3D Mesh Classification with Graph Neural Networks

Giovanni Bindi

Università degli Studi di Firenze

6th November 2020

Introduction

Aim

Being able to correctly classify faces identities.

Methodology

Training Graph Neural Networks.

Data

Using graph representation of 3D meshes.

Geometric Deep Learning - [1]

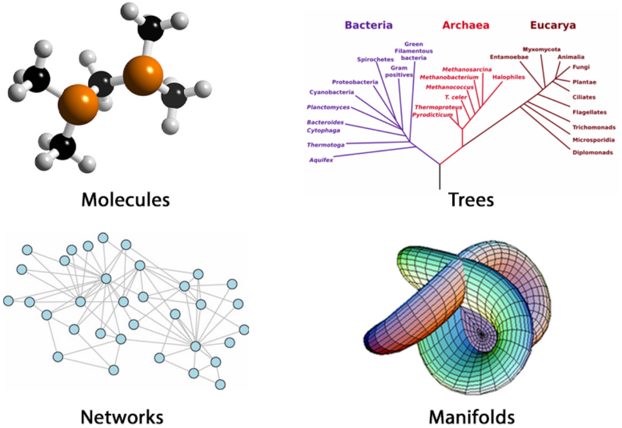


Figure 1: Geometric Deep Learning handles **non-euclidean** data

Graph Neural Network

Neural networks on graphs

- Firstly proposed by Gori *& al* [2] in 2005
- Definitive follow-up [3] in 2009
- **Idea**: use NNs to learn a node representation
- **How**: iteratively approaching the fixed point of a *dynamical system*
- Pro: Theoretical guarantees of convergence
- Con: Inefficient
- Con: Sharing parameters $\implies$ no hierarchical feature extraction
- Con: No edge information

Graph Convolution - 1

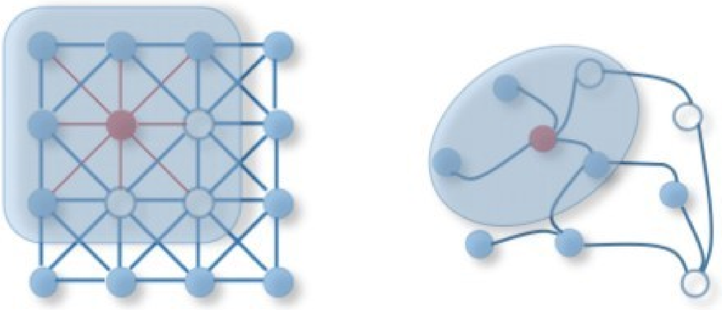


Figure 2: Different approaches for convolutional filters: images (*left*) vs. graphs (*right*)

Graph Convolution - 2

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph. Suppose we have a feature vector $\mathbf{x}_v \in \mathbb{R}^d$ for every node $v \in \mathcal{V}$. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{f \times d}$ we define:

$$\mathbf{g}_v = \mathbf{W}\mathbf{x}_v \tag{1}$$

Defining the set of neighbors (usually first-order) of v as $\mathcal{N}_v$, given an *activation function* $\sigma(\cdot)$, the v node hidden representation $\mathbf{h}_v \in \mathbb{R}^f$ is calculated as

$$\mathbf{h}_v = \sigma\left(\sum_{u \in \mathcal{N}_v} \alpha_{vu}\mathbf{g}_u\right) \tag{2}$$

There are many possibilities on the definition of α_{vu}, one of them is interpreting as an attention weight that measures the connective strength between the node v and its neighbor u.

Graph Attention Networks (GAT) - [4]

Inspired by the recent advances of attention models, the authors propose to define the α_{vu} coefficients as:

$$\alpha_{vu} = \mathrm{softmax}_u(e_{vu}) = \frac{\exp\left(e_{vu}\right)}{\sum_{w \in \mathcal{N}_v} \exp\left(e_{vw}\right)} \tag{3}$$

where

$$e_{vu} = a(\mathbf{g}_v, \mathbf{g}_u) = a(\mathbf{W}\mathbf{x}_v, \mathbf{W}\mathbf{x}_u) \tag{4}$$

and $a : \mathbb{R}^f \times \mathbb{R}^f \rightarrow \mathbb{R}$ is a *self-attention* mechanism: a single layer feedforward neural network with weights $\mathbf{a} \in \mathbb{R}^{2f}$. Finally:

$$\alpha_{vu} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\intercal[\mathbf{W}\mathbf{x}_v \| \mathbf{W}\mathbf{x}_u]\right)\right)}{\sum_{w \in \mathcal{N}_v} \exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\intercal[\mathbf{W}\mathbf{x}_v \| \mathbf{W}\mathbf{x}_w]\right)\right)} \tag{5}$$

GAT: Multi-Head Attention - [4]

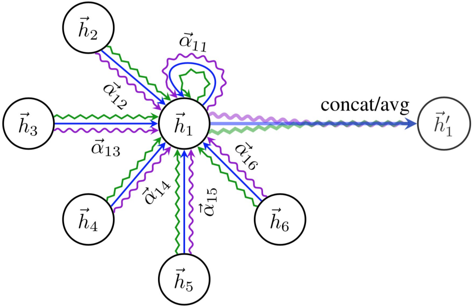


Figure 3: Multi-head attention with $H = 3$

Multi-head attention of order H: H copies (each with different parameters) of Eq. (5) $\rightarrow$ concatenated or averaged.

From Node To Graph Classification

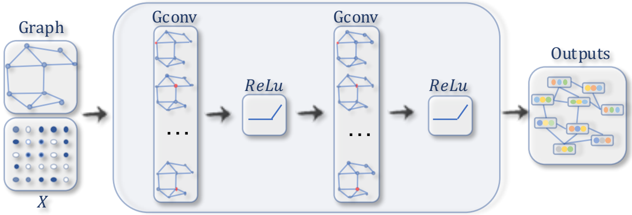


Figure 4: Learning nodes representations

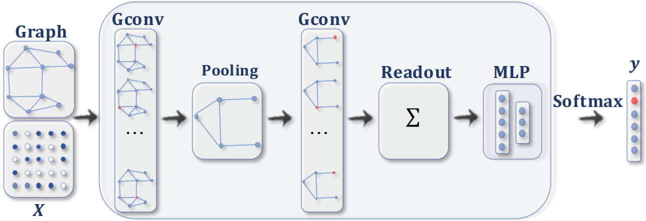


Figure 5: Graph classification

Pooling: Top-k Pooling - [5]

The top-k pooling mechanism has been applied: let $k \in [0, 1]$ and $\mathbf{H} \in \mathbb{R}^{N \times f}$ the feature matrix (with $N = |\mathcal{V}|$). Firstly

$$\mathbf{y} = \frac{\mathbf{H}\mathbf{p}}{\|\mathbf{p}\|_2} \tag{6}$$

is computed where $\mathbf{p} \in \mathbb{R}^f$ is a learnable vector of parameters, then

$$\mathbf{i} = \text{top-}k(\mathbf{y}, k) \tag{7}$$

and finally new features $\mathbf{H}'$ and adjacency matrix $\mathbf{A}'$:

$$\begin{cases} \mathbf{H}' = (\mathbf{H} \odot \sigma(\mathbf{y}))_{\mathbf{i}} \\ \mathbf{A}' = \mathbf{A}_{\mathbf{i},\mathbf{i}} \end{cases} \tag{8}$$

where, usually, $\sigma(\cdot) = \tanh(\cdot)$.

Readout: Global Attention - [6]

Suppose we want to describe our graph $\mathcal{G}$ with a vector $\mathbf{p}_{\mathcal{G}} \in \mathbb{R}^p$. Let $g : \mathbb{R}^f \to \mathbb{R}^p$ a neural network, then a possible solution is

$$\mathbf{p}_{\mathcal{G}} = \sum_{v \in \mathcal{V}} \alpha_v^p g(\mathbf{h}_v) \tag{9}$$

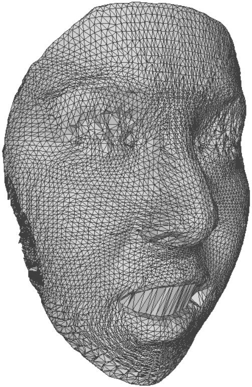
where $\alpha_v^p \in [0, 1]$ is a global attention coefficient for the node v calculated by another neural network f as

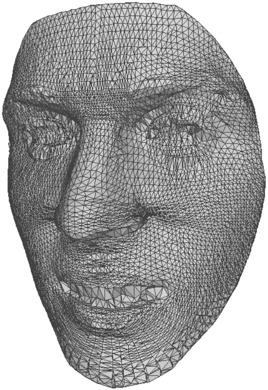
$$\alpha_v^p = \text{softmax}(f(\mathbf{H}))_v \tag{10}$$

where the softmax is applied across nodes, and not across features.

Data: Bosphorus & FRGC

1. Bosphorus [7]:
   - 105 identities
   - 2889 meshes
   - *mesh / id* ≈ 28
   - Every mesh has $\sim$ 6k nodes, $\sim$ 40k edges $\sim$ 13k faces
2. FRGC [8]:
   - 466 identities
   - 3948 meshes
   - *mesh / id* ≈ 8
   - Same structure as Bosphorus





Network Architecture & Hyperparameters

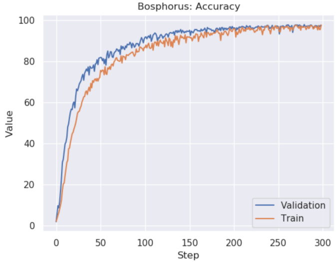
#	Layer
1	$\texttt{GAT}(3, 32, H = 2)$
2	`InstanceNorm`
3	$\texttt{Top-}k$
4	$\texttt{GAT}(64, 128, H = 2)$
5	`InstanceNorm`
6	$\texttt{Top-}k$
7	$\texttt{GAT}(256, 512, H = 2)$
8	`InstanceNorm`
9	$\texttt{GlobalAttention}(512, 512)$
10	`Dropout`
11	$\texttt{Dense}(512, 512)$
12	`Dropout`
13	$\texttt{Dense}(512, 256)$
14	`Dropout`
15	$\texttt{Dense}(256, C)$

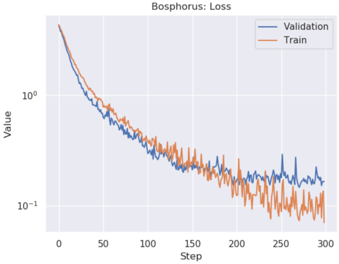
Pre-processing

- Norm vertices to $[0, 1]^3$
- Filtering out identities with number of meshes: $N_m < T_{low} \lor N_m > T_{up}$

- 70% train / 30% test split
- 3-Fold cross validation
- 300 epochs
- Batch size: 16
- Starting learning rate: $\lambda = 10^{-4}$
- Top-k with $k = 0.3$
- Dropout $p = 0.5$

Bosphorus

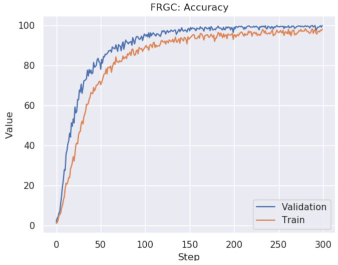


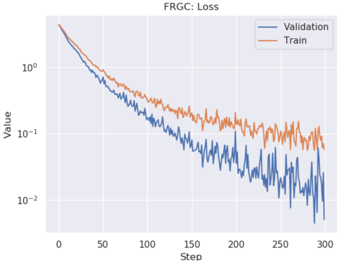


- Filtered out identities with $N_m > 34$ meshes $\rightarrow$ **1294** training and **555** testing examples
- **76** identities left
- $mesh \,/\, id \approx 24$

Dataset	Split	Accuracy
Bosphorus	Train	99.4 %
Bosphorus	**Test**	**90.6 %**

FRGC





- Filtered out identities with $N_m < 16$ meshes $\rightarrow$ **997** training and **472** testing examples
- **78** identities left
- $mesh\ /id \approx 18$

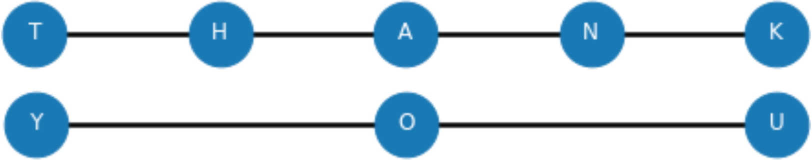
Dataset	Split	Accuracy
FRGC	Train	100 %
FRGC	**Test**	**92.1 %**

Conclusions & Further Developments

- CNN-like Graph Neural Network on 3D meshes
- Con: Did not reach SOTA performances
- Pro: Easily customizable model
- Could be a **baseline** for future architectures

Code

https://github.com/w00zie/3d_face_class


T
H
A
N
K
Y
O
U

References I

[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data", *CoRR*, vol. abs/1611.08097, 2016. arXiv: 1611.08097. [Online]. Available: http://arxiv.org/abs/1611.08097.

[2] M. Gori, G. Monfardini, and F. Scarselli, "A new model for earning in raph domains",, vol. 2, Jan. 2005, 729–734 vol. 2, ISBN: 0-7803-9048-2. DOI: 10.1109/IJCNN.2005.1555942.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model", *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

References II

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[5] H. Gao and S. Ji, "Graph u-nets", *CoRR*, vol. abs/1905.05178, 2019. arXiv: 1905.05178. [Online]. Available: http://arxiv.org/abs/1905.05178.

[6] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, *Gated graph sequence neural networks*, 2017. arXiv: 1511.05493 [cs.LG].

[7] A. Savran, N. Alyuz, H. Dibeklioglu, O. Celiktutan, B. Gokberk, B. Sankur, and L. Akarun, "Bosphorus database for 3d face analysis",, Jan. 2008, pp. 47–56. DOI: 10.1007/978-3-540-89991-4_6.

References III

[8] P. J. Phillips, P. Flynn, T. Scruggs, K. Bowyer, J. (Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge",, vol. 1, Jul. 2005, 947–954 vol. 1, ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.268.

[9] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance normalization: The missing ingredient for fast stylization", *CoRR*, vol. abs/1607.08022, 2016. arXiv: 1607.08022. [Online]. Available: http://arxiv.org/abs/1607.08022.

Vertex Normalization

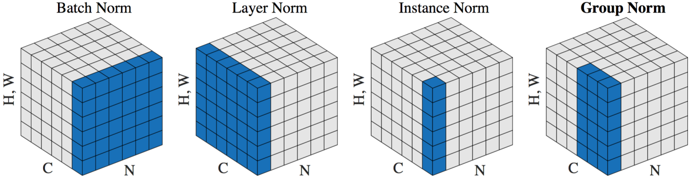
Let $\mathbf{x}_v \in \mathbb{R}^d$ the vector of initial features for node $v \in \mathcal{V}$.
Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\intercal \in \mathbb{R}^{N \times d}$ the feature matrix of the whole graph, being $N = |\mathcal{V}|$. For each node we define a mapping from $\mathbb{R}^d \to [0, 1]^d$ as following:

$$
\begin{aligned}
M &= \max_{i \in \{1,\ldots,N\}} \max_{j \in \{1,\ldots,d\}} \mathbf{X}_{ij} \\
m &= \min_{i \in \{1,\ldots,N\}} \min_{j \in \{1,\ldots,d\}} \mathbf{X}_{ij} \\
\alpha &= \frac{1}{M - m} \\
\mathbf{x}_v^n &= \alpha(\mathbf{x}_v - m\mathbf{1})
\end{aligned}
$$

where $\mathbf{1} = [1, \ldots, 1]^\intercal \in \mathbb{R}^d$. It is possible to show that this transformation scales the distances by a factor α^2:

$$
d(\mathbf{x}_v^n, \mathbf{x}_u^n) = \|\mathbf{x}_v^n - \mathbf{x}_u^n\|_2 = \alpha^2 \|\mathbf{x}_v - \mathbf{x}_u\|_2 = \alpha^2 d(\mathbf{x}_v, \mathbf{x}_u)
$$

Instance Normalization - [9]



Normalization is applied on each element $\mathbf{h}_v$ of every batch $\mathcal{B}$:

$$\mathbf{h}_v = \frac{\mathbf{h}_v - \mathbb{E}[\mathbf{h}_v]}{\mathrm{Var}[\mathbf{h}_v] + \epsilon} \odot \gamma + \beta$$

where $\mathbf{h}_v, \gamma, \beta \in \mathbb{R}^f$ and $\odot$ is the element-wise multiplication.