DIFF GRAPH U-NET: A DIFFERENTIABLE U-SHAPE GRAPH NETWORK FOR GRAPH CLASSIFICATION

Manxi Lin, Mengge Hu, Guangya Shen

Technical University of Denmark

ABSTRACT

Recently, graph convolutional networks (GCNs) have been demonstrated efficient in learning graph representations. Regarding the down-sampling and up-sampling of non-Euclidean data, most existing methods are flat and lack robustness. We visualize the process of a state-of-the-art work DiffPool, and develop a novel differentiable module for upsampling called DiffUnpool. DiffPool and DiffUnpool learn soft cluster assignment for nodes via GCNs and multi-layer perceptrons respectively. To address the graph classification problem, based on DiffPool and DiffUnpool, we further propose an end-to-end encoder-decoder architecture, diff graph U-Net. Different from other U-shape models before, diff graph U-Net learns node embeddings hierarchically, and collect global features in residual fashion. Our experimental results show that our model yields an overall improvement of accuracy on 4 different data sets, compared with previous methods.

Index Terms— GCN, graph classification, hierarchical pooling, U-Net

1. INTRODUCTION

Convolutional neural networks (CNNs)[1] have been proved to have the considerable capability in many artificial intelligence challenges, such as image classification, medical image analysis, and natural language processing. Graph convolutional Network (GCNs)[2] has drawn more and more attention in recent years. It is a kind of neural network that can be applied in graphs directly, which provides a convenient way for node-level, edge-level, and graph-level prediction tasks in data represented by graphs, such as protein structures, social graphs, and point clouds.

GCNs have been implemented in a wide variety of tasks. Among them, a very important application is graph classification [3] [4], whose task is to make a prediction on labels associated with entire graphs. Most existing methods in this field work in a flat way: they are consist of an encoder architecture to generate embeddings for all the nodes in the graph, and a global pooling module to fuse embeddings. Traditional GCN architecture is not able to percept the hierarchical information, that is, it ignores the hierarchical structure of graphs. What's worse, even in some existing methods with hierarchical feature extraction[5], there are many problems, where one noticeable is that the deep encoder architecture in these networks reduces the robustness and limits the performance of the method. It prevents the GCN models from further improvement, and it may even lead to network performance degradation. One possible solution is U-shape graph network. [6] introduces a novel graph U-Net, but with global pooling, the way lack of hierarchical learning. In addition, due to large-scale computations, GCNs are always time-consuming and computationally expensive.

With these in mind, in order to improve GCN's performance in graph classification scenario, here we propose a novel end-to-end differentiable graph unpooling operation (DiffUnpool). Based on the pioneering work[5] and DiffUnpool, we develop a U-shape network architecture for graph classification. In this framework, differentiable pooling modules (DiffPool), together with GCN modules, generate embeddings and coarsen the graph hierarchically. DiffUnpool modules are stacked to restore the graph to its original structure, with the help of embeddings from the previous DiffUnpool module, and from the corresponding DiffPool module. After that, embeddings of each node are aggregated together and then sent to a classifier. Experimental results demonstrate that our proposed model outperforms previous methods.

2. RELATED WORK

Graph classification with graph convolutional networks. There are a bunch of applications of GCNs, including node classification, link prediction and graph classification. Among them, Node classification is similar to the pixel-wise segmentation task in grid-like data, such as images. Graph classification is to make a prediction on labels associated with graphs. It's analogous to image classification. Many GCNs have been proposed to solve graph classification problems[4][5][7]. Most of these methods follow the general "message-passing" architecture developed by [2][8], in which the layer-wise forward-propagation operation in the l-th layer is defined as:

$$E_{l+1} = GCN(A, E_l, W_l) = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}E_lW_l) \quad (1)$$

, where E_l and E_{l+1} are the input and output node embeddings of the layer, $\tilde{A} = A + I$ is summation of the input adjacency matrix A and an identity matrix I, $\tilde{D} = \sum_j \tilde{A}_{ij}$ is a diagonal node degree matrix to normalize $\tilde{A}$, σ is an activation function, and W_l is a trainable weight matrix.

Node embeddings are feature vectors correspond with each node in a graph. The original embeddings put into the network are decided by the graph data. They could be coordinates of point clouds, types of atoms in a molecule or user features in a social graph. It should be noted that the one-hot encoded labels for node classification, e.g., types of molecules in a protein molecule, can be used as node embeddings in graph classification. It's also noticeable that in "message-passing" architecture, we do not consider edge features, even though one can easily extend the algorithm to support them.

Pooling in graph convolutional networks. It's proved by many studies[5][9][10] that pooling plays an important role in GCNs. Amongst them, an interesting work would be DiffPool[5], a differentiable graph pooling module analogous to spatial pooling operation in CNNs. Via constructing assignment matrix, DiffPool can be deployed in end-to-end GCN architectures in a hierarchical fashion. The cluster assignment matrix $S_l \in \mathbb{R}^{n_l \times n_{l+1}}$, where n_l and n_{l+1} are node numbers in layer l and $l+1$, can be considered as a soft clustering of each node at layer l to the next coarsened layer $l+1$. Given the l-th DiffPool module in a network, the assignment matrix S is computed by:

$$S_l = softmax(GCN_{pool}(A_l, E_l, W_l^p)) \quad (2)$$

, where GCN_{pool} operation is described by equation 1, $A_l \in \mathbb{R}^{n_l \times n_l}$, $E_l \in \mathbb{R}^{n_{l+1} \times d}$ and W_l^p are the input adjacency matrix, node embeddings and trainable weight matrix in this layer. $softmax$ is implemented to normalize the assignment matrix. d is the feature size of node embeddings.

With the help of assignment matrix, the adjacency matrix and node embeddings are updated as:

$$A_{l+1} = S_l^T A_l S_l, \in \mathbb{R}^{n_{l+1} \times n_{l+1}} \quad (3)$$

$$E_{l+1} = GCN_{emb}(A_{l+1}, S_l^T E_l, W_l^e), \in \mathbb{R}^{n_{l+1} \times d} \quad (4)$$

Note that since GCN_{pool} and GCN_{emb} are two independent GCN modules, their weight matrix W_l^p and W_l^e are different.

U-Nets and Graph U-Nets. Encoder-decoder architectures such as the U-Net[11] are state-of-the-art methods for image segmentation, due to their efficiency and robustness. Regarding graph data, [6] proposes Graph U-Nets (g-U-Nets) for node classification and graph classification in 2019. In g-U-Nets, when pooling, nodes are sampled by gPool, to generate a smaller graph. gPool is a global pooling based on 1D footprint of each node. Node embeddings are projected to 1D space by a trainable vector. Nodes with top-k values in the scalar projection, are selected to construct the new graph. The new adjacency matrix and node embeddings are formed by row/column extraction and GCNs respectively. Selected-node index from each gPool layer is stored and then used by the corresponding unpooling layers, to reconstruct the original graph structure by adding empty embeddings for unselected nodes. It's clear that g-U-Nets follows a flat way when pooling and unpooling graphs.

3. DIFF GRAPH U-NET

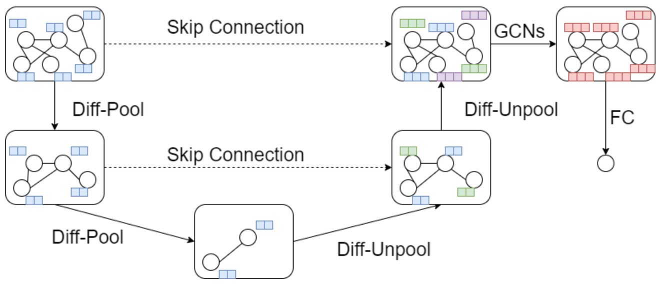


Fig. 1. Diff graph U-Net architecture

3.1. Differentiable unpooling module

Traditional up-sampling methods, e.g., deconvolution, are not applicable in graph data. Inspired by DiffPool, to develop a differentiable end-to-end graph-based encoder-decoder architecture, we propose the Differentiable unpooling module (DiffUnpool). DiffPool works as encoders in the network, to reduce feature map size and increase receptive field, while DiffUnpool plays the role of decoders, to reconstruct original graphs.

There are two GCNs in a DiffPool module, given by equation 2 and 4, leading to a considerable computational cost. For this reason, our DiffUnpool module is not simply an inverse operation of DiffPool. In layer l, the assignment matrix $S_{l,up} \in \mathbb{R}^{n_l \times n_{l+1}}$ is updated by a multi-layer perceptron (MLP)[12], MLP_s, as:

$$S_{l,up} = softmax(MLP_s(S_{l,pool}^T)) \quad (5)$$

, where $S_{l,pool} \in \mathbb{R}^{n_{l+1} \times n_l}$ is the assignment matrix from the corresponding pooling layer. For example, the first layer in unpooling process is related to the last layer in the pooling process.

The new adjacency matrix A_{l+1} is also computed by a MLP, MLP_a, as:

$$A_{l+1} = MLP_a(S_{l,up}^T A_l S_{l,up}) \quad (6)$$

, where A_l is the adjacency matrix from the previous layer.

To exploit graph representation, we employ a GCN to learn the topological information and generate node embeddings E_{l+1}:

$$E_{l+1} = \delta(GCN(A_{l+1}, S_{l,up}^T E_l, W_l)) \tag{7}$$

, where E_l stands for the input node embeddings of layer l. It does not represent node embeddings from the previous layer, but the output of skip connection, which would be mentioned later. W_l is a trainable matrix. δ means attention operation, it's defined as:

$$\delta(x) = softmax(x) \cdot x \tag{8}$$

, x is an arbitrary input vector. [10][13] have proved that attention mechanism improves deep neural networks' performance.

In the DiffUnpool module, compared with DiffPool, we replace one GCN with two independent MLPs. Ablation experiments demonstrate that not only training speed, but also the model performance has been ameliorated due to the replacement.

3.2. Network architecture

By stacking DiffPool and DiffUnpool modules, we develop a differentiable graph U-Net, as illustrated in figure 1.

Skip connection Skip connection is an important trick in U-Net to avoid potential information loss resulted from deep convolution. In diff graph U-Net, we implement skip connection for the same reason, but in a slightly different way.

Regarding the DiffUnpool layer l, we denote node embeddings from its previous layer as $E_{l-1,up} \in \mathbb{R}^{n_{l,up} \times d}$ and those from its corresponding pooling layer as $E_{l,p} \in \mathbb{R}^{n_{l,p} \times d}$, where d is the feature size of node embeddings, $n_{l,up}$ and $n_{l,p}$ are node number in the two layers respectively. If we set down-sampling and up-sampling ratio the same, $n_{l,up}$ would be equal to $n_{l,p}$. By skip connection, E_l is defined as:

$$E_l = MLP(E_{l-1,up} \oplus E_{l,p}) \in \mathbb{R}^{n_{l,p} \times 2d} \tag{9}$$

, where $\oplus$ means concatenation operation. Note that in most references, the output of the skip connection has the same size as the input feature vectors. That is, $E_l \in \mathbb{R}^{n_{l,p} \times d}$. However, in our case, the feature size increases over the unpooling process. Objectively speaking, our implementation increases the computational cost, whereas it keeps information in a higher degree.

Reconstruction loss Regarding graph classification problem, encoder-decoder architecture is implemented to generate a new graph structure that is slightly different from the original graph, to improve model robustness. Reconstruction loss is used to measure the similarity of the restored graph structure and the original one. We define the reconstruction loss as:

$$loss_{RC} = ||A_L - A_0||_1 / {n_0}^2 \tag{10}$$

, where $A_L \in \mathbb{R}^{n_0 \times n_0}$ and $A_0 \in \mathbb{R}^{n_0 \times n_0}$ are the adjacency matrix from the last unpooling module in U-Net, and that of the original graph, respectively. They have the same size.

During training, L_rcs is added to the classification loss. The overall loss is defined as:

$$loss = (1-\lambda)\cdot(loss_{CE}+loss_E+loss_{LP})+\lambda\cdot loss_{RC} \tag{11}$$

, where $loss_{CE}$ is cross-entropy loss for classification, $loss_E$ and $loss_{LP}$ are entropy regularization loss and link prediction loss from [5]. They work as auxiliaries to stabilize the training. The reconstruction factor λ ought to stay at a low value. An extremely-high λ may weaken the role of U-Net: the input and output of U-Net tend to be the same.

Global feature aggregation After a bunch of DiffPool and DiffUnpool modules, the input graph of U-Net is encoded and decoded to a new graph. To solve graph classification problem, we employ a GCN block, shown in figure 2, to capture the topological information in the new graph. We call it global feature aggregator. The block consists of two GCNs in residual fashion[14]. Given an input graph G, adjacency matrix A and node embeddings E, the output of the two GCNs and the block would be:

$$E_1 = GCN_1(A, E, W_1) \tag{12}$$

$$E_2 = GCN_2(A, E_1, W_2) \tag{13}$$

$$E_o = E + E_2 \tag{14}$$

, where W_1 and W_2 are weight matrix.

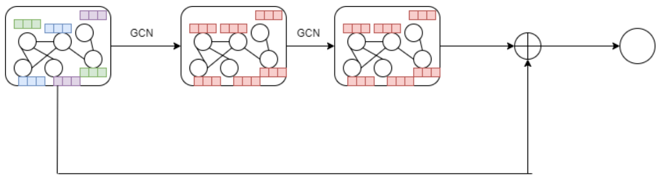


Fig. 2. Global feature aggregator

Residual structure helps improve model performance and avoid potential network degradation. After the global feature aggregator, graph embedding goes through a fully-connected layer and is then put into softmax function to make the final prediction.

4. EXPERIMENTS

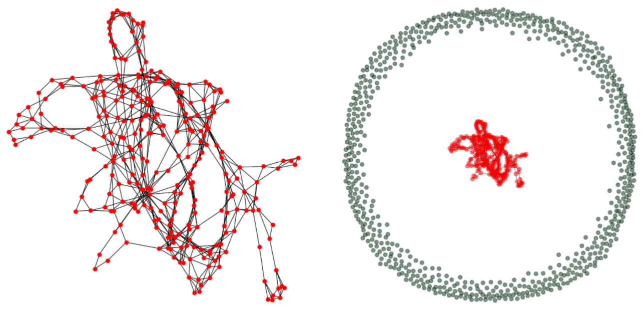
We evaluate our networks with previous state-of-the-art models on node classification.

Datasets We use four different data sets for graph classification to probe the ability of Diff graph U-Net. D&D[15] consists of 1178 protein structures. Each protein is represented by a graph. These proteins are labeled as enzymes or non-enzymes. ENZYMES[16] is a data set describing protein tertiary structures. It includes 600 graphs from 6 enzyme classes. Peking_1[17] contains 85 compounds from female rats. Each compound is labeled by whether its carcinogenic activity would cause cancer or not. PROTEINS[18] revolves 1113 protein structures from the Protein Data Bank. Simple features such as secondary-structure content, amino acid propensities, surface properties and ligands are used as node embeddings. The data set also has two different classes, enzymes and non-enzymes.

Baseline model We employ a GCN network including three DiffPool modules as the baseline. Regardless of size, the input graph is first down-sampled or up-sampled to 1000 nodes, and then pooled three times, with an assignment ratio of 10% at each time. After the DiffPool modules, there is only one node in the graph. We regard the node embedding as the graph's global features and deploy the softmax function to make a prediction. ReLU[19] works as the activation function in GCNs.

Diff graph U-Net Based on the architecture proposed in section 3.2, we build a diff graph U-Net consist of 3 DiffPool modules and 3 DiffUnpool modules. The pooling and unpooling ratio are set the same. For D&D and ENZYMES, the ratio is 50% and for the rest two, the ratio is 10%. To avoid overfitting, we apply L-2 regularization on weights. Dropout[20] is applied to global feature aggregator. Their hyperparameters are fine-tuned on each data set. Apart from these, SELU[21] replaces ReLU as the activation function in GCNs, due to its contribution to model convergence. Plus, the reconstruction factor λ in equation 11 is set 0.1.

4.1. Visualization of DiffPool



(a) Raw Graph (b) Sampled Graph

Fig. 3. Up-sampling: initialization

Before evaluation, we visualize the topological representation change of a graph in the baseline model, to illustrate what is happening inside our networks.

We pick one graph randomly from the D&D data set, which is presented in figure 3. The left one is the raw graph including 288 nodes and 757 edges, while the right one is the up-sampled graph, with 1000 nodes inside, where the 712 fill-in ones are colored green.

In the DiffPool module, an assignment matrix is deployed to soft assign nodes into different clusters. Rather than that in hard clustering, which means each node belongs to a determined group, in soft clustering, each node has a different probability to be allocated to various clusters, and its embedding is distributed into the clusters according to the probability.

Hard clustering is usually easier to visualize and understand. If we assign each node to the cluster with the highest probability, then the process would be hard clustering. The assignment result is demonstrated in figure 4. In the left figure, 100 nodes in 42 clusters are in different colors. In the right chart, node embeddings in each cluster are fused as 42 cluster embeddings, which are deemed to be the outputs of the pooling operation.

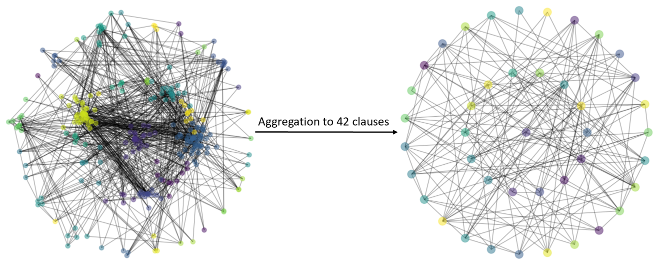


Fig. 4. Hard Assignment in the second DiffPool module

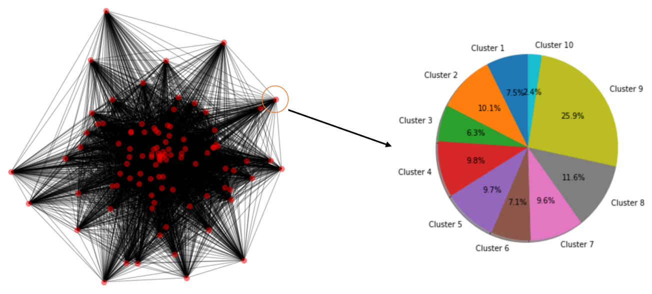


Fig. 5. Soft Assignment in the second DiffPool module

However, since it's very inflexible to assign each node to one cluster completely, and it's impossible to ensure group number in the hard assignment, in DiffPool module, soft clustering is employed. In the soft assignment, node embeddings are assigned to a given number of groups with respect to the assignment matrix, described by the equation 4. Each node has contributions to each cluster. In figure 5, the left chart illustrates the output graph of the first pooling module, which contains 100 nodes. The right pie chart shows the contribution of one node to the ten clusters. It's actually one row in the assignment matrix.

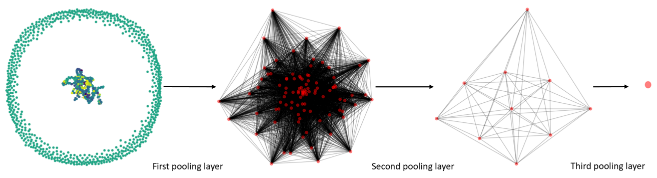


Fig. 6. Visualization of baseline pipeline

Figure 6 demonstrates the complete execution process of the baseline model. The input graph is pooled three times with a down-sampling rate of 10%, until there is only one node in the graph. Nodes in each module correspond with clusters in the previous module. After the last module, the node's feature vector, also called graph embedding, is sent into a classifier to make the final prediction.

4.2. Results for graph classification

We compare the performance of Diff Graph U-Net to the baseline. Each model is trained for 100 epochs and early stopping[22] is applied. The batch size is set to 20. Table 1 presents the accuracy results of the two methods on different data sets. 10-fold cross-validation is employed to convince accuracy. The final result is the average accuracy over the 10 iterations. Xavier initialization[23] is implemented.

Data sets	Classes	Method	
		Baseline	Diff graph U-Net
D&D	2	79.49	85.47
ENZYMES	6	55.00	58.33
Peking_1	2	77.6	87.5
PROTEINS	2	77.78	88.89

Table 1. Accuracy results of the two models

We observe that the diff graph U-Net outperforms the baseline model on all the four data sets. Regarding binary classification problems(D&D, Peking_1 and PROTEINS), diff graph U-Net has obviously better performance. Experiment results demonstrate that diff graph U-Net has an overall improvement over the previous methods[1].

Shall we completely owe the results to the increase of trainable parameters?[2] Since diff graph U-Net outperforms the baseline model on a small-scale data set, Peking_1, without severe over-fitting and degradation, we believe that there are factors other than parameter number in the structure improving the model performance. Neural networks are not always the deeper, the better.

5. DISCUSSION

5.1. Limitations

There are many limitations to our work. Because of the time limit, we haven't done all the ablation experiments and therefore not able to show them in the report. In addition, even though in the network structures, MLPs replace GCNs in many places to reduce computation, diff graph U-Net still costs a long time to train. More specifically, the time diff graph U-Net takes is roughly two times as long as that consumed by the baseline model.

Plus, one of our observations is that diff graph U-Net could be very unstable to train. There is considerable variation in accuracy results across different runs, even with the same hyperparameter setting. It's the reason behind our choice of 10-fold cross-validation.

Diff graph U-Net does not well in multi-class classification problem on ENZYMES, which is also an aporia in this field at present.

5.2. Further Work

Interesting future directions may include the following points, based on our current work.

- Develop a computational-efficient and light-weight network structure based on present work. Replacing soft assignment with hardware-friendly Hard assignment is a potential way.
- Apply diff graph U-Nets in other tasks. U-Net is designed for image segmentation originally. Diff graph U-Net ought to be more suitable for solving similar problems such as node classification.
- Employ data argumentation. The diff graph U-Net could be very unstable to train. One of the reasons is the small size of the data set. By deploying diff graph U-Net as a generator, we plan to develop a Siamese network for graph data argumentation.
- Study the relationship between U-Net's depth and its ability.

6. CONCLUSION

We propose a differentiable end-to-end encoder-decoder network, diff graph U-Net, for graph classification. It pools and unpools graph hierarchically and outperforms many former methods. To have a deeper glance into our work, check our Github repository.

[1]We also compare our accuracy results with those from graph U-Net[6]. On D&D and PROTEINS, graph U-Net reaches an accuracy of 82.43% and 77.68%, lower than diff graph U-Net's 85.47% and 88.89%.

[2]The baseline model has 3 modules while in diff graph U-Net, the number is 6.

7. REFERENCES

[1] Bottou L. Orr G. B. LeCun, Y. and K.-R. Muller, "Efficient backprop," in *Neural networks: Tricks of the trade*, 2012, pp. 9–48.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[3] B. Dai H. Dai and L. Song., "Discriminative embeddings of latent variable models for structured data," in *International Conference on Machine Learning*, 2016, pp. 2702–2711.

[4] J. Iparraguirre R. Bombarell T. Hirzel A. Aspuru-Guzik D. K. Duvenaud, D. Maclaurin and R. P. Adams., "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[5] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, 2018, pp. 4800–4810.

[6] Hongyang Gao and Shuiwang Ji, "Graph U-Nets," 2019.

[7] Daizong Liu, Hongting Zhang, and Pan Zhou, "Video-based facial expression recognition using graph convolutional networks," *arXiv preprint arXiv:2010.13386*, 2020.

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[9] Filippo Maria, Bianchi Daniele, and Grattarola Cesare, "Spectral Clustering with Graph Neural Networks for Graph Pooling," 2020.

[10] Junhyun Lee, Inyeop Lee, and Jaewoo Kang, "Self-Attention Graph Pooling," 2019.

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[12] Martin Riedmiller and AM Lernen, "Multi layer perceptron," *Machine Learning Lab Special Lecture, University of Freiburg*, pp. 7–24, 2014.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[15] Paul D Dobson and Andrew J Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[16] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[17] Ryan A. Rossi and Nesreen K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015.

[18] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt, "Scalable kernels for graphs with continuous attributes," *Advances in neural information processing systems*, vol. 26, pp. 216–224, 2013.

[19] Abien Fred Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, pp. 971–980, 2017.

[22] Rich Caruana, Steve Lawrence, and C Lee Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in neural information processing systems*, 2001, pp. 402–408.

[23] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.