1 Image Processing and Convolution

1.1 Preparation: Understanding Convolution

Think of convolution as a way to find patterns in an image by looking through a small window (kernel). As we slide this window across the image, we:

1. Look at a small neighborhood of pixels
2. Multiply each pixel by the corresponding value in our pattern-matching window (kernel)
3. Sum up these products to get a single number
4. Move the window and repeat

For example, if we want to detect vertical edges, we might use this kernel:

$$K = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

Let's see how it works on a small image region (3x3 pixels):

$$\begin{bmatrix} 10 & 80 & 10 \\ 10 & 80 & 10 \\ 10 & 80 & 10 \end{bmatrix}$$

We multiply each pixel by the corresponding kernel value and sum: $(10 \times -1 + 80 \times 1 + 10 \times 0) \times 3 = 210$. The high positive value (210) indicates a strong vertical edge was detected. Here are some common kernel patterns:

Vertical edge detection:

$$\begin{bmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{bmatrix} \quad \text{Looks for } \xrightarrow{\text{dark to bright}}$$

Horizontal edge detection:

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{Looks for } \downarrow \text{ dark to bright}$$

Blur/Smoothing:

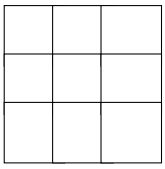
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{Averages all neighboring pixels}$$

1.2 Image Processing and Convolution

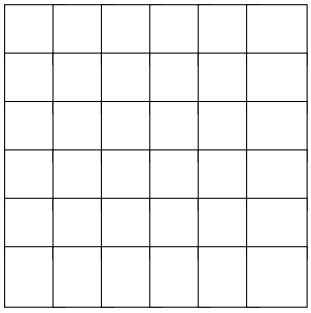
Consider a 6x6 grayscale image showing a diagonal line pattern:

80	10	10	10	10	10
10	80	10	10	10	10
10	10	80	10	10	10
10	10	10	80	10	10
10	10	10	10	80	10
10	10	10	10	10	80

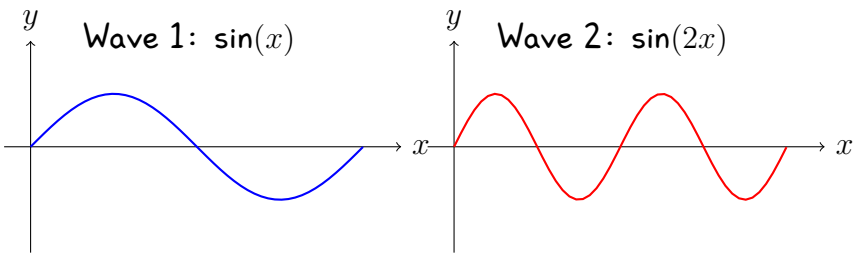
1. If we want to detect diagonal edges, which kernel of size 3x3 would be most appropriate? The kernel should have the values between -1 and 1.



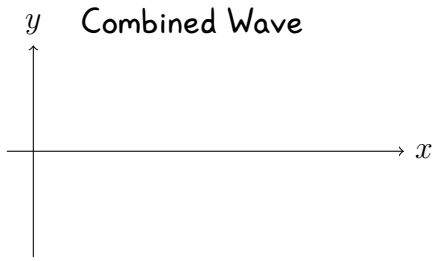
2. Apply your kernel to compute the convoluted image. No need to calculate the value of each pixel exactly but show your estimate by shading the pixels. For the boundary pixels, leave them blank since the kernel exceeds the boundary of the image.



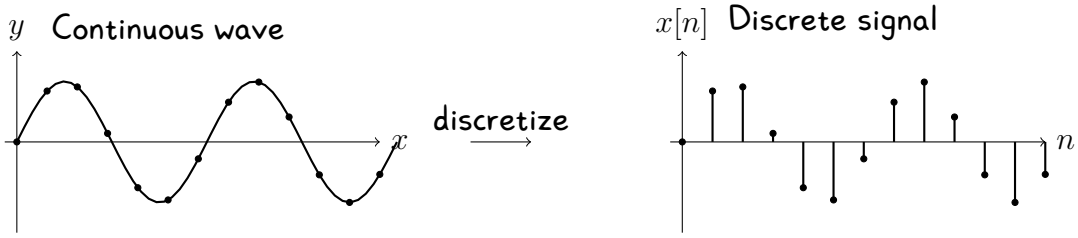
3. Now, let's learn how JPEG compression works. Consider this waves:



We combine these two waves by weighting the first wave by 1.5 and the second wave by 0.5. Combined Wave $= 1.5 \cdot \sin(x) + 0.5 \cdot \sin(2x)$. Draw the combined waves.



4. The Fourier transform is a reverse operation: it decomposes, not combines, waves into basic waves. The waves are continuous functions. But we can discretize them for computation as follows:



This results in a vector of values $[10, 80, 10, 80, 10, 80, 10, 80]$. Now, let's create a discretized mixed wave Z from X and Y as follows

$$Z = X + Y = [10, 90, 30, 90, 10, 70, -10, 70, 10] \tag{1}$$

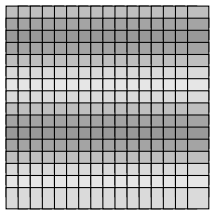
where

$$X = [10, 80, 10, 80, 10, 80, 10, 80, 10], \quad Y = [0, 10, 20, 10, 0, -10, -20, -10, 0] \tag{2}$$

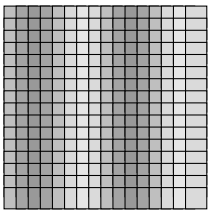
(a) If we apply a kernel $K = [-1, 1, -1]$ to this signal, what will be the resulting signal? What kind of frequencies will this kernel emphasize?

(c) If we apply a kernel $K = [1, 1, 1]$ to this signal, what will be the resulting signal? What kind of frequencies will this kernel emphasize?

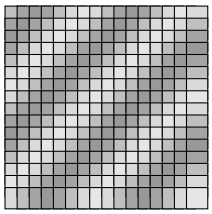
5. Just as 1D signals can be decomposed into sine waves, 2D images can be decomposed into 2D waves as follows. The Fourier transform can be applied to 2D images to decompose them into a sum of 2D waves.



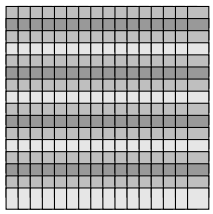
Horizontal (Low freq.)



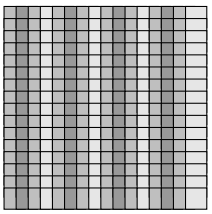
Vertical (Low freq.)



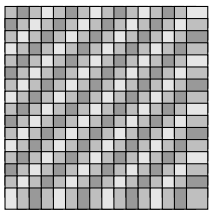
Diagonal (Low freq.)



Horizontal (High freq.)



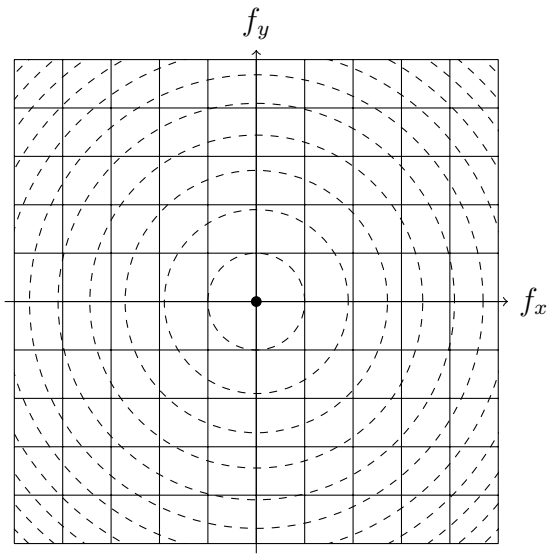
Vertical (High freq.)



Diagonal (High freq.)

Now, consider this checkerboard pattern on the left. Mark where you expect the highest magnitudes in the Fourier transform grid. The dashed circles represent the basis 2D waves in the Fourier domain.

80	10	80	10	80	10
10	80	10	80	10	80
80	10	80	10	80	10
10	80	10	80	10	80
80	10	80	10	80	10
10	80	10	80	10	80



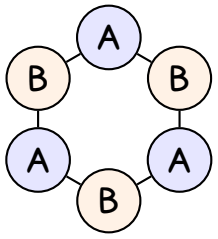
Fourier Transform Grid

6. The image can be mapped to the Fourier transform grid (called frequency domain). We can also map it back to the original image domain (called spatial domain). Thus, we can manipulate the image in the frequency domain to remove some waves from the original image. If we want to keep only the low-frequency components of the checkerboard pattern, what regions of the Fourier transform grid should we set to zero?

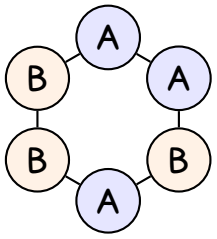
Weisfeiler-Lehman Test

A neural network's job is often graph classification (e.g., is this molecule toxic?). To do this, the network must be able to distinguish between two different graphs.

Below are two molecules (Graph G_1 and Graph G_2). They consist of atoms labeled A and B. If we present this to a neural network that simply counts labels, it will likely not be able to distinguish between the two graphs.



G_1



G_2

Part 1: Distinguishing Labeled Graphs

Let's work up a solution step by step. Since counting labels (global pooling) doesn't work, let's look at the **local structure** around each node.

Step 1: Message Passing. Create a **signature** for every node: '(My Label, {Neighbor Labels})'.

Graph G_1 Signatures:

- Any Node A: `(A, {B, B})`
- Any Node B: `(B, {A, A})`

Graph G_2 Signatures:

- Top Node (A): _____
- Top-Right Node (A): _____
- Bottom Node (A): `(A, {B, B})`

Step 2: Compression (Hashing). Assign a unique ID to every unique signature found across both graphs.

Signature	→	ID	Signature	→	ID
(A, {B, B})	→	1	(B, {A, B})	→	4
(B, {A, A})	→	2	(A, {A, A})	→	5
(A, {A, B})	→	3	(B, {B, B})	→	6

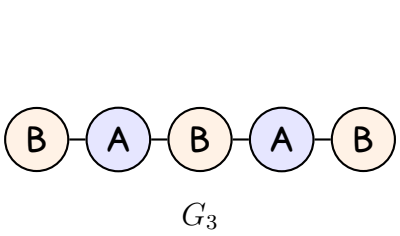
Step 3: Rewrite with Colors. List the set of Color IDs present in each graph.

- G_1 **Colors:** { 1, 1, 1, 2, 2, 2 }
- G_2 **Colors:** { __, __, __, __, __, __ }

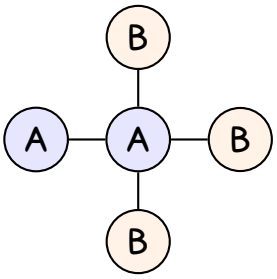
Conclusion: Are the sets of colors different? **[Yes / No]**

Exercise B: Structural Difference

Below are graphs G_3 (Line) and G_4 (Star). Both have **5 Nodes** (2 As, 3 Bs).



G_3



G_4

Task: Fill in the signatures '(My Label, {Neighbors})' for all nodes.

Graph G_3 (Line):

- Left B: _____
- Left A: _____
- Middle B: _____
- Right A: _____
- Right B: _____

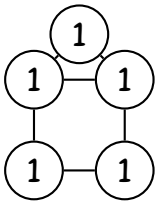
Graph G_4 (Star):

- Center A: _____
- Left A: _____
- Top B: _____
- Right B: _____
- Bottom B: _____

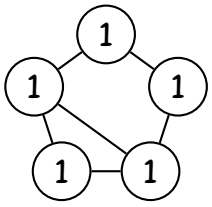
Compare: Does G_4 generate a signature that is completely impossible in G_3? **[Yes / No]**

Part 2: Identifying Isomorphic Graphs (Unlabeled)

Sometimes different drawings represent the same graph. A GNN should map these to the same embedding. Consider H_1 (The House) and H_2 (Pentagon w/ Chord).



H_1



H_2

Since there are no node labels (A or B), we initialize **all nodes** with the label "**1**". Perform the same Message Passing process as Part 1. Create signatures for every node: `(My Label, {Neighbor Labels})`.

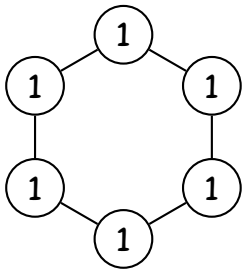
Signature Found	Number of nodes in H_1	Number of nodes in H_2
`(1, {1, 1})`		
`(1, {1, 1, 1})`		

Conclusion: Compare the counts in the table.

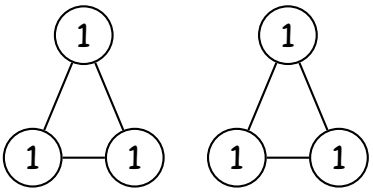
1. Are the "histograms" (counts of signatures) identical? [**Yes / No**]
2. If they are identical, the WL test cannot distinguish them (Likely Isomorphic).

Part 3: When WL Fails

Consider L_1 (Hexagon) and L_2 (Two Triangles). All labels "1".



L_1



L_2

Perform the same Message Passing process as Part 1, and fill in the table below.

Signature Found	Number of nodes in L_1	Number of nodes in L_2
`(1, {1, 1})`		

See if the counts in the table are identical (i.e., the WL test cannot distinguish them) or not (i.e., the WL test can distinguish them). What does this tell us about the power of the WL test?

Part 4: The Neural Connection (Mapping WL to GCN)

Great job working through the Weisfeiler-Lehman (WL) test by hand! Now let's see how this classic, step-by-step coloring idea shows up inside a Graph Convolutional Network (GCN). You can think of a GCN as a kind of "smooth" or "continuous" version of the WL process.

A typical GCN layer updates the features h_v of node v with the following recipe:

$$h_v^{(new)} = \sigma\left(W \cdot \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_v d_u}} h_u\right)$$

Notation:

- h_v = feature vector of node v (e.g., a vector like $[0.2, 1.5, -0.3]$)
- $\mathcal{N}(v)$ = the set of neighbors of node v
- d_v = degree of node v (number of neighbors)
- W = weight matrix (learnable parameters that transform features)
- σ = activation function (e.g., ReLU, sigmoid: adds nonlinearity)

Let's make some connections between the parts of this GCN equation and the steps you took in the WL test:

1. **The "Collection" Step:** In the WL test, you "collected" the labels of neighbors into a bag (a multiset), like {A, B, B}. Which part of the formula above does something similar?
   Answer: _____

2. **The "Hashing" Step:** Remember how the WL test assigned a new color (ID) by looking up the unique neighbor patterns? In neural nets, we don't use lookup tables—instead, which component here "mixes and matches" those patterns into new features?
   Answer: _____

3. **The "Color" Step:** While the WL test spit out new integer colors (like 3, 7, or 11), GCNs make continuous vectors. The final step is deciding when a neuron "lights up" for a certain pattern—which we do with a nonlinear activation!
   Answer: _____

Final Reflection

If a standard GCN is kind of a "continuous" version of the WL test, what does your finding in Part 3 (about the limitations) suggest about what GCNs can and can't distinguish?