STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIX**, Number 1, 2024
DOI: 10.24193/subbi.2024.1.02

3D DEFORMABLE OBJECT MATCHING USING GRAPH NEURAL NETWORKS

MIHAI-ADRIAN LOGHIN

Abstract. Considering the current advancements in computer vision it can be observed that most of it is focused on two dimensional imagery. This includes problems such as classification, regression, and the lesser known object matching problem. While object matching ca be viewed as a solved problem in a two dimensional space, for a three dimensional space there is a long way to go, especially for non-rigid objects. The problem is focused on matching a given object to a target object. We propose a solution based on Graph Neural Networks that tries to generalize over multiple objects at once, based on self-attention and cross-attention blocks for the network. To test our solution, we utilised five convolutional operators for the layers of the model. The convolutional operators we compared included GCNConv, ChebConv, SAGEConv, TAGConv, and FeaStConv. This paper aims to find the best operators for our architecture and the task. Our approach obtained favourable results for predicting the barycentric weights for the model, while struggling on predicting the triangle indexes. The best results were obtained for the models using GCNConv, for the triangles index prediction and FeaStConv for the barycentric coordinates prediction.

1. Introduction

In the most recent years we have observed multiple new technologies that require or are improved by usage of 3D models. This ranges from applications in medicine [13] to research oriented papers about simulations of 3D environments. Still, very few papers talk about how we can relate 3D objects to real-world objects. By creating a solution for relating 3D objects to real-world

Received by the editors: 29 November 2023.

2010 *Mathematics Subject Classification.* 68T45, 68U05.

1998 *CR Categories and Descriptors.* I.2.10 [**ARTIFICIAL INTELLIGENCE**]: Vision and Scene Understanding – *3D/stereo scene analysis*; I.3.5 [**COMPUTER GRAPHICS**]: Computational Geometry and Object Modeling – *Geometric algorithms, languages, and systems*.

Key words and phrases. graph neural networks, object matching, 3D objects, deformable objects.

objects, software such as NextMed [13] could be improved. On a similar aspect, the validity of such an application is given by frameworks like the one presented in [26]. By starting this research we could end up with applications that would easily allow to overlap digital reconstructions over real-world objects automatically. This could be especially useful in medicine, where we could have a 3D recreation of an organ that could be moulded automatically over the real organ.

In this paper, we will explore one of the first steps in relating real-world objects to those from a virtual world. The process consists of first seeing if it is even possible to relate 3D objects to one another by focusing on deformable objects. While 2D image matching or object matching can be considered an optimised problem [16, 19, 30], the same cannot be said for cases where the objects can suffer various deformations such as cuts, holes, or surface changes. Furthermore, very few, if any, experiments have focused on creating a reference-object agnostic solution for the problem.

Throughout this paper, we will present the overall field of study, how we constructed a dataset for the problem and how the model for different kinds of solutions have been made. Our focus will be on exploring solutions that work based on multiple inputs, data from multiple datasets, and multiple shape classes for the objects. We aim to obtain a good comparison between multiple model architectures for deformable object matching, mainly focusing on comparing convolutional operators that work on graphs, representing 3D objects. The solution that we provide will be based on using self and cross-attention blocks in a graph neural network to take into account both the deformed shape and the target shape.

Our paper is structured into eight sections starting with the introduction. In Section 2 we present the current state-of-the-art and other approaches in the field of study, to properly define it. Section 3 will be used to discuss the problem of object matching, and which solutions and datasets for the problem are currently available. After defining the problem, Section 4 will be used to define what graph neural networks are and what the different convolution operators for those networks are. We will present our proposed model design and its evaluation in Section 5. The results and experimental setup will be presented in Section 6 and then they will be further discussed in Section 7. We will go over a few conclusions and further research possibilities in Section 8.

2. Related work

To start off, we need to focus on what has already been accomplished in this field. As such, in this section we will present the already solved problem or existing models related to the problem at hand.

At the very beginning of research related to object matching, most of the solutions were focused on works related to two dimensional images. While articles such as [17, 19, 32] have already solved the problem, they also indicate possible approaches for solutions in higher dimensions. Article [16] in particular shows the possibility of using graph neural networks (GNN) for the problem with very promising results.

Deformable objects, especially if related to matching between three dimensional objects, have been especially of interest in challenges for the field. A clear example of this are the articles [4, 8, 9, 22] related to the SHREC event in various years. The results from the challenge show that the current state-of-the-art is represented by Partial Functional Maps, followed closely behind by an approach using Random Forests. This indicates the possibility of using a machine learning approach for the problem and creating a model that can learn to associate one model to multiple objects. With partial functional maps, the authors of [22] managed to obtain a matching percentage of around 80% for deformations involving both cuts and holes. This method creates a form of mapping between the points of one object to another, treating them like functions.

On the side of usefulness for the field, we do not need to look further than the field of robotics. Matching a predefined 3D object to a real-world counterpart has been essential in multiple papers [20,31,34], including even solutions outside of robotics [26] based on rigid objects. Still, the problem in robotics is that it relies heavily on using an almost perfect environment and object representation to the 3D counterpart.

Other methods for deformable object matching are based on point cloud data. One of those solutions can be found in [23], focusing on human-shaped 3D objects. The authors present a method that uses learning based on functional maps, focusing only on heaving self-attention blocks that embed the objects. They claim to achieve state-of-the-art results obtaining an error of $5.4e-2$. A similar solution, closer in structure to what we propose, for learning point cloud matching is presented in [12]. This solution provides a network based on self and cross-attention blocks using the Chamfer distance as its loss function. Compared to the previous solution, the authors validate their approach on multiple shape classes for the 3D objects. On SHREC 2019, they obtained an accuracy of 15.3% and an error of 5.6, beating the other methods they compared with. Given that both approaches obtained state-of-the-art

results, it is easy to say that in regards to using point clouds the solutions are well optimised. Still, the discrepancy between the error and accuracy for the last method shows the need for further investigation of the architecture and of the results obtained by other methods.

Considering everything that has been mentioned so far, it should be clear that the problem itself still requires a lot of improvements. This is especially true when it comes to constructing a machine learning approach for the problem, as we will be doing in the following sections.

3. Deformable object matching

In this section, we will discuss the problem of 3D deformable object matching, some already existing solutions for the problem and how a dataset for the problem looks like. To do all of this, we will separate our findings into two subsections. The first subsection will be for the larger problem, while the second will be dedicated to defining our dataset.

3.1. **Problem definition.** Deformable object matching, compared to matching rigid bodies, is a lot more complicated than it might seem. It requires finding a way to transform the given object into something that more closely resembles the reference object. The main problem here is finding the correct type of transformation needed, as it goes beyond basic transformations such as translation, rotation and scaling [5, 26, 29]. Regarding this, we can use barycentric weights to solve the problem [4, 8, 22].

$$p = \sum_{i=1}^{3} \lambda_i * v_i, where : \sum_{i=1}^{3} \lambda_i = 1 \text{ [14]} \tag{1}$$

Considering Equation 1, we can transform any point from a deformed object to a position that more closely matches that of the target object. In the equation, λ_i with $i \in \{1, 2, 3\}$ represents the barycentric weights for a given triangle from target object. The weights are chosen in such a way that by multiplying each weight to the vertices of the triangles and then summing them we get a approximation of a node from the deformed object to the target object [4,8,14,22]. The transformation could lead to an association like that from Fig. 3, given the rightmost result.

3.2. **Dataset.** An important part of our project was gathering relevant and consistent data from across multiple datasets. In the end, we have found two datasets with very similar formats that still offered their own unique elements for the problem. The datasets that were used are SHREC 2016 [4,22] and SHREC 2019 [8]. The first dataset offers deformations such as body

movements, holes and cuts. On the other hand, the second dataset offers deformations such as surface changes, wrinkles, and joint movements.

In total, the datasets offer 276 three dimensional objects spread across 11 classes. The classes are for the most part evenly distributed, with the main exceptions being the human class having the highest number of objects and the glove class having the lowest number of objects [4, 8, 22]. For our testing environment, we have constructed a dataset with a 70-30 train-test split. We have split the dataset evenly for all the classes.

4. Graph Neural Networks (GNN)

As stated in Section 2, the current state-of-the-art is represented by the use of Functional Maps, closely followed behind by the use of Random Forests [4, 8, 9, 22]. Both of the mentioned approaches fail to properly generalize to different types of damage. While there is another approach using Graph Neural Networks [21], even that one still bases its final layer on Functional Maps. We are researching the possibility of using GNNs without any other attachments. For that, we will look into how different types of convolutions can improve the results.

Defining graph neural networks is essential for our research. As such, we will be looking into the general definition for this machine learning architecture and how it can be used to suit our needs. We defined multiple subsections for this case, to understand not just the general definition, but also the subsequent definitions needed for our model architecture.

4.1. **Definition.** Graph neural networks are a variant of neural networks such that they can work on graphs. In our case, a graph is defined as $G = (V, E)$, where V represents the set of nodes in the graph and E is the set of edges that connect the nodes. We can also associate a matrix $A \in \mathbb{R}^{N \times N}$ to the graph, representing the adjacency matrix. Using this, and the power of neural networks we can create predictions at node, edge, and graph level [18, 35, 37].

Defining a mathematical expression for GNNs requires taking into consideration information about graph structures and neural networks at once. This consists of using information regarding nodes, edges, and learnable weights, which are represented in the following equation:

$$h_i^{t+1} = f(h_i^t W_i + \sum_{j \in N_i} \frac{1}{c_{ij}} h_j^t W_j) \text{ [35, 37]} \tag{2}$$

To understand GNNs even better, we will analyse Equation 2. In this equation h_i^t represents the vector representation of node i at time t, W_i represents the learnable weights for the given node i (which are not always present), c_{ij}

represents a non-weighted connection between nodes i and j, N_i is considered the node neighbourhood of node i and W_j represents the learnable weights for the given node j. In the equation, f represents a propagation function and if needed, can be interpreted as an activation function. This equation stands at the base of how most, if not all, GNNs are constructed [35, 37]. While it might suffer changes from implementation to implementation, the main idea remains the same.

4.2. **GCNConv.** This graph convolution was created for the task of semi-supervised classification. Still, this does not mean it can not be used for scenarios such as ours. Considering this, we will look at what the main aspects of this convolutional operator are and what kind of results were obtained using it in the original presentation of the method [18,37]. If we were to start from equation 2, we can form a new equation, with a similar structure.

$$H^{t+1} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}H^tW^t) \text{ [18]} \tag{3}$$

Equation 3 represents the propagation rule of the network. Similarly, we work just like before with the node level information, like in Equation 2. Here H^t represents the activation results of layer l and for $t = 0$ it represents the results of the initial graph, similarly to how h_i^t was used in the previous equation. W^t represents, just as before, the learnable weights for layer l. σ denotes the activation function for the layer, and it can take multiple forms. In the original paper, the authors have used both softmax and ReLU as possible approaches. $\widetilde{D}_{ii} = \sum_{j=0} \widetilde{A}_{ij}$ is considered a diagonal degree matrix, for which $\widetilde{A} = A + I$ is the adjacency matrix with self inserted loops [18].

In the original evaluation, the authors of the method have shown through rigorous testing that their method could beat other approaches. In their experiments, they have noted improvements over multiple runs. They have also shown how their method can get over 80% accuracy where other methods would only get at most 75% [18].

This convolutional operator has also been proven to be time-efficient. It was shown that for the most part it performs the same on GPU and CPU, only slowing in performance when there are more edges in the graph. For a number of 1k edges the authors have noted a performance of around 10^{-3} seconds/epoch and only at 10M nodes did they reach 10 seconds/epoch on GPU [18].

4.3. **ChebConv.** The ChebConv operator was created with the idea of creating GNNs with fast localized spectral filtering. This means that the authors

implemented ideas similar to pooling in their approach [6]. Those can be better portrayed by the following equation:

$$H^{t+1} = \sum_{k=0}^{K-1} \Theta_k T_k(\widetilde{\Lambda}) \text{ [6]} \tag{4}$$

This convolutional operator is described in full in Equation 4. Here, Θ is a vector of polynomial coefficients, more precisely Chebyshev coefficients. $T_k(\widetilde{\Lambda})$ is a Chebyshev polynomial of order k and it is determined recurrently. To start the recurrent process we consider $\widetilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I$, where I is the identity matrix and Λ is the diagonal matrix formed by the frequencies of the graph, $T_0(\widetilde{\Lambda}) = H$, and $T_1(\widetilde{\Lambda}) = \widetilde{\Lambda}H$. We can then consider the recurrent function $T_k(\widetilde{\Lambda}) = 2\widetilde{\Lambda}T_{k-1}(\widetilde{\Lambda}) - T_{k-2}(\widetilde{\Lambda})$. The authors have noted that the entire operation would have a computation cost of $O(K|\epsilon|)$ [6].

For the experimental phase of the research, the authors have revisited various datasets for which solutions exist using classical CNNs and other machine learning methods. Those datasets include MNIST and 20NEWS. For the MNIST dataset, they have shown an almost exact performance with CNNs. Still, the proposed architecture outperformed other methods in terms of time efficiency. As for the 20NEWS dataset, the model did not manage to outperform the Multinomial Naive Bayes approach. Although it did not perform better in this case, the authors noted that their proposed architecture still outperforms other fully connected neural networks [6].

In their research, they have also tackled the influence of graph quality on the results. They have noted that the way the graph is constructed is the most important part for their operator to work. For this, they considered a comparative use of image and text graphs. What their study has shown is that the method works best on image graphs, but due to the limitation of text graphs, it cannot outperform the current state-of-the-art [6].

4.4. **SAGEConv.** This method was created on the assumption that not all nodes in the graph need to be used for determining the embedding of one node. SAGEConv came as a way to essentially improve the already existing approach of using low-dimensional node embeddings for large graphs. As such, they proposed a framework called GraphSAGE [15]. The framework also stands as one of the more popular approaches for graph convolutions [37]. In our research we are only interested in the convolutional operator of this framework and how it can help us. We will detail our findings and explain what we will be using from this framework.

The SAGEConv operator can be described in the form of two equations, of which the latter is optional. This idea comes from showing the operator in two ways, a standard equation operator and an optimised operator that can be used for further improving the model [15].

$$h_i^{t+1} = W_1 h_i^t + W_2 * mean_{j \in N_i}(h_j^t) \text{ [15]} \tag{5}$$

$$h_j^{t+1} = \sigma(W_3 h_j^t + b) \text{ [15]} \tag{6}$$

We can formulate the equation for the convolutional operator based on the information from Equations 5 and 6. In those equations W_i for $i \in \{1, 2, 3\}$ represent the weight matrices that will change during the training process. As previously used, σ can represent any activation function. In the original article, the authors have not presented any functions that would be more favourable to be used. h_i represents the feature vector for a node v_i [15].

In the implementation of the convolutional operator, the most interesting part is the relation between Equation 5 and Equation 6. By that, we are referring to the fact that the relations can work independently of one another, at least according to the PyTorch Geometric implementation. We can then consider that the combined use of the two equations is an improved version of the convolutional operator over the graph [10, 15]. In our work, we only considered the base version, without the additional improvement to the method. This is due to it requiring a more complicated implementation of our model.

For the qualitative evaluation of the operator, the authors have noted a comparison against four other methods. Those methods include the DeepWalk algorithm, a random classifier, logistic regression, and a hybrid between raw features and DeepWalk embeddings. They have also noted extended versions of their algorithm that use the operator from Subsection 4.2, an LSTM, a mean operator, and a pooling operator [15, 18].

The testing took place on three datasets. Those datasets were based on citations, Reddit posts, and the PPI dataset. On all three of those, the proposed algorithm has outperformed all other methods, with the most notable results being from the GCNConv and LSTM variants. The algorithms were tested on both supervised and unsupervised environments. The authors have noted that their method does generalize across graphs [15].

4.5. **TAGConv.** In the previous three subsections, we have considered the use of the more popular convolutional operators. Now it is time to get into more problem-specific operators, mainly those that were created to work directly with 3D objects. Graph convolutions can defined on the spectral or vertex domains, of which the authors of TAGConv have chosen the latter.

The name of the model stands for Topology Adaptive Graph Convolution and it is mainly based on the idea that the network/operator will adapt to the topology of the graph [7].

$$H^{t+1} = \sum_{k=0}^{K} (D^{-\frac{1}{2}} A D^{-\frac{1}{2}})^k H^t W_k \text{ [7, 10]} \tag{7}$$

Since we are interested in the convolutional operator from the entire network, we will only look at that part of its equation. The operator is represented by Equation 7 and it is adjusted to fit its PyTorch Geometric implementation. In this case, we consider K to have the base value of 3, representing the number of hops. A represents the adjacency matrix of the graph and $D_{ii} = \sum_{j=0} A_{ij}$ the diagonal degree matrix. Considering that, $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is the normalization of the diagonal matrix. As always, W represents the learnable weights for the convolutional operator [7, 10].

This operator, like the previous ones, has been tested mainly on standard benchmark datasets. Those datasets include Pubmed, Citeseer, and Cora. While the model was tested against other standard methods such as DeepWalk and deep convolutional neural networks, it also tested against other graph neural networks such as ChebNet and GCN, presented in the first two subsections. According to the authors, TAGConv outperforms the other methods on all three datasets, with an accuracy of over 80% on two of the datasets [7].

While topology is an important part of defining geometric forms, it is not enough. For the next subsection, we will be looking at our final operator, created to work perfectly with 3D objects. Just like now, we will be looking at it from a theoretical and applied perspective.

4.6. **FeaStConv.** For our final operator, as previously stated, we will focus on direct applications to 3D objects. This operator was constructed from the need to create something like convolutional neural networks, but for 3D shapes. Its name stands for Feature-Steered Graph Convolutions and its original evaluation was done directly on 3D meshes for a variety of problems regarding shape analysis [33].

FeaStConv can be better described in the following equation:

$$h_i^{t+1} = \frac{1}{|N_i|} \sum_{j \in N_i} \sum_{k=1}^{K} q_k(h_i^t, h_j^t) W_k h_j^t \text{ [10, 33]} \tag{8}$$

The equation, like in the previous subsection, is defined according to its implementation in the original article and the PyTorch Geometric implementation. Equation 8 describes the convolutional operator in full. The most unique elements of the equation are represented by K and q_k representing the number of attention heads and an activation function, respectively. In this equation $q_k(h_i, h_j) = softmax_j(u_k^T(h_j - h_i) + c_k)$, where W_k, u_k and c_k are trainable parameters [10, 33].

As mentioned, above, the operator was tested and experimented with using problems related to 3D shape analysis. Those problems included 3D shape correspondence and part labelling. For 3D shape correspondence, the model was compared to other methods such as PointNet, ACNN, GCNN and MoNet. The model proposed by the authors outperformed all other models by a lot, having an accuracy of 98% at most and 88% at least. For the part labeling problem, they used a dataset based on ShapeNet [2] and compared it with four other methods, some of which were mentioned before. While it did not outperform any of the other methods, except one where the difference was of 0.1%, the model still got overall similar results [33].

5. Model and evaluation

We need to properly define the model architecture and evaluation on our data. We will explore this in two subsections, dedicated to each subject. Following that, we will focus on the experiments defined by us.

All of the implementation effort was done using PyTorch and PyTorch Geometric for the loss function and for the implementation of the convolutional operators, respectively. We have chosen those frameworks, based on the number of operations that they had implemented and their usage in other papers [10, 25].

5.1. **Model.**

Our model architecture is based on the idea of using self and cross-attention blocks. Those blocks are a necessity for the problem, as we have to work with multiple 3D objects at once for one result. As a reference for constructing our model we have used SuperGlue [30] and later validated it based on articles that came out during our research that tried to use similar architectures for the use of self-attention and cross-attention blocks [21]. To further emphasise the validity of our approach, we also considered looking into approaches that try to solve the problem in other contexts, such as point cloud data [12]. We only used other models as architectural references, rather than for specific layer parameters.

Fig. 1 is a visualization of our model and how it works. The desired output, as referenced in Section 3 is an [n,4] vector. In this case, n represents the number of nodes in the graph and 4 is the size of the output for each node,

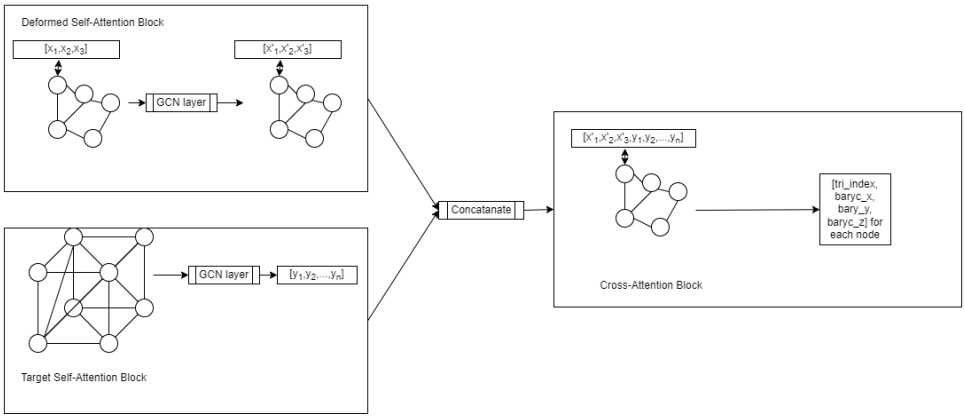


FIGURE 1. A diagram describing the Graph Convolution Network (GCN) created to address the problem of object matching. The model works using a dual input: the deformed object and the reference object. In the third layer of the model, the two outputs are concatenated and the final result will be a tensor of size four representing: the triangle of correspondence, and the barycentric weights for the given triangle.

representing the triangle index and barycentric weights. The self-attention blocks are meant for reinterpreting the object graphs based on node neighbourhoods and the cross-attention block is meant to learn how the two graphs can interact.

For the cross-attention block and deformed self-attention block we have used the approach of reducing the number of hidden channels after each individual convolution. First, it is divided by 2 and then by 4. In the case of the reference/target self-attention block, we used the opposite approach as we wanted to obtain a larger embedding of the graph at the end. As such, we first multiply the number of hidden channels by 2 and then by 4. Between the convolutions, we have used a ReLU activation function and for the output we have used the ReLU for the triangle output and SoftMax for the barycentric part of the output.

5.2. **Training and evaluation.** We have decided to evaluate our model using three loss functions. The loss functions were determined based on the required outputs of the model in regard to the dataset. As such, we have determined that there needs to be a loss for the triangle index $L1_{Triangle}$, one

for the barycentric weights $L1_{Baryc}$ and one for the combined loss of the two $L1_{Combined}$.

$$L1_{combined} = L1_{triangle} + L1_{baryc} \tag{9}$$

For our experiments, we have used the standard definition for the loss function, which is used to determine $L1_{Triangle}$ and $L1_{Baryc}$. In Equation 9 we can observe the combined loss function which represents the sum of the other loss functions and which was used to train the model [3, 11, 24]. To complete the training requirements, we also need to mention the use of the Adam optimizer with a learning rate of $3e-4$, which was chosen based on its proven improvements in performance [1, 36].

We also consider the mean geodesic error as a possible loss function, as it was used on the SHREC 2016 dataset [22]. In the end, we opted against this. After further research, the function did not seem standard enough to be selected.

6. Experiments

6.1. **Setup.** To cover all possible architectural and data combinations, we have decided to explore the problem in four main ways. Each of the three experiment types will have a role in the evaluation of the model in a new environment to see how it reacts under new conditions. The experiments are: [4, 8, 22]

- Experiments on the combined dataset from SHREC 2016 and 2019 [4, 8, 22]
- Experiments on the SHREC 2016 dataset [4, 22]. To evaluate if the standalone dataset offers a better training environment.
- Experiments on the SHREC 2019 dataset [8]. Just like the previous point, it will be used to evaluate the consistency of the results on a standalone dataset. Additionally, it will also help in defining which dataset is better as a training set,
- Experiments on some of the single classes from the SHREC 2016 dataset [4, 22]. An experiment type that will help determine if the problems that we found are related to the use of multiple classes or due to the model that is being used.

Additionally, the experiments will be done for all the convolutional layers defined in Section 4. Besides the change in convolutional layers, we also applied changes in terms of hidden channels, for a number of 20 epochs each and a batch size of 20. An exception for the batch size was used in the case of the single class experiments, where we used a batch size of one.

For the experiments done on the entire dataset we have only used the best combinations obtained from the other experiments. We also reduced the number of epochs by five and expended the batch size by ten, due to time constraints and computational availability in the used environment.

Conv. type	Ch.	Embed	$L1_{Triangle}$		$L1_{Baryc}$		$L1_{Combined}$	
			train	test	train	test	train	test
GCNConv	16	100	7.4E+03	2.2E+04	4.2E-01	1.3E+00	7.4E+03	2.2E+04
		500	5.7E+03	1.7E+04	4.3E-01	1.4E+00	5.7E+03	1.7E+04
	32	100	7.3E+03	2.3E+04	4.1E-01	1.3E+00	7.3E+03	2.3E+04
		500	5.3E+03	1.6E+04	4.6E-01	1.4E+00	5.3E+03	1.6E+04
ChebConv	16	100	1.1E+07	2.7E+07	4.4E-01	1.4E+00	1.1E+07	2.7E+07
		500	7.0E+06	1.6E+07	4.4E-01	1.4E+00	7.0E+06	1.6E+07
	32	100	2.9E+07	6.7E+07	4.6E-01	1.4E+00	2.9E+07	6.7E+07
		500	8.6E+06	1.9E+07	4.5E-01	1.4E+00	8.6E+06	1.9E+07
SAGEConv	16	100	7.4E+03	2.3E+04	4.5E-01	1.4E+00	7.4E+03	2.3E+04
		500	6.8E+03	2.0E+04	4.4E-01	1.4E+00	6.8E+03	2.0E+04
	32	100	5.9E+03	1.8E+04	**4.0E-01**	**1.2E+00**	5.9E+03	1.8E+04
		500	**5.7E+03**	**1.7E+04**	4.2E-01	1.3E+00	**5.7E+03**	**1.7E+04**
TAGConv	16	100	7.8E+03	2.3E+04	4.2E-01	1.2E+00	7.8E+03	2.3E+04
		500	7.1E+03	2.3E+04	4.1E-01	1.3E+00	7.1E+03	2.3E+04
	32	100	7.2E+03	2.3E+04	4.3E-01	1.3E+00	7.2E+03	2.3E+04
		500	6.0E+03	1.9E+04	4.2E-01	1.3E+00	6.0E+03	1.9E+04
FeaStConv	16	100	9.9E+03	3.1E+04	3.5E-01	1.1E+00	9.9E+03	3.1E+04
		500	9.4E+03	2.9E+04	3.6E-01	1.1E+00	9.4E+03	2.9E+04
	32	100	9.5E+03	3.0E+04	3.6E-01	1.1E+00	9.5E+03	3.0E+04
		500	8.0E+03	2.5E+04	3.8E-01	1.2E+00	8.0E+03	2.5E+04

TABLE 1. The results for a single class in the SHREC 2016 dataset [22]. The class was chosen so that it would have a single reference object.

6.2. **Results.** Table 4 is the table that contains the evaluation of the model in the context of the full dataset that was originally presented in Section 3. As for Tables 2 and 3, they represent the results for individual datasets that formed the full dataset. Finally, Table 1 presents the model aggregated results when evaluated on a single class.

The scope of the results is to determine the best possible combination. Our research has so far only focused on determining the best architectural combination for a graph neural network for the task. We only considered longer amounts of training for the best combination, due to our limited computational power. Furthermore, the experiments were designed in such a way as to allow us to find the weakest links in our dataset and approach. We will touch up more on our decisions in the next section.

Conv. type	Ch.	Embed	$L1_{Triangle}$		$L1_{Baryc}$		$L1_{Combined}$	
			train	test	train	test	train	test
GCNConv	16	100	6.3E+03	1.9E+04	4.2E-01	1.3E+00	6.3E+03	1.9E+04
		500	5.5E+03	1.7E+04	4.3E-01	1.3E+00	5.5E+03	1.7E+04
	32	100	5.9E+03	1.8E+04	4.3E-01	1.4E+00	5.9E+03	1.8E+04
		500	5.4E+03	1.7E+04	4.6E-01	1.4E+00	5.4E+03	1.7E+04
SAGEConv	16	100	6.3E+03	1.9E+04	4.6E-01	1.4E+00	6.3E+03	1.9E+04
		500	6.2E+03	1.9E+04	4.4E-01	1.4E+00	6.2E+03	1.9E+04
	32	100	5.8E+03	1.7E+04	4.1E-01	1.3E+00	5.8E+03	1.7E+04
		500	**5.5E+03**	**1.6E+04**	4.3E-01	1.3E+00	**5.5E+03**	**1.6E+04**
TAGConv	16	100	6.7E+03	2.1E+04	4.2E-01	1.3E+00	6.7E+03	2.1E+04
		500	6.3E+03	2.0E+04	4.2E-01	1.3E+00	6.3E+03	2.0E+04
	32	100	6.2E+03	1.9E+04	4.3E-01	1.3E+00	6.2E+03	1.9E+04
		500	6.2E+03	2.0E+04	4.6E-01	1.4E+00	6.2E+03	2.0E+04
FeaStConv	16	100	8.3E+03	2.6E+04	**3.8E-01**	**1.2E+00**	8.3E+03	2.6E+04
		500	-	-	-	-	-	-
	32	100	-	-	-	-	-	-
		500	-	-	-	-	-	-

TABLE 2. Our model's results on the SHREC 2016 dataset [22].

Conv. type	Ch.	Embed	$L1_{Triangle}$		$L1_{Baryc}$		$L1_{Combined}$	
			train	test	train	test	train	test
GCNConv	16	100	6.3E+03	2.0E+04	4.7E-01	1.5E+00	6.3E+03	2.0E+04
		500	5.7E+03	1.8E+04	4.5E-01	1.4E+00	5.7E+03	1.8E+04
	32	100	5.5E+03	1.7E+04	4.3E-01	1.4E+00	5.5E+03	1.7E+04
		500	5.5E+03	1.8E+04	**4.1E-01**	**1.3E+00**	5.5E+03	1.8E+04
SAGEConv	16	100	6.6E+03	2.1E+04	4.2E-01	1.3E+00	6.6E+03	2.1E+04
		500	6.1E+03	1.9E+04	4.3E-01	1.4E+00	6.1E+03	1.9E+04
	32	100	5.8E+03	1.8E+04	4.6E-01	1.5E+00	5.8E+03	1.8E+04
		500	**5.5E+03**	**1.7E+04**	4.4E-01	1.4E+00	**5.5E+03**	**1.7E+04**
TAGConv	16	100	6.5E+03	2.2E+04	4.5E-01	1.5E+00	6.5E+03	2.2E+04
		500	6.1E+03	2.1E+04	4.6E-01	1.5E+00	6.1E+03	2.1E+04
	32	100	6.2E+03	2.0E+04	4.5E-01	1.4E+00	6.2E+03	2.0E+04
		500	6.1E+03	4.0E-01	4.5E-01	1.5E+00	6.1E+03	2.1E+04
FeaStConv	16	100	7.9E+03	2.6E+04	4.5E-01	1.4E+00	8.0E+03	2.6E+04
		500	-	-	-	-	-	-
	32	100	7.6E+03	2.5E+04	4.4E-01	1.4E+00	7.6E+03	2.5E+04
		500	-	-	-	-	-	-

TABLE 3. Our model's results on the SHREC 2019 dataset [8].

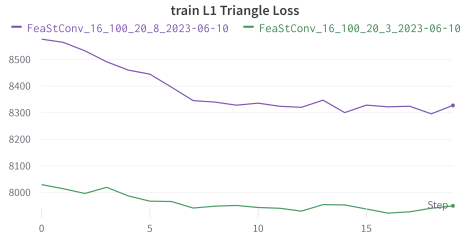
Conv. type	Ch.	Embed	$L1_{Triangle}$		$L1_{Baryc}$		$L1_{Combined}$	
			train	test	train	test	train	test
GCNConv	32	500	**5.4E+03**	**1.7E+04**	4.5E-01	1.4E+00	**5.4E+03**	**1.7E+04**
SAGEConv			5.6E+03	1.7E+04	4.3E-01	1.3E+00	5.6E+03	1.7E+04
TAGConv			6.2E+03	2.0E+04	4.5E-01	1.4E+00	6.2E+03	2.0E+04
FeaStConv	16	100	8.3E+03	2.6E+04	**3.8E-01**	**1.2E+00**	8.3E+03	2.6E+04

TABLE 4. Our model's results on the dataset formed by combining the SHREC 2016 and 2019 datasets [8, 22].

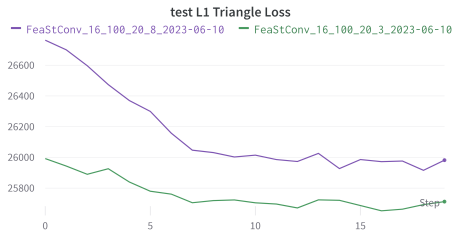
7. Discussion

When confronted with the problem of predicting large numbers for the triangle indexes, we have determined a shortcut for the training procedure. The outputs here are in the tens of thousands and are rather hard to learn for a neural network. To simplify the process we have tested multiplying the output with multiples of ten. The best results were obtained when multiplying the output by 100.

In our approach, we have encountered several benefits and some downfalls. To start off, we have observed a huge under-performance while using ChebConv as observed in Table 1. Since we started with single class experiments to get an initial idea of how to continue the rest of the experiments, we have removed all experiments using ChebConv architectures from the other experiment types. We motivate this choice by arguing that if an architecture under-performance on a single class, it has no way of performing better when put against multiple classes.



(A) Train results.



(B) Test results.

Figure 2. A visual representation of the triangle index loss results using FeaStConv for the individual datasets. The purple line represents the results for the SHREC 2016 [4, 22] dataset, while the green line represents the results for the SHREC 2019 [8] dataset.

To further remove some of the experimental difficulties for our complete dataset, we have experimented with the individual datasets too. The experiments can be seen in Tables 2 and 3. We have come to understand that there is not a huge performance difference between the models on the two datasets, some of which are further established by the results from Fig 2. Related to the graphical results are the results for the FeaStConv architecture. We have observed that this convolutional operator performs poorly for the triangle index prediction, but outperforms all other models on the barycentric predictions. Furthermore, it is the only architecture that works better with a smaller size

embedding and smaller size channels. Some problems can observed, as there are empty cells in the tables. The model seemed to have caused a memory overflow for the GPU in our environment. Considering this, we had to give up on running some of our experiments.

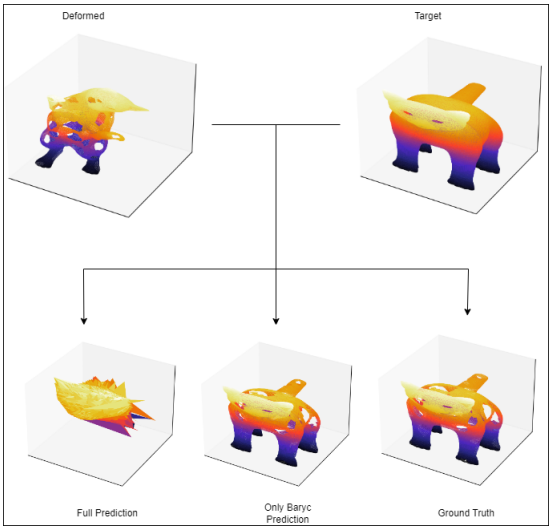


FIGURE 3. An illustration of the predictions for the seventh image with holes type damage from SHREC 2016 [4, 22]. This diagram contains, from left to right, the full prediction of the model after 20 epochs, only the barycentric prediction with correct triangles and the ground truth for this case.

For our final few experiments, one with each adequate model, we have chosen the bigger size architectures. The only exception to this rule is the previously presented model. In Table 4, one can observe the results for what we considered to be the best model. The observations made so far, on the smaller datasets, have remained true. As such, we can consider that the best model architecture is the one using GCNConv, followed by the one using FeaStConv.

To get a better representation of our model's performance, we will now be referring to Fig. 3. Here the reader is free to observe how the model performs when only using the barycentric predictions together with the correct triangle indexes. When using the full output, the shape ends up being clustered to something alike a centre of gravity. This does not mean that all the predictions are wrong, but the wrong prediction can have a huge effect on the model's performance.

Considering only the barycentric part of the output, the model's performance becomes almost indistinguishable from the ground truth. This further indicates that the next steps of our research should be focused on making the model perform better on the triangle index part of the output. Several methods could be used here, such as using an output that gives the indexes of three nodes and only later validating the correctness of this output.

8. Conclusions and further research

In our experiments, we have shown the potential of using various model architectures for deformable object matching. The best overall results were obtained using the GCNConv model with 32 starting hidden channels and an embedding size of 500, for the prediction of the triangle index. In the case of predicting the barycentric weights, the best results were obtained using the FeaStConv model, with of 16 starting hidden channels and an embedding size of 100.

We consider that the results give the right direction to continue developing the GNN model. Seeing that two distinct architectures have given the best results on the two targets of the model, we might want to look more into using them. A combined environment for the two convolutions is not out of the question, nor verifying their hyper-parameters.

Considering the conclusions and every experiment done so far, we have taken into consideration a few possible approaches for further research. Those approaches consist of changes and additions to the model and the dataset.

A possible improvement to the dataset is to consider several basic geometric forms such as cubes, spheres, and cones and then apply random deformations over them. This could show the potential of using synthetic data to train a more robust model.

As mentioned in the very first section, the final scope of this research would be to see if it can match a complex 3D object to a partial reconstruction of the object from a 2D image. This could be done by considering the use of a state-of-the-art depth estimation model such as MiDaS [27,28], which would help us evade the need to use cameras that already have the technology implemented.

Considering the possible new environment, starting from a picture, there will also be the possibility of using a new dataset constructed from that. The dataset could consist of the partial 3D objects given by the depth model and an associated complete 3D object. The only problem with this idea is the need to have more annotated data.

References

[1] Adedigba, A. P., Adeshina, S. A., Aina, O. E., and Aibinu, A. M. Optimal hyperparameter selection of deep learning models for COVID-19 chest x-ray classification. *Intell Based Med 5* (Apr. 2021), 100034.

[2] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[3] Christoffersen, P., and Jacobs, K. The importance of the loss function in option valuation. *Journal of Financial Economics 72*, 2 (2004), 291–318.

[4] Cosmo, L., Rodol, E., Bronstein, M. M., Torsello, A., Cremers, D., and Sahillioglu, Y. Partial Matching of Deformable Shapes. In *Eurographics Workshop on 3D Object Retrieval* (2016), A. Ferreira, A. Giachetti, and D. Giorgi, Eds., The Eurographics Association.

[5] Cosmo, L., Rodol, E., Masci, J., Torsello, A., and Bronstein, M. M. Matching deformable objects in clutter. In *2016 Fourth International Conference on 3D Vision (3DV)* (2016), pp. 1–10.

[6] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2016), NIPS'16, Curran Associates Inc., p. 38443852.

[7] Du, J., Zhang, S., Wu, G., Moura, J. M. F., and Kar, S. Topology adaptive graph convolutional networks. *CoRR abs/1710.10370* (2017).

[8] Dyke, R. M., Stride, C., Lai, Y.-K., Rosin, P. L., Aubry, M., Boyarski, A., Bronstein, A. M., Bronstein, M. M., Cremers, D., Fisher, M., Groueix, T., Guo, D., Kim, V. G., Kimmel, R., Lhner, Z., Li, K., Litany, O., Remez, T., Rodol, E., Russell, B. C., Sahilliolu, Y., Slossberg, R., Tam, G. K. L., Vestner, M., Wu, Z., and Yang, J. Shape correspondence with isometric and non-isometric deformations. In *Eurographics Workshop on 3D Object Retrieval* (2019), S. Biasotti, G. Lavou, and R. Veltkamp, Eds., The Eurographics Association.

[9] Dyke, R. M., Zhou, F., Lai, Y.-K., Rosin, P. L., Guo, D., Li, K., Marin, R., and Yang, J. SHREC 2020 Track: Non-rigid shape correspondence of physically-based deformations. In *Eurographics Workshop on 3D Object Retrieval* (2020), T. Schreck, T. Theoharis, I. Pratikakis, M. Spagnuolo, and R. C. Veltkamp, Eds., The Eurographics Association.

[10] Fey, M., and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[11] Gao, X., and Fang, Y. A note on the generalized degrees of freedom under the l1 loss function. *Journal of Statistical Planning and Inference 141*, 2 (2011), 677–686.

[12] Ginzburg, D., and Raviv, D. Dual geometric graph network (dg2n) iterative network for deformable shape alignment. In *2021 International Conference on 3D Vision (3DV)* (Dec. 2021), IEEE.

[13] Gonzlez Izard, S., Snchez Torres, R., Alonso Plaza, ., Juanes Mndez, J. A., and Garca-Pealvo, F. J. Nextmed: Automatic imaging segmentation, 3d reconstruction, and 3d model visualization platform using augmented and virtual reality. *Sensors 20*, 10 (2020).

[14] Grozdev, S., and Dekov, D. Barycentric coordinates: Formula sheet. *International Journal of Computer Discovered Mathematics 1* (03 2016), 72–82.

[15] Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 10251035.

[16] Joo, H., Jeong, Y., Duchenne, O., and Kweon, I. S. Graph-based shape matching for deformable objects. In *2011 18th IEEE International Conference on Image Processing* (2011), pp. 2901–2904.

[17] Kim, J., and Shontz, S. M. An improved shape matching algorithm for deformable objects using a global image feature. In *Advances in Visual Computing* (Berlin, Heidelberg, 2010), G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammound, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, and L. Avila, Eds., Springer Berlin Heidelberg, pp. 119–128.

[18] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR abs/1609.02907* (2016).

[19] Lazarou, M., Li, B., and Stathaki, T. A novel shape matching descriptor for real-time static hand gesture recognition. *Computer Vision and Image Understanding 210* (2021), 103241.

[20] Li, Y., Wang, Y., Case, M., Chang, S.-F., and Allen, P. K. Real-time pose estimation of deformable objects using a volumetric approach. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2014), pp. 1046–1052.

[21] Lian, Y., and Chen, M. Ca-cgnet: Component-aware capsule graph neural network for non-rigid shape correspondence. *Applied Sciences 13*, 5 (2023).

[22] Lhner, Z., Rodol, E., Bronstein, M. M., Cremers, D., Burghard, O., Cosmo, L., Dieckmann, A., Klein, R., and Sahillioglu, Y. Matching of Deformable Shapes with Topological Noise. In *Eurographics Workshop on 3D Object Retrieval* (2016), A. Ferreira, A. Giachetti, and D. Giorgi, Eds., The Eurographics Association.

[23] Marin, R., Rakotosaona, M.-J., Melzi, S., and Ovsjanikov, M. Correspondence learning via linearly-invariant embedding, 2020.

[24] Muthukumar, V., Narang, A., Subramanian, V., Belkin, M., Hsu, D., and Sahai, A. Classification vs regression in overparameterized regimes: Does the loss function matter? *J. Mach. Learn. Res. 22*, 1 (jan 2021).

[25] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[26] Prisacariu, V. A., and Reid, I. D. Pwp3d: Real-time segmentation and tracking of 3d objects. *International Journal of Computer Vision 98*, 3 (Jul 2012), 335–354.

[27] Ranftl, R., Bochkovskiy, A., and Koltun, V. Vision transformers for dense prediction. *ArXiv preprint* (2021).

[28] Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., and Koltun, V. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020).

[29] Rodol, E., Bronstein, A. M., Albarelli, A., Bergamasco, F., and Torsello, A. A game-theoretic approach to deformable shape matching. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 182–189.

[30] Sarlin, P., DeTone, D., Malisiewicz, T., and Rabinovich, A. Superglue: Learning feature matching with graph neural networks. *CoRR abs/1911.11763* (2019).

[31] Schulman, J., Lee, A., Ho, J., and Abbeel, P. Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 1130–1137.

[32] Veltkamp, R. Shape matching: similarity measures and algorithms. In *Proceedings International Conference on Shape Modeling and Applications* (2001), pp. 188–197.

[33] Verma, N., Boyer, E., and Verbeek, J. Dynamic filters in graph convolutional networks. *CoRR abs/1706.05206* (2017).

[34] Wu, Y., Yan, W., Kurutach, T., Pinto, L., and Abbeel, P. Learning to manipulate deformable objects without demonstrations. In *Robotics Science and Systems* (07 2020), pp. 65–76.

[35] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems 32*, 1 (2021), 4–24.

[36] Yang, L., and Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing 415* (2020), 295–316.

[37] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI Open 1* (2020), 57–81.

Department of Computer Science, Babes-Bolyai University, 1, M. Kogalniceanu Street, 400084, Cluj-Napoca, Romania

Email address: mihai.loghin@ubbcluj.ro