EAS 563: GNN for Chart Analysis

Karan Bali
Department of Computer Science
SUNY Buffalo
Buffalo, New York
kbali@buffalo.edu

Abstract

The task of this project was to gain acquaintance with Graphical Neural Networks(GNN) and its application in the domain of chart analysis. At the beginning of the project, the basic knowledge about the field of GNN and Graphical Machine Learning (GML) was attained using available literature. After that, an experiment was conducted to correctly classify "Tick-Labels" related to Task-3 of the [2] ICPR Chartinfo-2020 competition. The experiment was successful as the GNN was able to correctly classify the "Tick-Labels" with a relatively good F1 score.

1 Introduction

This project was part of a course in a university course setting. The project was a combination of independent study a capstone project, where one gets to learn about a newly emerging field and try out its application as a capstone project. Considering the recent successes ubiquitous significance, "Graphical Neural Networks" (GNN) was chosen as the topic of the project.

Graphs are one of the most important "Natural" data structures present around us in our daily lives. They are present in Natural forms like molecules, the nervous system of organisms, and the genetic evolution of species. They are also present in virtual forms like banking systems, Nodal transport systems, and most importantly World Wide Web(WWW).

Recent advances in GNN have given the field of Graphical machine learning a new momentum altogether. "GNN" is one of the most popular topics in many popular research conferences.

One of the objectives of our experiment was to apply a Graph convolution Network on charts dataset for the task of a Binary semantic classification of Tick-Labels.

As we were dealing with a new concept, "Literature review" formed a big part of the project. Hence, We'll go through the literature review needed to understand the experiment in the next part.

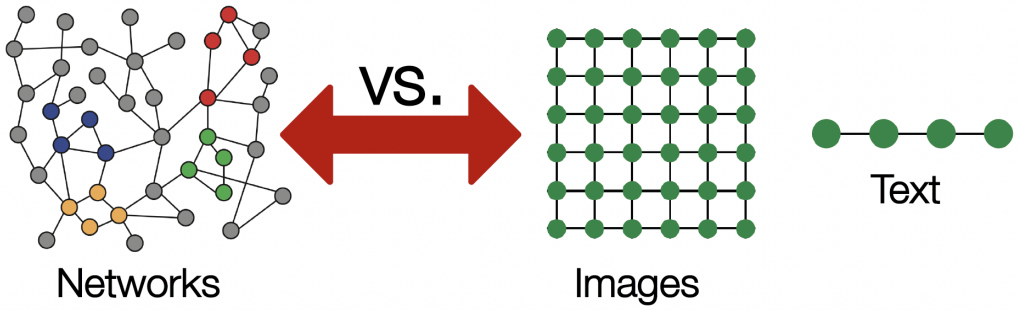
2 Literature Review

Firstly, we would like to cite thank Jure Leskovec for creating a wonderful introductory course on graphical machine learning. The [1] "Stanford CS224W: Machine Learning with Graphs" course was instrumental in getting adequate knowledge about the chosen topic. We've used and cited some of the images from course slides, which are open to the public for non-commercial purposes.

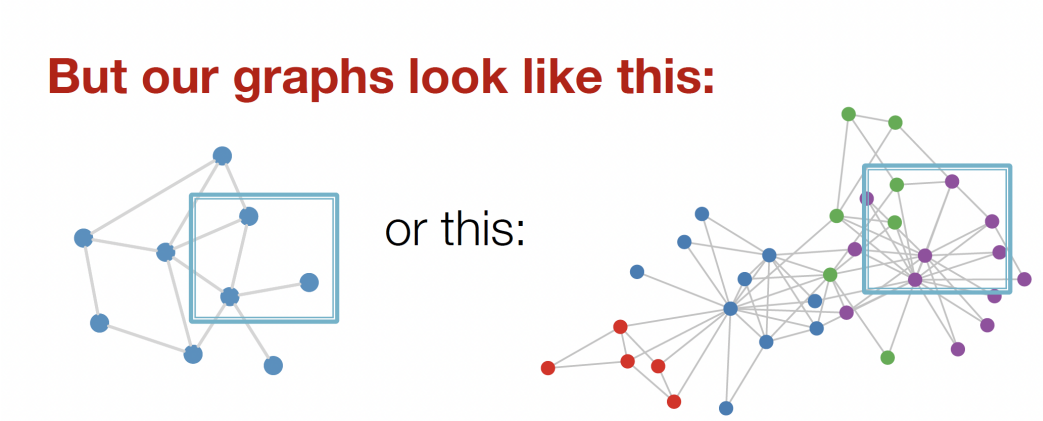
We assume that the reader has a basic knowledge of the main concepts of deep learning. Throughout the literature review, we'll build upon that the concepts of Graphical machine learning.

The efficacy of machine learning algorithms in dealing with conventional datasets like that of images speech is already well established. These algorithms assume the "topological structure" and

"ordering" of the given dataset. This assumption is not possible in a graphical dataset. For example, a pixel in an image can have definite upper, lower, left right neighbors but a graph node doesn't have any definite convention to identify such horizontal vertical neighbors. Another problem with a graphical dataset is its dynamic nature. Unlike an image, a graph can keep growing over time.



Ref: [1]



Ref: [1]

Hence, we need a new mechanism to apply the normal toolkit of machine learning algorithms in a graphical construct. This whole domain refers to Graphical Neural Networks(GNN) Graphical Machine Learning(GML). The conventional ML algorithms either depend upon specially hand-crafted features or learn those features on their own using neural networks. GNN derives and adopts many techniques from Graph theory network embeddings. To apply conventional ML algorithms to a graphical dataset, we need to somehow get a vector representation of that graphical dataset. Thus, we try to transform graphical data into an embedding space. We can use that representation vector for further downstream tasks which are unique to a graphical dataset. Ideally, this embedding space should reflect the ground truth, where similar nodes are closer to each other.

The convention of defining a graph is based on defining the nodes of a graph with a unique ID and a feature vector for that node. A graph also consists of an adjacency matrix that defines the edges/connections between those nodes. A graph can also contain features for each edge.

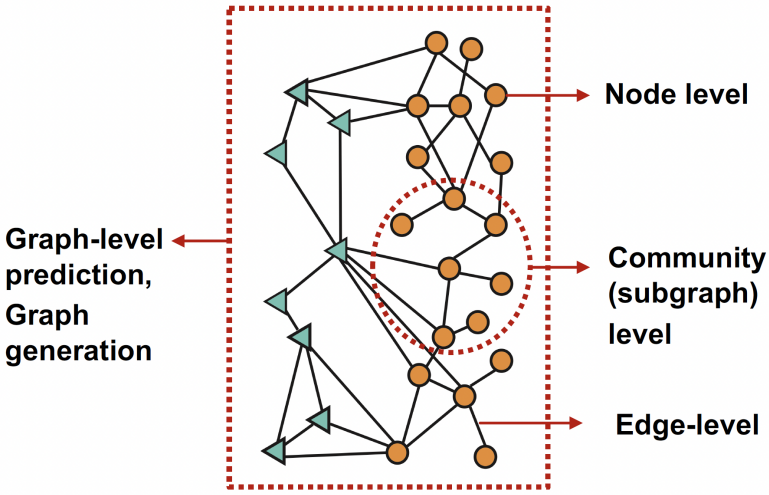
Given: $G = (V, E)$

Here Graph(G) is defined by the set of Nodes(V) and edges(E). Our aim is to:

$$\text{Learn a function: } f : V \rightarrow R$$

Where a function(f) maps from nodes to a real value(R).

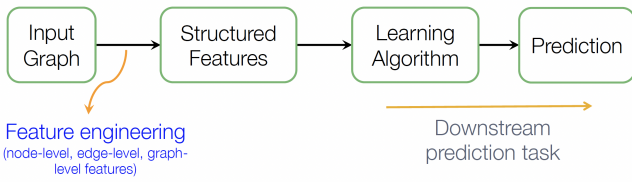
Coming to downstream tasks, we can have mainly 3 types of tasks specific to a graphical dataset. The 3 tasks are Node-level predictions, Edge-level predictions, and Graph-level (sub-Graph-level) predictions.



Ref: [1]

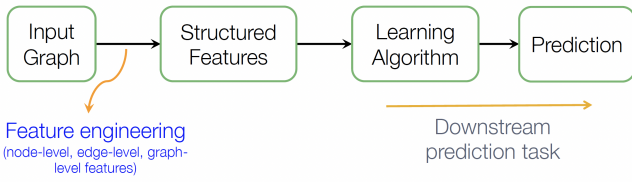
Graph representation learning has evolved similarly to conventional ML algorithms. The world of Artificial intelligence has seen traditional hand-crafted features getting replaced in Neural Networks, which learn these features on their own. Similarly, earlier graph learning methods used specially designed hand-crafted features. Later on, this step was replaced by GNN.

Traditional ML pipeline:



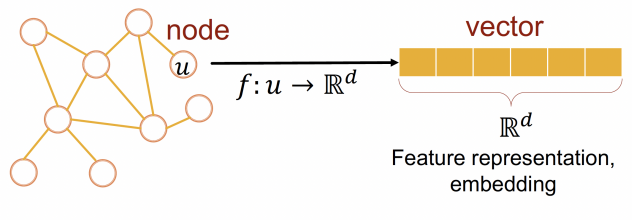
Ref: [1]

Deeplearning pipeline:



Ref: [1]

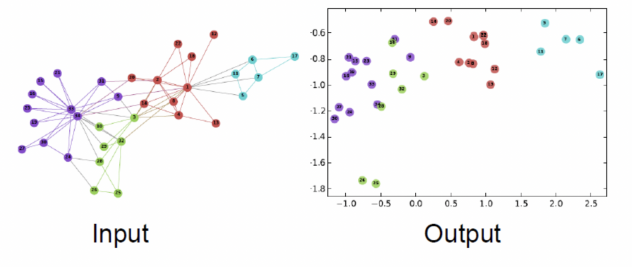
To classify nodes of a graph, we obtain a high-level feature representation of the node and use any conventional ML algorithms for further prediction tasks. To calculate node features, we can use various importance-based features (i.e. signifying the importance of a node in a graph) or structure-based features (i.e. signifying structure in the local neighborhood of a node in a graph).



Ref: [1]

However, when the graph grows larger, many of the above methods fail. Thus, the Random walk strategy is a good alternative in these scenarios. In Random walk, we get a node embedding using Negative sampling random K-hop walk for a particular node. Then we can formulate a simple log-likelihood objective function that aims to increase the similarity score of similar nodes.

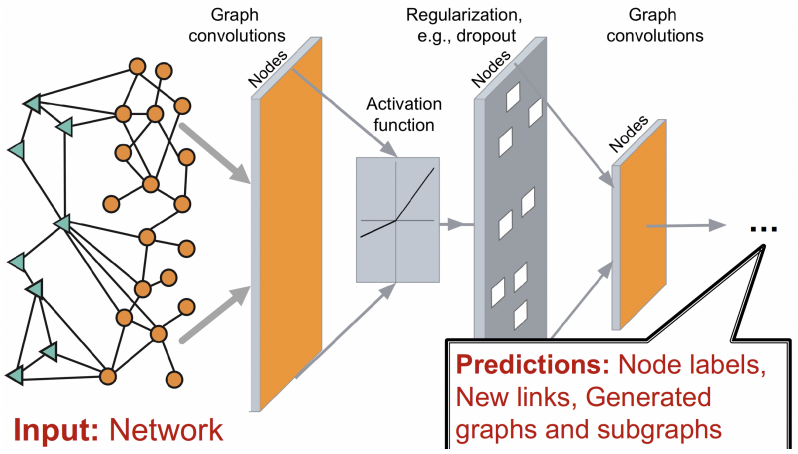
Node2Vec is a slight modification of Random walk. Instead of doing a simple random walk, Node2Vec opts for either a Breadth-first walk or a Depth-first walk with some probability. A Breadth-first step helps to capture the local information of a node a Depth-first step helps to capture the global information of a node.



Ref: [1]

To predict links of a graph, we get a high-level feature representation of the link, thus, a pair of nodes. To calculate link features, we can use various distance-based features (i.e. features that signify the shortest distance between nodes), local neighborhood features (i.e. signifying similarity in the local neighborhood of 2 nodes), or global neighborhood features (i.e. signifying global structure of the graph concerning 2 nodes)

To predict a whole graph, we get a high-level feature representation of the graph in itself, thus. To calculate graph features, we can use the Graphlet kernel method. Similar to the bag-of-words model in NLP, we obtain a representation of a graph based on the number of graphlets present in a graph. As counting graphlets is NP-hard, we can also use the Weisfeiler-Lehman kernel to design a computationally efficient feature descriptor for a graph. Weisfeiler-Lehman kernel method uses the color refinement method to capture the graph structure.



Ref: [1]

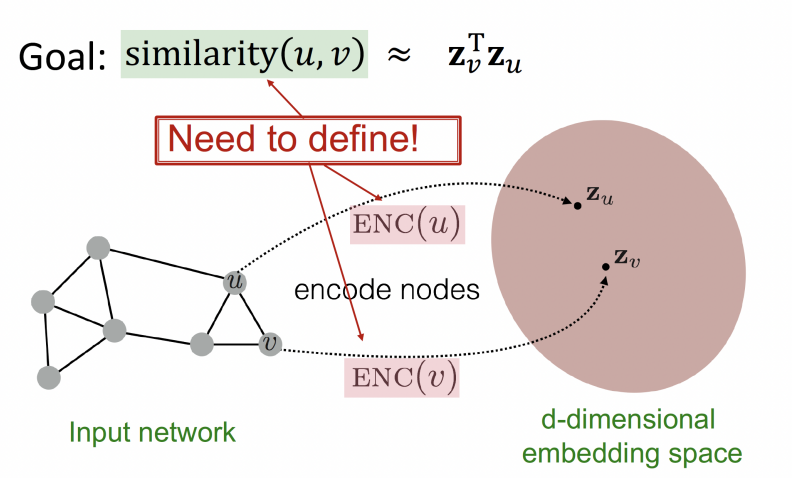
To graph embeddings, we can use three approaches. The first approach is to simply use the graph embedding method on subgraphs and then get an average value for the whole graph. The second ap-

proach is similar to the first approach, but we add a virtual node on every sub-graph before applying a graph embedding method. The third method uses sampled anonymous walk embedding and then concatenates those embeddings to get final graph embedding.

As mentioned above, we can use specially hand-crafted features specific to a task as feature representation for the node, link, or graph. We can either use these features directly or in an Encoder-Decoder setting.

2.1 Encoder-Decoder

An Encoder-Decoder has 2 components. The encoder takes the feature representation and maps it to an embedding space. The decoder takes that embedding representation and maps it to a similarity score. Now the whole task of the Encoder-Decoder setup is to learn perfect weights that reflect the ground truth (i.e. similarity score between similar nodes should be high and vice versa).



Ref: [1]

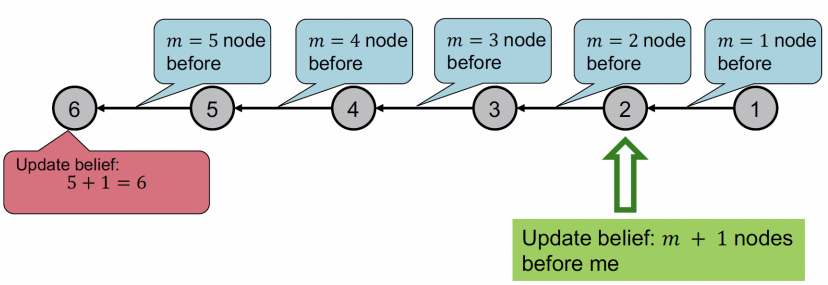
2.2 Message Passing Algorithm

Before we try to understand the Message passing Algorithm (MP), we'll go over some other topics needed to understand MP.

In our whole approach, there is an assumption about the correlation in the data in general. This correlation is caused by 2 factors, Homophily, and Influence. Homophily refers to the tendency of like data points being together. Influence refers to the tendency of a data point to get affected by its connections or neighborhood.

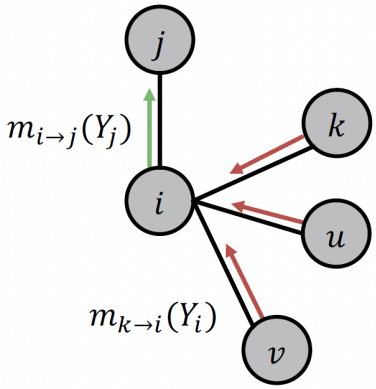
Based on this information we can define three types of classifiers, Local classifier, Relational classifier, and Collective inference. The Local classifier is the simplest version that uses just the features of the node itself to classify. Thus, it makes no use of network information. The Relational classifier uses node attributes of the node itself and its neighbors. Thus, it makes some use of correlation caused by homophily influence within the nodes of a graph. The Collective inference applies a relational classifier to each node iteratively till the inconsistency between the network settles down. Thus, it not only uses correlation but also propagates it through the network.

The Collective inference, also known as Loopy belief propagation, forms the basis of the MP algorithm. In the MP algorithm, each node can pass a message/belief to its next neighbor. Thus, each node has a prior belief before receiving a message. This prior belief is updated based on the summation of incoming passed messages. The messages from all neighbors are weighted by a label-label potential matrix that captures the relative importance of all neighbors of a node. Finally, an updated message is sent to the outbound neighbors, where the whole process is repeated. In the end, we get a graph with an updated belief for all its nodes. This is a small summary of the MP algorithm. MP algorithm is parallelizable but its convergence is not guaranteed in the case of cyclic graphs.



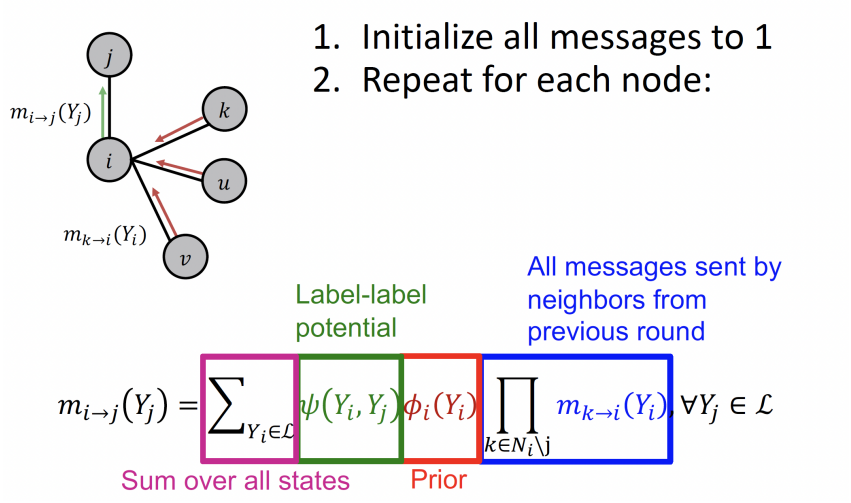
Ref: [1]

As mentioned earlier, Graphical datasets are different from conventional datasets. There is no sense of orientation in a definite way. This means that a graph is permutation invariant. Thus a normal Convolution neural network that works on the assumption of local information and ordering is useless on a graph. To use Convolution in a graphical setting, we need to get some measure local neighborhood in a graphical dataset. This local neighborhood is defined by K-hop neighbors. Where "K" is a hyperparameter defining the shortest path from the node to the K-hop neighbor.



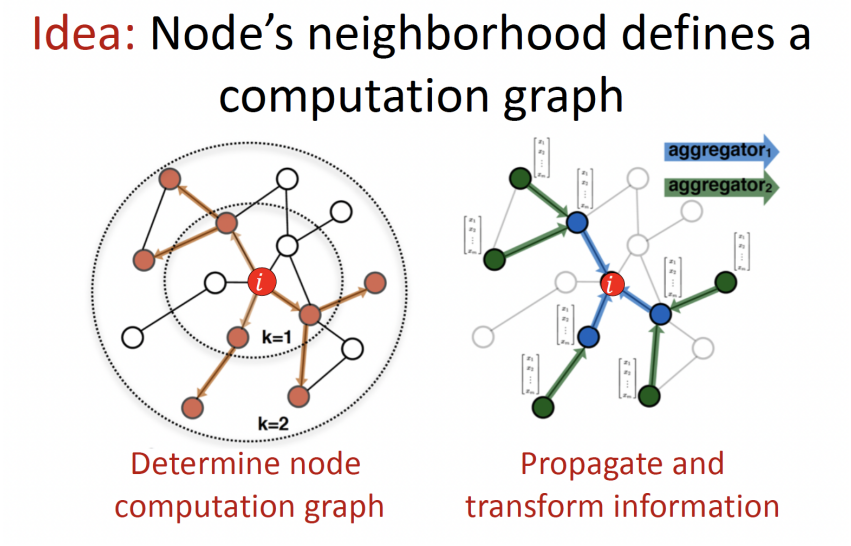
Ref: [1]

Getting the idea from the MP algorithm, we can get node embedding for K-hop local neighborhood. Thus We aggregate the information from K-hop neighbors after passing their features through a neural network. After aggregating, we can pass this belief to K-1 hop neighbors. We can repeat this process till we are left with a final node embedding. To visualize this operation, we can think of Layer 0 consisting of raw features of K-hop neighbors of a node. Layer 1 aggregate these features and passes them through a neural network to update the beliefs of K-1 neighbors. Layer 2 again aggregate and passes the updated beliefs of K-1 neighbors through another neural network to update the beliefs of K-2 neighbors. This process is repeated till we are left with node embeddings. Each neural network in each subsequent layer has got its own set of weights and biases to train. The final node embedding can be compared with the ground truth to get a final loss for the prediction. This loss is used to calculate the gradient that needs to be propagated back through all layers.



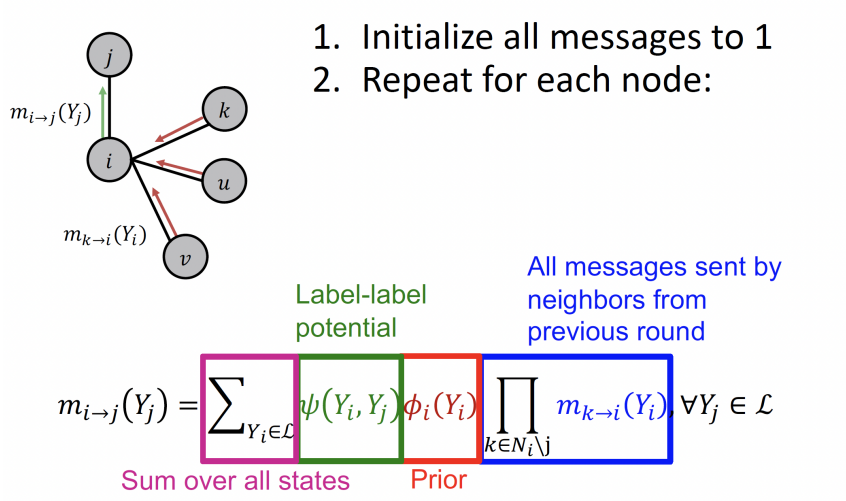
Ref: [1]

During backpropagation, all weights biases of neural networks get updated. This form a simple GNN or Graphical Convolution Network (GConv).



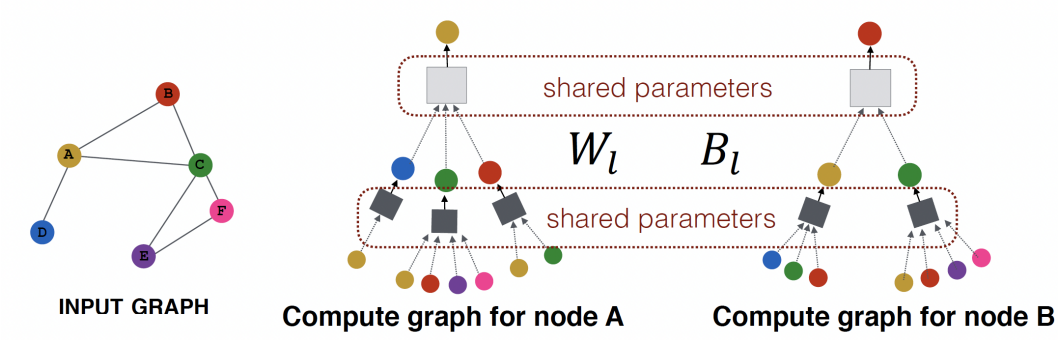
Ref: [1]

Thus, each node in a graph automatically generates a computation graph based on its neighborhood.

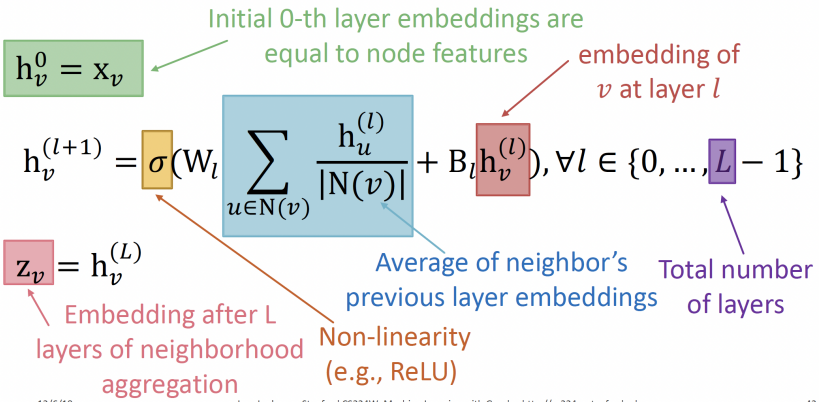


Ref: [1]

The neural network of each layer process particular distant neighbors. Thus, they are inductive in nature. weight and biases of a particular layer are shared by all nodes. They can generalize to a new graph or node. We can train GNN on one set of graphs and test the same trained model on another set of graphs. However, the result from both the test set will depend upon the similarity of the training testing dataset.



Ref: [1]



Ref: [1]

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(l+1)} = \sigma\left(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$z_v = h_v^{(L)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

h_v^l: the hidden representation of node v at layer l

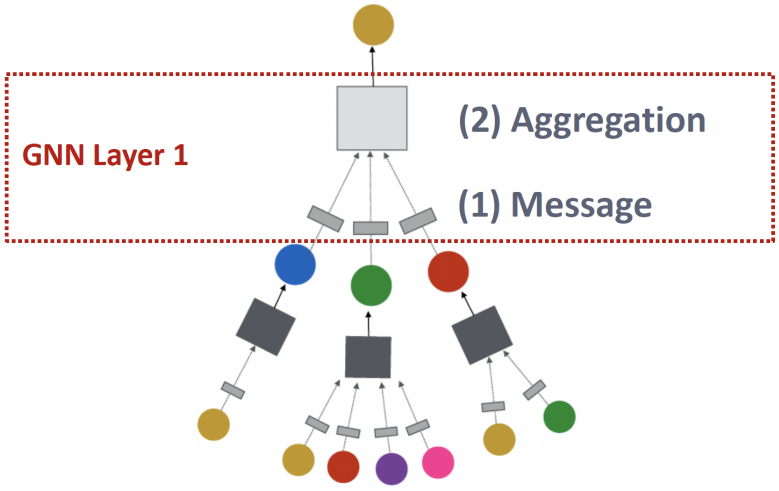
- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

Ref: [1]

As mentioned above, a GNN layer consists of two operations, message formation, and aggregation. We can combine both of these two operations into a single operation. The neural network will have its trainable weights and biases.

GNN Layer = Message + Aggregation

- **Different instantiations under this perspective**
- **GCN, GraphSAGE, GAT, …**



Ref: [1]

The Aggregation function in a GCN is a simple summation. However, we can have different types of aggregation functions like mean, pooling, max. We can also pass messages through another MLP or LSTM to get an aggregated message. This variation of GCN is called the GraphSAGE algorithm.

- **GraphSAGE:**

Concatenate neighbor embedding and self embedding

$$\mathrm{h}_v^{(l+1)} = \sigma([\mathrm{W}_l \cdot \mathrm{AGG}\left(\left\{\mathrm{h}_u^{(l)}, \forall u \in N(v)\right\}\right), \mathrm{B}_l \mathrm{h}_v^{(l)}])$$

Flexible aggregation function instead of mean

Ref: [1]

Another variant of GCN uses an attention mechanism to weigh messages from different neighbors based on their importance. This variation is called a Graph Attention Network (GAT).

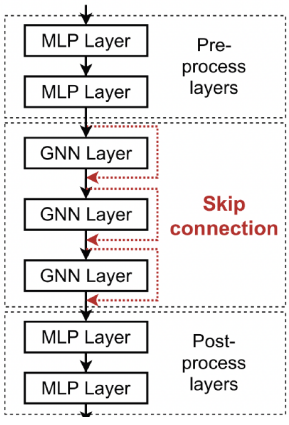
Graph Attention Networks

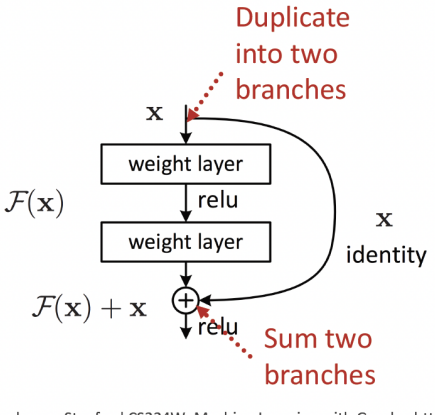
$$\mathbf{h}_v^{(l)} = \sigma(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)})$$

Attention weights

Ref: [1]

Similar to Attention in GAT, We can apply many of the other deep learning optimization strategies like Batch Normalization Dropout in GNN. Many aspects like non-linear activation functions of normal deep learning algorithms are shared by GNN. We can also add a Residual skip connection between GNN layers. This helps in the flow of gradients between different layers.

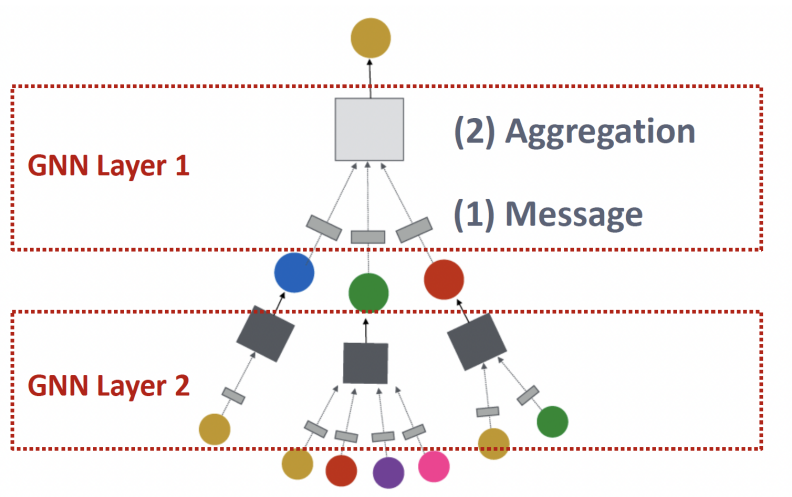




Ref: [1]

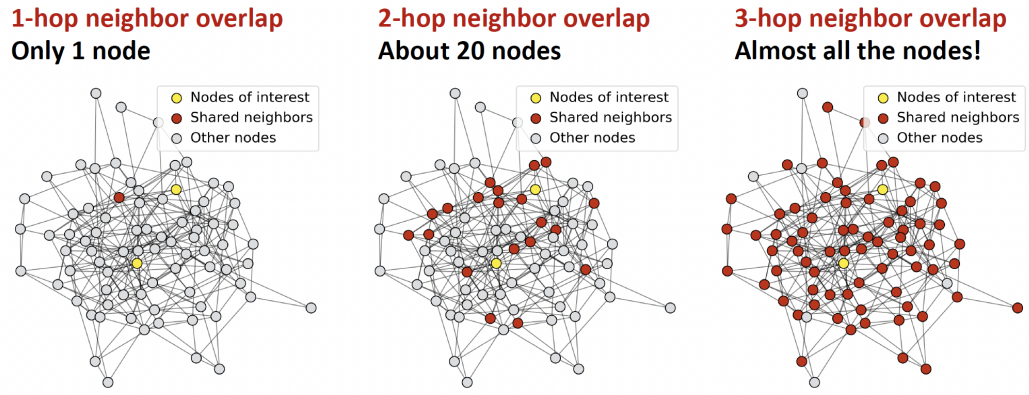
We have already explained that a GNN is formed by stacking several GNN layers, which performs message formation aggregation functions. More number of layers brings more expressive power to a model.

Stacking 2 GNN layers to form a GNN model:



Ref: [1]

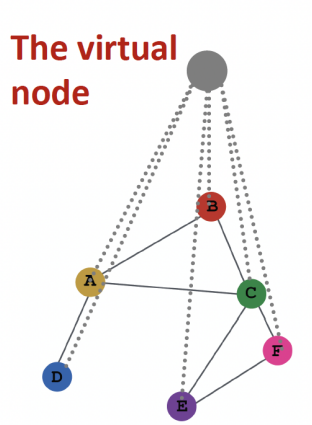
However, there's a downside too. Stacking too many GNN layers causes "Over-smoothing". The problem of over smoothing is related to the receptive field. The receptive field of a node in a GNN is the set of nodes that are used to calculate the final node embeddings of a graph. As we increase the number of layers, the receptive field grows exponentially. Thus, many nodes will have overlapping receptive fields. This means many nodes will have similar node embeddings. This goes against our objective of having node embeddings to differentiate between nodes. Thus, we usually stack a limited amount of layers so that a balance between expressive power receptive field could be maintained.



Ref: [1]

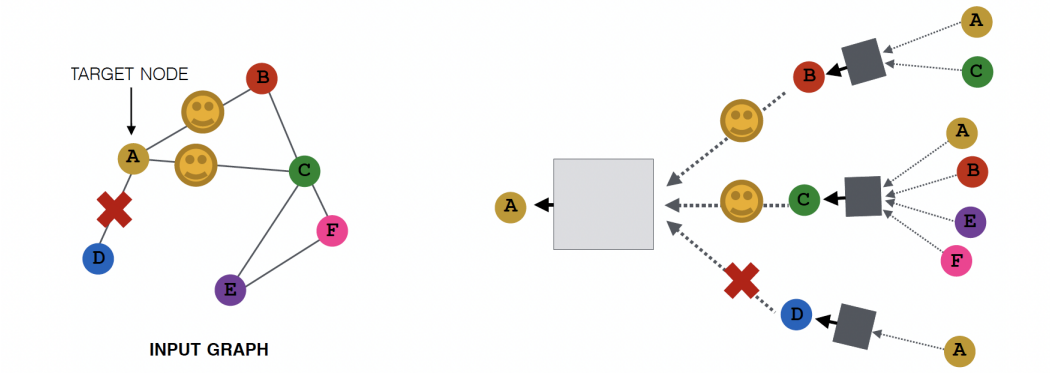
Many times a plain graphical dataset can't be used in a GNN due to some specific issue. For example, a large graph like a community network of Facebook is too big for any computational capacity. We can also have a dense or a sparse graph, where message passing is too costly or inefficient. Thus, in these situations, we need some graph manipulation techniques to bail us out of the situation. For this, we can use either graph feature manipulation or graph structure manipulation. To handle a dense graph, we sample nodes and edges from all possible options. Thus, we don't need to traverse over all the neighbors. Similarly, if a graph is too large, we can sample subgraphs to compute embeddings from that large graph. Conversely, if a large is too sparse, we can add virtual edges nodes to make the graph denser. We can also use feature augmentation in the last scenario.

Adding Virtual node to a sparse graph:



Ref: [1]

Samlping from a large graph:

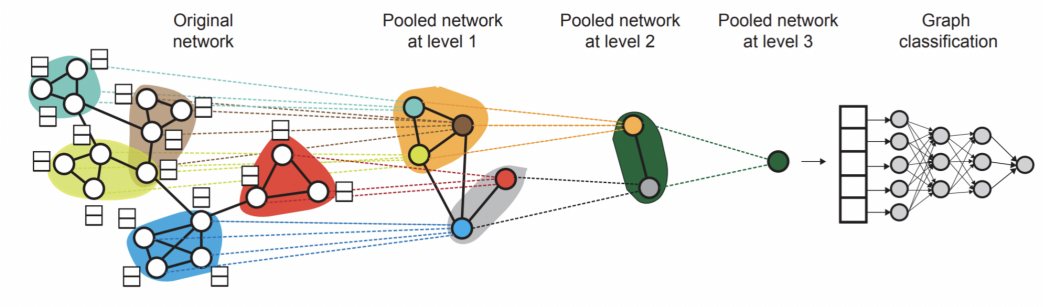


Ref: [1]

The last part of GNN consist of prediction heads, loss function evaluation metrics. All these choices are task-specific. The prediction head for a node classification task is similar to any classification task in a normal neural network. The task is to take a final 'd-dimensional' embedding and predict a class for each node using a softmax layer. The prediction head for an edge classification task takes 2 node embeddings of the edge-defining nodes and transforms them into a single embedding. We can achieve this by either concatenation of 2 node embeddings or some alternative operation like dot product. We can also apply Multi-head attention with edge-defining node embeddings to classify the edge into multiple classes. The prediction head for a graph is very similar to the aggregation layer in a GNN layer, where we combine all node embeddings of a graph into a single node embedding. This final node embedding can be used to classify the graph.

One alternative way to predict large graphs is to use "Hierarchical Pooling". In hierarchical pooling, we use two GNN, GNN A GNN B. GNN A is used to compute node embeddings and GNN B is used to cluster nodes. After each round, all node embeddings belonging to the same cluster get aggregated into one single embedding. The whole process is repeated until we are left with one final embedding for the graph.

Hierarchical pooling

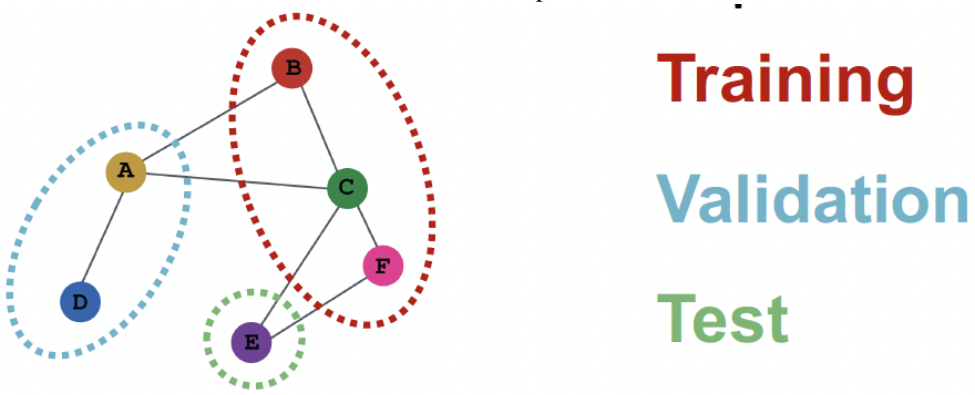


Ref: [1]

The loss functions used in GNN are similar to conventional neural networks depend on the type of task. Thus, Cross-Entropy loss for classification Mean Squared loss for regression are most common. For evaluation also, we use normal task-specific metrics. For example, Accuracy, ROC AUC, RMSE, F1 score, etc.

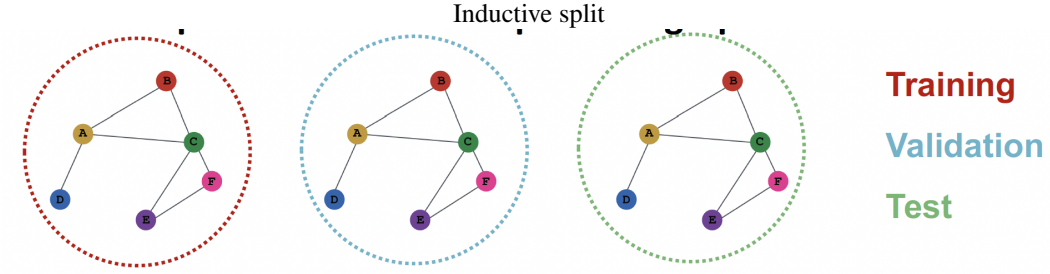
In the last, We'll go through the training process for the GNN. The first task of training is to split the dataset into a training, validation, and testing dataset. However, splitting the graphical dataset is not as simple as the image dataset. In an image dataset, each image in itself is an independent data point. However, in a graph, each data point (i.e. a node) is related to other data points (i.e. other nodes). This peculiar situation requires a different approach. Thus, we can have two types of approaches to split a graphical dataset, Transductive Inductive. Each dataset split masks a set of nodes, edges depending upon particular tasks. In a Transductive split, the input graph is common to all dataset splits. In an Inductive split, the input graph is partitioned into different splits, each split can access the nodes within a split.

Transductive split



Ref: [1]

Inductive split



Ref: [1]

Training for node prediction and graph prediction is simpler than training for a link prediction task. For link prediction, we mask edges progressively disclose them during training, evaluation, or testing stages. There are also other methods to split. In short, even splitting of datasets is task-specific.

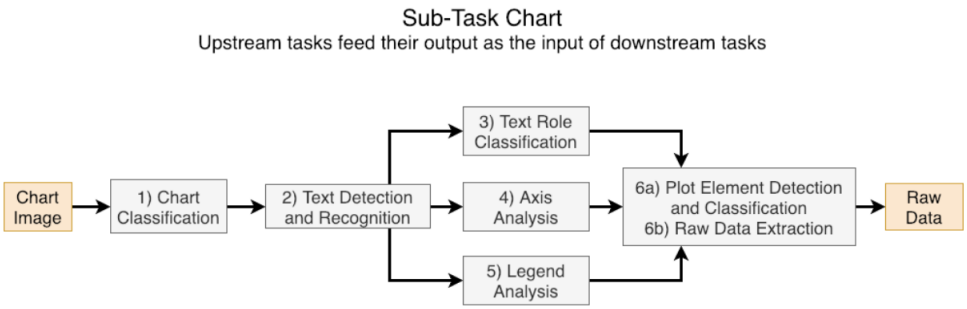
Figure 1: Ref: [1]

3 Experiment details

After going through all of the literature review, We'll explain the modalities related to our experiment. It has been observed that GNN is well suited for Geometric pattern learning. With this approach, we thought of applying GNN on the Chart analysis dataset related to the ICPR ChartInfographics-2020 competition.

3.1 ICRP ChartInfographics-2020 competition

[2] This competition was composed of a series of 6 sub-tasks for chart data extraction, which when put together as a pipeline goes from an input chart image to a CSV file representing the data used to create the chart. Entrants in the competition chose to participate in any number of sub-tasks, which were evaluated in isolation, such that solving previous sub-tasks is not necessary. Additionally, methods that perform the whole chart data extraction pipeline from the chart image without receiving intermediate inputs were also evaluated.[2]



Ref: [2]

3.2 Problem Objective

We chose Task-3 of the ICPR ChartInfo-2020 competition to test the practical applications of the efficacy of GNN in learning a geometrical pattern. For the problem, we try to find an alternative method of achieving the task and compare it with that of the top scorers of the competition.

[2] For text to be useful in chart interpretation, its semantic role should be identified. This sub-task focuses on identifying the role of each text block in a chart image, and text bounding boxes and transcripts are provided as input. Competing systems are expected to classify each bounding box into one of the following roles: 1)Chart Title 2)Axis Title 3)Tick Label 4)Tick Grouping 5)Legend Title 6)Legend Label 7)Value Label 8)Data-Marker-Label 9)Other. Task-3 dealt with 9 classes. However, only the Tick-Labels of the data points follow a geometric pattern to some extent. All other classes are more random in nature. Especially the classes belonging to the plot area are totally random as they depend upon the data points related to the graph. Hence, we decided to apply a GNN for only Tick-labels. Thus, the problem became a binary classification problem.

Thus, to train a GNN on Tick-labels we needed to pre-process the original data into a more graphical structure that could be used to 1) define the graphical structure between different nodes connected by different edges, and 2) convert the original task into a binary classification problem. The bounding boxes parameters detected in task-2 were used to define the node and its features. The edges between different nodes were defined using "Delaunay Triangulation". But before that, we needed to choose a data set.

3.3 Dataset

There were two datasets for the Chartinfo-2020 competition. [2] "Adobe Synth" dataset contained synthetic dataset constructed by chart libraries like Matplotlib, etc. [2] "UB PMC" dataset contained real scanned chart images and annotations. For our experiment, we chose the [2] "UB PMC dataset".

3.4 Data preprocessing

The original dataset contained images and JSON annotations in a specific format. The sample data is available at the competition's official site. We had to transform the JSON annotations into an appropriate CSV format that could be used to form a graphical dataset.

3.5 Deep Graph Library

We use DGL to formulate a graphical dataset and data structure and a custom data loader. A DGL is a framework-agnostic library that consists of built-in API functions specially used for training GNN's. There are alternatives to DGL like Pytorch Geometric, spektral. Each one of the alternatives has its advantages.

3.6 Delaunay Triangulation/Tessalation

To define a graphical data structure, we need to define the edges between the detected nodes from Task-2. For this, we used the concept of Delaunay Triangulation to connect 2 nearest nodes to a node with an edge, thus forming a triangle. We performed Delaunay triangulation for all points. The final edges and nodes defined the graph data structure.

3.7 Experiment Hypothesis

As mentioned above, there were a total of 6 tasks that could be completed and evaluated in isolation. To test the application of GNN on a charts dataset, We chose a subtask of Task-3. For Task-3, we had to do a semantic classification of all text-based bounding boxes detected/given as ground truth for Task-2. Our experiment was to run a GNN on the given dataset to correctly classify "Tick Labels" in task-3.

We intentionally chose to classify "Tick-Labels" as most of the time they follow an "L-shaped" geometric pattern. We believed that a simple GCN model should be able to learn this geometric pattern.

The results we got were good enough. The results clearly showed that GCN was correctly able to learn the pattern for the "Tick-Labels".

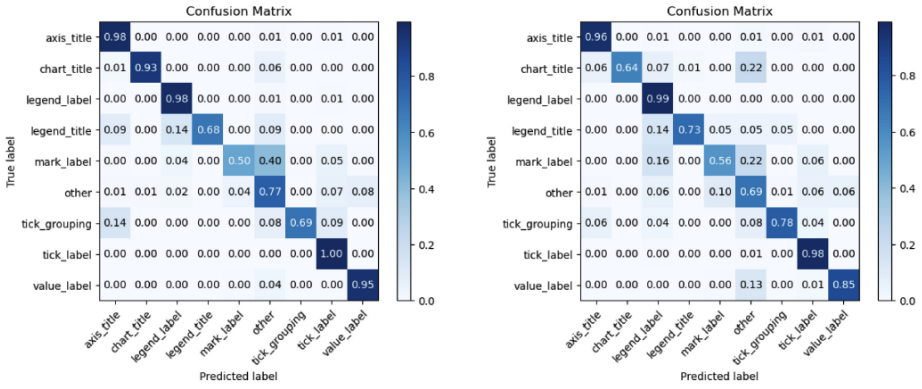
4 Results

Class	Overall	horizontal bar	line	scatter	vertical bar	vertical box
Average F1 Score:	0.92	0.83	0.945	0.95	0.91	0.898

5 Evaluation

We compared our results with the top scorer of the ICPR competition. As we completed only a sub-task of task-3, we compared the class-specific score of the top scorers. Ideally, we would have

compared it with other competitors' Tick-label scores also but their data wasn't available. The competition used "Averaged" F1-score to evaluate task-3. However, The confusion matrix for the 2 top-scorers was available.



(a) Lenovo-SCUT-Intsig (b) Magic

Fig. 3. Confusion matrices for Task 3 for the top-2 participants.

Ref: [3]

We can observe the performance of our GNN for classifying "Tick-labels" lower in comparison to the methods used by the top 2 scorers. But, The overall score of all the competitors for task-3 was also available. Hence, if we compare the results of other competitors for the whole of task-3, the performance of our GNN was comparable. We know that this is not a correct comparison, but we don't have the access to class-specific scores of other competitors.

Table 5. Task 3 results: Average F-measure across all predicted text role classes.

Team	Adobe Synth	UB PMC
Lenovo-SCUT-Intsig	**1.00**	**0.859**
Magic	**0.999**	0.817
DeepBlueAI	**0.999**	0.772
Py	–	0.654

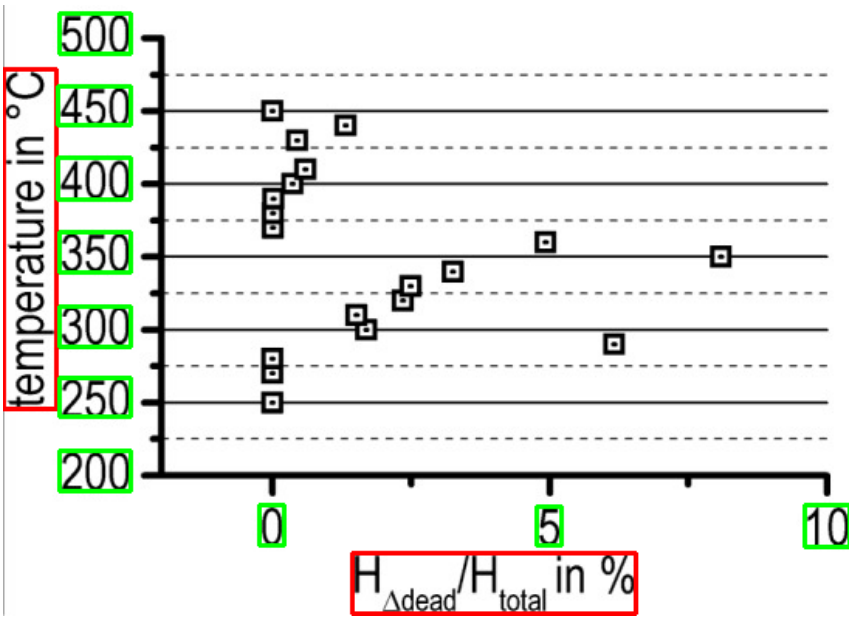
Ref: [3]

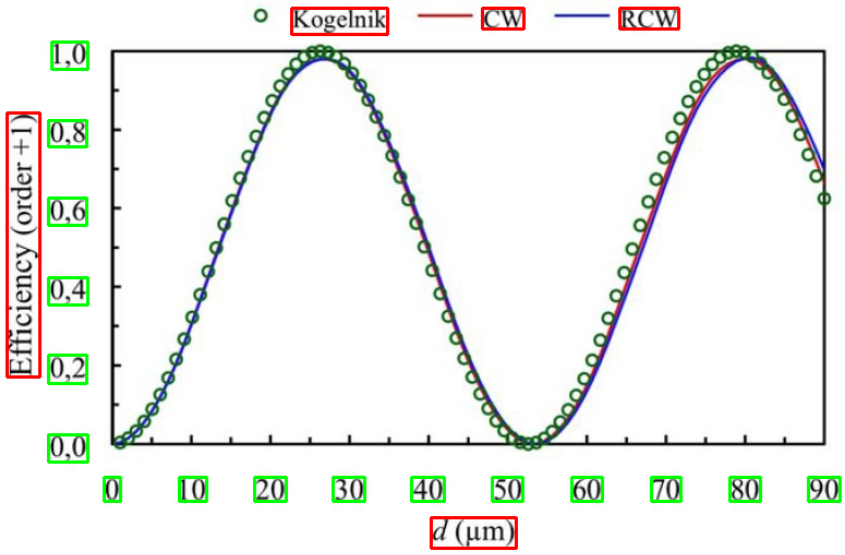
6 Output samples

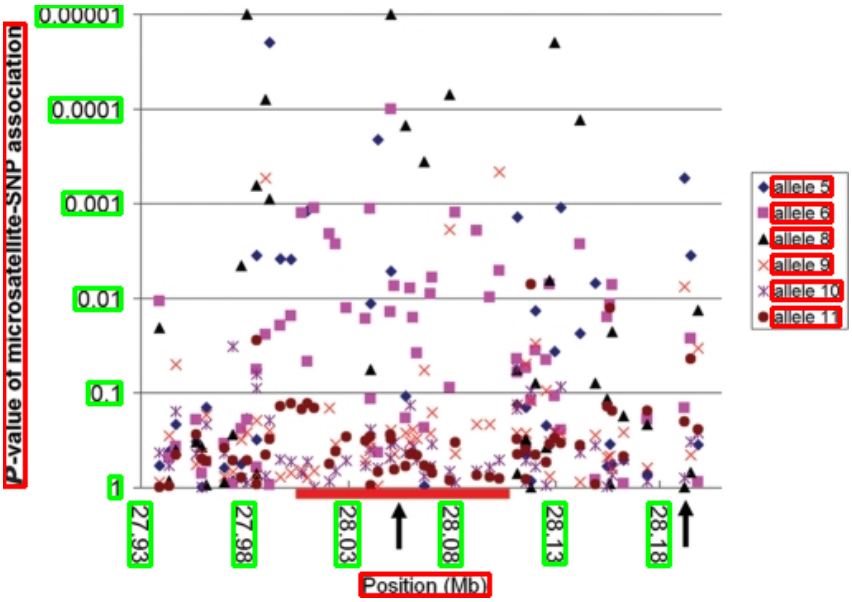
Color labels for True Positive, True Negative, False Positive, False Negative classes:

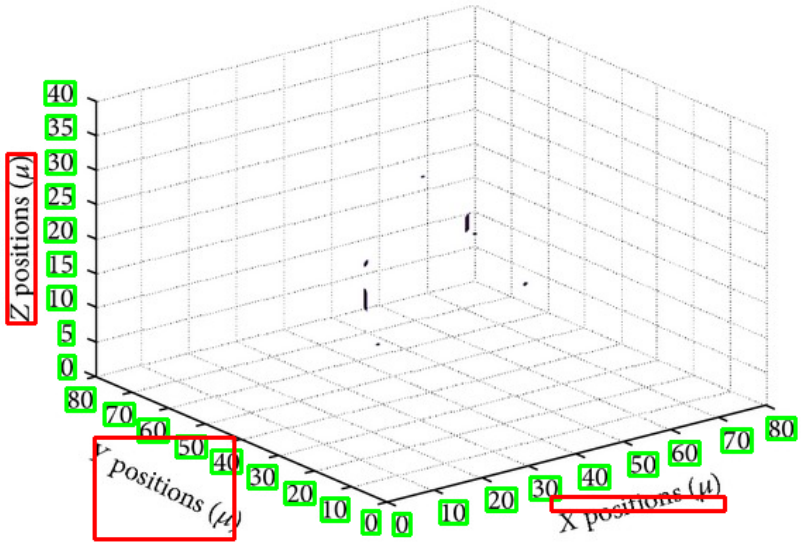
1) True Positive: Green
2) True Negative: Red
3) False Positive: Green beige
4) False Negative: Grey

6.1 Good samples

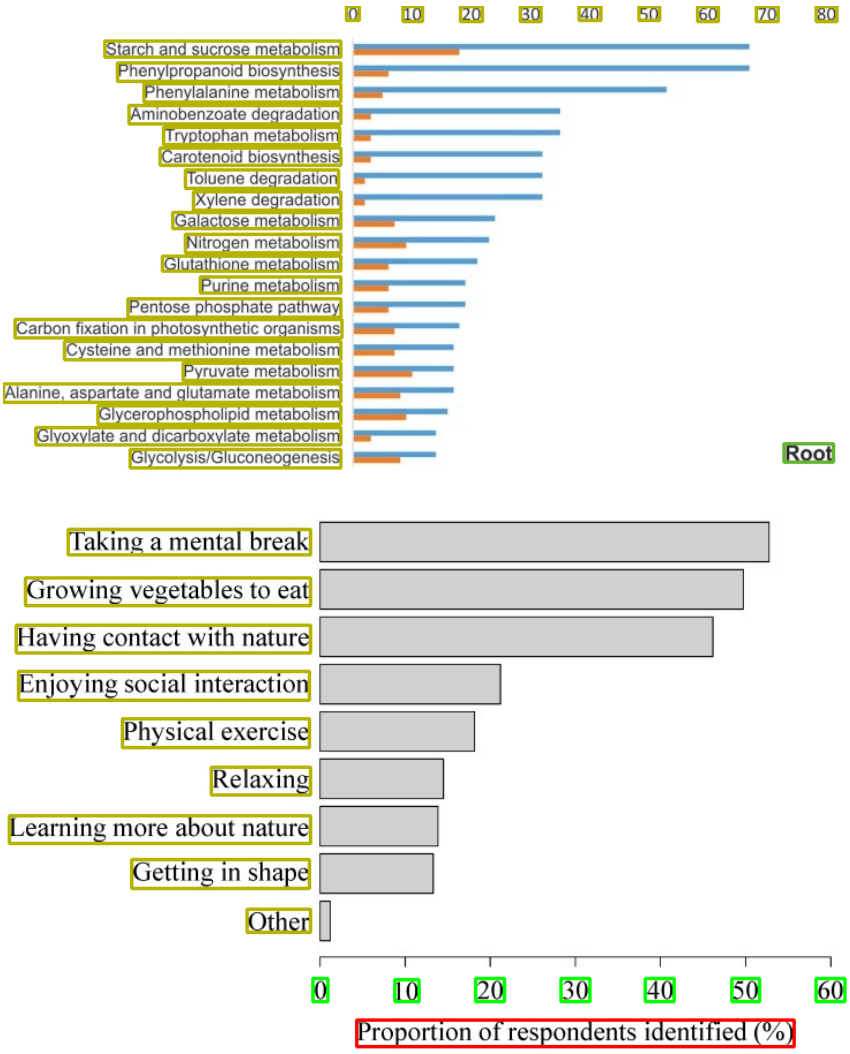




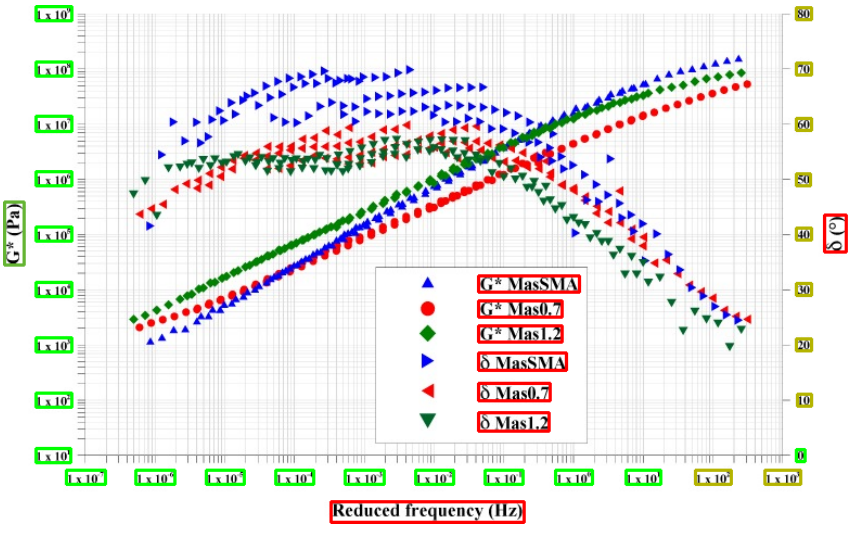

P-value of microsatellite-SNP association
0.00001
0.0001
0.001
0.01
0.1
1
27.93
27.98
28.03
28.08
28.13
28.18
Position (Mb)
allele 5
allele 6
allele 8
allele 9
allele 10
allele 11


Z positions (μ)
40
35
30
25
20
15
10
5
0
80
70
60
50
40
30
20
10
0
Y positions (μ)
0
10
20
30
40
50
60
70
80
X positions (μ)

6.2 Bad samples



6.3 Mixed results



7 Conclusion

The main objective of the whole project was to learn more about graphical machine learning and execute some practical applications. With that in mind, literature review formed an important part of the project. A considerable time was spent learning about the different aspects of GNN and its variants. The remaining time was spent on executing the GCN experiment on the ICPR ChartInfographics-2020 competition. Both of these core objectives were met. GNN was able to learn the geometric patterns of tick-labels with a relatively good accuracy. The evaluation F1-score "Horizontal Bar class" was slightly below overall F1-score. The "Scatter class" had best F1-score. The plotted evaluation samples provided a more deeper intuition of correct and incorrect classification.

8 Future work

However, the execution of Task-3 as a whole is still yet to be completed. But for executing the whole of Task-3, we can't depend upon just plain GNN. We'll need a more complex computer vision toolkit to do that. Hence, the last part of the paper mentions some of the future works that could be taken to complete task-3.

References

[1] Jure Leskovec, Stanford CS224W: Machine Learning with Graphs, Link: http://web.stanford.edu/class/cs224w/

[2] ICRP ChartInfographics 2020 competition, Link: https://chartinfo.github.io/

[3] Davila K., Tensmeyer C., Shekhar S., Singh H., Setlur S., Govindaraju V. (2021) ICPR 2020 - Competition on Harvesting Raw Tables from Infographics. In: Del Bimbo A. et al. (eds) Pattern Recognition. ICPR International Workshops and Challenges. ICPR 2021. Lecture Notes in Computer Science, vol 12668. Springer, Cham. https://doi.org/10.1007/978-3-030-68793-9_27