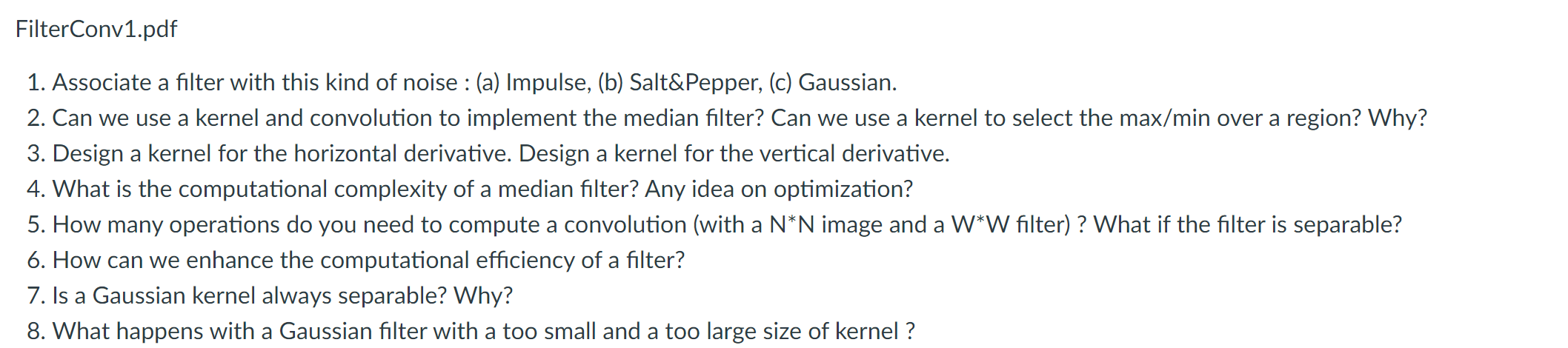
# Ch1

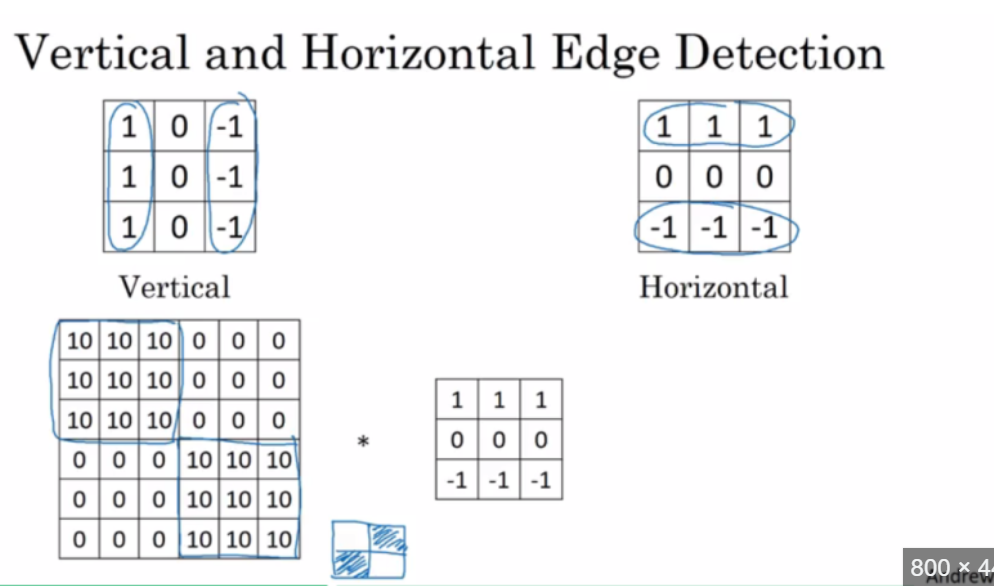
## Filterconv1



1. impulse noise means that it is extremely high than its neighbours, so we need a filter that can pass low value and prevent the high value.

median filter

we can use a gaussian filter to filter the gaussian noise

1. No, it is not liner computation.
2. 
3. N\*N, N^2, NlogN, it is determined by the sorting algorithm
4. N\*N\*W\*W, N\*N\*W\*2
5. try to find a separable W\*W filter
6. If the kernel could be separated, it needs to meet the limitations.

*Yes, a Gaussian kernel is always separable. This property is one of the reasons why Gaussian filters are commonly used in image processing and signal processing tasks.*

*A separable kernel is one that can be expressed as the outer product of two vectors. In the case of a Gaussian kernel, it can be represented as a product of two one-dimensional Gaussian functions. Mathematically, a 2D Gaussian kernel can be defined as:*

*G(x, y) = exp(-((x^2 + y^2) / (2 \* sigma^2))) / (2 \* pi \* sigma^2)*

*This 2D Gaussian kernel can be separated into two 1D Gaussian kernels along the x and y directions:*

*G(x, y) = G(x) \* G(y)*

*Where:*

*- `G(x)` is the 1D Gaussian kernel along the x-axis.*

*- `G(y)` is the 1D Gaussian kernel along the y-axis.*

*Separability is a desirable property because it simplifies the convolution operation and reduces the computational cost. Instead of applying a 2D convolution, you can apply two 1D convolutions sequentially, which is computationally more efficient.*

*Keep in mind that this separability property holds specifically for Gaussian kernels. Not all kernels used in image or signal processing are separable.*

1. the output imagine would be very smooth. It will depend on the number of sigma and theta.

## FilterEdges

1. **What is the difference between convolution with a Sobel kernel and computing the derivative of an image?**

The gradient points in the direction of most rapid increase in intensity. To calculate image gradient, we can use Sobel kernel convolution to approximately calculate the derivative of an image. The Sobel operator uses two 3x3 convolution kernels, one for detecting changes in intensity in the ***horizontal direction*** (Sobel-x) and the other for the ***vertical direction*** (Sobel-y). A derivative can be computed using various methods, and the Sobel operator is just one of the methods to approximate it.

There are other methods like the Scharr and Prewitt operators or even computing the true gradient using analytical methods (which is not always feasible for discrete data like images).

1. **Aren't function like derivative of a Gaussian too complicated to be used for real-time edge detection on large images?**

The derivative of a Gaussian is indeed more complicated than simpler operators like the Sobel kernel. However, It provides better localization of edges than simpler operators, and It's less sensitive to noise since the Gaussian function inherently smoothens the image while taking the derivative.

Besides, the choice of operator will always depend on the specific requirements of the task – both in terms of computational efficiency and the quality of the results.

1. **Why is the derivative of Gaussian filter preferred to the Gaussian of a derivative?**

\*\*Derivative of a Gaussian\*\*: create a Gaussian filter and then compute its derivative.

- This operation smoothens the image first (with the Gaussian) and then finds the rate of change (with the derivative).

- The result is an edge detector that is less sensitive to noise, as the Gaussian filter acts as a smoothing agent before the derivative operation is applied.

Not losing too much info.

\*\*Gaussian of a Derivative\*\*: first compute the derivative of the image and then apply the Gaussian filter.

- Such an operation would first highlight the edges (and noise) and then attempt to smoothen the result.

- This isn't commonly done because the initial derivative operation would be highly sensitive to noise, and the subsequent Gaussian filter might blur the important edge details that the derivative revealed.

1. \*\*Noise Sensitivity\*\*: Real-world images often contain noise. By applying a Gaussian filter first (smoothing the image), the derivative operation **becomes less sensitive to noise**. This means the edges you detect are more likely to be genuine edges in the subject rather than artifacts of noise.

2. \*\*Better Edge Localization\*\*: The "derivative of a Gaussian" approach tends to **provide better localization of edges**. This is because the smoothing occurs first, ensuring that the detected edges are more representative of broader structures in the image rather than minute details or noise.

1. **Can edge be reliably found without pre-processing the grayscale image first? Is it recommended to apply directly a Sobel filter on a grayscale image?**

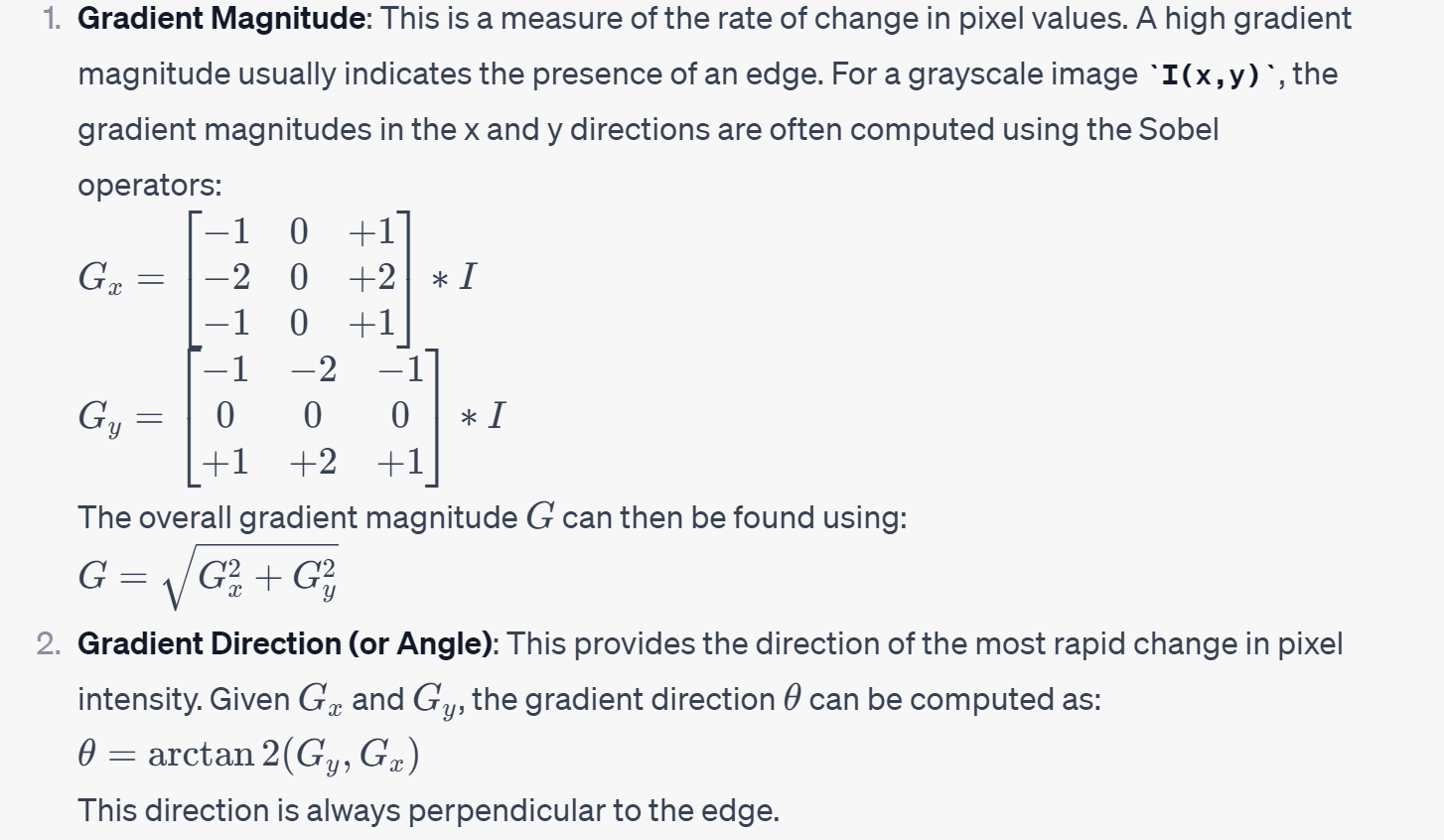
If there is little noise in the grayscale image, maybe edge can be found. But it is not recommended to apply directly a Sobel filter on a grayscale image.

Images, especially those captured by real-world sensors like cameras, are often corrupted by various types of noise (like Gaussian noise, speckle noise, etc.). Applying edge detection directly to such noisy images can result in spurious edges, making the results unreliable.

I think applying the Sobel filter directly to a noisy grayscale image may lead to many false edges. it's common to preprocess the image (often using a Gaussian blur) before applying the Sobel filter or any other edge detection technique.

1. **How does finding the gradient direction works and how does it relates to Canny edge detection?**

Edge detection often involves finding rapid changes in pixel intensity in an image. The gradient of an image provides a vector representation of the direction and magnitude of the most rapid change in intensity.



**\*\*Relation to Canny Edge Detection:\*\***

Canny edge detection is a multi-step algorithm designed to detect a wide range of edges in images.

1. \*\*Gradient Computation\*\*: As the first major step, Canny uses Sobel filters (or similar operators) to compute the gradient magnitude and direction for each pixel in the image.

2. \*\*Non-maximum Suppression\*\*: This step ensures that the edges are thin. Using the gradient direction, each pixel is checked against its neighbors along the gradient direction**. If the pixel has the highest gradient magnitude compared to its neighbors in that direction, it's preserved; otherwise, it's suppressed (set to zero). This results in thin edge lines.**

3. \*\*Linking and Thresholding\*\*: The gradient magnitudes are thresholded to identify strong, weak, and non-relevant pixels. Strong and weak edge pixels are determined based on high and low threshold values, respectively. Pixels with gradient magnitudes above the high threshold are considered strong edge pixels, while those between the two thresholds are considered weak edge pixels. Edge tracking is then performed by hysteresis: weak pixels connected to strong pixels are retained, while other weak pixels are suppressed.

So, the gradient direction is essential in the Canny edge detection algorithm, especially during the non-maximum suppression step, ensuring that the detected edges are thin and accurate.

1. **Why do we need hysteresis in Canny edge detection?**

Hysteresis in the Canny edge detection algorithm is used to address the problem of ambiguity in edge detection.

Briefly, the main reasons for using hysteresis are:

1. \*\***To Eliminate Weak Edges**\*\*: Many edges detected by gradient computations might be weak or due to noise. Hysteresis helps in distinguishing between true and false weak edges.

2. \*\***To Ensure Edge Continuity**\*\*: By using two threshold values (high and low), hysteresis ensures that if a weak edge pixel is connected to a strong edge pixel, it's considered part of the edge, making the overall edge appear continuous. On the other hand, weak edge pixels not connected to strong edge pixels are treated as noise and discarded.

Thus, hysteresis helps to keep genuine edge structures while discarding noise and spurious edges, resulting in cleaner and more accurate edge maps.

1. **How does Canny edge detector thinning work? What is the purpose of this step?**

The thinning step in the Canny edge detector is achieved through a process called "non-maximum suppression."

\*\*How it works\*\*:

After calculating the gradient magnitude and direction, for each pixel, non-maximum suppression checks if the pixel is a local maximum in the direction of the gradient (i.e., perpendicular to the edge direction). If the pixel's magnitude is greater than its neighbors in the gradient direction, it is kept. Otherwise, it's set to zero.

\*\*Purpose\*\*:

The main purpose of this step is to ensure that the edges in the output image are thin, typically one-pixel wide. This eliminates the response to the same edge being broader than a single pixel, which can occur due to variations in intensity. Thinning the edges makes them more accurate and easier to analyze in subsequent processing tasks.

1. **Why do we need to care about interpolated pixels for non-maximum suppression and how are they obtained?**

In the context of the Canny edge detector, non-maximum suppression checks each pixel to determine if it's a local maximum in its gradient direction, meaning that it compares each pixel's gradient magnitude with those of its neighbors in the direction of the gradient. However, this gradient direction is not always perfectly vertical or horizontal, so direct neighbors might not capture the true direction accurately.

\*\*Why we care\*\*:

1. \*\*Sub-pixel Accuracy\*\*: The gradient direction often doesn't align with the image's grid-like pixel arrangement. Ignoring interpolation can lead to inaccurate edge thinning.

2. \*\*Better Edge Localization\*\*: Interpolated values offer a more precise estimate of where the maximum gradient (i.e., the edge) occurs between pixels, leading to better edge representation.

\*\*How interpolated pixels are obtained\*\*:

1. \*\*Bilinear Interpolation\*\*: Given a gradient direction, we can find the two nearest pixel intensities in that direction. Bilinear interpolation is then used to compute a weighted average of these intensities based on the gradient's angle, producing an interpolated value.

2. The current pixel's gradient magnitude is compared against the interpolated values to determine if it's a local maximum.

In essence, interpolated pixels in non-maximum suppression allow the method to be more accurate and sensitive to the actual gradient direction, ensuring the detected edges are as accurate and thin as possible.

## Hough

1. **In practice, how do we go from Sobel to the Hough Transform?**
2. The sobel transform is used to detect edges in images, it can help find the places in the image where there is a rapid change in intensity or color. Through measuring the gradient of the image intensity at each point, we can find the direction and magnitude of the brightest edges.
3. Then we threshold the image and thin the edges, find these pixels considered as edges.
4. Using hough transform for line detection.

For every point or pixel that Sobel produced, they can be represented by a line in Hough space. If there are many curves intersecting at one particular value in the hough space, that means many pixels form a line with that particular r and theta.

Iterater each pixel, and draw the lines, then voting, find the most voted value in polar space. Find the local maximum

1. **Why is it useful to do the hough transform using the polar parameterization instead of the cartesian representation of points?**
2. Firstly, if use y = mx + b to represent the line, the vertical lines would make m to become infinite, and b is on a very wide range of values.
3. But in polar parameterization, both r the theta are bounded. By using voting, it is easier to mange the discrete and finite accumulator space.
4. I think it is more intuitive in the polar space than in a cartesian space
5. **Why is the Hough transform called a “voting” method?**

Basically, each feature point in the image space "votes" for potential parameterized models in a parameter space.

For each edge pixel in the image, the hough transform will plot a sinusoidal curve in the polar space, representing all the lines that could go through that particular pixel.

When there is another edge pixel, it will plot a sinusoidal curve in the polar space too, when they pass the same point in the polar space, it called voting, that means these two original points may be on the same line.

Thus, the voting means that every pixel "suggests" for" all possible lines (in the parameter space) that could pass through that pixel. And the peak in the voting results show that if many points are voting for a similar line parameterization, then there's likely a line in the image with those parameters.

1. **How does one handle multiple lines (objects) in the Hough transform? What is the consequence? Propose solutions.**

Handling multiple lines (or objects) with the Hough transform is a common challenge. It maybe have some problems:

**Accumulator Overlap**: When multiple lines are close to each other or have similar orientations, their sinusoids in the polar space may overlap, making it difficult to distinguish between the lines.

**High Computation**: If an image has many edge points and multiple lines, the accumulator array can get populated rapidly, leading to increased computation.

**Ambiguities in Peak Detection:** Multiple lines can lead to several peaks in the accumulator space. Identifying the most significant peaks or distinguishing between closely spaced peaks can be a challenge.

Proposed Solutions:

**Thresholding:** Apply a threshold to the accumulator space. Only peaks (lines) with votes above a certain threshold will be considered significant. This helps in eliminating minor lines or noise.

**Non-Maximum Suppressio**n: After the voting process, non-maximum suppression can be applied to the accumulator space to retain only the local maxima. This helps in distinguishing between closely spaced lines.

**Peak Distance:** Implement a minimum distance between peaks. If peaks in the accumulator space are too close, they can be merged into a single line or the one with fewer votes can be suppressed.

**Gradient Direction:** Use gradient direction information from the edge detection phase. Lines that are likely to be part of the same object will have similar gradient directions. This can aid in distinguishing between different line segments.

Pick the maximum peak?

1. **How is the gradient used in the Hough transform?**

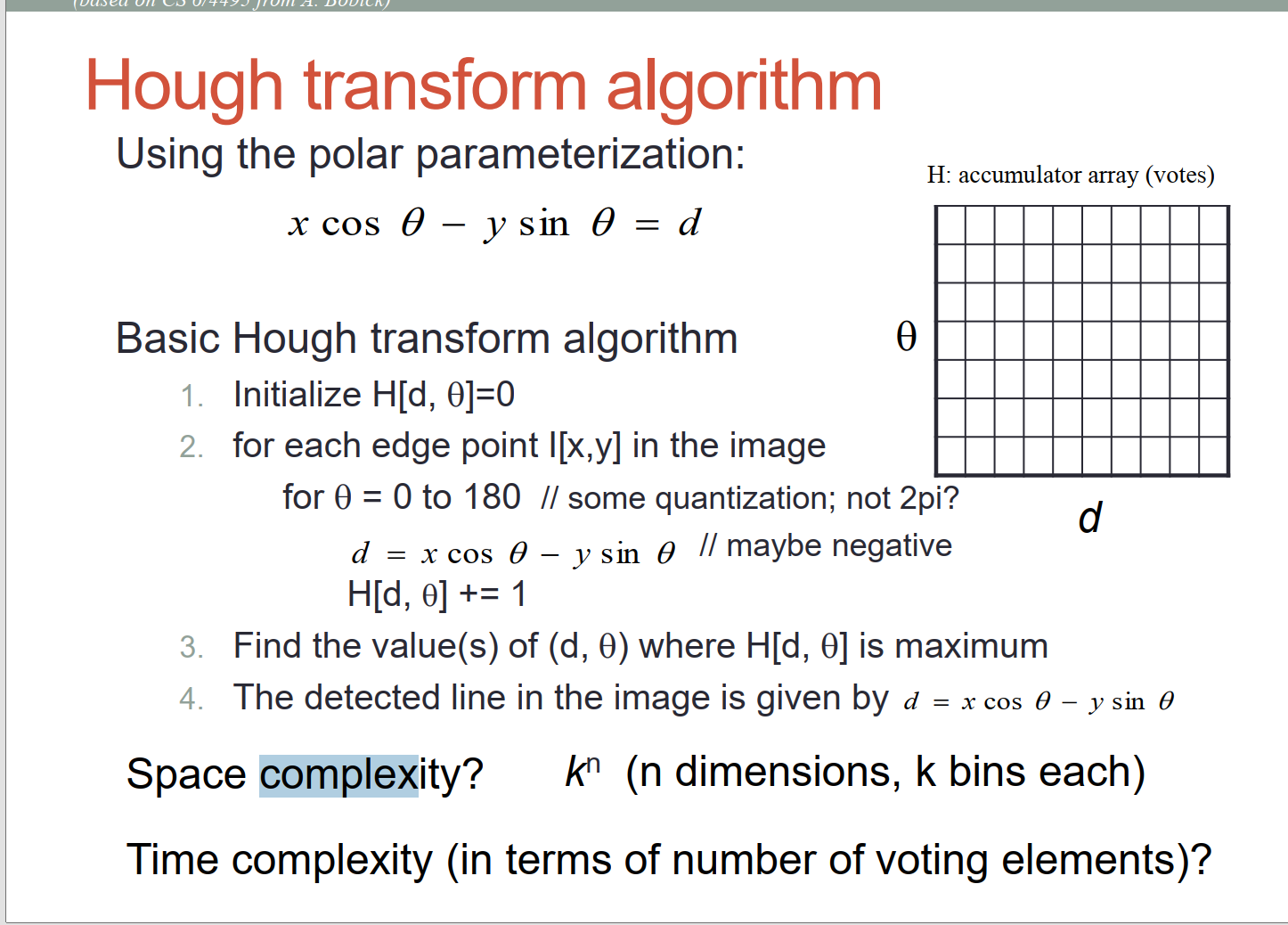
The gradient in the Hough transform is used primarily in two ways:

**1. \*\*Edge Detection\*\***: Before applying the Hough transform, edge detection techniques that rely on gradient computations (like the Canny edge detector) are employed to identify edges in the image.

**2. \*\*Orientation Guidance\*\***: The gradient direction at an edge pixel indicates the orientation of the line that's perpendicular to the edge. This orientation information can guide the voting process in the Hough transform, making it **more targeted towards possible line orientations.**

In summary, the gradient aids in detecting edges and provides orientation information for the Hough transform voting process.

1. **What is the complexity of the hough transform?**



The complexity of the Hough transform primarily depends on the number of edge pixels and the dimensions of the parameter space, I guess.

For line detection using the polar coordinate system:

Suppose there are N edge pixels in the image, If I need to iterate over all possible \(\theta\) values to compute and vote for the corresponding \(r\) value. This results in a complexity of \(O(N)\) for each edge pixel.

This complexity assumes line detection in the Hough transform. For other shapes or configurations, the complexity might differ based on the number of parameters and their discretization.

N points, k bins, Z dimensions, times complexity nk^2

1. **List solutions to reduce/limit the Hough transform complexity.**
2. Coarser Discretization: Reduce the granularity of the parameter space. This means fewer bins in the accumulator, but at the cost of precision.
3. Probabilistic Hough Transform: **Instead of considering all edge pixels, randomly sample a subset, reducing computation time.**
4. Region of Interest (ROI) Selection: **Process only specific regions of the image where you expect to find the shapes.**
5. **Pre-filtering: Use techniques like non-maximum suppression or edge thinning to reduce the number of edge pixels before applying the Hough transform.**
6. Parallelization: Utilize parallel processing capabilities to handle different portions of the image or parameter space simultaneously.
7. **Optimized Data Structures: Use efficient data structures and algorithms to update and search the accumulator array.**
8. **Gradient Guidance:** Utilize gradient direction to narrow down potential θ values for voting, rather than looping over all possibilities.
9. By employing these strategies, the computational burden of the Hough transform can be significantly reduced.
10. **How to perform a smoothing in the accumulator array?**

To perform smoothing in the accumulator array for the Hough transform:

**1. \*\*Gaussian Filter\*\*:** Apply a Gaussian filter (or kernel) to the accumulator array. This will help in smoothing out local variations and emphasizing prominent peaks.

**2. \*\*Mean Filter\*\*:** Apply a mean (or average) filter over the accumulator. This can help reduce sharp local peaks.

**3. \*\*Weighted Accumulation\*\***: When updating the accumulator, instead of incrementing by 1 for each vote, use a weighted value based on edge strength or gradient magnitude.

These smoothing techniques can help in consolidating votes in the accumulator array and making peak detection more robust against noise and minor variations.

## Frequency

1. **The Fourier tra**nsform is for periodic functions, how does it appl**y to images then?**

Fourier transform is a mathematical tool to transform a signal from its original domain to the frequency domain. It is not just for periodic functions maybe.

The images can be seen as functions, their domain is spatial, and the frequencies in an image relate to the changes in pixel values. We can use two dimensional fourier transform to apply for the image.

The applications:

1. Filtering: removing the high frequencies from an image’s fourier transform.
2. Compression: when most energy is concentrated in low frequencies, by preserving and discarding the less important higher frequencies, we can compress the image.

Infinite: see the picture as a part of an infinite function

1. Can the Fourier Transform be used for edge detection? How?

Yes, it can be used in edge detection, it can be used for high-pass filtering. Also, the inverse fourier transform need to be applied for the image to bring it back to the sparial domain.

However, because the Fourier transform-based edge detection is more global in nature, affecting the entire image, whereas spatial domain operators can be more localized.

1. Explain the vertical and horizontal white line in the center of the Fourier image spectrum in most example.

*1. \*\*DC Component\*\*: The very center of the Fourier spectrum represents the DC component (or zero frequency component). This is essentially the average brightness of the entire image. It's often the brightest spot in the Fourier spectrum.*

2. \*\*Low Frequencies\*\*: As you move away from the center, the frequencies increase. The vertical line represents the horizontal frequency components of the image, and the horizontal line represents the vertical frequency components of the image.

**The \*\*vertical white line\*\* corresponds to changes in the image that happen horizontally (e.g., vertical edges or patterns).**

**The \*\*horizontal white line\*\* corresponds to changes in the image that occur vertically (e.g., horizontal edges or patterns).**

*3. \*\*Symmetry\*\*: The 2D Fourier Transform of a real-valued image is symmetric about its center. This means that the left half of the spectrum is a mirror image of the right half, and the top half is a mirror image of the bottom half.*

*4. \*\*Centering the Spectrum\*\*: Often, for visualization purposes, the spectrum is shifted so that the DC component (and those white lines) is at the center of the image. This is accomplished using a Fourier shift operation. If this shift isn't done, the DC component is at the top-left corner, and the spectrum's appearance is different, albeit the information is the same.*

*5. \*\*Brightness\*\*: Those white lines might not always be white; their appearance (in terms of brightness and clarity) depends on the content of the image and the range of frequencies present. However, in many typical images, especially those with strong vertical and horizontal elements, these lines can be quite pronounced.*

**In summary, the vertical and horizontal lines in the center of a Fourier image spectrum represent the low-frequency components of the image, indicative of broad changes or patterns in the image across the respective orthogonal directions.**

1. What is "ringing"? Why can you most prominently see it in JPEG compressed line drawing?

"ringing" refers to the unwanted oscillations near sharp transitions in images, and it's particularly noticeable in JPEG-compressed line drawings due to the nature of the lossy compression process and the sharp contrasts inherent to line drawings.

**JPEG Compression: JPEG is a lossy compression method that uses the Discrete Cosine Transform (DCT) to convert blocks of the image into the frequency domain. Once in the frequency domain, many of the higher frequency coefficients (those representing finer details) are quantized or discarded to achieve compression. When the image is decompressed, these missing details can lead to artifacts, including ringing.**

Sharp Transitions: **Line drawings typically have sharp edges and high contrast, which means they contain a lot of high-frequency information. When you compress these drawings using JPEG, the DCT tries to approximate these sharp transitions using a limited set of basis functions (cosines). The imprecision introduced by this quantization leads to oscillations near the transitions, manifesting as ringing.**

Gibbs Phenomenon: The ringing artifacts are closely related to the Gibbs phenomenon, a concept from Fourier series analysis. The Gibbs phenomenon describes the overshoot (or "ringing") that occurs when representing a discontinuous function (like a step function or a sharp edge) using a finite series of sinusoids.

Prominence in Line Drawings: Unlike photographs that usually have smoother transitions and a plethora of tones, line drawings have simple, high-contrast areas, making any introduced artifact, such as ringing, much more noticeable.

1. Explain the JPEG compression and decompression pipeline.

**JPEG Compression:**

1. \*\*Color Space Conversion\*\*: Convert the image from RGB to YCbCr color space. This separates the image into a luminance component (Y) and two chrominance components (Cb and Cr).

2. **\*\*Subsampling\*\***: The human eye is more sensitive to luminance (brightness) than chrominance (color). JPEG takes advantage of this by optionally subsampling the chrominance channels, often reducing their resolution by half (known as 4:2:0 subsampling).

3. **\*\*Block Splitting\*\***: Split the image into 8x8 pixel blocks.

4. **\*\*Discrete Cosine Transform (DCT)\*\***: For each 8x8 block, apply the DCT. This transforms the block from the spatial domain into the frequency domain. The result is an 8x8 matrix where the top-left corner represents the lowest frequency (DC component) and the bottom-right corner represents the highest frequency.

5. \*\*Quantization\*\*: Divide each value in the DCT'd block by a corresponding value in a quantization matrix. Then round to the nearest integer. The quantization matrix is designed to retain more visually important information while reducing or removing less important details. This is where the lossy part of JPEG compression primarily occurs.

6. \*\*Entropy Coding\*\*: Encode the quantized values using Huffman or arithmetic coding. This takes advantage of the frequency of occurrence of certain patterns to represent the image in fewer bits.

7. \*\*File Formatting\*\*: Store the compressed data, along with metadata (like quantization tables and Huffman tables), in a file format, typically with a .jpg or .jpeg extension.

**JPEG Decompression:**

The decompression process is essentially the reverse of the compression process:

1**. \*\*Entropy Decoding\*\***: Decode the Huffman or arithmetic coded data to retrieve the quantized DCT values.

2. **\*\*Dequantization\*\***: Multiply each value in the decoded blocks by the corresponding value in the quantization matrix to reverse the quantization step.

3. **\*\*Inverse DCT (IDCT)\*\***: For each block, apply the IDCT to transform it from the frequency domain back to the spatial domain.

4. **\*\*Block Combination\*\***: Combine the 8x8 blocks to reconstruct the image.

5. \*\*Chrominance Upsampling (if necessary)\*\*: If chrominance channels were subsampled during compression, they are upsampled to match the luminance channel's resolution.

6. \*\*Color Space Conversion\*\*: Convert the image back from YCbCr to RGB color space.

After these steps, you get a reconstructed image. It's worth noting that due to the lossy nature of JPEG compression, the decompressed image might not be exactly identical to the original, especially at lower quality settings. However, at high quality settings, the differences might be visually negligible.

1. JPEG uses DCT to compress images, this makes sense for BW images, how is color represented with JPEG?

JPEG does use the Discrete Cosine Transform (DCT) to compress images, and while the DCT process can be applied directly to grayscale (BW) images, handling color images requires a few additional steps:

1. \*\*Color Space Conversion\*\*: **JPEG typically starts by converting the standard RGB (Red, Green, Blue) representation of an image to the YCbCr color space.**

**- \*\*Y\*\* component represents the luminance (brightness) of the color, roughly corresponding to the grayscale version of the image.**

**- \*\*Cb\*\* and \*\*Cr\*\* represent chrominance (color) components. Cb gives information about how much blue is present and Cr gives information about how much red is present.**

2. \*\*Subsampling\*\*: Before applying DCT, JPEG can optionally perform chroma subsampling. **This is based on the observation that the human eye is more sensitive to variations in brightness than in color. Thus, the Cb and Cr components can be sampled at a lower resolution than the Y component to reduce data size without a significant perceived drop in quality.** A common subsampling scheme is 4:2:0, where for every 4x4 block of pixels, we take the Y values for each pixel but only sample Cb and Cr values at half the horizontal and vertical resolution.

3. \*\*DCT Compression\*\*: After (optionally) subsampling, the Y, Cb, and Cr components are each compressed using DCT separately. Each component is divided into 8x8 blocks, and DCT is applied block-wise, just like with grayscale images.

4. \*\*Quantization and Encoding\*\*: Post DCT, each of the components undergoes the same quantization and encoding processes. Depending on the settings, the quantization might differ between the Y and the Cb/Cr channels, often preserving more detail in the luminance than in the chrominance.

5. \*\*Decompression\*\*: When decompressing, each of the Y, Cb, and Cr channels is processed separately. If subsampling was applied, the Cb and Cr channels would be upsampled to match the Y channel's resolution. Finally, the decompressed YCbCr values are converted back to RGB for display.

In summary, while the DCT process remains central to JPEG compression, **color images introduce an additional layer of complexity.** By working in the YCbCr color space and optionally using chroma subsampling, JPEG can more efficiently represent and compress color information in a way that aligns with human visual perception.

1. Having a quantization table, how do you get back the image after compression?

The quantization table in JPEG compression is used to reduce the precision of the Discrete Cosine Transform (DCT) coefficients, which leads to substantial compression. However, it also introduces loss in image quality, making JPEG a lossy compression method.

To reconstruct the image after JPEG compression, you would essentially reverse the steps that were used during compression. Here's a brief overview focusing on the use of the quantization table:

**1. \*\*Decoding\*\*:** Start by decoding the JPEG file. This will give you the quantized DCT coefficients for each 8x8 block in the image, along with other information like the quantization tables.

**2. \*\*Dequantization\*\***: For each 8x8 block of quantized DCT coefficients, multiply each coefficient by the corresponding value in the quantization table. This step "undoes" the quantization process by scaling the coefficients back up.

- For instance, if a DCT coefficient was divided by 10 during the quantization process and rounded to 3, after dequantization, it would be multiplied by 10 to give a value of 30.

**3. \*\*Inverse DCT (IDCT)\*\***: Perform the Inverse Discrete Cosine Transform on each 8x8 block of dequantized coefficients. This converts each block from the frequency domain back to the spatial domain, yielding pixel values.

**4. \*\*Merging Blocks\*\***: Combine the 8x8 blocks to form the full image.

**5. \*\*Color Channel Handling (if it's a color image)\*\***:

- If chroma subsampling was used during compression, upsample the Cb and Cr channels to the resolution of the Y channel.

- Convert the image from the YCbCr color space back to the RGB color space.

After these steps, you'll have a reconstructed image. **It's important to note that due to the lossy nature of the JPEG compression (especially because of quantization), the reconstructed image will not be exactly the same as the original image.** The difference in quality will *largely depend on the values in the quantization table: a more aggressive quantization will result in higher compression but lower image quality, while a less aggressive quantization will retain more quality at the expense of less compression.*

1. JPEG is good for Human vision, how about Computer Vision ?

JPEG compression was designed with human vision in mind. The compression techniques, particularly the use of the YCbCr color space and chroma subsampling, capitalize on the characteristics and limitations of human visual perception to achieve significant data reduction with minimal perceived loss of quality.

However, when it comes to computer vision, the nuances of JPEG compression can sometimes introduce complications:

1. **\*\*Lossy Compression Artifacts\*\***: The artifacts introduced by JPEG's lossy compression (like ringing, blocking, and loss of high-frequency details) might not be noticeable to the human eye in many cases, especially at high-quality settings. **However, these artifacts can adversely affect computer vision algorithms that rely on precise pixel values or fine details.**

2. **\*\*Edge Detection\*\*: Algorithms that detect edges in images can be particularly sensitive to the artifacts introduced by JPEG compression. The ringing artifacts around sharp edges or high contrast areas can lead to false edges being detected.**

3. \*\*Feature Extraction\*\*: Modern computer vision techniques, especially those based on deep learning, often rely on extracting intricate features from images. Compression artifacts can distort these features, potentially reducing the accuracy of the model.

4. \*\*Texture Analysis\*\*: Similar to edge detection, texture analysis can be impacted by the loss of high-frequency details due to JPEG compression.

5. \*\*Consistency and Repeatability\*\*: If an image undergoes multiple cycles of compression and decompression (for instance, being saved, edited, and saved again multiple times as a JPEG), this can introduce cumulative artifacts that further degrade the image quality. This inconsistency can be problematic for computer vision tasks.

6. \*\*Speed\*\*: Decoding JPEG images requires additional computational overhead. In real-time computer vision applications, this might introduce unwanted latency.

Given these challenges, when is it appropriate to use JPEG in computer vision?

- **\*\*Dataset Storage\*\***: Due to the sheer size of many image datasets, it might be practical to store them using JPEG compression to save space, especially if the slight loss in image fidelity doesn't significantly impact the downstream task.

- **\*\*Fine-tuning Compression Levels\*\***: If JPEG is used, one can fine-tune the compression level to strike a balance between size and quality. Using a high-quality setting might reduce the adverse effects on computer vision tasks.

- **\*\*Real-world Scenarios\*\***: If the end application will be processing JPEG images (e.g., a system analyzing user-uploaded photos), it might make sense to train or test algorithms on JPEG-compressed data to replicate real-world conditions.

However, for critical tasks or when precision is paramount, lossless image formats like PNG or TIFF might be preferred over JPEG. It's always essential to consider the specific requirements and constraints of the computer vision task at hand when choosing an image format.

Dectecting lose information