Going into the networks of Life

Final Report - ML in Network Science

Dwivedi Deepesh
b00792828@essec.edu
DSBA Master - CentraleSupélec

Giacopelli Nicolò
b00794705@essec.edu
DSBA Master - CentraleSupélec

Liu Dayu
b00789480@essec.edu
DSBA Master - CentraleSupélec

Remadi Adel
b00804320@essec.edu
DSBA Master - CentraleSupélec

Abstract

This report draws its interest in bioinformatics and the innovations brought by the AI revolution in what concerns the study of the primary sources of Life. In particular, the present project attempts to predict protein structures' enzymatic functions starting from their graph representation, before trying to design graph structural approximations of enzymes through the use of generative models. Indeed, function prediction and function design are both considered as key tasks in the domain of protein engineering and are at the core of several real life applications such as drug discovery, gene editing, and antibodies therapeutics. In that regard, 2 graph classification and 1 graph generation tasks were performed on the PROTEINS and ENZYMES datasets and a total of 7 graph Neural Network models were designed and compared in the process. The generated proteins were then evaluated using the best performing classifier to assess the quality of the developed generation process.

Keywords: bioinformatics, proteins, enzymes, graph classification, graph generation

1 Introduction and motivation

Proteins are complex macromolecules that are essential for a wide range of biological processes. They are made up of long chains of amino acids that fold into specific three-dimensional structures, referred to as *protein-folding*, which in turn determine their biological function. Proteins play a critical role in maintaining the structure and function of cells, and they are involved in very different processes such as enzyme catalysis, signaling, defense, storage and transport. Enzyme catalysis, specifically, is of particular interest. In fact, enzymes are a specific type of protein that catalyze chemical reactions in living organisms. They are highly specific, meaning that each enzyme can catalyze a particular chemical reaction or set of reactions. They function by binding to specific substrates and speeding up their chemical reactions, so that they become essential for many metabolic processes in living organisms, and they play a critical role in maintaining cellular homeostasis. Notwithstanding their importance, they are still subject of intense research and still not completely understood.

The first key task for what concerns the study of these micro-biological structures is the ability to discriminate between proteins that attend an enzymatic function and those that do not. Being able to distinguish between enzymes and other proteins is important because it allows researchers to understand how biochemical reactions are regulated and how different enzymes work together to carry out specific biological processes. It also enables the development of targeted therapies for diseases that result from enzyme deficiencies or dysfunctions. This can be done by studying the unique three-dimensional structure of a protein and its active sites, since enzymes perform their function due to their specific substrates that bind with specific chemical reactors. This specific type of protein engineering tasks can be referred to as function prediction, and is considered as one of the major challenges in that field. In our project, we evaluated state-of-the-art deep learning techniques in **Graph-level classification** to test the predictive ability of several models. The aim was to leverage on a graph representation of the proteins' structure to exploit both the information contained in the sequence of amino acids, as well as the spatial information in order to infer the protein's function.

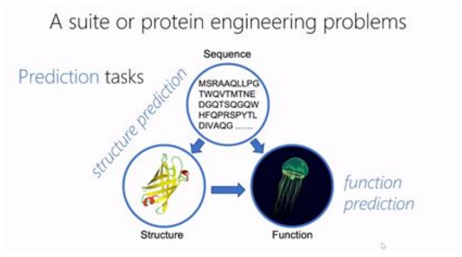
A suite or protein engineering problems
Prediction tasks
Sequence
MSRAAQLLPG TWQVTMTNE DGQTSQGQW HFQPRSPYTL DIVAQG ......
structure prediction
Structure
Function
function prediction

Figure 1. Key Prediction tasks in Protein Engineering

The second important task, which hinges on the first, is the Design task, which starts from the identification of the biological function of an enzyme and aims at reconstructing the protein, by defining its spatial location as well as the amino acid sequence by which it is composed. The generation of potential candidates for enzymatic structures is indeed a booming area of research, with ground breaking potential implications. We approached this task through the constantly improving techniques of **Graph generation**, in order to simulate protein structures close enough to available examples of enzymes. One key motivation for using Graph Neural Networks (GNNs) for enzyme generation is that enzymes are inherently graph-like structures, composed of amino acid residues that interact with each other through various physical and chemical interactions. Traditional machine learning methods are often limited in their ability to capture these complex interactions, but GNNs have been shown to be effective at modeling graph data and extracting meaningful features. This can lead to new insights into enzyme activity, new drug targets, and more efficient enzyme engineering, with potential benefits for a range of fields, from biomedicine to biotechnology and environmental science. The timingness of the task is demonstrated by recent papers like the one of AlphaFold by DeepMind in 2021 ([3]), which we set as an inspiration for this project.

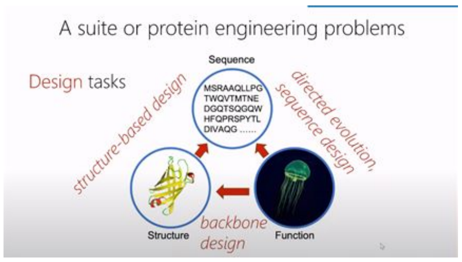

Figure 2. Key Design tasks in Protein Engineering

2 Problem definition and related work

Our project developed around the tasks of Protein Design and Function Prediction, which are strictly related, in order to see at what point positive externalities on one side could help for the other. In particular, we used publicly available datasets of protein structures and their graph representation in order to train an Enzyme Classifier and a Generator independently, pushing the boundaries as much as possible in both tasks, to then evaluate the ability of our Generator to foul a Classifier trained on the full dataset. For this objective, we evaluated state-of-the-art techniques based on Graph Neural Networks, to test their appropriateness on a smaller scale for this important area of research of molecular biology.

Graph Classification The first task we plan to tackle is a Graph Classification one. In this context, we want to learn a function $f : \mathcal{G} \rightarrow \mathcal{Y}$ predicting a property of the graph. For instance, if $\mathcal{G}$ is the set of molecular graphs, it may be interesting to be accurate in predicting the absence or presence of toxicity. In our case, our objective is to either distinguish whether a given protein is an enzyme or not, as well as to predict the specific class of enzyme. In this field, the main strand of research have passed from Graph kernels to Graph Neural Networks around the year 2016. We followed [12] which benchmarks different architectures based on the interaction of different graph Message Passing methods and Pooling techniques. Many pooling techniques have been recently developed, which can be broadly defined into two categories, namely **1**. those that implement a strategy to downsample a graph to the most salient nodes and **2**. those that aim at obtaining the graph embeddings by exploring the hierarchical structure of each input graph. In the first group, the Self-Attention Graph Pooling technique is comprised (from [6]), which selects a sparse subsample of the embedding matrix Z based on self-attention weights, that is probability distribution over the set of nodes. In our case, the self-attention weights are computed through a Graph Convolutional Network as described in [4], followed by a sparse softmax transformation (described in [7]). In this group we could also consider traditional readout techniques like mean and maximum over the set of nodes, whose performance has been evaluated as described in the following. The second group comprehends techniques like DiffPool (from [13]) and HGPSL Pooling (from [15]), which aim at assigning nodes to clusters in a soft way after each layer of Graph Convolutions, by creating hierarchies. The former exploits two parallel GCNs to predict the embedding matrix Z and the matrix of soft cluster assignment after each layer. The latter is instead a state-of-the-art technique which firstly selects different examples of subgraphs according to a node information score which summarizes the degree of novelty brought by the node in its subgraph and then re-learns the structure of the selected graph by re-weighting the edges.

Graph Generation was the second more challenging task in which we evaluated the power of simulation of Graph Generation techniques for tackle enzyme design. In particular, we exploited previous knowledge in GANs applied to Graph structures (described in [11]), and VGAE, or Variational Graph Auto-Encoders (from [5]). The former consists of two neural networks - a generator network G that generates new data, and a discriminator network D that evaluates the realism of the generated data, which compete in a min-max gain to reach a Nash Equilibrium, at which hopefully the generator is able to simulate good candidates for the data it trained on. The second case is a more complicated technique that introduces a structure on the latent space, penalizing deviations from a multivariate Gaussian distribution, together with a reparametrization technique, in order to successfully exploit the decoder part of the architecture and transform randomly sampled noise in a graph structure. The following describes the steps we took to address these two strictly related tasks, together with the architecture we evaluated.

3 Methodology and Evaluation

The general approach will be the one of benchmarking different techniques and methods to perform two different but interrelated task of classification and generation.

Graph Classification Our plan is then to benchmark techniques of supervised graph classification with GCN ([4]), and Deep Graph CNN ([14]) and GraphSAGE ([2]). Different architectures will be considered, as well as specific modules, so that different aggregator functions like Top-K and hierarchical pooling module ([15]) will be tried out, as well as the Attention mechanism ([10]). At the same time, another benchmark will be constituted by unsupervised embedding techniques, such as graph2vec ([9]) and Unsupervised Graph-level embedding (from [1]), which can incorporate a supervised loss for classification. Comparing different embedding techniques will prove to be extremely interesting in relation to the second task of graph generation.

Datasets In our project we considered two datasets from an open-source project named TUDataset, started in 2016 ([8]). The datasets are called PROTEINS and ENZYMES. The former concerns a binary classification problem of proteins into enzyme/non-enzyme, while the latter comprises (only) enzymes assigned to one of the 6 EC functional classes, each providing information about the type of reaction catalyzed by the enzyme. The first dataset contains 1113 graphs (avg n. of nodes $n = 40$, avg n. of edges $m = 73$), comprising 450 enzymes and 663 non-enzymes. The second contains 600 enzymes ($n = 33$, $m = 62$), and is a much harder dataset to benchmark on. The two datasets are not perfectly substitutable, even if they concern the same topic, due to the specifities of the information and the way the presence of an edge is defined. Since our purpose for generation was being able to generate protein recognized as enzyme by a trained classifier, and due to the reason above, we decided to focus our attention on the PROTEINS dataset for the generative task. The graph classification task is instead performed on both datasets, and we proceed to show our results. The main reason for chosing the TUDataset is that it is a widely used dataset in **Benchmarking** of algorithms. Proteins and enzymes can be naturally represented as graphs, with nodes representing amino acids and edges representing the connections between them. The TUDataset's graph-based representation is well-suited for capturing the complex structural information of proteins, which is crucial for accurate classification. By utilizing this dataset, we can leverage the power of graph neural networks for protein and enzyme classification.

3.1 Graph Classification on ENZYMES

In this section, we tackle the problem of enzyme classification (6 categories) using graph neural networks (GNNs) on the ENZYMES dataset provided by TUDatasets. The classification of enzymes into these six categories is based on the type of reaction they catalyze and the mechanism by which they do so.

Model implementation We implemented two GNN models, a basic Graph Convolutional Network (GCN) model and a Graph Attention Network (GAT) model, and compared their performance. The dataset is split into training, validation, and test sets, and we evaluated the models based on their classification accuracy. Our results demonstrate the effectiveness of GNNs in the enzyme classification task and highlight the importance of the TU dataset of enzymes for studying enzyme function and protein structure. Graph Convolutional Network (GCN) model consists of 3 GraphConv layers from DGL, followed by a linear layer for classification. We used the leaky ReLU activation function between layers for non-linearity. The model performs message passing to learn node representations and then averages the node representations to obtain a graph embedding. The final linear layer is used to classify the graph representation. Graph Attention Network (GAT) model extends the GCN model by incorporating an attention mechanism. We used the GATConv layers from DGL and implemented a multi-head attention mechanism. Similar to the GCN model, the GAT model performs message passing to learn node representations, averages the node representations for a graph embedding, and uses a linear layer for classification. Here we used 3 GATConv layers with 5 attention heads which gave the highest accuracy.

Training and Evaluation We trained both models using the Adam optimizer and the cross-entropy loss function for 150 epochs. The training is done on the training set, and the models' performance is evaluated on the validation set every five epochs. After training, we test the models on the test set to obtain their classification accuracy.

Graph Classification - ENZYMES	
Model	Test Accuracy
GCN	50 %
GAT	62 %

In this study, we evaluated the performance of two graph neural network models, a basic Graph Convolutional Network (GCN) model and a Graph Attention Network (GAT) model, for the enzyme classification task on the ENZYMES dataset. We measured the models' performance in terms of classification accuracy. The GAT model achieved an accuracy of 62.5%, while the GCN model had an accuracy of 50%.
The **GAT** model emerged as the superior performer in the enzyme classification task. This model leverages an attention mechanism to identify the most informative neighboring

nodes for each node during the message passing phase, allowing it to capture more relevant information for enzyme classification. The attention mechanism enables the model to adaptively assign different importance levels to neighboring nodes, effectively capturing the complex relationships between amino acid residues in enzymes' protein structures. The **GCN** model, although not as effective as the GAT model, still achieved a respectable classification accuracy of 50%. GCNs are a simpler graph neural network architecture, lacking the attention mechanism present in GATs. As a result, the GCN model may struggle to capture the intricate relationships between amino acid residues in enzymes' protein structures, which could explain its lower performance compared to the GAT model.

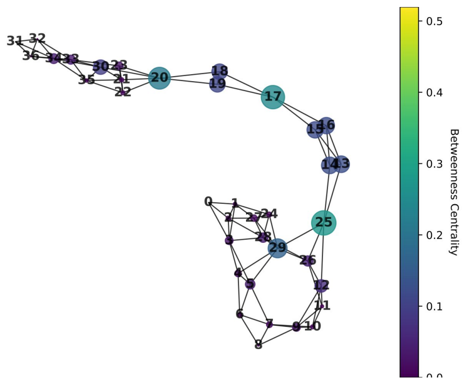

Figure 3. Betweenness Centrality of Nodes

Discussion Our results demonstrate the potential of graph neural networks for enzyme classification tasks, with the GAT model outperforming the basic GCN model. The GAT model's attention mechanism allows it to capture more relevant information from neighboring nodes, contributing to its higher accuracy. This highlights the importance of considering the complex relationships between amino acid residues when classifying enzymes based on their protein structures. The performance gap between the GAT and GCN models suggests that incorporating attention mechanisms in graph neural network models can lead to improved performance in enzyme classification tasks. Future work could explore additional GNN architectures, such as Graph Isomorphism Networks (GINs) and ChebNet, to further assess their potential in enzyme classification tasks. Moreover, the development of more sophisticated readout functions for GNNs could be explored to improve performance. In this study, we employed a simple readout function that computes the mean of all node representations to obtain the graph embedding. Alternative readout functions, such as pooling or attention-based methods, could potentially yield better performance. In conclusion, our results emphasize the potential of graph neural networks, particularly GAT models, for enzyme classification tasks. By leveraging the attention mechanism and the rich structural information present in the ENZYMES dataset, GNNs can contribute to advancing computational biology, drug discovery, and general understanding of the molecular mechanisms underlying enzyme catalysis. Based on the betweenness centrality analysis of the enzyme protein interaction network, we observed that some nodes, representing amino acid residues, have higher betweenness centrality values. These nodes can be considered as critical residues for the enzyme's function, as they connect different regions of the enzyme and facilitate communication between functional sites, structural stability, and evolutionary conservation.
The interpretation of the betweenness centrality analysis in the context of enzyme function can be summarized as follows:

1. Communication between functional sites: Nodes with high betweenness centrality are important in connecting the active site, substrate binding pocket, dimer interface, and other regions involved in the enzyme's conformational changes during catalysis. These residues may play a crucial role in transmitting information.
2. Structural stability: High betweenness centrality residues often participate in a network of hydrogen bonds and electrostatic interactions that contribute to the enzyme's structural stability. These interactions help maintain the proper orientation of active site residues, allowing efficient substrate binding and catalysis.
3. Evolutionary conservation: The high betweenness centrality residues in the enzyme may be conserved across orthologs from different species. This evolutionary conservation indicates their importance in maintaining the enzyme's function and suggests that they are under strong selective pressure to preserve their roles in the catalytic process.

In conclusion, the betweenness centrality analysis of the enzyme protein interaction network provides valuable insights into the critical residues that contribute to the enzyme's function. By identifying these key residues, we can better understand the molecular mechanisms underlying enzyme catalysis and potentially design targeted mutations or small molecule inhibitors to modulate the enzyme's activity for therapeutic purposes.

3.2 Graph Classification on PROTEINS

Our analysis subsequently focused on the PROTEINS dataset, accessible directly from the TUDataset collection as well. This is the dataset on which we performed both a classification and a generation task (covered in section 3.3), so that our efforts went towards building a strong classifier, to probe for the positive externalities mentioned in the problem definition of this project.

Model Implementation Our efforts went towards the benchmarking of different pooling techniques for graph classification. For all the models, we tried to stick to a similar setup in order to compare the methodologies, and only coarse finetuning. The dataset was divided in training (75%), validation (15%) and holdout sample. An Early stopping technique based on accuracy on the validation dataset has been used, which stopped training after a series of epochs with no improvement in the metric. In our case the patience was set to 15. The width of all networks has been maintained around a magnitude of 128, with a variable depth depending on the pooling technique (from 5 up to 10). AdamW optimizer has been chosen for training (AMSGrad variant), with weight decay (0.01), and a learning rate of $2 \cdot 10^{-4}$. Training batch size has been fixed to 5 graphs.

Mean and Max readout We started with a simple **Mean** and **Max** readout setting, which aggregate the embeddings of all the nodes in a graph and predict through a classification head. The number of hidden layers has been fixed to 5, with an ELU activation function. As the table below shows, we found the Max pooling to be a surprisingly competitive technique, very robust around 75% of accuracy on the test dataset.

DiffPool Differential Pooling ([13]) aims at learning a hierarchical representation of the graph, by assigning the nodes of the graph to a cluster in a soft way. The assignment matrix S^l at layer l, of dimension $n_l \times n_{l+1}$, maps the number of clusters of the l layer to the new number n_{l+1}, with n_0 initialized at the number of nodes of the graph. The assignment matrix is learned through a Graph Neural Network $\text{GNN}_{l,\text{pool}}$ operator followed by a softmax activation. This procedure is independent from the one of learning the feature embeddings Z_l, which is done with a different $\text{GNN}_{l,\text{embed}}$. The adjacency matrix A_{l+1} and the feature matrix X_{l+1} are then updated from Z_l and S^l. A dropout rate parameter determines how many nodes are selected after the pooling, and we fixed it to 0.4.

The table below shows that DiffPool is the second best performing technique, and it indeed shows a stable training behaviour and a good performance in discrimination, as showed by the confusion matrix in Figure 4.

Graph Classification - PROTEINS			
Model	Test Dataset		
	Acc	F1	AUC
Mean readout	67.26 %	0.433	0.629
Max readout	75.595 %	0.63	0.82
DiffPool	76.19 %	0.672	0.806
SAGPool	72.024 %	0.657	0.724
HGPSLPool	**79.167** %	textbf0.706	**0.828**

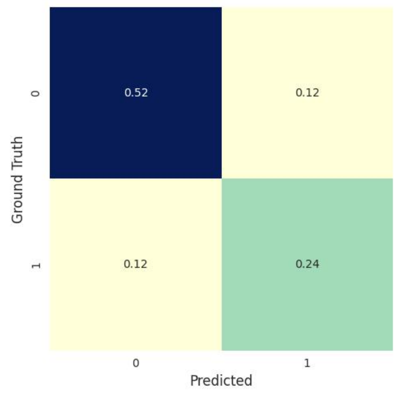

Figure 4. Confusion matrix on holdout - DiffPool

SAGPool Self-Attention Graph Pooling ([6]) exploits a self-attention mask to select a subgraph that is kept as input for the successive graph convolution. The self-attention scores are computed very similarly to a graph convolution, that is $Z = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}X\Theta_{att}\right)$, with a softmax activation on top and an additional linear layer Θ_{att}. The top $k * N$ nodes are then kept for the next convolution, where k is the pooling ratio. In our experiments, this technique is overpassed by a simple Max readout method.

HGPSLPool As the table shows, our best results are obtained with a state-of-the-art technique called HGPSLPool. Hierarchical Graph Pooling with Structure Learning (HGPSLPool) is a sophisticated model that combines graph pooling and structure learning. It first selects a subset of nodes based on their importance scores and then learns a new graph structure on the pooled nodes. Once the new subgraph is built, the new structure is re-learnt by assigning a weight to any potential edge, even if not existing ex-ante. The model can be configured to use different settings, such as sampling neighbors, sparse attention, and structure learning. There are 2 important components for HGPSLPool Model, namely a Graph Convolutional Network and a Node Information Score. The former, as we know, incorporates both node features and the local graph structure. In your code, the GCN layer is defined as a class that inherits from the MessagePassing class. The forward method of the GCN class takes the input node features and an edge index (which represents the graph structure) and computes the output node features by aggregating information from neighboring nodes. The aggregation is performed using a combination of the input features and edge weights, resulting in updated node features that incorporate information from the local neighborhood. Regarding the Node Information Score, it is computed by aggregating the input node features using the forward method of the Node Information Score class. The higher the score, the more information is contained in the node, and this score is later used in the HGPSLPool model to determine which nodes to keep during the pooling process. The forward method of the NodeInformationScore class takes the input node features, edge index, and edge weight and computes an information score for each node in the graph using a message passing approach. The table above shows the results we obtained when benchmarking different pooling techniques for graph classification, and it shows that indeed HGPSLPooling was found to be the best performing one, even if DiffPool stands its way.

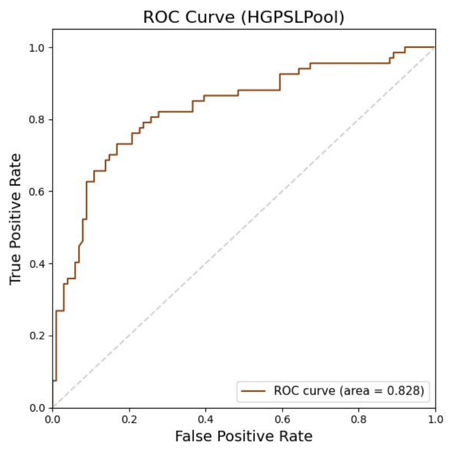

Figure 5. ROC Curve - HGPSLPool

3.3 Graph Generation on PROTEINS

In this final section, we address the challenging task of graph generation, and more precisely the process of designing promising Enzyme approximations using the PROTEINS dataset. Two generative models were explored, GraphGAN and VGAE, and the quality of their respective generative processes were assessed using the HGPSLPool classifier presented in section 3.2.

GraphGAN This type of model required the creation of both a graph discriminator and a graph generator. Each are neural network models with antagonistic roles. While the generator was trained to generate protein graphs that would be classified as enzymes by the discriminator, the latter had the task of discriminating generated enzymes with real ones sampled from the PROTEINS dataset. This type of architecture is very sensitive and challenging to train, and both models must have a well-balanced power in their respective task to reach a promising equilibrium. For the generator, we first performed a statistical analysis on the real data to obtain the means and standard deviations of the number of nodes grouped by their attributes. We then used these distributions as the starting point for the generator to sample specific number of nodes from each type. These were set as inputs into the generator, which generated a latent representation matrix and an "n" value that determines the percentage of edges that should be connected in the subsequent inner product decoder. We used this inner product decoder to generate a graph by taking the n% highest inner product scores as the connected edges. Then, both the generated graphs and real enzyme graphs were fed into the discriminator, which was trained to identify the source of each of its inputs. The parameters of both models were updated simultaneously based on opposing loss functions. Figure 6 below illustrate the overall pipeline that was designed to train the graphGAN generator.

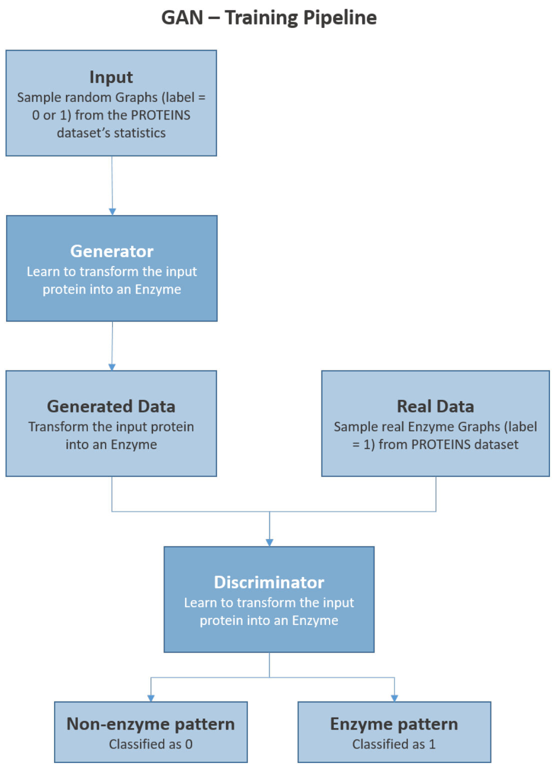
GAN – Training Pipeline
Input
Sample random Graphs (label = 0 or 1) from the PROTEINS dataset's statistics
Generator
Learn to transform the input protein into an Enzyme
Generated Data
Transform the input protein into an Enzyme
Real Data
Sample real Enzyme Graphs (label = 1) from PROTEINS dataset
Discriminator
Learn to transform the input protein into an Enzyme
Non-enzyme pattern
Classified as 0
Enzyme pattern
Classified as 1

Figure 6. GAN - Training Pipeline

VGAE For this model, we used a GCN-based VGAE Encoder to learn a latent representation Z from the real enzymes (label=1) of the PROTEINS dataset. We then built a noise generator model, composed of a GAT layer and multiple linear layers. The noise generator had the purpose to increase the variance of the latent representations and facilitate the model's ability to generate new designs. The noisy latent representation is then passed through a decoder composed of a series of linear layers and a final inner product. The output of the linear layers is used along with the real edge index to compute the reconstruction loss and update the generator model's parameters. The inner product creates a symmetric ranking matrix on which only the top 25% values led to the generation of an edge. The overall pipeline of the VGAE graph generation process is illustrated below in Figure 7.

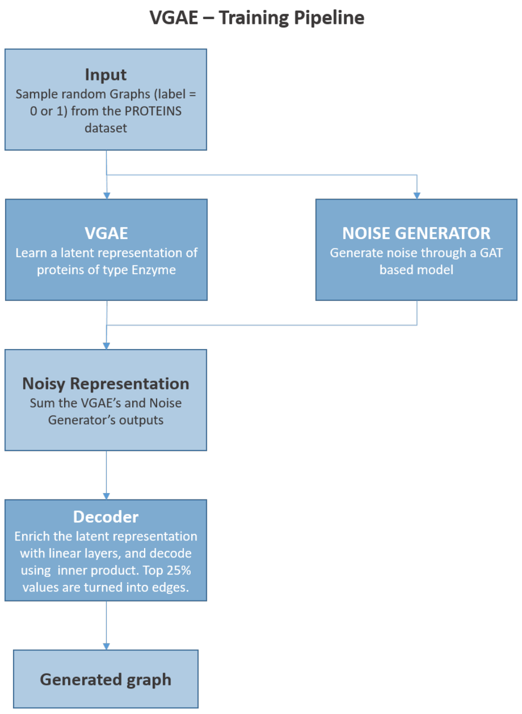
VGAE – Training Pipeline
Input
Sample random Graphs (label = 0 or 1) from the PROTEINS dataset
VGAE
Learn a latent representation of proteins of type Enzyme
NOISE GENERATOR
Generate noise through a GAT based model
Noisy Representation
Sum the VGAE's and Noise Generator's outputs
Decoder
Enrich the latent representation with linear layers, and decode using inner product. Top 25% values are turned into edges.
Generated graph

Figure 7. VGAE - Training Pipeline

Discussion on the generative pipelines As a comparison between the two models, the VGAE had the nice property of generating a wider variety of protein graphs. Indeed, the graphGAN model was bounded by attributes distributions while the VGAE took any type of graph proteins as inputs (enzyme or not) and tried to generate new enzymes out of it. On the other hand, however, even though it produced less variety of graphs, the graphGAN had the nice property of integrating the distributions of the node attributes, which the VGAE model did not have. We initially made an attempt to include the learning of generating node attributes directly into the layers of the VGAE's decoder, which was eventually not fruitful and therefore discarded from the final pipeline. A few examples of the generated proteins from the VGAE model are illustrated below in Figure 8:

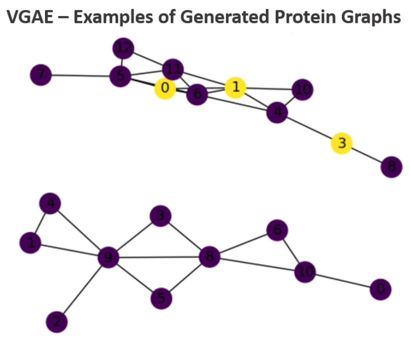

Figure 8. VGAE- Examples of generated protein graphs

In order to evaluate the quality of the generation process of each models, 1000 proteins were generated using the presented generative models. As these models were trained to generate proteins with the property of being enzymes, the evaluation metric was defined as the percentage of generated proteins that would be classified as an enzyme by the best classifier presented in section 3.2, namely the HGPSLPool classifier which obtained 79% accuracy on the PROTEINS dataset. The evaluation pipeline is described below.

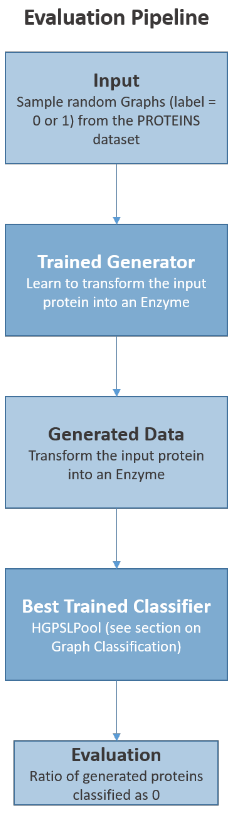
Evaluation Pipeline
Input
Sample random Graphs (label = 0 or 1) from the PROTEINS dataset
Trained Generator
Learn to transform the input protein into an Enzyme
Generated Data
Transform the input protein into an Enzyme
Best Trained Classifier
HGPSLPool (see section on Graph Classification)
Evaluation
Ratio of generated proteins classified as 0

Figure 9. Evaluation Pipeline

The obtained results for both generative models can be seen in the below table. It appears that about half of the generated proteins were successfully classified as enzymes. This result seems promising and may be increase through further tuning. It also indicates the existence of positive externalities between the tasks of function prediction and function design in protein engineering. Indeed, passing generated samples into a classifier allowed to filter out candidates and would help further analyses to be performed on the most promising subsets.

	Ratio classified as enzymes by HGPSLPool
GAN	51 %
GAE+NN	45 %

4 Conclusions

In conclusion, our work focused on classifying enzymes and proteins using Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT), as well as generating protein graphs with Generative Adversarial Networks (GAN) and Graph Autoencoders (VGAE).

4.1 Enzyme classification

Our best results were achieved using the GAT model, outperforming the GCN model. This demonstrates the potential of GATs in capturing complex relationships in graph-structured data for enzyme classification tasks.

4.2 Protein classification

We achieved the highest accuracy using the **HPSLPool** model, showing its effectiveness in classifying protein sequences. Future work could explore additional graph pooling techniques to further improve performance.

4.3 Protein sequence generation

Using GAN and VGAE, we generated proteins graphs with the aim to approximate the enzymatic function. The generated samples were classified with the HGPSLPool model. The results showed that about half of the generated graphs were correctly classified as enzymes. This highlights the potential of generative models for protein design tasks.

4.4 Future work

Building on our findings, future research could investigate alternative GNN architectures and training techniques to further improve enzyme and protein classification. Additionally, exploring more advanced generative models and optimizing their hyperparameters could enhance the quality of generated protein sequences.
The generation of protein sequences using generative models like GANs and GAEs combined with NNs can have a significant impact on various applications in the field of bioinformatics, drug discovery, and synthetic biology. By generating novel protein sequences, researchers can explore previously uncharted areas of the protein sequence space, potentially leading to the discovery of new protein structures, functions, and interactions. For example, let's consider the application of drug discovery. One of the primary goals in this field is to identify and develop molecules that can effectively bind to specific target proteins, modulating their functions and thus treating a particular disease. By generating novel protein sequences, researchers can potentially identify new protein targets that may be involved in various diseases. Additionally, these generated sequences could serve as templates for designing new drugs that bind to these newly discovered targets with high specificity and potency. Furthermore, in synthetic biology, the generation of novel protein sequences can lead to the creation of custom-designed proteins with desired properties or functions. These proteins can then be used in various applications, such as industrial biotechnology for the production of biofuels, pharmaceuticals, or enzymes with improved catalytic properties. By generating and exploring new protein sequences, researchers can potentially engineer proteins that are more stable, efficient, or capable of performing novel functions.
In summary, the generation of protein sequences using generative models can help advance the fields of bioinformatics, drug discovery, and synthetic biology by providing a diverse set of novel protein sequences for researchers to investigate. These generated sequences can lead to the discovery of new protein targets, aid in the design of new drugs, and facilitate the engineering of custom proteins with desired properties, ultimately contributing to the development of innovative solutions for various challenges in these fields.

References

[1] Yunsheng Bai, Hao Ding, Yang Qiao, Agustin Marinovic, Ken Gu, Ting Chen, Yizhou Sun, and Wei Wang. 2019. Unsupervised inductive graph-level representation learning via graph-graph proximity. *arXiv preprint arXiv:1904.01098* (2019).
[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
[3] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.
[4] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[5] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[6] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *International conference on machine learning*. PMLR, 3734–3743.
[7] Andre Martins and Ramon Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*. PMLR, 1614–1623.
[8] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663* (2020).
[9] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017).
[10] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* 1050, 20 (2017), 10–48550.
[11] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
[12] Jiaxing Xu, Jinjie Ni, Sophi Shilpa Gururajapathy, and Yiping Ke. 2022. A Class-Aware Representation Refinement Framework for Graph Classification. *arXiv preprint arXiv:2209.00936* (2022).

[13] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems* 31 (2018).

[14] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[15] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954* (2019).