Team Capybaras: Quantum Edge Detection

Timothy Gao, Cassiel Graullera, Tina Wang, Noah Cooney

MIT BWSI

Contents

1 Author Notes

This is a hybrid algorithm, which means it includes both Quantum and classical parts. The Quantum Edge Detection offer an exponential speed up over classical edge detection, and the time complexity is discussed. However, the classical pre and post processing of the image does not offer a speed-up (every pixel must be visited at least once), so the time complexity is not discussed.

A lot of this information can be found on the markdown cells of Main.ipynb. Also, this work is inspired by the literature `https://arxiv.org/pdf/2012.11036.pdf`, but many were made to tackle different challenges during implementation (e.g., coming up with adaptive thresholding techniques).

To try the algorithm on a custom image, you can use Main_Pipeline.py, which is outlined below:

1. Classical Image Pre-processing
    - Strip colored image down to one color

      OR
    - Convert to Gray Scale, via $0.2989 \cdot R + 0.5870 \cdot B + 0.1140 \cdot G$
    - For larger images, image gridding and solve for each grid
    - (Optional) Apply additional image filters
2. Quantum Probability Image Encoding (QPIE) (2)
3. QHED (3)
4. Classical Post-Processing (4)
    - For larger images, combining grids

For full implementation details, visit the individual Python scripts and notebooks. A lot of it is not discussed in this paoer.

2 Quantum Probability Image Encoding (QPIE)

2.1 How QPIE Represents Classical Images

After classical image filtering, each element in the image grid now represents the color intensity in that cell. Let $c = grid[i][j]$, and index of c as $i * N + j$. So the indices would look like this for a 3x3:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

In QPIE representation, the image becomes

$$|\text{Img}\rangle = \sum_{i=0}^{2^n-1} c_i \, |i\rangle$$

or

$$|\text{Img}\rangle = \sum_{i=0}^{2^n-1} grid[\frac{i}{N}][i \mod N] \, |i\rangle$$

(n is number of qubits)

This means that we need $n = \lceil \log_2 N^2 \rceil$ qubits in order to index all N^2 pixels in our image. Since we are encoding color intensity into a quantum state, $\sum c_i^2 = 1$ s all probabilities must add up to one.

2.2 Amplitude Encoding

To accomplish this while maintaining relative differences, we set $c_i = \frac{c_i}{\sqrt{\sum c_i^2}}$ (Divide by Root mean squared) so now

$$\sum (\frac{c_i}{\sqrt{\sum c_i^2}})^2 = \frac{1}{\sum c_i^2} \cdot \sum c_i^2 = 1$$

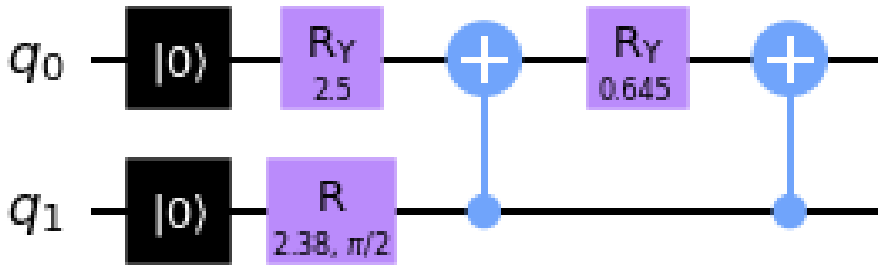
, as desired

Note that to actually obtain this, we need to perform $\mathcal{O}(N^2)$, but since it's only part of the classical pre-processing, we don't include it when calculating time complexity for the quantum part.

2.3 Creating the Quantum Image

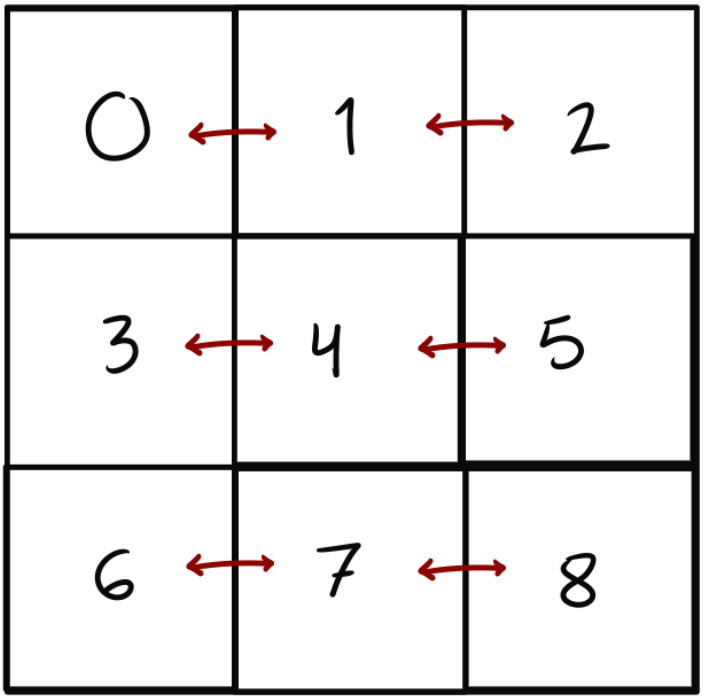
After amplitude encoding, we can use a combination of CNOT and Arbitrary Rotation Gates to achieve the desired amplitudes (c) for all quantum states states.

For example, to encode (0, 128, 192, 255):



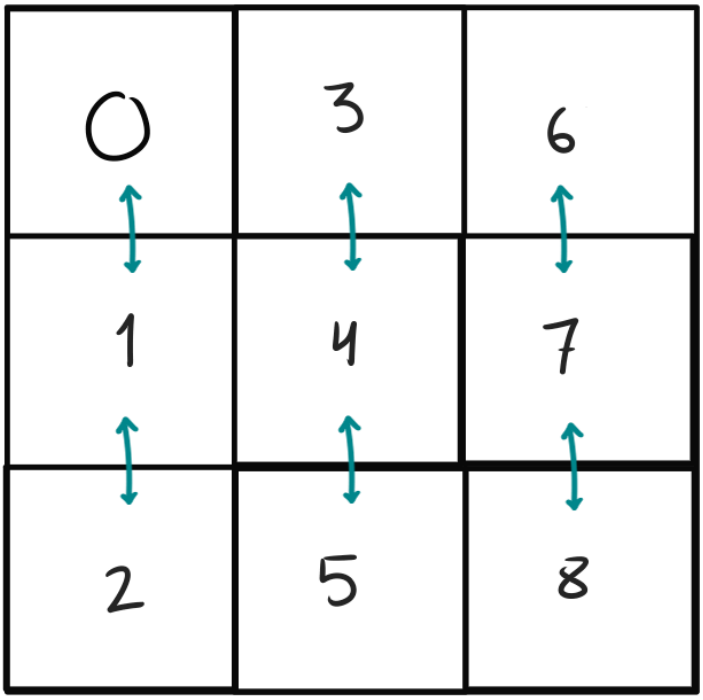
Quiskit has a nice implementation of this algorithm for us (here), which we can call through initialize such circuits through a .initialize() call on the QuantumCircuit.

In QPIE, we're indexing the grid as $|\text{Img}\rangle = \sum_{i=0}^{2^n-1} grid[\lfloor \frac{i}{N} \rfloor][i \mod N] \, |i\rangle$. Therefore, if we obtain differences between adjacent indicies, it will only obtain horizontal differences:



(the red indicates the differences)

In order to obtain all differences for the entire 2D grid, one solution is to simply take the transpose the image, then apply the entire algorithm to both images.



When we combine both images together at the end we will now have adjacent differences both vertically and horizontally.

Time complexity: $\mathcal{O}(n) = \mathcal{O}(log(N^2))$ for initializing amplitudes on $n = \lceil \log_2 N^2 \rceil$ number of qubits.

3 QHED

3.1 Hadamard Magic

The first is a simple Hadamard Gate on qubit 0, which is highlighted in red below:

Index 0: 000
Index 1: 001
Index 2: 010
Index 3: 011
Index 4: 100

Index 5: 101

... (Example with 3 qubits)

Notice how it alternates, which makes sense when we consider $\mod 2$. This property works neatly with a Hadamard Gate. Recall the definition of a Hamadard Gate:

$$|0\rangle \to \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}$$

$$|1\rangle \to \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$$

So let's consider applying the Hamadard Gate on the 0th qubit
Performing this action (H_0) on Index 0 (Let's donate this as $I_{2^n-1}[0]$) yields:

$$\frac{1}{\sqrt{2}} c_0|000\rangle + c_0|001\rangle)$$

H_0 on $I_{2^n-1}[1]$ yields:

$$\frac{1}{\sqrt{2}} c_0|000\rangle - c_0|001\rangle)$$

H_0 on $I_{2^n-1}[2]$ yields:

$$\frac{1}{\sqrt{2}} c_0|010\rangle + c_0|010\rangle)$$

so on... The signs will alternate, because the 0th qubit alternates.
Formally, we are performing $I_{2^{n-1}} \otimes H_0$.
The Hadamard Gate H is defined as

$$\tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\downarrow$$

$$I_{2^{n-1}} \otimes H_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix}$$

Applying this unitary to the QPIE representation $|\text{Img}\rangle = \sum_{i=0}^{N-1} c_i \, |i\rangle$

$$(I_{2^{n-1}} \otimes H_0) \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N-2} \\ c_{N-1} \end{bmatrix} \rightarrow \frac{1}{\sqrt{2}} \begin{bmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_2 + c_3 \\ c_2 - c_3 \\ \vdots \\ c_{N-2} + c_{N-1} \\ c_{N-2} - c_{N-1} \end{bmatrix}$$

This matrix gives us access to the gradient between the pixel intensities of neighboring pixels c_i and c_{i+1}, through $(c_i - c_{i+1})$.

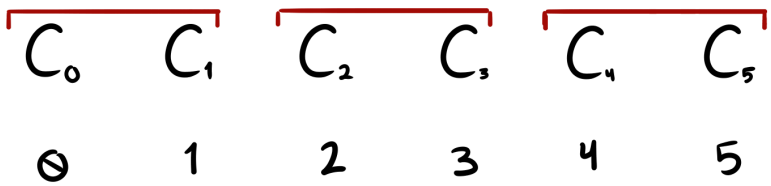
Intuitively, this should also make sense, again considering the alternating 0th qubit between 0/1 and how H changes the sign accordingly.

This Hadamard gate is applied in $\mathcal{O}(1)$ on a quantum computer, allowing us the compute adjacent-cell-differences very quickly, while on a classical computer the same algorithm takes time proportional to the number of pixels, which is N^2 or 2^n.

Time Complexity: $\mathcal{O}(1)$

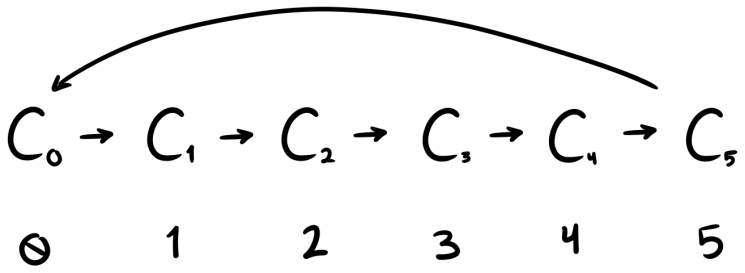
3.2 Amplitude Permutation

However, this only obtains the horizontal pixel differences for even-odd pixel pairs, if we examine the matrix closely.

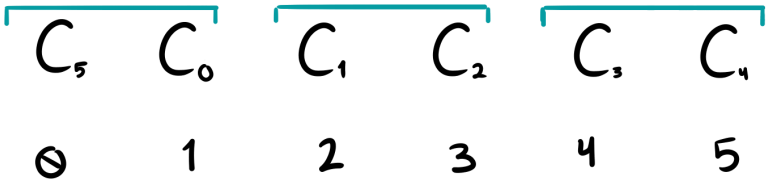


Highlighted in red below for n = 3 example

One way to account for this is by performing an amplitude permutation, shifting the entire register right and wrapping around the rightmost element:



Amplitude permutation



Result after permutating

This gives us access to also the odd-even pixel pairs now, so we now have all adjacent pixel differences. In post-processing, we can take care not to include irrelevant differences like $(c_5 - c_0)$.

We can easily perform this permutation operation with the unitary

$$D_{2^{n+1}} = \begin{bmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

In practice, we don't actually rotate. Instead, this unitary just "extends", as follows:

$$(c_0, c_0, c_1, c_1, c_2, c_2, \dots, c_{N-2}, c_{N-2}, c_{N-1}, c_{N-1})^T \rightarrow (c_0, c_1, c_1, c_2, c_2, c_3, \dots, c_{N-2}, c_{N-1}, c_{N-1}, c_0)^T$$

This is why we need an extra ancillary qubit.

If only we could apply this transformation directly to our quantum state...

Well, it turns out we can! This unitary actually correponds to the Decrement gate. Hence, we can efficently decompose this unitary into a set of single- and multi-controlled-X rotations on a register of multiple qubits as shown by Fijany and Williams and Gidney.

In Qiskit, we can use this function to directly apply the matrix transformation to our quantum state, with .unitary().

So after utilizing the unitary:

$$(I_{2^n} \otimes H) \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_1 \\ c_2 \\ c_2 \\ c_3 \\ \vdots \\ c_{N-2} \\ c_{N-1} \\ c_{N-1} \\ c_0 \end{bmatrix} \rightarrow \begin{bmatrix} c_0 + c_1 \\ c_0 - c_1 \\ c_1 + c_2 \\ c_1 - c_2 \\ c_2 + c_3 \\ c_2 - c_3 \\ \vdots \\ c_{N-2} + c_{N-1} \\ c_{N-2} - c_{N-1} \\ c_{N-1} + c_0 \\ c_{N-1} - c_0 \end{bmatrix}$$

Now, we have access to the resultant horizontal gradient values $c_i - c_{i+1}$ for all possible pairs of adjacent qubits.

Time Complexity: $\mathcal{O}(n)$ from applying the unitary.

4 Classical Post Processing

4.1 Getting the Image Back

To use our results from QHED, we mark notable edges by "thresholding", i.e., marking cells that have significant adjacent differences as edges.

We store the marked/unmarked cells as a binary value (1 for marked, 0 for unmarked).

We use $2 \cdot i + 1$ to select only indices where it's subtraction between adjacent cells (which we care about) and not the addition.

Also, we take care not to include indices where $i + 1 \mod N = 0$ because that is where $c_{N-1} - c_0$ occurs.

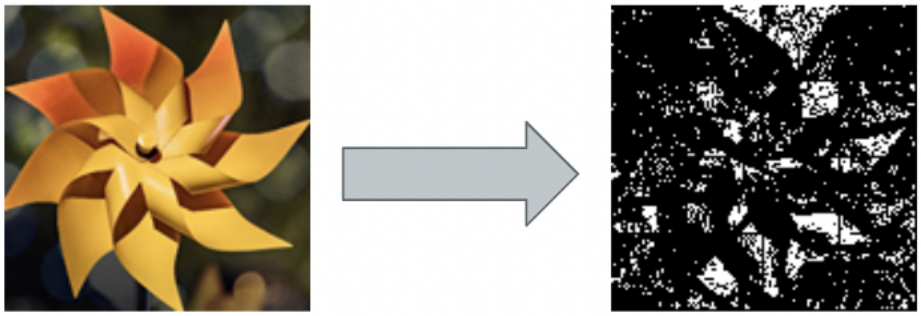
Finally, we can combine our vertical transposed image results with the horizontal original image results with a simple OR operator.

4.2 Thresholding

Initially, we used absolute thresholding - we always used the standard value 10^{-15}.

This works fine for certain images, and fine for most 32x32 images, which we were originally testing.

However, as we implemented image gridding (See $Main_Pipeline.py$) for larger images, we realized absolute thresholding is only a temporary solution. For more details on this part, check out our video, in the problems we faced section. Essentially, we realized that between images, since the value $\sum c_i^2$ can be drastically different (especially for larger images), and the distribution of $c_{i-1} - c_i$ values can also vary between images. As a result, although QPIE maintains the relative difference between pixels, it does not maintain the absolute difference (i.e., a two pixel difference is larger for smaller values of $\sum c_i^2$, thus it does not make sense for us to threshold by a set absolute value. In our results, this is why we saw that our algorithm works fine for some images but not for others.



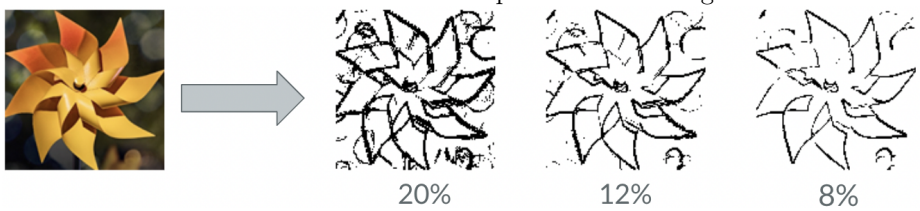
To solve this problem, we invented two strategies to threshold adaptively (according to the $\sum c_i^2$ of image):

1. K-Best Adaptive Thresholding
    - In the final cumulated results of all the $c_{i-1} - c_i$ differences, we take the K best (largest absolute value) differences and treat them as edges
    - In practice, this value of K is a percentage for how many values we turn into edges in the final images, and the actual K is computed on the fly via multiplying by size of c_i difference array
    - Effectively, this allows us to account for differences in $\sum c_i^2$ between these differences will scale the entire array of $c_{i-1} - c_i$ differences evenly
2. Max Adaptive Thresholding
    - In the final cumulated results of all the $c_{i-1} - c_i$ differences, we take all values $\leq K \cdot (max c_{i-1} - c_i)$ where K is some percentage
    - Effectively, this allows us to account for differences in $\sum c_i^2$ between these differences will scale $max c_{i-1} - c_i$ accordingly

We found that both thresholding methods are useful under different circumstances, but $K = 0.2(20\%)$ K-Best Adaptive Thresholding works best for general cases. Now, with these strategies, we can now tackle any image of any

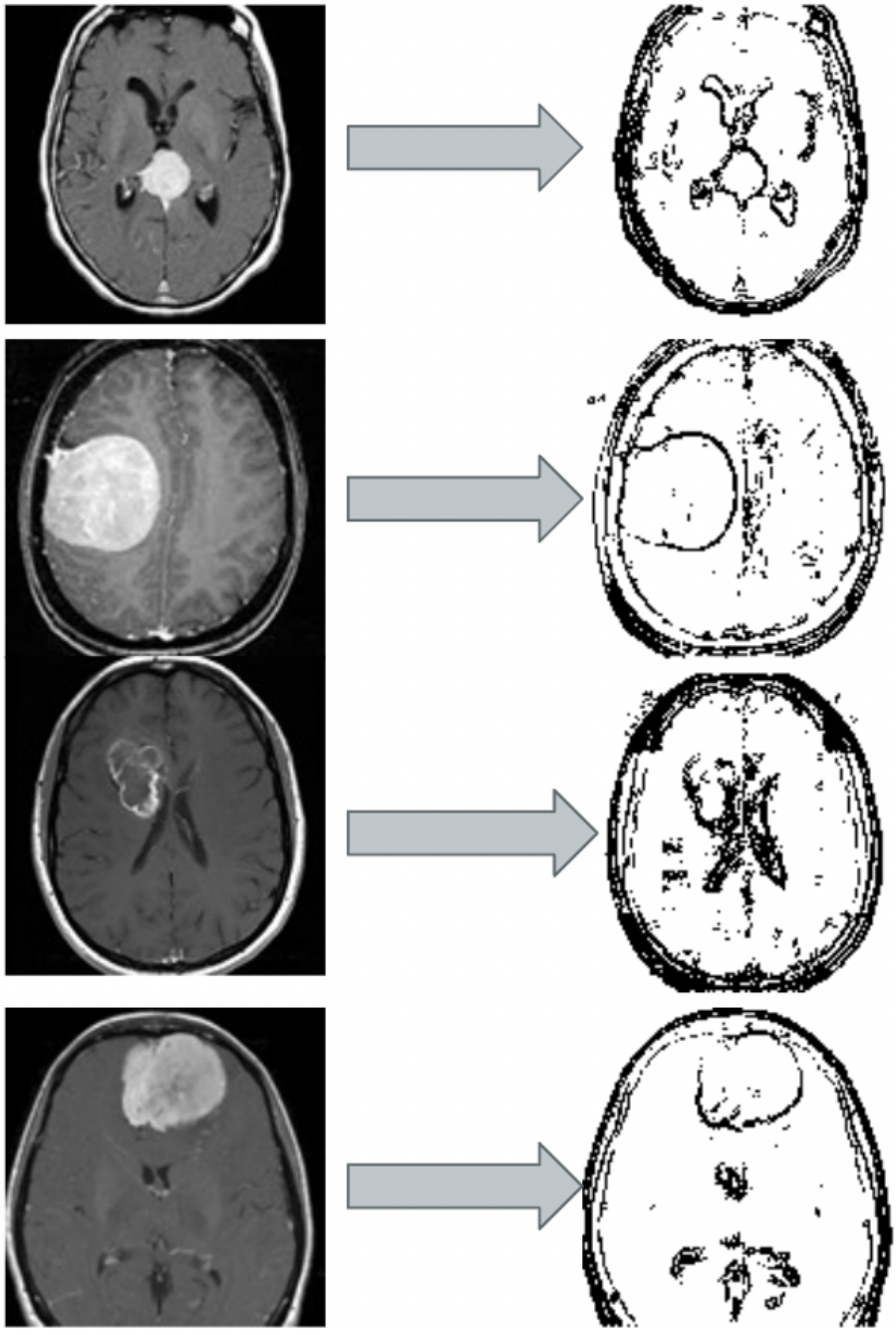
size. Image gridding, adaptive thresholding, and everything else is implemented in the $Main_Pipeline.py$ file. The $Mainnotebook$ does not including image gridding but instead takes averages of N/g side length squares, where g is the desired grid size for computation.

With different values of K via K-Best Adaptive Thresholding:



5 Results

30% K-Best Adaptive Thresholding



70% Max Adaptive Thresholding

17% K-Best Adaptive Thresholding

6 Acknowledgements

Thank you to MIT and Lincoln Labs for sponsoring and providing us with the opportunity to participate in the BWSI Beaverworks Summer Institute.