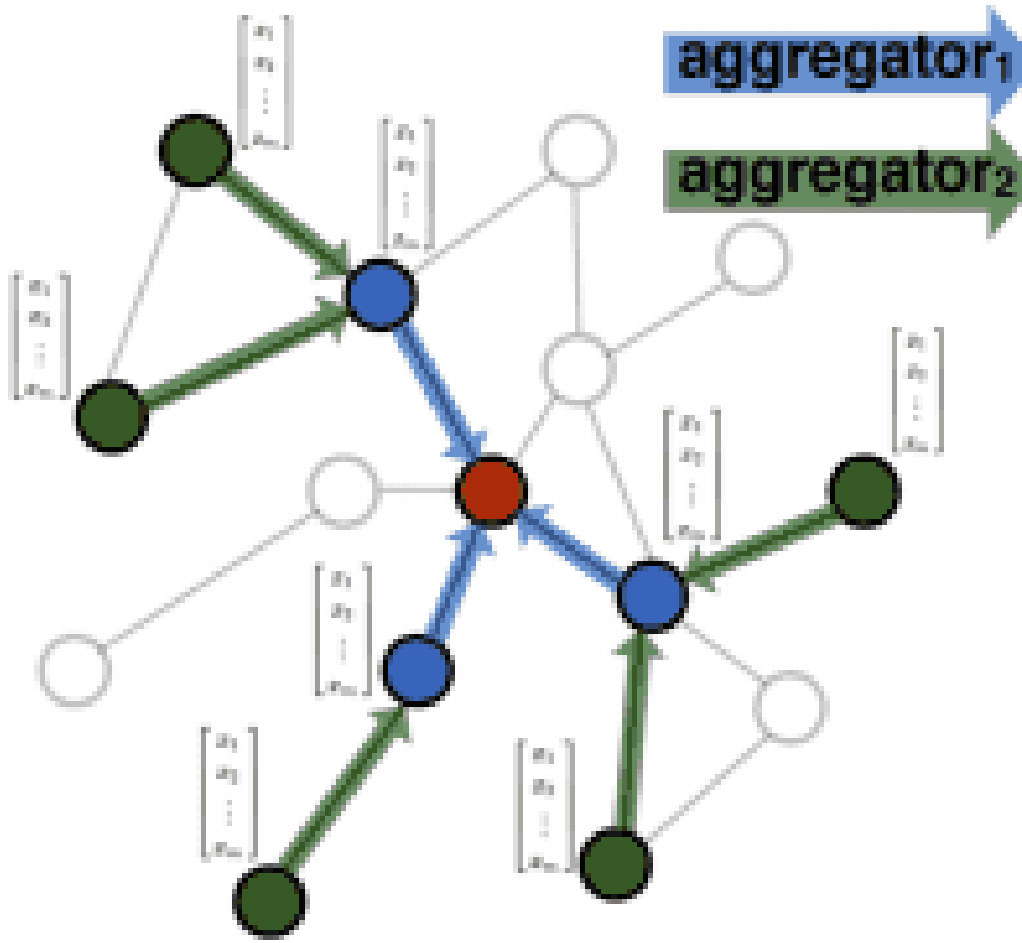
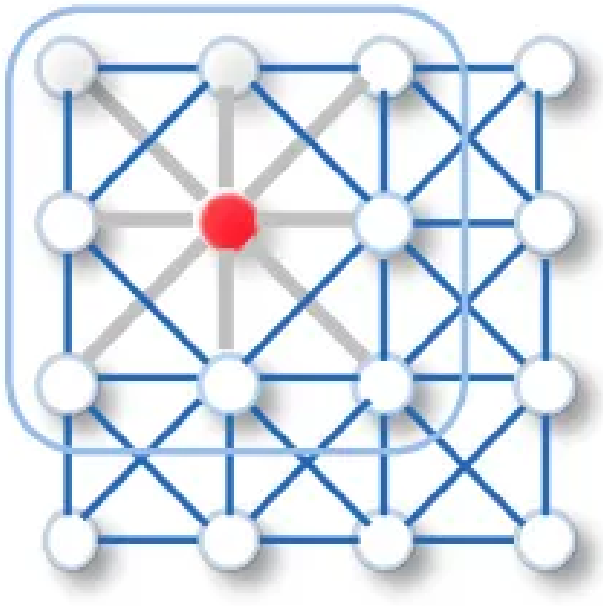
Graph Neural Networks 🧠

From Images to Graphs



Aggregate feature information from neighbors

Motivation: Extend Convolutional Neural Networks (CNNs) to Graphs 🤔



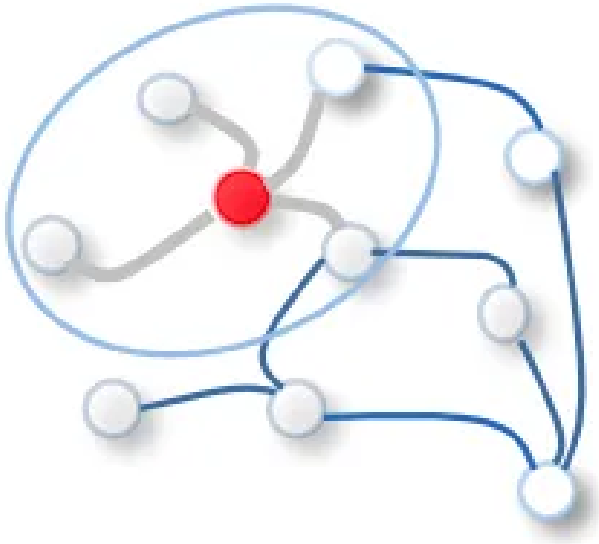
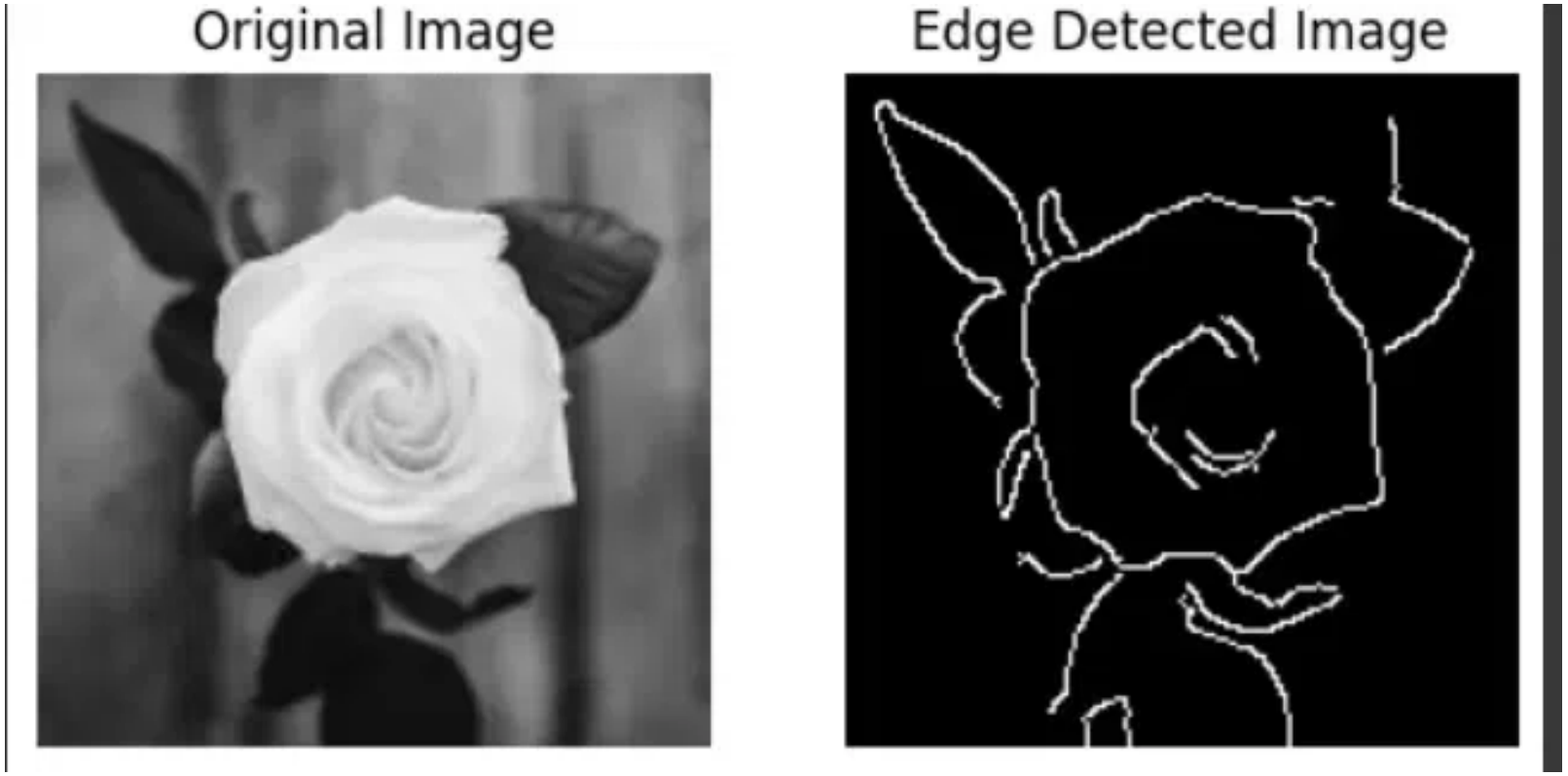


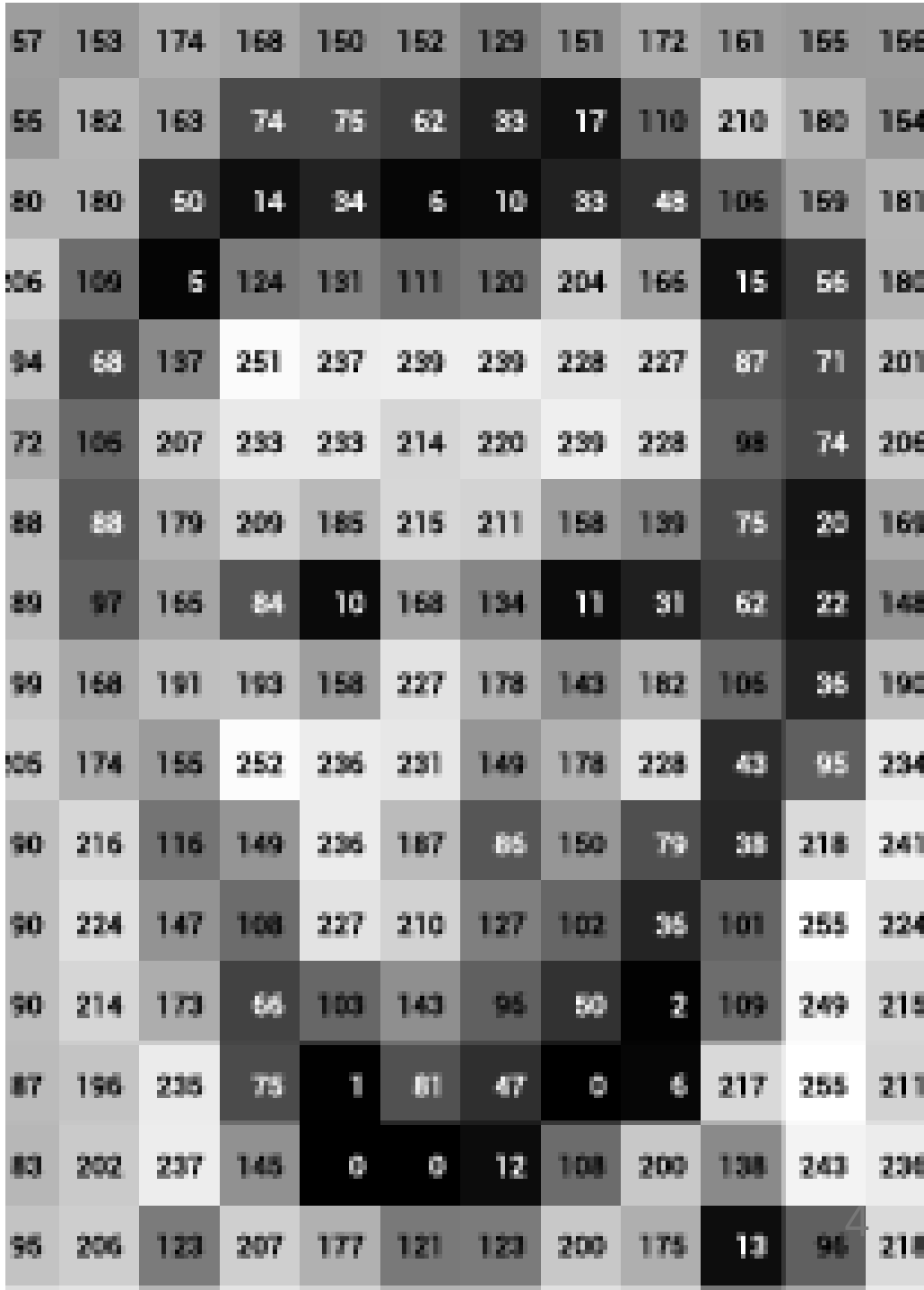
Image Processing Fundamentals: Edge Detection 🌄



Basic Image Processing 📸

- Image = 2D matrix of pixel values
- Each pixel represents brightness/color
- Example grayscale image:

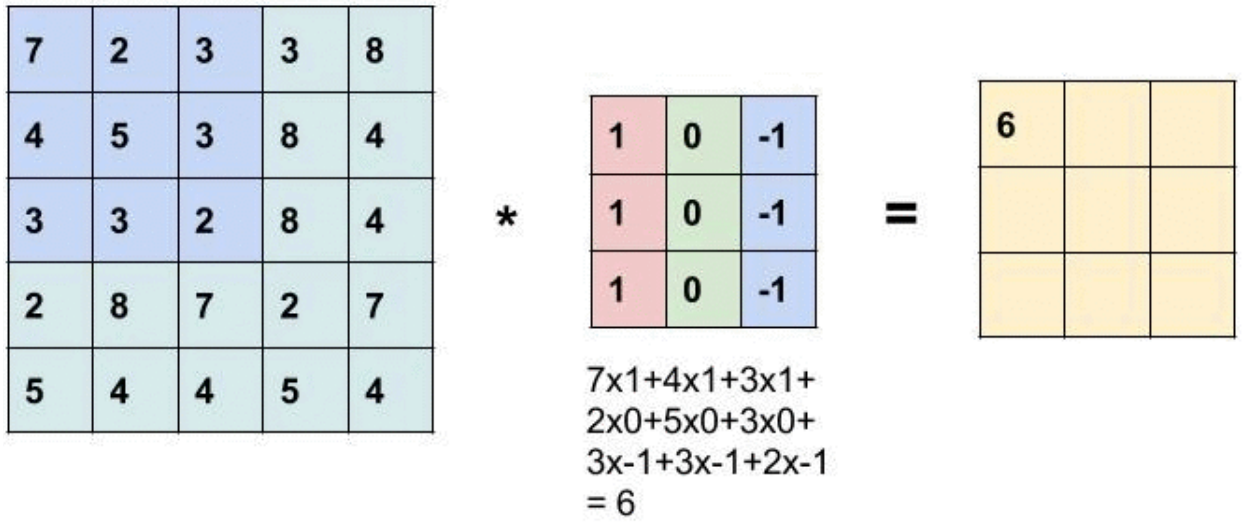
$$X = \begin{bmatrix} 10 & 10 & 80 & 10 & 10 & 10 \\ 10 & 10 & 80 & 10 & 10 & 10 \\ 10 & 10 & 80 & 10 & 10 & 10 \\ 10 & 10 & 80 & 10 & 10 & 10 \\ 10 & 10 & 80 & 10 & 10 & 10 \\ 10 & 10 & 80 & 10 & 10 & 10 \end{bmatrix}$$



Convolution: Spatial Domain 🔁

- Slide kernel over image
- Multiply and sum values
- Example kernel (vertical edge detection):
- Demo

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



Convolution is Complicated 😬

Example:

Suppose we have an image X and a kernel K as follows:

$$X = [X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6]$$
$$K = [K_1 \quad K_2 \quad K_3]$$

The convolution is given by

$$X * K = \sum_{i=1}^{6} X_i K_{7-i}$$

Or equivalently,

$$X * K = [X_1K_3 + X_2K_2 + X_3K_1 \quad X_2K_3 + X_3K_2 + X_4K_1 \quad X_3K_3 + X_4K_2 + X_5K_1 \quad X_4K_3 + X_5K_2 + X_6K_1]$$

Let's make it simpler using the convolution theorem!

What is the convolution theorem?

Suppose two functions f and g and their Fourier transforms F and G. Then,

$$\underbrace{(f * g)}_{\text{convolution}} \leftrightarrow \underbrace{(F \cdot G)}_{\text{multiplication}}$$

The Fourier transform is a one-to-one mapping between f and F (and g and G).

But what is the Fourier transform 🙃?

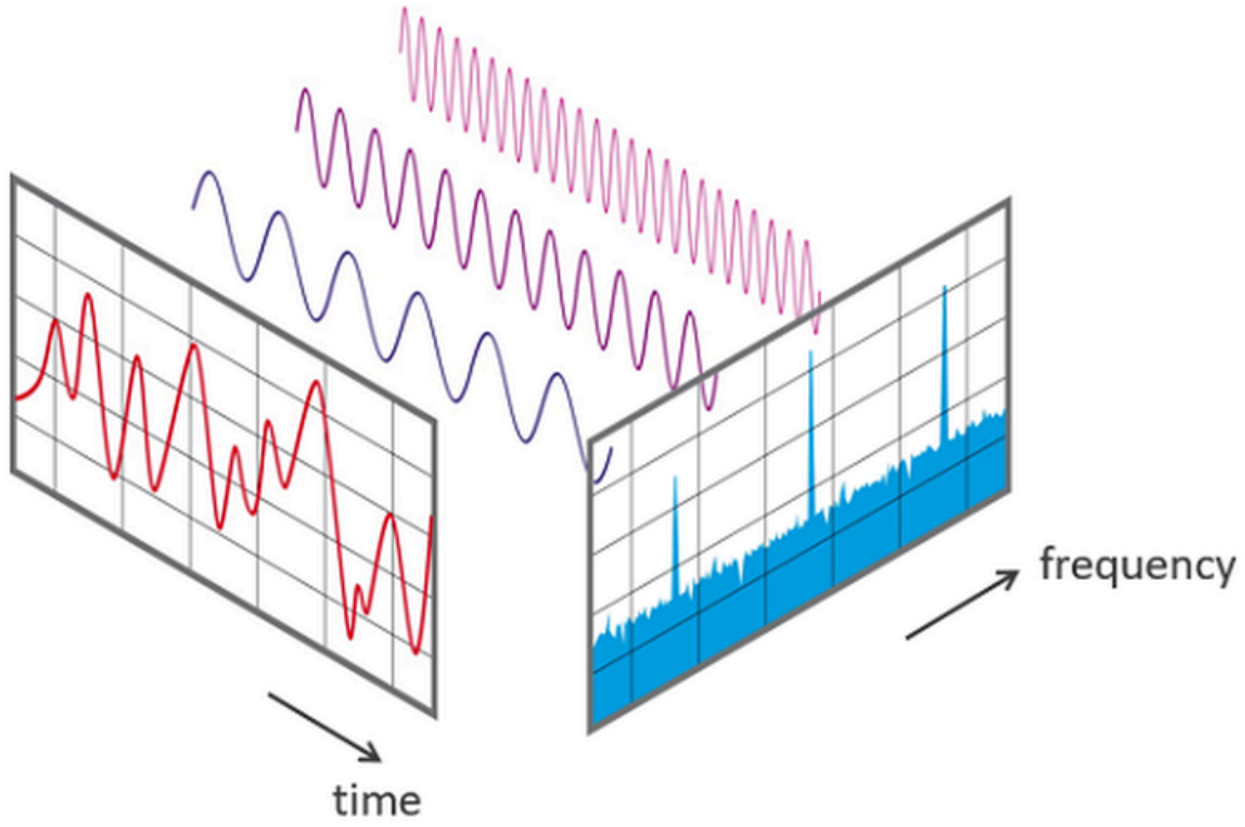
Fourier Transform: The Basics 🌊

Transform a signal from:

- Time/Space domain ➡️ Frequency domain

Key Concept:

- Any signal can be decomposed into sum of sine and cosine waves
- Each wave has specific frequency and amplitude

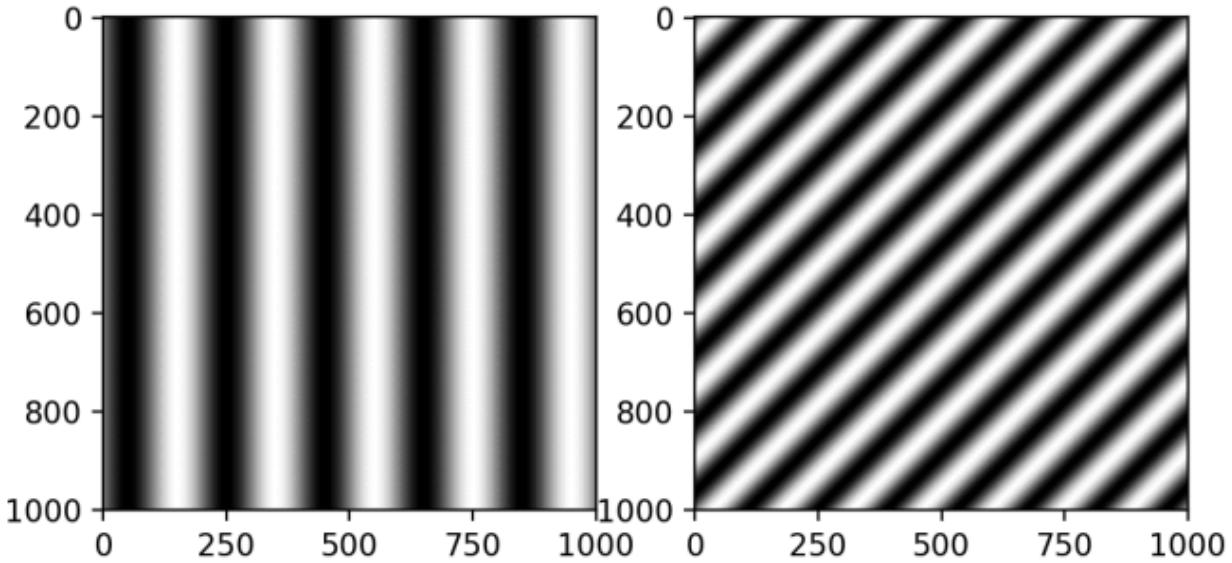


2D Fourier Transform 📊

2D Fourier Transform decomposes image into sum of *2D* waves.

$$\mathcal{F}(X)[h,w] = \sum_{k=0}^{H-1}\sum_{\ell=0}^{W-1} X[k,\ell] \cdot \underbrace{e^{-2\pi i\left(\frac{hk}{H}+\frac{w\ell}{W}\right)}}_{2D \text{ wave}}$$

For image X with size $H \times W$.



Edge Detection in Frequency Domain 🔍

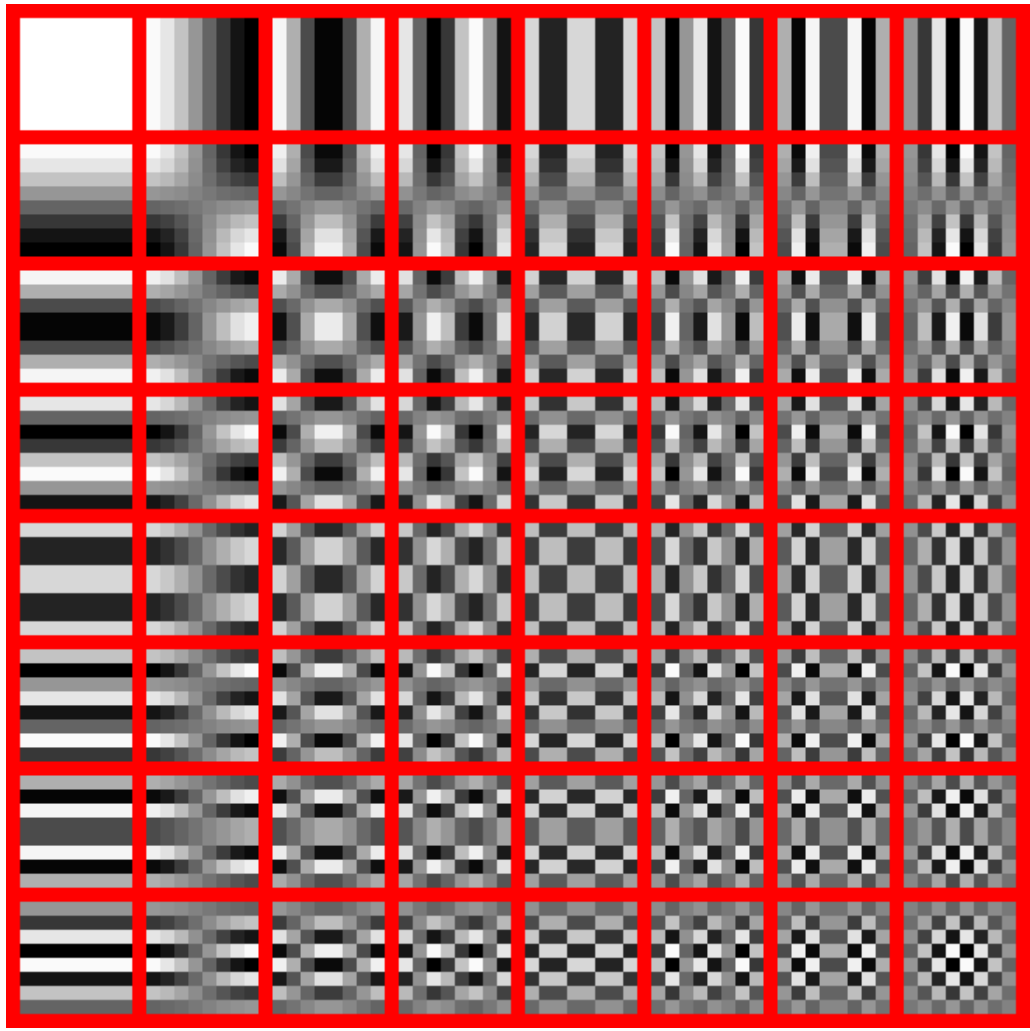
Original Image ➡️ Fourier Transform ➡️ Apply Filter ➡️ Inverse Transform

JPEG Compression 📸

1. Divide image into 8x8 blocks
2. Apply Discrete Cosine Transform (similar to Fourier)
3. Quantize frequencies
    - Keep low frequencies
    - Discard high frequencies
4. Encode efficiently

Benefits:

- Smaller file size
- Maintains visual quality
- Exploits human visual perception

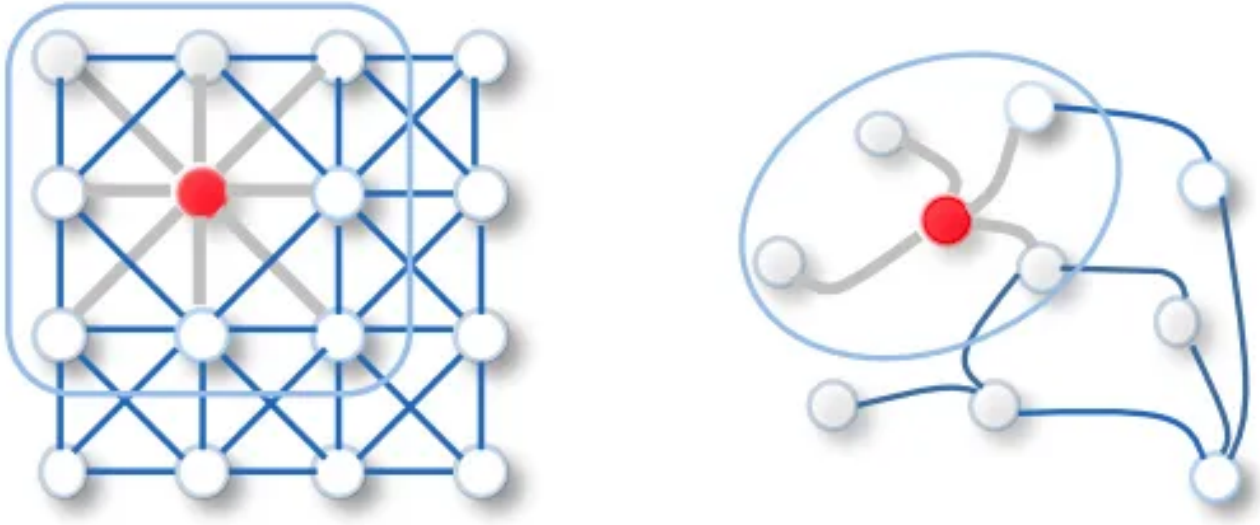


Coding Exercise

Coding Exercise

From Images to Graphs

- Image = 2D grid of pixels
- Through a convolution, a pixel value is influenced by its neighbors
- We can represent this neighborhood structure using a graph and define **convolutions on graphs**!



Graph Fourier Transform 🔁

- Just like signals, graphs have a "frequency" domain.
- Suppose we have a graph of N nodes, each node has a feature x_i.
- **The total variation** measures the smoothness of the node features:

$$J = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}(x_i - x_j)^2 = \mathbf{x}^\top \mathbf{L}\mathbf{x}$$

where $\mathbf{L}$: Graph Laplacian, x_i: Node features, A_{ij}: Adjacency matrix

Q: What x makes the total variation smallest (most smooth) and largest (most varying)? 🤔

The eigendecomposition of the Laplacian:

$$\mathbf{L}\mathbf{x} = \lambda\mathbf{x}$$

By multiplying both sides by $\mathbf{x}^\top$, we get

$$\mathbf{x}^\top\mathbf{L}\mathbf{x} = \lambda$$

This tells us that:

1. The eigenvectors with small eigenvalues represent **low-frequency** signals.
2. The eigenvectors with large eigenvalues represent **high-frequency** signals.

Decomposing the Total Variation

The total variation can be decomposed as follows ($\mathbf{u}_i$ is the eigenvector of the Laplacian):

$$J = \mathbf{x}^\top \mathbf{L}\mathbf{x} = \mathbf{x}^\top \left(\sum_{i=1}^{N} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top \right) \mathbf{x} = \sum_{i=1}^{N} \lambda_i (\mathbf{x}^\top \mathbf{u}_i)(\mathbf{u}_i^\top \mathbf{x})$$

$$= \sum_{i=1}^{N} \lambda_i \underbrace{||\mathbf{x}^\top \mathbf{u}_i||^2}_{\text{alignment between } \mathbf{x} \text{ and } \mathbf{u}_i}$$

Key Insight:

- The total variation is now decomposed into the sum of different frequency components $\lambda_i \cdot ||\mathbf{x}^\top \mathbf{u}_i||^2$.
- λ_i acts as *a filter (kernel)* that reinforces or passes the signal $\mathbf{x}^\top \mathbf{u}_i$.

Spectral Filtering 🎛️

Low-pass Filter:

$$h_{\text{low}}(\lambda) = \frac{1}{1 + \alpha\lambda}$$

High-pass Filter:

$$h_{\text{high}}(\lambda) = \frac{\alpha\lambda}{1 + \alpha\lambda}$$

Properties:

- Controls which frequencies pass
- Smooth or sharpen signals

Spectral Graph Convolution 🔄

Key idea: Let the kernel be a learnable parameter:

$$\mathbf{L}_{\text{learn}} = \sum_{k=1}^{K} \theta_k \mathbf{u}_k \mathbf{u}_k^\top$$

Layer operation:

$$\mathbf{x}^{(\ell+1)} = h\left(L_{\text{learn}} \mathbf{x}^{(\ell)}\right)$$

Where:

- θ_k: Learnable parameters
- h: Activation function
- K: Number of filters
- https://arxiv.org/abs/1312.6203

Multi-dimensional Features 📊

Let's consider the case where the node features are multi-dimensional. We want to map the f_{in}-dimensional features to f_{out}-dimensional features.

Key idea: Learn a separate filter for every combination of individual input and output features.

$$\mathbf{X}'_i = h\left(\sum_{j=1}^{f_{\text{in}}} L_{\text{learn}}^{(i,j)} \mathbf{X}_j^{(\ell)}\right), \quad \forall i \in \{1, \ldots, f_{\text{out}}\}$$

Where:

- $\mathbf{X} \in \mathbb{R}^{N \times f_{\text{in}}}$
- $\mathbf{X}' \in \mathbb{R}^{N \times f_{\text{out}}}$
- $L_{\text{learn}}^{(i,j)} = \sum_{k=1}^{K} \theta_{k,(i,j)} \mathbf{u}_k \mathbf{u}_k^\top$

Limitations of Spectral GNNs ⚠️

1. **Computational Cost**:
    - Eigendecomposition: $O(N^3)$
    - Prohibitive for large graphs
2. **Spatial Locality**:
    - Non-localized filters
    - Global node influence

ChebNet: A Bridge Solution

Key Idea: Approximate filters using Chebyshev polynomials

$$\mathbf{L}_{\text{learn}} \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})$$

where:

- T_k: Chebyshev polynomials, i.e., $T_0(x) = 1, T_1(x) = x$, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$
- $\tilde{\mathbf{L}}$: Scaled Laplacian, i.e., $\tilde{\mathbf{L}} = 2\lambda_{\max}^{-1}\mathbf{L} - \mathbf{I}$
- **Key property**:
  - A node can influence other nodes within K hops away.
  - Faster computation
- https://arxiv.org/abs/1606.09375

From Spectral to Spatial GNNs 🔄

Evolution of Graph Convolutional Networks:

1. Spectral GNNs (full eigen) ➡️ https://arxiv.org/abs/1312.6203
2. ChebNet (polynomial approx) ➡️ https://arxiv.org/abs/1606.09375
3. ▶️ ***GCN (first-order approximation)*** ➡️ https://arxiv.org/abs/1609.02907
4. Modern spatial GNNs

Graph Convolutional Networks 🌐

A Simple Yet Powerful Architecture

Kipf & Welling (2017)

Motivation 🤔

Problems with Previous Approaches:

- Spectral GNNs: Computationally expensive
- ChebNet: Still complex
- Need for simpler, scalable solution

Goals:

- Simple architecture
- Linear complexity
- Good performance

From ChebNet to GCN 🔄

ChebNet's Formulation:

$$h_{\theta'} * x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})x$$

First-Order Approximation:

$$g_{\theta'} * x \approx \theta'_0 x + \theta'_1 (L - I_N)x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

Further Simplification $\theta = \theta_0 = -\theta_1$:

$$g_\theta * x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x$$

Recipe of GCN 📊

- **Step 1**: Add self-loops to the adjacency matrix.

$$A' = A + I_N$$

- **Step 2**: Compute the normalized adjacency matrix.

$$\tilde{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

  where $\tilde{D}_{ii} = \sum_j A'$ is the degree matrix of A'.

- **Step 3**: Perform the convolution.

$$Z = \tilde{A} X^{(\ell)}$$

- **Step 4**: Let different dimensions communicate with each other.

$$Z' = Z W^{(\ell)}$$

- **Step 5**: Apply non-linear activation.

$$X^{(\ell+1)} = \sigma(Z')$$

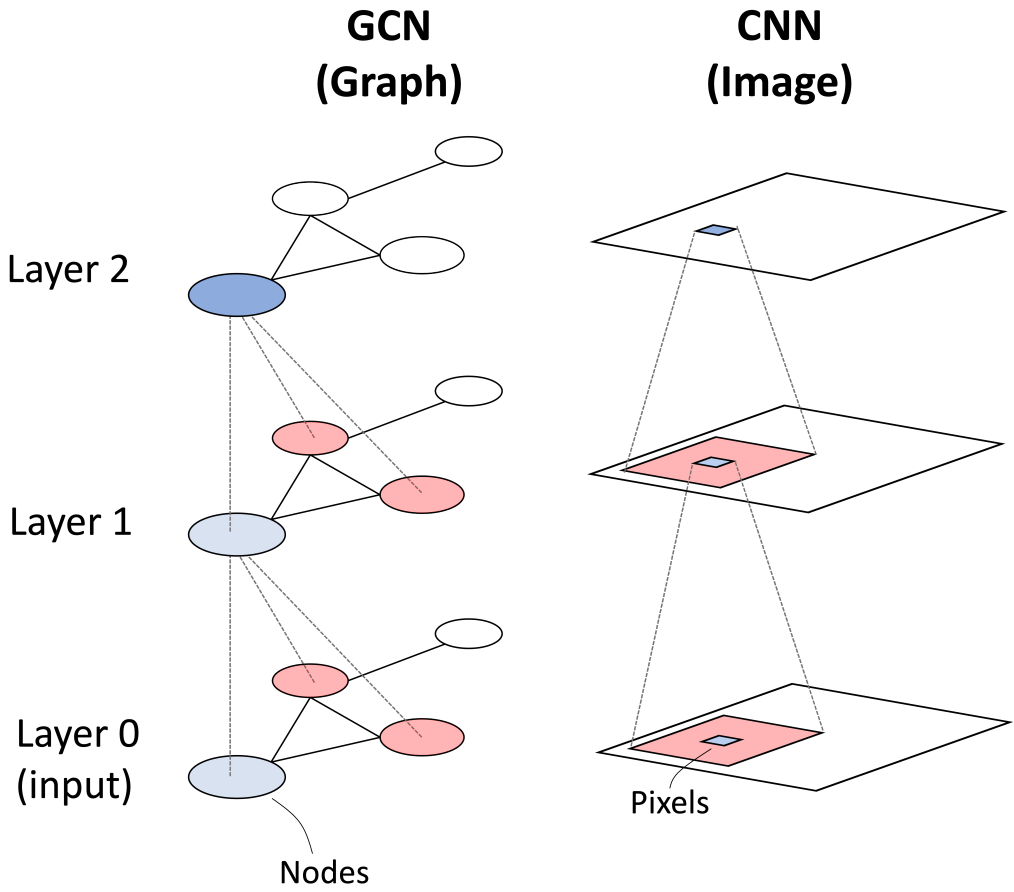
Renormalization Trick 🔧

- GCN is a powerful when it is deep (multiple convolutions).
- But, as the depth increases, the training becomes unstable due to **vanishing/exploding gradients**.

Solution: Add self-connections with renormalization:

$$\tilde{A} = A + I_N$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$



Extensions and Variants 🔄

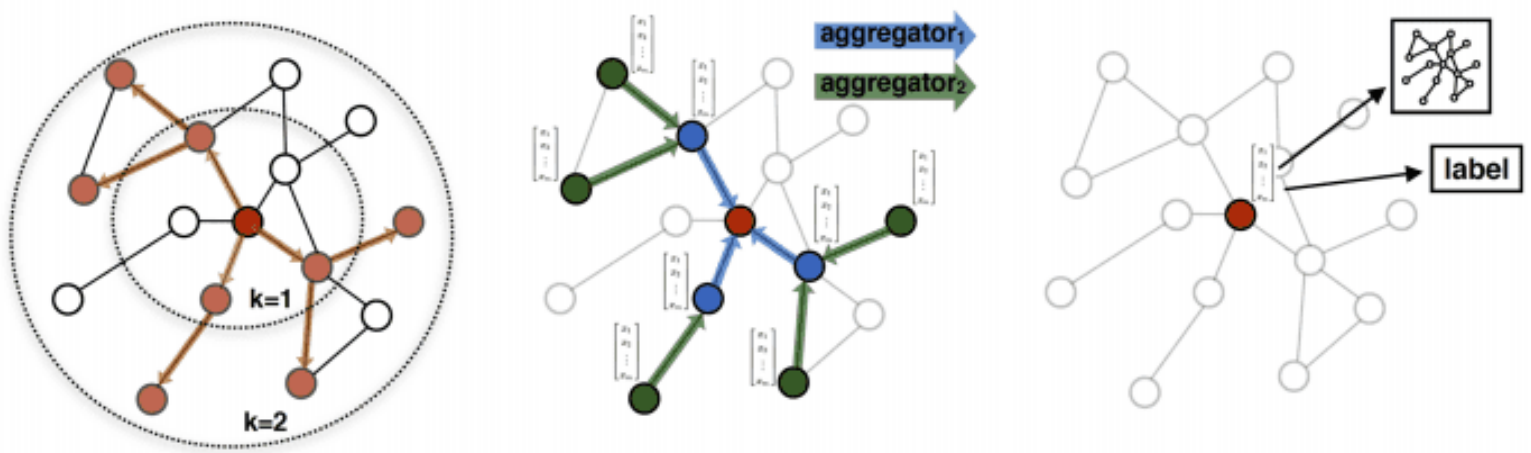
1. **GraphSAGE**
    - Sampling-based approach
    - Inductive capability
2. **GAT**
    - Attention mechanisms
    - Weighted aggregation
3. **GIN**
    - Theoretically more powerful
    - WL test equivalent

GraphSAGE 🎯

- Previous GNNs are transductive: require the entire graph structure during training and cannot generalize to unseen nodes.
- GraphSAGE is **inductive**: can generalize to unseen nodes.

Key idea: Sample a fixed number of nodes within the neighborhood for each node.

- Allow for localized computation for each node.
- Learns transferable patterns



1. Sample neighborhood

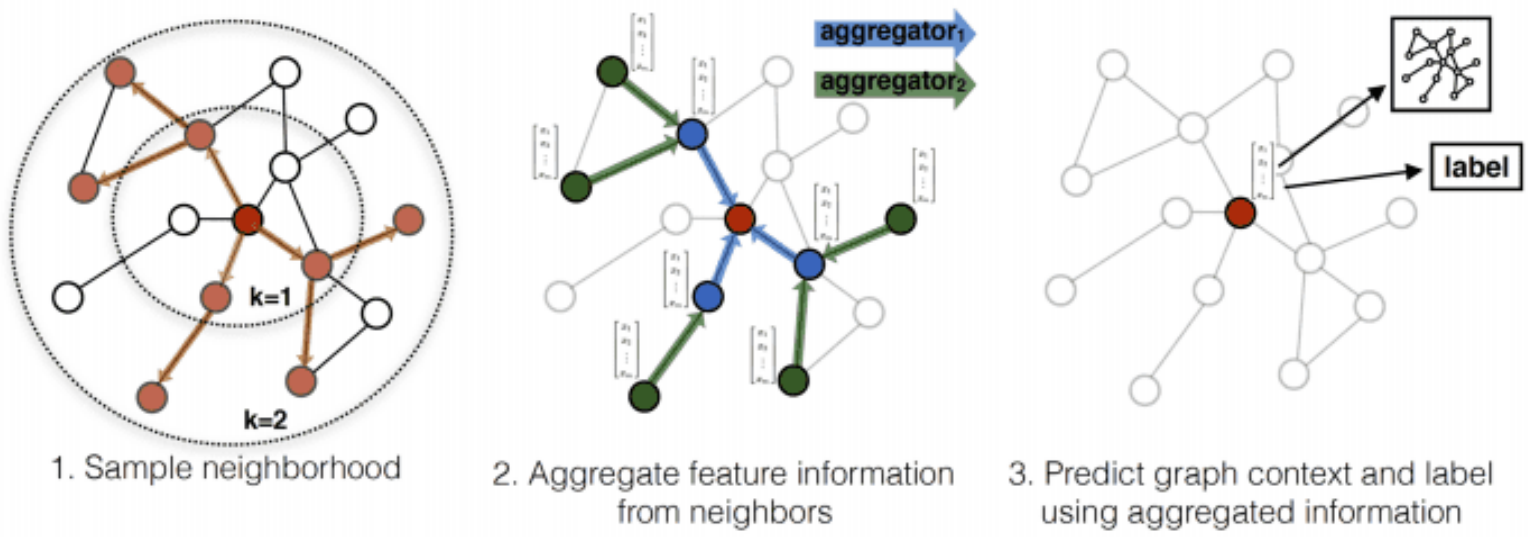
2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Another Key idea: Aggregation function

- Treat the self node feature and the neighborhood features differently

$$h_v^{(k+1)} = \text{CONCAT}\left(\underbrace{h_v^{(k)}}_{\text{self node feature}}, \underbrace{\text{AGGREGATE}}_{\text{Sum/Mean/Max/LTCM}}\left\{h_u^{(k)} \mid u \in \mathcal{N}(v)\right\}\right)$$



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Graph Attention Networks (GAT) 👀

Attention Mechanism:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})}$$

$$e_{ij} = \text{LeakyReLU}(\vec{a}^{\top}[\underbrace{Wh_i^{(k)}}_{\text{self node feature}}, \underbrace{Wh_j^{(k)}}_{\text{neighbor node feature}}])$$

Features:

- Learn importance of neighbors
- Multiple attention heads
- Dynamic edge weights

Graph Isomorphism Network (GIN) 🔍

Based on Weisfeiler-Lehman Test:

$$h_v^{(k+1)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)})\cdot h_v^{(k)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k)}\right)$$

Key Features:

- Maximally powerful GNNs
- Theoretical connections to graph isomorphism
- Learnable or fixed ϵ

Summary 📝

Key Takeaways:

- GNNs extend CNNs to irregular structures
- Multiple architectures available:
    - GCN: Simple and effective
    - GraphSAGE: Scalable and inductive
    - GAT: Attention-based
    - GIN: Theoretically powerful

Future Directions:

- Scalability
- Expressiveness
- Applications