq.layers.PQC`, to train the model circui

To classify these images, Farhi et al. proposed taking the expectation of a readout qubit in a paramet returns a value between 1 and -1.

```
# Build the Keras model.
model = tf.keras.Sequential([
    # The input is the data-circuit, encoded as a tf.string
    tf.keras.layers.Input(shape=(), dtype=tf.string),
    # The PQC layer returns the expected value of the readout gate, range [-1,1].
    tfq.layers.PQC(model_circuit, model_readout),
])
```

Next, describe the training procedure to the model, using the `compile` method.

Since the the expected readout is in the range `[-1,1]`, optimizing the hinge loss is a somewhat natur

```
y_train_hinge = 2.0*y_train-1.0
y_test_hinge = 2.0*y_test-1.0
```

Second, use a custiom `hinge_accuracy` metric that correctly handles `[-1, 1]` as the `y_true` labels a `tf.losses.BinaryAccuracy(threshold=0.0)` expects `y_true` to be a boolean, and so can't be used w

```
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)
```

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


print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
pqc (PQC)                    (None, 1)                 32        
=================================================================
Total params: 32
Trainable params: 32
Non-trainable params: 0
_________________________________________________________________
None
```

Train the quantum model

Using fewer examples just ends training earlier (5min), but runs long enough to show that it is making

```
EPOCHS = 3
BATCH_SIZE = 32

NUM_EXAMPLES = 500 #len(x_train_tfcirc)


x_train_tfcirc_sub = x_train_tfcirc[:NUM_EXAMPLES]
y_train_hinge_sub = y_train_hinge[:NUM_EXAMPLES]
```

Training this model to convergence should achieve >85% accuracy on the test set.

```
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))

qnn_results = model.evaluate(x_test_tfcirc, y_test)
```

```
Train on 500 samples, validate on 2037 samples
Epoch 1/3
500/500 [==============================] - 230s 460ms/sample - loss: 1.0001 - hinge_accu
Epoch 2/3
500/500 [==============================] - 228s 455ms/sample - loss: 1.0000 - hinge_accu
Epoch 3/3
500/500 [==============================] - 227s 455ms/sample - loss: 1.0000 - hinge_accu
2037/2037 [==============================] - 26s 13ms/sample - loss: 1.0010 - hinge_accu
```

Note: The training accuracy reports the average over the epoch. The validation accuracy is evaluated

3. Classical neural network

While the quantum neural network works for this simplified MNIST problem, a basic classical neural n
QNN on this task. After a single epoch, a classical neural network can achieve >98% accuracy on the h

In the following example, a classical neural network is used for for the 3-6 classification problem usin
instead of subsampling the image. This easily converges to nearly 100% accuracy of the test set.

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

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1179776
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,198,721
Trainable params: 1,198,721
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_train,
          y_train,
          batch_size=128,
          epochs=1,
          verbose=1,
          validation_data=(x_test, y_test))

cnn_results = model.evaluate(x_test, y_test)
```

```
Train on 12214 samples, validate on 2037 samples
12214/12214 [==============================] - 7s 540us/sample - loss: 0.1211 - accuracy
2037/2037 [==============================] - 0s 92us/sample - loss: 0.0297 - accuracy: 0
```

The above model has nearly 1.2M parameters. For a more fair comparison, try a 37-parameter model,

```
def create_fair_classical_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(4,4,1)))
    model.add(tf.keras.layers.Dense(2, activation='relu'))
    model.add(tf.keras.layers.Dense(1))
    return model


model = create_fair_classical_model()
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
```

```
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 16)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 34        
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 3         
=================================================================
Total params: 37
Trainable params: 37
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_train_bin,
          y_train_nocon,
          batch_size=128,
          epochs=20,
          verbose=2,
          validation_data=(x_test_bin, y_test))

fair_nn_results = model.evaluate(x_test_bin, y_test)
```

```
Train on 11210 samples, validate on 2037 samples
Epoch 1/20
11210/11210 - 1s - loss: 0.6685 - accuracy: 0.4980 - val_loss: 0.6565 - val_accuracy: 0.
Epoch 2/20
11210/11210 - 0s - loss: 0.6536 - accuracy: 0.4980 - val_loss: 0.6431 - val_accuracy: 0.
Epoch 3/20
11210/11210 - 0s - loss: 0.6433 - accuracy: 0.4981 - val_loss: 0.6323 - val_accuracy: 0.
Epoch 4/20
11210/11210 - 0s - loss: 0.6338 - accuracy: 0.5027 - val_loss: 0.6216 - val_accuracy: 0.
Epoch 5/20
11210/11210 - 0s - loss: 0.6245 - accuracy: 0.5189 - val_loss: 0.6124 - val_accuracy: 0.
Epoch 6/20
11210/11210 - 0s - loss: 0.6161 - accuracy: 0.5502 - val_loss: 0.6039 - val_accuracy: 0.
Epoch 7/20
11210/11210 - 0s - loss: 0.6099 - accuracy: 0.5905 - val_loss: 0.5977 - val_accuracy: 0.
Epoch 8/20
11210/11210 - 0s - loss: 0.6056 - accuracy: 0.6085 - val_loss: 0.5922 - val_accuracy: 0.
Epoch 9/20
11210/11210 - 0s - loss: 0.6016 - accuracy: 0.6291 - val_loss: 0.5880 - val_accuracy: 0.
Epoch 10/20
11210/11210 - 0s - loss: 0.5994 - accuracy: 0.6335 - val_loss: 0.5863 - val_accuracy: 0.
Epoch 11/20
11210/11210 - 0s - loss: 0.5983 - accuracy: 0.6336 - val_loss: 0.5851 - val_accuracy: 0.
Epoch 12/20
11210/11210 - 0s - loss: 0.5974 - accuracy: 0.6384 - val_loss: 0.5845 - val_accuracy: 0.
Epoch 13/20
11210/11210 - 0s - loss: 0.5966 - accuracy: 0.6402 - val_loss: 0.5836 - val_accuracy: 0.
Epoch 14/20
11210/11210 - 0s - loss: 0.5962 - accuracy: 0.6402 - val_loss: 0.5830 - val_accuracy: 0.
Epoch 15/20
11210/11210 - 0s - loss: 0.5956 - accuracy: 0.6403 - val_loss: 0.5825 - val_accuracy: 0.
Epoch 16/20
11210/11210 - 0s - loss: 0.5951 - accuracy: 0.6403 - val_loss: 0.5822 - val_accuracy: 0.
Epoch 17/20
11210/11210 - 0s - loss: 0.5948 - accuracy: 0.6403 - val_loss: 0.5820 - val_accuracy: 0.
Epoch 18/20
11210/11210 - 0s - loss: 0.5944 - accuracy: 0.6403 - val_loss: 0.5816 - val_accuracy: 0.
Epoch 19/20
11210/11210 - 0s - loss: 0.5941 - accuracy: 0.6417 - val_loss: 0.5814 - val_accuracy: 0.
Epoch 20/20
11210/11210 - 0s - loss: 0.5938 - accuracy: 0.6433 - val_loss: 0.5811 - val_accuracy: 0.
2037/2037 [==============================] - 0s 53us/sample - loss: 0.5811 - accuracy: 0
```

4. Comparison

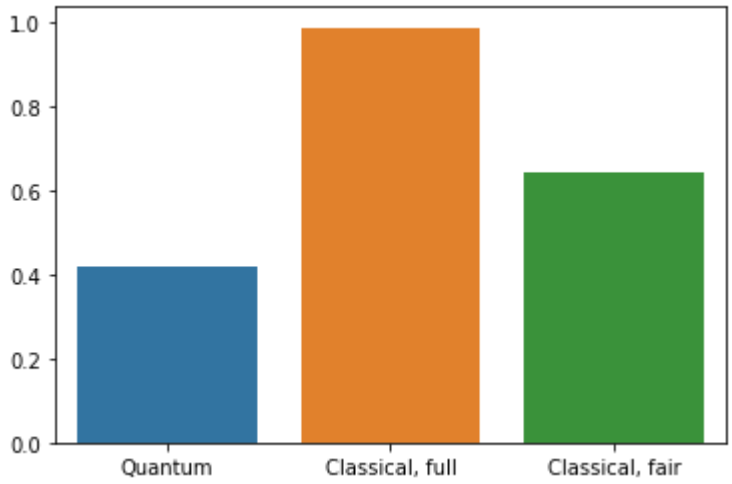
Higher resolution input and a more powerful model make this problem easy for the CNN. While a clas (~32 parameters) trains to a similar accuracy in a fraction of the time. One way or the other, the class outperforms the quantum neural network. For classical data, it is difficult to beat a classical neural ne

```
qnn_accuracy = qnn_results[1]
cnn_accuracy = cnn_results[1]
fair_nn_accuracy = fair_nn_results[1]
```

```
fair_nn_accuracy = fair_nn_results[1]

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6d54676fd0>
```



Reference

Farhi et al.