Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

by M. Defferrard, X. Bresson, P. Vandergheynst, 2016

Paper presentation

Hubert Leterme

July 8th, 2020

Table of Contents

- **Convolutional neural networks (CNNs)**, introduced in 1989 by Yann LeCun [1], are able to learn **local stationary structures** which are composed into **multi-scale patterns**. They led to **breakthroughs in image, video and sound recognition tasks**.
- Defferard and colleagues [2] introduced a **generalization of CNNs to graphs**, which encode **complex data structures** lying on irregular or non-euclidean domains.

Introduction

- **Convolutional neural networks (CNNs)**, introduced in 1989 by Yann LeCun [1], are able to learn **local stationary structures** which are composed into **multi-scale patterns**. They led to **breakthroughs in image, video and sound recognition tasks**.
- Defferard and colleagues [2] introduced a **generalization of CNNs to graphs**, which encode **complex data structures** lying on irregular or non-euclidean domains.
- Main challenges:
  - Construct a **convolution operator on irregular grids**;
  - Design **strictly localized filters**, as in standard CNNs;
  - Compute forward- and backward-propagation with a **linear complexity** w.r.t. the filter support's size and the number of edges;
  - Design an **efficient pooling operator** (which yields smaller graphs by grouping vertices together);
  - Obtain **high experimental performance** on both standard image and more complex data recognition tasks.

Section 2

Background on CNNs

Description of a convolutional layer

Forward-propagation:

$$\mathbf{X}_s \xrightarrow{\text{Conv}(\mathbf{W})} \mathbf{Y}_s \xrightarrow{\text{ReLu}} \mathbf{A}_s \xrightarrow{\text{Pool}} \mathbf{Z}_s$$

with:

- $\mathbf{X}_s \in \mathbb{R}^{C\times M\times M}$, $\mathbf{Y}_s, \mathbf{A}_s \in \mathbb{R}^{D\times M\times M}$ and $\mathbf{Z}_s \in \mathbb{R}^{D\times N\times N}$ (feature maps for the s-th training sample);
- $\mathbf{W} \in \mathbb{R}^{D\times C\times \mu\times \mu}$ (convolution kernels – **trainable parameters**);
- $C, D > 0$ (number of input and output feature maps);
- $M, N > 0$ such that $N < M$ (sizes of the input and output feature maps);
- $\mu \ll M$ (size of the convolution kernels).

- Conv: convolutional layer (see slide 7);
- ReLu: rectified linear unit (non-linear pointwise operation);
- Pool: pooling operator (e.g. max pooling).

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0\,..\,D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

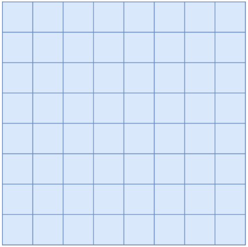


Figure: c-th input feature map $\mathbf{X}_{s,c} \in \mathbb{R}^{M \times M}$, for a given $c \in [0\,..\,C-1]$

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0\,..\,D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

0	0	0	0	0	0	0	0	0	0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0									0
0	0	0	0	0	0	0	0	0	0

Figure: c-th input feature map, extended with zeros (padding)

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0 \,..\, D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

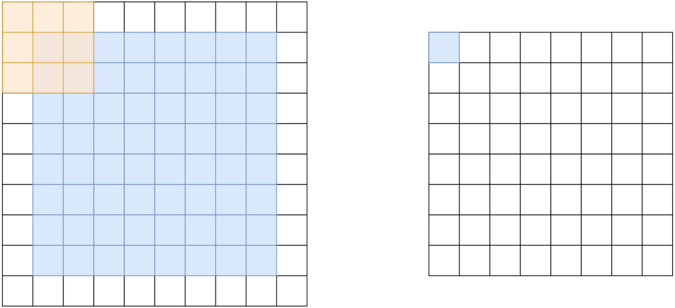


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0\,..\,D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

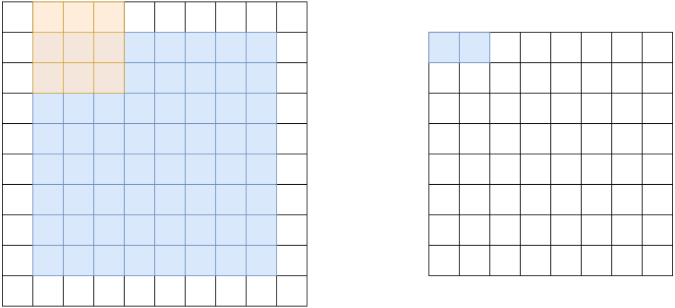


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0\,..\,D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

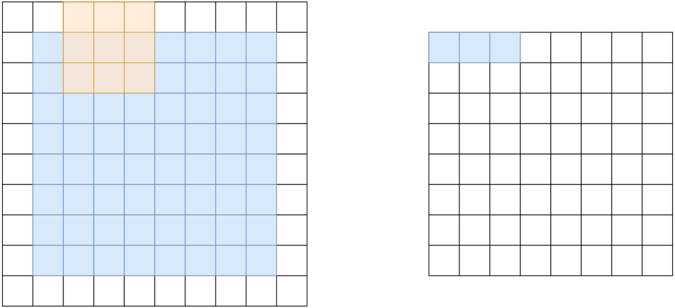


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0\,..\,D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

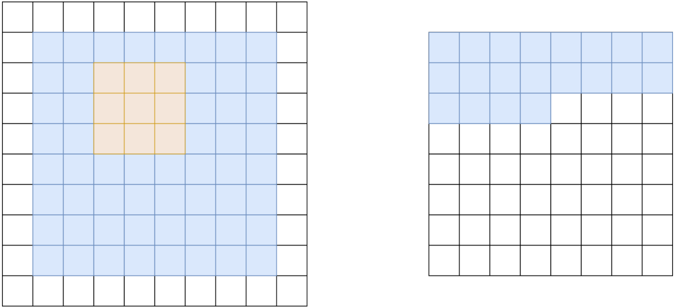


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0 \,..\, D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

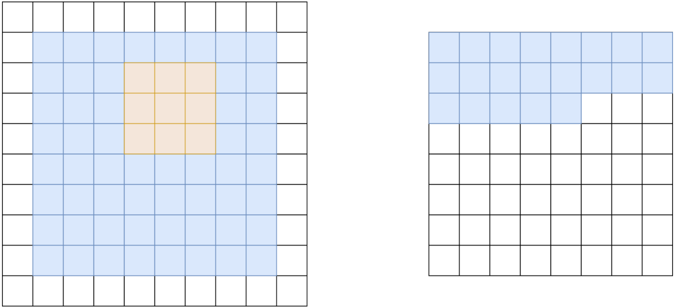


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Computation of the d-th output feature map, for $d \in [0 \,..\, D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

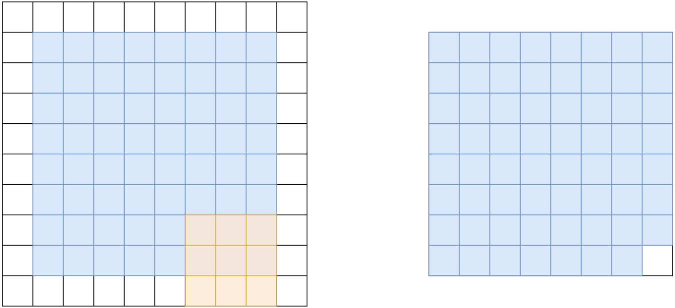


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Illustration of a convolutional layer

Computation of the d-th output feature map, for $d \in [0 \,..\, D-1]$:

$$\mathbf{Y}_{s,d} = b_d + \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

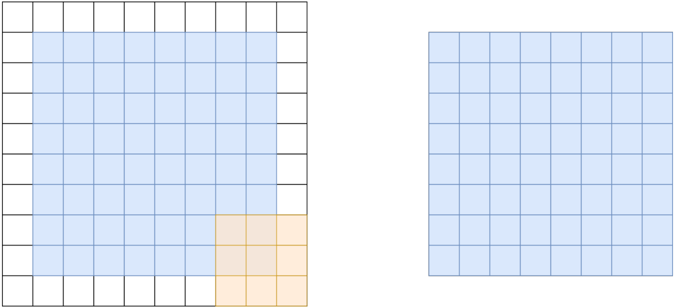


Figure: Cross-correlation mapping the c-th input $\mathbf{X}_{s,c}$ (left, in blue) to the d-th output, using the kernel $\mathbf{W}_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(\mathbf{W}_{d,c} \star \mathbf{X}_{s,c})$.

Convolution versus cross-correlation

- Matrix convolution product:

$$(\mathbf{U} * \mathbf{V})[m,n] = \sum_{i\in\mathbb{Z}} \sum_{j\in\mathbb{Z}} \mathbf{U}[i,j] \cdot \mathbf{V}[m-i, n-j]$$

- Cross-correlation:

$$(\mathbf{U} \star \mathbf{V})[m,n] = \sum_{i\in\mathbb{Z}} \sum_{j\in\mathbb{Z}} \mathbf{U}[i,j] \cdot \mathbf{V}[m+i, n+j]$$

Proposition 2.1

$$\mathbf{U} \star \mathbf{V} = \overline{\mathbf{U}} * \mathbf{V}$$

where $\overline{\mathbf{U}}[m,n] = \mathbf{U}[-m,-n]$.

Let's assume the following values have already been computed:

- E: loss computed over a minibatch of S samples;
- $\left\{\nabla_{(\mathbf{Y}_{s,d})}E \middle| s \in [0\mathinner{..}S-1]\,, d \in [0\mathinner{..}D-1]\right\}$: gradients w.r.t. the outputs;

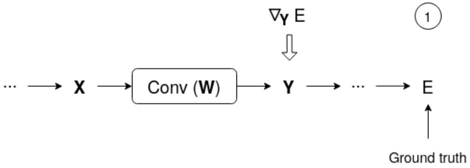
Then, backpropagates the gradient in $\mathcal{O}(SCD \cdot \mu^2 N^2)$:

$$\nabla_{(\mathbf{W}_{d,c})}E = \sum_{s=1}^{S} \left(\nabla_{(\mathbf{Y}_{s,d})}E\right) \star \mathbf{X}_{s,c}$$

$$\nabla_{(\mathbf{X}_{s,c})}E = \sum_{d=1}^{D} \left(\nabla_{(\mathbf{Y}_{s,d})}E\right) * \mathbf{W}_{d,c}$$

Finally, update the weights using stochastic gradient descent:

$$\mathbf{W}_{d,c} \leftarrow \left(\mathbf{W}_{d,c} - \eta \cdot \nabla_{(\mathbf{W}_{d,c})}E\right)$$



Let's assume the following values have already been computed:

- E: loss computed over a minibatch of S samples;
- $\left\{\nabla_{(\mathbf{Y}_{s,d})}E \middle| s \in [0\,..\,S-1], d \in [0\,..\,D-1]\right\}$: gradients w.r.t. the outputs;

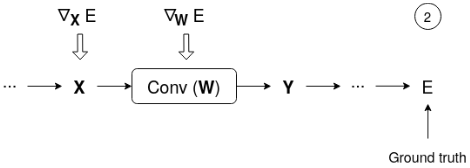
Then, backpropagates the gradient in $\mathcal{O}(SCD \cdot \mu^2 N^2)$:

$$\nabla_{(\mathbf{W}_{d,c})}E = \sum_{s=1}^{S}\left(\nabla_{(\mathbf{Y}_{s,d})}E\right) \star \mathbf{X}_{s,c}$$

$$\nabla_{(\mathbf{X}_{s,c})}E = \sum_{d=1}^{D}\left(\nabla_{(\mathbf{Y}_{s,d})}E\right) * \mathbf{W}_{d,c}$$

Finally, update the weights using stochastic gradient descent:

$$\mathbf{W}_{d,c} \leftarrow \left(\mathbf{W}_{d,c} - \eta \cdot \nabla_{(\mathbf{W}_{d,c})}E\right)$$



For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j \in \{0,1\}} \Big(\mathbf{Y}_{s,d}[2m+i,\, 2n+j] \Big)$$

with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2)\times(N/2)}$.

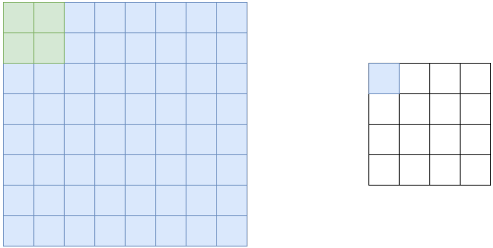
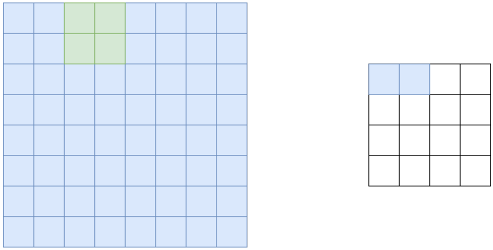


Illustration of a pooling layer
Example of max pooling with size 2 × 2

For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j \in \{0,1\}} \Big(\mathbf{Y}_{s,d}[2m+i,\, 2n+j] \Big)$$

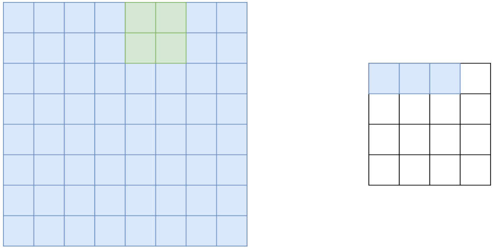
with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2)\times(N/2)}$.



For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \Big(\mathbf{Y}_{s,d}[2m + i,\, 2n + j] \Big)$$

with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$.



For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j \in \{0,1\}} \left(\mathbf{Y}_{s,d}[2m+i,\, 2n+j] \right)$$

with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2)\times(N/2)}$.

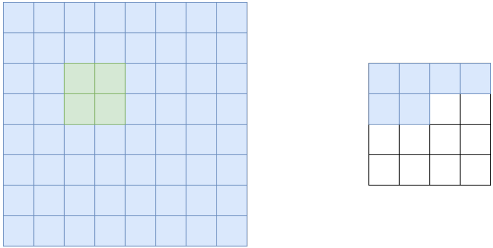
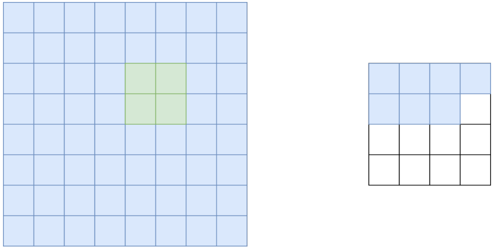


Illustration of a pooling layer
Example of max pooling with size 2×2

For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j \in \{0,1\}} \left(\mathbf{Y}_{s,d}[2m+i,\ 2n+j] \right)$$

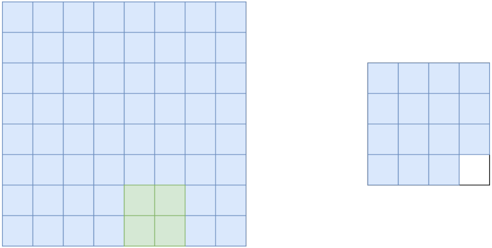
with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$.



For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j\in\{0,1\}} \left(\mathbf{Y}_{s,d}[2m+i,\ 2n+j]\right)$$

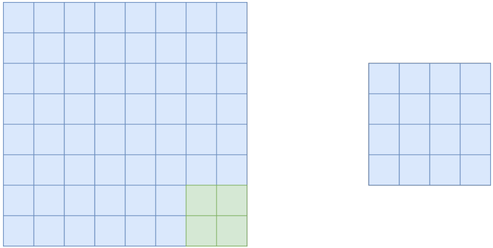
with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N\times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2)\times(N/2)}$.



For any sample s and any output d:

$$\mathbf{Z}_{s,d}[m,n] = \max_{i,j \in \{0,1\}} \left(\mathbf{Y}_{s,d}[2m+i,\, 2n+j] \right)$$

with $\mathbf{Y}_{s,d} \in \mathbb{R}^{N \times N}$ and $\mathbf{Z}_{s,d} \in \mathbb{R}^{(N/2)\times(N/2)}$.



Section 3

Convolution layers on graphs

Notations

- Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, with:
  - $|\mathcal{V}| = N$;
  - $\mathbf{A} \in \mathbb{R}^{N \times N}$ such that $\mathbf{A}_{ij} \neq 0 \implies (i, j) \in \mathcal{E}$.
- $\mathbf{L} \in \mathbb{R}^{N \times N}$: positive semidefinite reference matrix for $\mathcal{G}$;
- $\mathbf{U}, \mathbf{\Lambda} \in \mathbb{R}^{N \times N}$, with:
  - $\mathbf{U} = [\mathbf{u}_0, \dots, \mathbf{u}_{N-1}]$: eigenvectors of $\mathbf{L}$ (graph Fourier modes);
  - $\mathbf{\Lambda} = \mathrm{diag}(\lambda_0, \dots, \lambda_{N-1})$: eigenvalues of $\mathbf{L}$ (graph frequencies);

  such that $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$;
- Input signal $\mathbf{x} \in \mathbb{R}^N$, defined on the nodes of $\mathcal{G}$;
- $\hat{\mathbf{x}}$: graph Fourier transform of $\mathbf{x}$, such that $\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}$.

From classical to graph convolutions

Forward-propagation for any sample s and any output d:

Classical CNN:

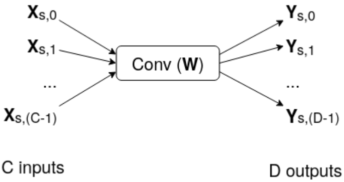
$$\mathbf{Y}_{s,d} = \sum_{c=0}^{C-1} (\mathbf{W}_{d,c} \star \mathbf{X}_{s,c}) = \sum_{c=0}^{C-1} \left(\overline{\mathbf{W}_{d,c}} * \mathbf{X}_{s,c}\right)$$

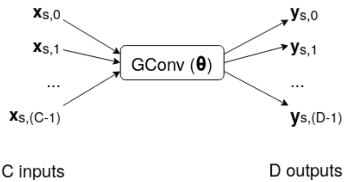
according to proposition 2.1.

Graph CNN:

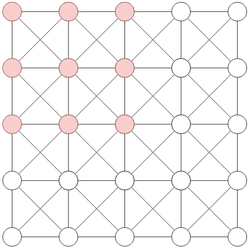
$$\mathbf{y}_{s,d} = \sum_{c=0}^{C-1} (\overline{\boldsymbol{\theta}_{d,c}} *_{(\mathcal{G})} \mathbf{x}_{s,c})$$

where $*_{(\mathcal{G})}$ has to be defined.

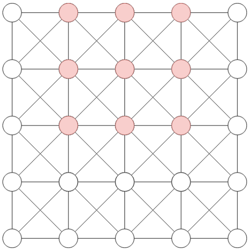




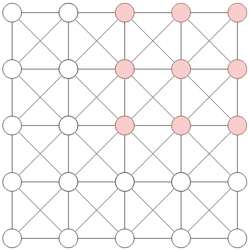
Issue with spatial convolution: **no unique definition of translation** on graphs.



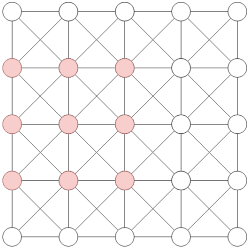
Issue with spatial convolution: **no unique definition of translation** on graphs.



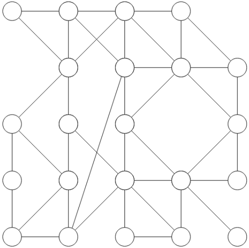
Issue with spatial convolution: **no unique definition of translation** on graphs.



Issue with spatial convolution: **no unique definition of translation** on graphs.



Issue with spatial convolution: **no unique definition of translation** on graphs.



Graph convolution via Fourier transform

Instead, use convolution properties **in the Fourier domain**:

$$\mathbf{G}^{(d)}(\mathbf{x}_s) = \mathbf{y}_{s,d} = \sum_{c=0}^{C-1} (\overline{\boldsymbol{\theta}_{d,c}} *_{(\mathcal{G})} \mathbf{x}_{s,c})$$

$$= \underbrace{\sum_{c=0}^{C-1} \underbrace{\mathbf{U} \cdot \underbrace{g^{(d,c)}(\boldsymbol{\Lambda}) \cdot \overbrace{\mathbf{U}^\top \mathbf{x}_{s,c}}^{\widehat{\mathbf{x}_{s,c}}}}_{\text{filtering}}}_{\text{inverse Fourier transform}}}_{\text{sum over all inputs}}$$

with $g^{(d,c)} : \mathbb{R}_+ \to \mathbb{R}$ and $g^{(d,c)}(\boldsymbol{\Lambda}) = \text{diag}\left(g^{(d,c)}(\lambda_0), \dots, g^{(d,c)}(\lambda_{N-1})\right)$.

$\Longrightarrow$ For any input c and output d, $g^{(d,c)}$ only needs to be defined on the graph frequencies $\lambda_0, \dots, \lambda_{N-1}$, giving a weight to the corresponding eigenspaces.

Parametrization of graph filters

First approach: non-parametric filters

Let $\boldsymbol{\theta}_{d,c} \in \mathbb{R}^N$ such that $g^{(d,c)}(\lambda_n) = \boldsymbol{\theta}_{d,c}[n]$ for any $n \in [0 \mathinner{.\,.} N-1]$.

- ✗ Not localized in space;
- ✗ N trainable parameters ($\ll N$ for standard CNNs);
- ✗ Filtering operation: $\mathcal{O}(N^2)$ (linear complexity for standard CNNs).

Let $\boldsymbol{\theta}_{d,c} \in \mathbb{R}^K$ $(K \ll N)$ such that for any $n \in [0\,..\,N-1]$:

$$g^{(d,c)}(\lambda_n) = \sum_{k=0}^{K-1} \boldsymbol{\theta}_{d,c}[k] \cdot \lambda_n^k$$

- ✓ K-**localized filters**: for any $i, j \in [0\,..\,N-1]$, $\mathbf{y}_{s,d}[j]$ is influenced by $\mathbf{x}_{s,c}[i]$ only if $d_{\mathcal{G}}(i,j) \leq (K-1)$, where $d_{\mathcal{G}}$ denotes the minimum number of edges connecting vertices i and j;
- ✓ K **trainable parameters**, which is equal to the filter spatial extension (similarly to standard 1D CNNs);
- ✗ Filtering operation: $\mathcal{O}(N^2)$ (linear complexity for standard CNNs).

Let $\boldsymbol{\theta}_{d,c} \in \mathbb{R}^K$ $(K \ll N)$ such that for any $n \in [0\,..\,N-1]$:

$$g^{(d,c)}(\lambda_n) = \sum_{k=0}^{K-1} \boldsymbol{\theta}_{d,c}[k] \cdot T_k(\tilde{\lambda}_n)$$

with $\tilde{\lambda} = 2\lambda/\lambda_{\max} - 1$ and $T_k \in \mathcal{P}_k(\mathbb{R})$ (Chebyshev polynomials) such that:

$$\begin{cases} T_0(u) = 1 \\ T_1(u) = u \\ T_k(u) = 2uT_{k-1}(u) - T_{k-2}(u) \quad \text{for any } k \geq 2 \end{cases}$$

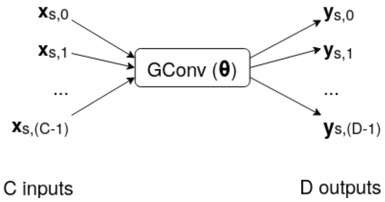
- ✓ K-**localized filters**: for any $i, j \in [0\,..\,N-1]$, $\mathbf{y}_{s,d}[j]$ is influenced by $\mathbf{x}_{s,c}[i]$ only if $d_{\mathcal{G}}(i,j) \leq (K-1)$, where $d_{\mathcal{G}}$ denotes the minimum number of edges connecting vertices i and j;
- ✓ K **trainable parameters**, which is equal to the filter spatial extension (similarly to standard 1D CNNs);
- ✓ **Fast filtering operation** with complexity $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(N^2)$ (takes advantage of the sparsity of $\mathbf{L}$).

Forward-propagation in $\mathcal{O}(SCD \cdot K|\mathcal{E}|)$:

$$\begin{aligned}\mathbf{y}_{s,d} &= \sum_{c=0}^{C-1} (\overline{\boldsymbol{\theta}_{d,c}} *_{(\mathcal{G})} \mathbf{x}_{s,c}) \\ &= \sum_{c=0}^{C-1} (\mathcal{T}_{\mathbf{L}}(\mathbf{x}_{s,c}) \cdot \boldsymbol{\theta}_{d,c})\end{aligned}$$

with:

- $\boldsymbol{\theta}_{d,c} \in \mathbb{R}^K$ vector of Chebyshev coefficients;
- $\mathcal{T}_{\mathbf{L}} : \mathbb{R}^N \to \mathbb{R}^{N \times K}$, computed in $\mathcal{O}(K|\mathcal{E}|)$ with K recursive computations.

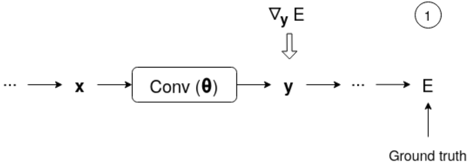


Gradient backpropagation in $\mathcal{O}(SCD \cdot K|\mathcal{E}|)$ (assuming $|\mathcal{E}| \sim N$):

$$\nabla_{(\boldsymbol{\theta}_{d,c})} E = \sum_{s=0}^{S-1} \left[\mathcal{T}_{\mathbf{L}}(\mathbf{x}_{s,c})^\top \cdot \nabla_{(\mathbf{y}_{s,d})} E \right]$$

$$\nabla_{(\mathbf{x}_{s,c})} E = \sum_{d=0}^{D-1} \left[\mathcal{T}_{\mathbf{L}} \left(\nabla_{(\mathbf{y}_{s,d})} E \right) \cdot \boldsymbol{\theta}_{d,c} \right]$$

where the loss E is computed over a minibatch of S samples.



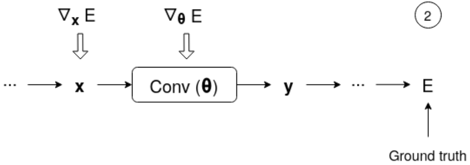
Stochastic gradient descent: $\boldsymbol{\theta}_{d,c} \leftarrow \left(\boldsymbol{\theta}_{d,c} - \eta \cdot \nabla_{(\boldsymbol{\theta}_{d,c})} E \right)$

Gradient backpropagation in $\mathcal{O}(SCD \cdot K|\mathcal{E}|)$ (assuming $|\mathcal{E}| \sim N$):

$$\nabla_{(\boldsymbol{\theta}_{d,c})} E = \sum_{s=0}^{S-1} \left[\mathcal{T}_{\mathbf{L}}(\mathbf{x}_{s,c})^\top \cdot \nabla_{(\mathbf{y}_{s,d})} E \right]$$

$$\nabla_{(\mathbf{x}_{s,c})} E = \sum_{d=0}^{D-1} \left[\mathcal{T}_{\mathbf{L}} \left(\nabla_{(\mathbf{y}_{s,d})} E \right) \cdot \boldsymbol{\theta}_{d,c} \right]$$

where the loss E is computed over a minibatch of S samples.



Stochastic gradient descent: $\boldsymbol{\theta}_{d,c} \leftarrow \left(\boldsymbol{\theta}_{d,c} - \eta \cdot \nabla_{(\boldsymbol{\theta}_{d,c})} E \right)$

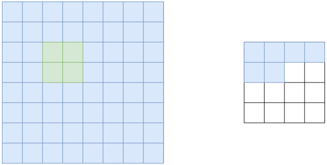
Section 4

Pooling layers on graphs

Forward-propagation for any sample s and any output d:

Classical CNN:

$$\mathbf{Z}_{s,d}[\boldsymbol{n}] = \max_{\boldsymbol{i}\in\{0,1\}^2} \left(\mathbf{Y}_{s,d}[2\boldsymbol{n}+\boldsymbol{i}]\right)$$



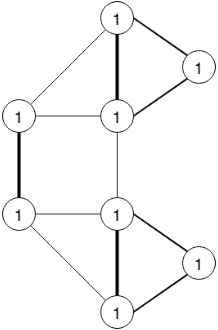
Graph CNN:

$$\mathbf{z}_{s,d}[n] = \max_{m\in\pi_n} \left(\mathbf{y}_{s,d}[m]\right)$$

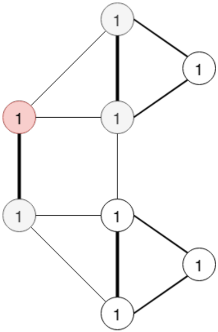
where $\pi_n \subset [0\,..\,N-1]$ denotes the set of neighboring nodes that are reduced into one in the output graph.

Goal: find a graph structure $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathbf{W}')$ with $|\mathcal{V}'| = N' = \lceil N/2 \rceil$ and a grouping $\{\pi_n\}_{n\in[0..N'-1]}$, such that local geometric structures are preserved.

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

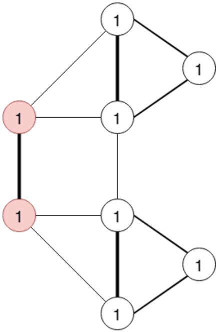


- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



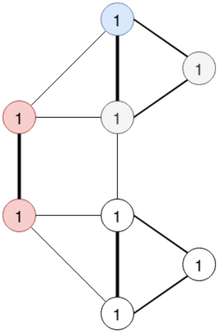
Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

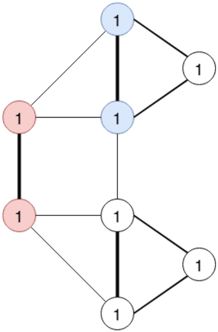


Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

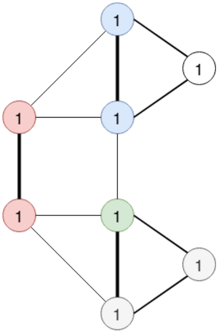


- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

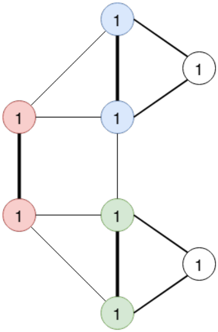


Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

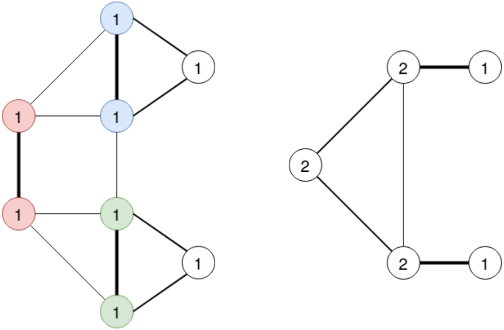


- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

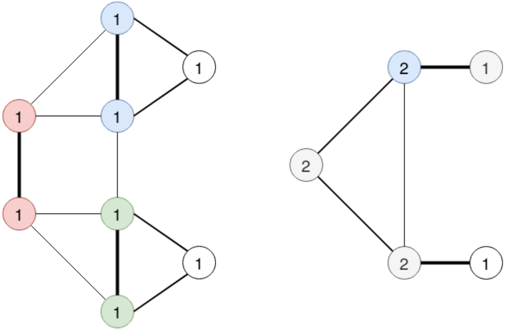


Graph coarsening using Graclus multilevel clustering algorithm

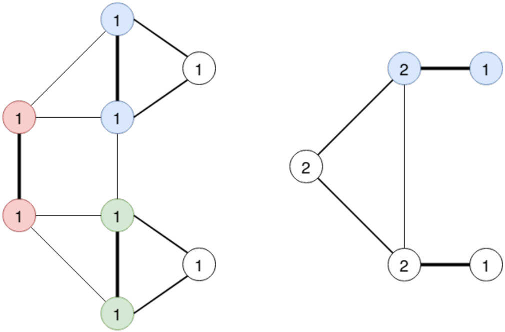
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.

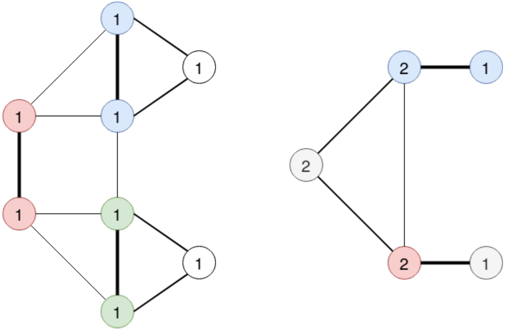


- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



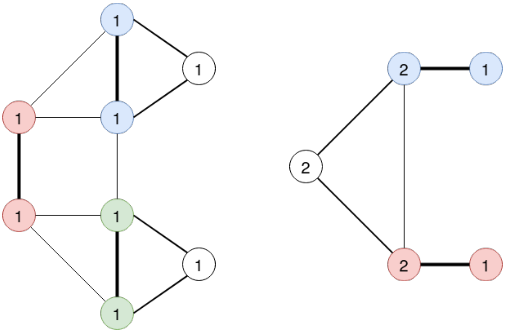
Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



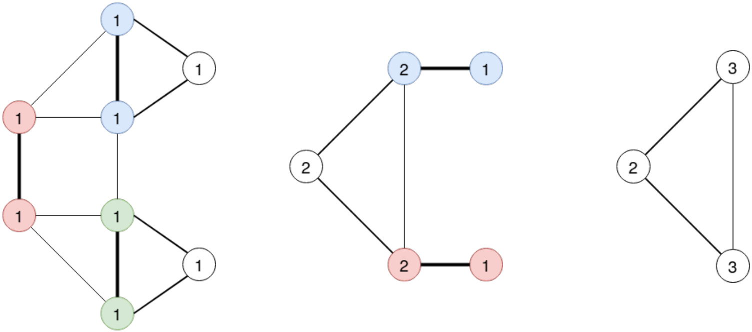
Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



Graph coarsening using Graclus multilevel clustering algorithm

- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.



Fast pooling algorithm

Idea: rearrange vertices such that the pooling operation is computed over 2 consecutive nodes:

$$\forall n \in \left[0 \mathinner{.\,.} N' - 1\right], \pi(n) = \{2n, 2n+1\}$$

Then:

$$\mathbf{z}_{s,d}[n] = \max_{i \in \{0,1\}^2} \left(\mathbf{y}_{s,d}[2n+i]\right)$$

Fast pooling algorithm

Idea: rearrange vertices such that the pooling operation is computed over 2 consecutive nodes:

$$\forall n \in [0 \,..\, N' - 1], \pi(n) = \{2n, 2n+1\}$$

Then:

$$\mathbf{z}_{s,d}[n] = \max_{i \in \{0,1\}^2} \left(\mathbf{y}_{s,d}[2n+i] \right)$$

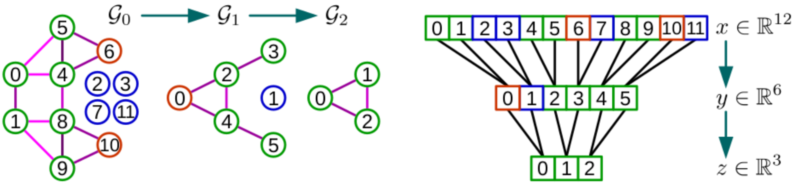


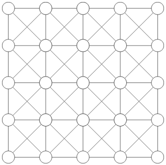
Figure: From [2]

Section 5

Numerical experiments

Applying graph CNN on image classification

- Sanity check for the model: it should at least perform well on standard image classification tasks.
- 8-NN[1] similarity graph of the 2D grid:



with weights: $\mathbf{A}[i,j] = \exp\left(-\frac{\|\mathbf{z}_j - \mathbf{z}_i\|_2^2}{\sigma^2}\right)$, where $\mathbf{z}_i \in \mathbb{R}^2$ is the coordinate of pixel i on the grid.

Model	Accuracy
Classical CNN	99.33
Proposed graph CNN	99.14

Figure: Classical vs graph CNN

		Accuracy		
Dataset	Architecture	Non-Param (2)	Spline (7) [4]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Figure: Different models of graph CNNs

[1] nearest neighbors

- **Text categorization** problem on the 20NEWS dataset [4].
- Using a **bag-of-words model** [5]: each document (input data) is represented as a vector $\mathbf{x} \in \mathbb{R}^N$ with $N = 10,000$ (**most common words** in the corpus). $\mathbf{x}[i]$ is the number of occurrences of word i in the document.

- **Text categorization** problem on the 20NEWS dataset [4].
- Using a **bag-of-words model** [5]: each document (input data) is represented as a vector $\mathbf{x} \in \mathbb{R}^N$ with $N = 10,000$ (**most common words** in the corpus). $\mathbf{x}[i]$ is the number of occurrences of word i in the document.

- Word2vec embedding [5]: each word i is **semantically represented** as a vector $\boldsymbol{z}_i \in \mathbb{R}^d$ using (e.g. $d = 640$).
- Data structure: **16-NN graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, with:
  - $|\mathcal{V}| = 10,000$;
  - $|\mathcal{E}| = 132,834$ edges (connections between the nearest neighbors, using the Euclidean distance induced by the word2vec embedding);
  - weights: $\mathbf{A}[i,j] = \exp\left(-\frac{\|\boldsymbol{z}_j - \boldsymbol{z}_i\|_2^2}{\sigma^2}\right)$.
- Model trained for 20 epochs using **Adam optimizer** [6] and initial learning rate $\eta = 0.001$.

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Figure: Proposed model (GC32) is beaten by multinomial Bayes classifier but outperforms fully-connected newtorks with much less parameters.

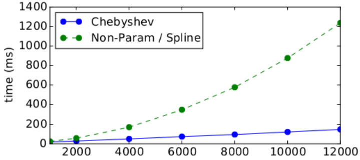


Figure: Linear complexity of the proposed model w.r.t. the data dimensionality N (vs $\mathcal{O}(N^2)$ for non-parametric CNNs or graph CNNs introduced in [7]).

Dataset	Architecture	Accuracy		
		Non-Param (2)	Spline (7) [4]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Figure: Different models of graph CNNs

Influence of graph structure on prediction accuracy

Architecture	8-NN on 2D Euclidean grid	random
GC32	97.40	96.88
GC32-P4-GC64-P4-FC512	99.14	95.39

Figure: MNIST

bag-of-words	word2vec pre-learned	word2vec learned	approximate	random
67.50	66.98	68.26	67.86	67.75

Figure: 20NEWS

- "bag-of-words": naive embedding;
- "learned": embedding learned with word2vec [5];
- "approximate": approximate nearest-neighbors algorithm used for larger datasets.

$\Longrightarrow$ The quality of the results **strongly depend on the graph structure**. It must be designed in order to fulfill **assumptions of locality and stationarity**, as in classical CNNs.

Conclusion and perspectives

Defferrard and colleagues [2] proposed a model of graph CNN able to extract local and stationary features from the data. Improvements with respect to previous graph CNNs [7] are:

- **strictly localized** convolution filters;
- **computational efficiency** which is comparable to classical CNNs;
- **higher test accuracy**.

Future work:

- Explore applications to fields where the data naturally lies on graphs, with explicit information about its structure;
- Learn optimal graph structure in parallel to CNN parameters (instead of using a pre-defined one).

References I

Yann LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel.
Backpropagation Applied to Handwritten Zip Code Recognition.
Neural Computation, 1989.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst.
Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.
In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3844–3852. Curran Associates, Inc., 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2323, 1998.

Thorsten Joachims.
A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.
Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

References II

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean.
Efficient Estimation of Word Representations in Vector Space.
arXiv:1301.3781 [cs], September 2013.

Diederik P. Kingma and Jimmy Ba.
Adam: A Method for Stochastic Optimization.
arXiv preprint arXiv:1412.6980, pages 1–15, 2014.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun.
Spectral Networks and Locally Connected Networks on Graphs.
arXiv:1312.6203 [cs], May 2014.