Pooling in CNNs
Pooling in CNNs
Pooling in CNNs
Pooling in CNNs
Pooling in CNNs

Diagram showing pooling in a CNN.
Pooling in CNNs
Roadmap

Things we are going to cover:

- A “message passing” for pooling
- Methods
- Global pooling
- Open questions
• Graph: nodes connected by edges;
• $A$, $N \times N$ adjacency matrix;
• $D = \text{diag}([d_1, \ldots, d_N])$, diagonal degree matrix;
• $X = \begin{bmatrix} x_1, \ldots, x_N \end{bmatrix}$, $x_i \in \mathbb{R}^F$, node attributes or “graph signal”;
• $e_{ij} \in \mathbb{R}^S$, edge attribute for edge $i \to j$,
Strategy 1: aggregate same attributes (Candy Crush pooling).
Graph coarsening by example

Strategy 2: aggregate cliques.
Strategy 3: keep only some types/colors.
Three main questions [1]

1. How to identify groups of related nodes?
2. How to get new node attributes from the groups?
3. How to connect the new nodes?

Step 1: Select
Selecting nodes

Example 1: partition.

\{ \textcircled{blue}, \textcircled{red}, \textcircled{orange} \} \quad \{ \textcircled{grey}, \textcircled{green}, \textcircled{pink} \} \quad \{ \textcircled{yellow} \}
Selecting nodes

Example 1: partition.
\{ \text{nodes 1, 2, 3} \} \quad \{ \text{nodes 4, 5} \} \quad \{ \text{node 6} \}

Example 2: cover (possible overlaps).
\{ \text{nodes 1, 2, 3} \} \quad \{ \text{nodes 4, 5} \} \quad \{ \text{node 6} \}

Example 3: sparse.
Selecting nodes

Example 1: partition.
\[
\{ \text{color 1} \} \quad \{ \text{color 2} \} \quad \{ \text{color 3} \}
\]

Example 2: cover (possible overlaps).
\[
\{ \text{color 1} \} \quad \{ \text{color 2} \} \quad \{ \text{color 3} \}
\]

Example 3: sparse.
\[
\{ \text{color 1} \} \quad \{ \text{color 2} \} \quad \{ \text{color 3} \}
\]
The **selection** stage computes $K$ supernodes:

$$\text{Sel} : \mathcal{G} \mapsto S = \{S_1, \ldots, S_K\}.$$  

Each supernode is a set of nodes associated with a score:

$$S_k = \{(x_i, s_i) \mid s_i \in \mathbb{R}_{>0}\},$$
Spectral clustering [3]

The low-frequency eigenvectors naturally cluster the nodes.

Spectral clustering [3]

The low-frequency eigenvectors naturally cluster the nodes.

Idea: run k-means clustering (or similar) using the first few eigenvectors.

Spectral clustering [3]

The low-frequency eigenvectors naturally cluster the nodes.

**Idea:** run k-means clustering (or similar) using the first few eigenvectors.

Equivalent to finding the **minimum normalized k-cut** [2].

---


Mincut vs. Maxcut

Mincut
Node decimation [5]

**Alternative**: use the highest-frequency eigenvector to do something similar to a regular subsampling.

---


Node decimation [5]

**Alternative**: use the highest-frequency eigenvector to do something similar to a regular subsampling.

Equivalent to finding the maximum 2-cut [4].

---


Mincut vs. Maxcut

Mincut

Maxcut
Some problems

Problems with spectral methods:

- Computing eigenvectors is expensive \((O(N^3))\);
- They do not consider attributes.

But we get the general idea...
Step 2: Reduce
The reduction stage aggregates the supernodes in a permutation-invariant way:

\[ \text{Red} : \mathcal{G}, S_k \mapsto x'_k \]

Typical approach is to take a weighted sum (weights given by the scores in the supernodes):

\[ X' = SX \quad (\in \mathbb{R}^{K \times F}) \]
Step 3: Connect
Connecting supernodes

The **connection** function decides whether two supernodes are connected (and, in case, computes the associated attributes):

\[ \text{Con} : G, S_k, S_l \mapsto e'_{kl} \]

Typical approach is again to take a **weighted sum** of edges between two supernodes:

\[ A' = SAS^\top \ (\in \mathbb{R}^{K \times K}) \]
Putting everything together:

\[ S = \{S_k\}_{k=1:K} = \text{Sel}(G); \]

Selection

\[ \mathcal{X}' = \{\text{Red}(G, S_k)\}_{k=1:K}; \]

Reduction

\[ \mathcal{E}' = \{\text{Con}(G, S_k, S_l)\}_{k,l=1:K}; \]

Connection

---

Methods
Pooling methods

A few ideas:

1. **Graclus** [6]: visit nodes randomly, merge pairs that maximize \( \frac{a_{ij}}{w_i} + \frac{a_{ij}}{w_j} \); \(^1\)

   In [7], they reduce supernodes with element-wise max.

---


\(^1\) \(w_i\) is a “weight” assigned to the node, e.g., its degree.
A few ideas:

1. **Graclus** [6]: visit nodes randomly, merge pairs that maximize $\frac{a_{ij}}{w_i} + \frac{a_{ij}}{w_j}$; ¹

   In [7], they reduce supernodes with element-wise max.

2. **Clique Pooling** [8]: merge together cliques.

¹ $w_i$ is a “weight” assigned to the node, e.g., its degree.
Pooling methods

A few ideas:

1. **Graclus** [6]: visit nodes randomly, merge pairs that maximize \( \frac{a_{ij}}{w_i} + \frac{a_{ij}}{w_j} \).
   
   In [7], they reduce supernodes with element-wise max.

2. **Clique Pooling** [8]: merge together cliques.

3. **LaPool** [9]: select “leaders” that have higher local variation \( \|LX\| \) w.r.t. all their neighbors. Create clusters by assigning nodes to nearest leader.

---


\( w_i \) is a “weight” assigned to the node, e.g., its degree.
Key idea: learn to output $S^T$ by giving node features $X$ as input to a neural network.

- **DiffPool** [10]: GNN for $S^T$, regularize with “link prediction” loss;

$$\phi(X) = S^T = [\mathbb{R}^{N \times K}]$$

Key idea: learn to output $S^T$ by giving node features $X$ as input to a neural network.

- **DiffPool** [10]: GNN for $S^T$, regularize with “link prediction” loss;
- **MinCutPool** [11]: MLP for $S^T$, regularize with “minimum cut” loss (same objective as spectral clustering);

$$\phi(X) = S^T = \begin{bmatrix} \mathbb{R}^{N \times K} \end{bmatrix}$$

Key idea: learn to output $S^T$ by giving node features $X$ as input to a neural network.

- **DiffPool** [10]: GNN for $S^T$, regularize with “link prediction” loss;
- **MinCutPool** [11]: MLP for $S^T$, regularize with “minimum cut” loss (same objective as spectral clustering);
- **Deep Graph Mapper** [12]: combine Mapper [13] and GCN [14] to compute clusters.

$\phi(X) = S^T = \in \mathbb{R}^{N \times K}$

---


- Select: \( S^\top = \text{MLP}(X) \)
- Reduce: \( X' = SX \)
- Connect: \( A' = SAS^\top \)
- MinCut loss: \( \mathcal{L}_c = -\frac{\text{Tr}(SAS^\top)}{\text{Tr}(SDS^\top)} \)
- Orthogonality loss:
  \[
  \mathcal{L}_o = \left\| \frac{SS^\top}{\|SS^\top\|_F} - \frac{I_K}{\sqrt{K}} \right\|_F
  \]

Top-K methods

**Problem**: computing $S$ with neural network is likely to yield a very dense matrix.

Can we learn a sparse selection?
Top-K methods

\[ X = \begin{bmatrix} \phi(X) \end{bmatrix} \]

Features
Top-K methods

\[ X = \phi(X) = \text{Sorted Top-K} \]
Top-K methods

$$X = \begin{array}{c}
\text{Features} \\
\end{array}$$

$$\phi(X) = \begin{array}{c}
\text{Scores} \\
\text{Sorted}
\end{array}$$
Top-K methods

\[
X = \begin{bmatrix}
\text{Features}
\end{bmatrix}
\]
\[
\phi(X) = \begin{bmatrix}
\text{Scores}
\end{bmatrix}
\]

Top-K

Sorted
Top-K methods

\[ X = \begin{array}{c}
\text{Features} \\
\end{array} \]

\[ \phi(X) = \begin{array}{c}
\text{Scores} \\
\text{Sorted} \\
\text{Top-K} \\
\end{array} \]
Top-K methods

Different ways of computing the selection indices $i$:

- Select with a simple linear projection $\theta \in \mathbb{R}^F$ [15];
- Select with a GNN [16];
- Train the selection with a supervised objective (needs ground truth for which nodes to keep) [17].

Top-K methods

Reduce: $X' = X_i$ - Connect: $A' = A_{i,i}$

Problems:

- Top-k selection is **non-differentiable** (no way of training $\phi$). Solved by **gating** (multiplying) the node attributes with the scores.
- Graph is likely to be **disconnected** or simply **cut off** (like in the image on the right). Not really solvable...
Main properties of pooling operators

- **Dense vs. Sparse**: how many nodes are selected for the supernodes ($O(N)$ vs. $O(1)$);
- **Fixed vs. Adaptive**: how many supernodes does the selection compute (e.g., DiffPool/MinCut are fixed);
- **Model-free vs. Model-based**: learn to pool from data or based on a *model* of how to pool;
Global pooling
Global Pooling

In CNNs, after convolution, we usually flatten out the images to give a vector as input to a MLP:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\]

The graph equivalent must be \textit{invariant to permutations} of the nodes:
Global Pooling

Once again, there are many ways to do this:

- Sum, average, product, max;
- Weighted sum with attention [18];
- Sum and then apply a neural network [19];

Open questions

- Does pooling really work? [20]
  - Which task benefit from it, a priori?
- Can we make a pooling layer that is dense, model-free, and adaptive?


