CONTENTS

List of Figures

List of Tables

List of Output Screens

Symbols and Abbreviations

DFD	Data Flow Diagrams
UML	Unified Modelling Language
RAM	Random Access Memory
SRS	Software Requirements Specification
GPL	General Public License
WWW	World Wide Web
OOP	Object Oriented Programming
GUI	Graphical User Interface
AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Networks
2D	Two Dimensional
3D	Three Dimensional
UI	User Interface
I/O	Input-Output
GPU	Graphics Processing Unit
API	Application Programming Interface
.Py	Python File Extension
URL	Uniform Resource Locator
SVM	Support Vector Machine
CPU	Central Processing Unit
OS	Operating System
GB	Gigabyte
DL	Deep Learning
CNN	Convolution Neural Networks
CART	Classification and Regression Tree
IG	Information Gain
GNN	Graph Neural Network

Abstract

The computer software industry is perhaps one of the fastest growing industries in the world today. With the increasing cost of software development, it is apparent that effective techniques for estimating software costs are essential in order to control the costs and make software more competitive. Because of the highly dynamic nature of the Software development, it becomes more and more difficult to get a correct software effort estimation. A number of algorithmic techniques have been developed and used for software cost estimation. However, calculations from several of such techniques indicate that estimates of development effort and cost differ considerably and hence may pose problems to software managers in committing resources and controlling costs. Software cost estimation is the process of predicting the effort required to develop a software system. Many estimation models have been proposed over the last 30 years. These estimation models are implemented in various frameworks for accurately predicting software effort.

In this project we implement a similar framework using the concept of neural networks, to be precise we use the Graphical Neural Network concept. We implement the algorithm on the ISBSG dataset to predict the software effort.

The predicted accuracy values are compared with various algorithms already at use in the market. The project aims to achieve a high accuracy value by implementing GNN.

Various software metrics are used to predict the overall software effort and for further development of the project.

1. Introduction

1.1 Motivation

Estimation of effort and duration of software development has become a topic of growing importance. This is not surprising. It often happens that software is more expensive than estimated and completion is later than planned. Moreover, it turns out that much software does not meet the demands of the customer. There are a number of examples of such automation projects.

The development costs of the automation of the education funding in The Netherlands proved to be three times as much as expected. Delays and wrong payments are a daily occurrence (Volkskrant, 24 June 1987). The development of the software for the purpose of the house-rent subsidies, produced to government order, proved to be twice as much as planned (NRC Handelsblad, 28 February 1989). In September 1989 the Dutch media announced as front-page news the results of a governmental audit concerning the automation for the police. It proved to be an expensive disaster. The development costs of a computerized identifying system were US$43 million instead of the estimated US$21 million. Furthermore, the system did not answer the formulated goals. The findings of a well-known Dutch consultancy organization (Berenschot) were that the costs of the automation of the registration of the Dutch population at the municipal offices were more than twice as much as were estimated (Volkskrant, 5 January 1990). A few years ago, the estimates of the costs were about US$25 million. New calculations show that there is a deficit of more than US$30 million.

A field study by the Eindhoven University of Technology gives an overview of the present state of the art of the estimation and control of software development projects in 598 Dutch organizations. The most remarkable conclusions are:

- 35% of the participating organizations do not make an estimate
- 50% of the responding organizations record no data on an ongoing project
- 57% do not use cost-accounting
- 80% of the projects executed by the participating organizations have overruns of budgets and duration
- the mean overruns of budgets and duration are 50%

Van Lierop et al. 2 measured extensively whether development activities were executed according to plan. They investigated the reasons for the differences between plan and reality, and overall, 80 development activities were measured. For all these activities 3203 hours were planned but 3838 hours were used, which means an overshoot of 20% on average of the planned number of hours. The duration of the activities (in days) proved to be 28% longer on average than planned. For all the activities 406 days of duration were planned, while the actual number of days proved to be 526.

In the literature the impression is given, mistakenly, that software development without overshoots of plans and budgets is not possible. This impression is inaccurate, and other measurements confirm this. These show that 6% of all the activities had a shorter duration than planned and 58% were executed according to plan and were ready exactly on time. With regard

to the development effort, it appeared that 25% of the activities needed less effort than estimated and 30% needed precisely the estimated effort. The reasons for the differences between plan and reality prove to be very specific for the development situation. In the organization where the measurements were taken the reasons were mainly related to things underestimation of the quantity of work, underestimation of the complexity of the application, and specifications which proved to be unrealistic from a technical point of view. In other organizations, where similar measurements were taken, other reasons were discovered. As a result, other control actions are, of course, necessary. This conclusion fits well with the results of research carried out by Beers. Thirty experienced software developers, project managers, and others, were asked to give the reasons for unsuccessful software projects. The answers can be summarized briefly as 'many minds, many thoughts'. It was not possible to indicate just one reason. A long list of all kinds of reasons were given.

It is alarming that it is so difficult for organizations to control the development of software. This is sufficient reason to emphasize that software development cost estimation and control should take its place as a fully-fledged branch within discipline of software development and a reliable automated pipeline should be developed.

1.2 Problem Definition

The main question, when confronting the above-mentioned problems, is what it is that makes software cost estimation so difficult. There are many reasons and, without going into detail, some can be listed as
follows:
(1) There is a lack of data on completed software projects. This kind of data can support project management in making estimates.
(2) Estimates are often done hurriedly, without an appreciation for the effort required to do a credible job. In addition, too often it is the case that an estimate is needed before clear specifications of the system requirements have been produced. Therefore, a typical situation is that estimators are being pressured to write an estimate too quickly for a
system that they do not fully understand.
(3) Clear, complete and reliable specifications are difficult to formulate, especially at the start of a project. Changes, adaptations and additions are more the rule than the exception: as a consequence
plans and budgets must be adapted too.
(4) Characteristics of software and software development make estimating difficult. For example, the level of abstraction, complexity, measurability of product and process, innovative aspects, etc.
(5) A great number of factors have an influence on the effort and time to develop software. These factors are called 'cost drivers'. Examples are size and complexity of the software, commitment and participation of the user organization, experience of the development team. In general, these cost drivers are difficult to determine in operation.
(6) Rapid changes in information technology (IT) and the methodology of software development are a problem for a stabilization of the estimation process. For example, it is difficult to predict the influence of new workbenches, fourth and fifth
generation languages, prototyping strategies, and so on.
(7) An estimator (mostly the project manager) cannot have much experience in developing estimates, especially for large projects. How many 'large' projects can someone manage in, for example, 10
years?

(8) An apparent bias of software developers towards underestimation. An estimator is likely to consider how long a certain portion of the software would take and then to extrapolate this estimate to the rest
of the system, ignoring the non-linear aspects of software development, for example co-ordination and management.
(9) The estimator estimates the time it would take to perform the task personally, ignoring the fact that a lot of work will be done by less experienced people, and junior staff with a lower productivity
rate.
(10) There exists a serious mis-assumption of a linear relation between the required capacity per unit of time and the available time. This would mean that software developed by 25 people in two years could be accomplished by 50 people in one year. The
assumption is seriously wrong. According to Brooks the crucial corollary is: 'Adding people to a late project only makes it later'.
(11) The estimator tends to reduce the estimates to some
degree, in order to make the bid more acceptable.

1.3 Objective of Project

The objective of this project is to introduce techniques for estimating the cost and effort required for software production. As mentioned above in the problem definition, there are a lot of challenges in existing models for software cost estimation. This project aims to eradicate these challenges and make the process as efficient and reliable as possible. Application of data science algorithms such as Neural Network concepts are widely used, one such Neural Network applied is the Convolutional Neural Network. Application of CNN was a major step in the automation of software cost estimation.

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.

Since the 1950s, the early days of AI, researchers have struggled to make a system that can understand visual data. In the following years, this field came to be known as Computer Vision. In 2012, computer vision took a quantum leap when a group of researchers from the University of Toronto developed an AI model that surpassed the best image recognition algorithms and that too by a large margin.

The AI system, which became known as AlexNet (named after its main creator, Alex Krizhevsky), won the 2012 ImageNet computer vision contest with an amazing 85 percent accuracy. The runner-up scored a modest 74 percent on the test.

At the heart of AlexNet was Convolutional Neural Networks a special type of neural network that roughly imitates human vision. Over the years CNNs have become a very important part of many Computer Vision applications and hence a part of any computer vision
course online.

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyse visual imagery. Now when we think of a neural

network, we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

Despite the power and resource complexity of CNNs, they provide in-depth results. At the root of it all, it is just recognizing patterns and details that are so minute and inconspicuous that it goes unnoticed to the human eye. But when it comes to understanding the contents of an image it fails.

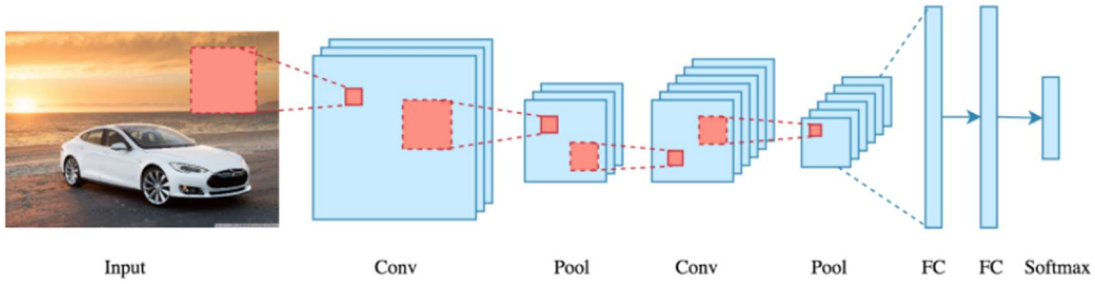


Fig 1. Convolutional flow

Let's take a look at this example. When we pass the below image to a CNN it detects a person in their mid-30s and a child probably around 10 years. But when we look at the same image we start thinking of multiple different scenarios. Maybe it's a father and son day out, a picnic or maybe they are camping. Maybe it is a school ground and the child scored a goal and his dad is happy so he lifts him.

Fig 2. Example Figure for CNN

These limitations are more than evident when it comes to practical applications. For example, CNNs were widely used to moderate content on social media. But despite the vast resources of images and videos that they were trained on it still isn't able to completely block and remove inappropriate content.

Several studies have shown that CNNs trained on ImageNet and other popular datasets fail to detect objects when they see them under different lighting conditions and from new angles.

To overcome the limitations of CNN we apply a new Neural Network which is Graph Neural Network for better results. Many real-world tasks can be transformed into the graph related tasks, including Software Cost Estimation. For example, predicting the metrics present in software are usually transformed into the Software Cost Estimation. The predicted metrics will thus be helpful for further development research. Recent development on graph neural network (GNN) provides the most promising approach to solve the Software Cost Estimation

task. Nevertheless, most GNNs learn the node embeddings through a relatively “shallow” neighbourhood which contains information only two or three hops away from each node.

Graph Neural Network, as how it is called, is a neural network that can directly be applied to graphs. It provides a convenient way for node level, edge level, and graph level prediction task.

The intuition of GNN is that nodes are naturally defined by their neighbours and connections. To understand this, we can simply imagine that if we remove the neighbours and connections around a node, then the node will lose all its information. Therefore, the neighbours of a node and connections to neighbours define the concept of the node.

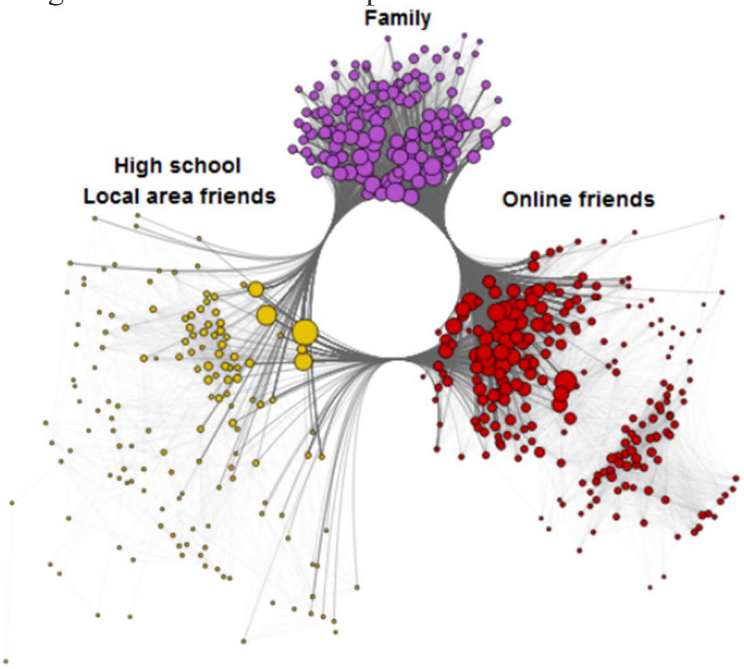


Fig 3. A social media graphs

Having this in mind, we then give every node a state *(x)* to represent its concept. We can use the node state *(x)* to produce an output *(o)*, i.e., decision about the concept. The final state *(x_n)* of the node is normally called “node embedding”. A simple yes or no would suffice. The task of all GNN is to determine the “node embedding” of each node, by looking at the information on its neighbouring nodes.

1.4 Limitations of Project

The number of GNN layers is limited due to the Laplacian smoothing. Thus, the number of hidden layers in GNN usually is set to two or three. More layers will possibly result in the Laplacian smoothing result which will reduce the performance of learning. Recently a much deeper version of GCN is proposed that the number of hidden layers can be added to hundred.

1.4.1 Graph Structure Limitation

Turns out conventional GNN's are not able to distinguish simple graph structures. GCN and GraphSAGE both fail to distinguish the below two graphs, and their network structure. If you start with input node features that are uniform, but different edge combinations, we

want the GNN to capture that structural difference. But in this case both models fail to do so.

1.4.2 Noise Vulnerability in GNN

Deep neural networks are vulnerable to adversarial attacks. For example, in image recognition task we can add a layer of noise data to an image in a way it is not recognizable to humans but you can fool the neural network to identify it as some other image.

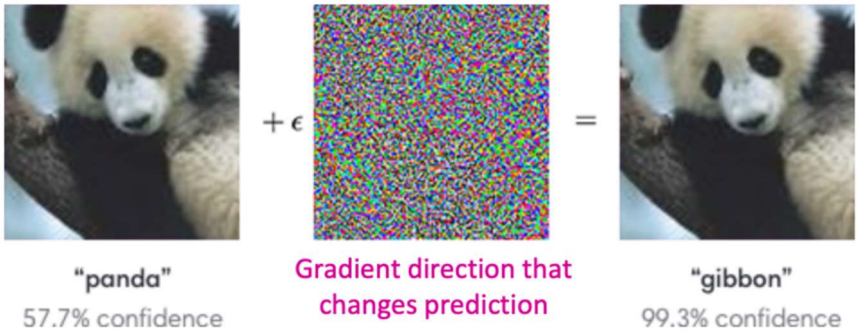


Fig 5. Noise vulnerability in GNN

2. Literature Survey

2.1 Introduction

Software cost estimation is one of the most challenging tasks in software engineering. Over the past years the estimators have used parametric cost estimation models to establish software cost, however the challenges to accurate cost estimation keep evolving with the advancing technology. Planned effort and actual effort have been comparison in detail through applying on NASA projects. This project uses Graph neural networks for software cost estimation. Artificial Neural Network represents a complex set of relationship between the effort and the cost drivers and is a potential tool for estimation. The proposed model automates the software cost estimation task and helps project manager to provide fast and realistic estimate for the project effort and development time that in turn gives software cost.

Many cost estimation methods are available. Still research is going on in this area in order to find more accurate cost estimation method. In this section, the work done in this field is presented.

Neha Sharma et. al. (2013) proposed model to assess parameters for NASA software project dataset using a genetic algorithm (GA). The result shows that the three models (Organic model, Semi-detached model and Embedded model) take much larger time and performs inferior than proposed model [1].

A. Idri and A. Zahi (2013) evaluate and compare the classical analogy and fuzzy analogy for software cost estimation on a web software dataset. They estimated accuracy, tolerance of imprecision and uncertainty of cost drivers showing the usefulness of Fuzzy Analogy for software cost estimation [2].

M. Azzeh (2013) analyses the potential of Use Case Point estimation model for global projects and uses this as a basis to discuss three proposed factors (Global team trust, Global team composition and Culture value) that helps in managing the global software project development [3].

A. Kaushik et. al. (2013) mapped the COCOMO model by using a Feed Forward Back Propagation neural network. The proposed model takes identity function at the input layer and sigmoid function at the hidden and output layer. The model uses COCOMO and COCOMO NASA 2 dataset to train and test the network [4].

Anupama Kaushik et. al. (2012) made an analysis to use the fuzzy logic in the COCOMO model and provided in-depth review and comparison of software effort estimation models. They presented an overview of fuzzy approaches in COCOMO's effort estimation [5, 6].

Surendra Pal Singh et. al. (2012) analyses neural network using Bayesian Regularization training algorithm and produces reduced condition numbers as compared to the fuzzy model having membership functions. It is concluded that the neural network model using Bayesian

Regularization training algorithm is a more stable model than the fuzzy model having membership functions [7].

Abeer Hamdy (2012) developed an adaptive fuzzy model for software effort estimation. A fuzzy logic-based component with COCOMO81 intermediate model is used to improve its accuracy and sensitivity. The proposed model uses genetic algorithms (GA) to tune the fuzzy sets parameters [8].

Swarup Kumar et. al. (2011) proposed fuzzy software cost estimation model that handles ambiguousness, obscurity and a comparison is made with other popular software cost estimation models. Fuzzy logic method is used to address the problem of obscurity and vagueness exists in software effort drivers to estimate software effort [9].

Gharehchopogh (2011) made a case study for software cost estimation using Neural Network (NN) architecture for predicting necessary effort of new software. The results indicate that the NN model offers the very best algorithmic method to predict and estimate software costs.

2.2 Existing System

The existing system for this project utilizes, decision trees or support vector regressor or random forest or CNN logic, the mentioned algorithms operate as follows:
Decision Trees:

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types:

1. Categorical Variable Decision Tree: Decision Tree which has a categorical target variable then it called a Categorical variable decision tree.
2. Continuous Variable Decision Tree: Decision Tree has a continuous target variable then it is called Continuous Variable Decision Tree.

Example: - Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customers is a significant variable but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree

to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

Important Terminology related to Decision Trees

1. Root Node**:** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. Splitting: It is a process of dividing a node into two or more sub-nodes.
3. Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.
4. Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.
5. Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. Branch / Sub-Tree**:** A subsection of the entire tree is called branch or sub-tree.
7. Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example. Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

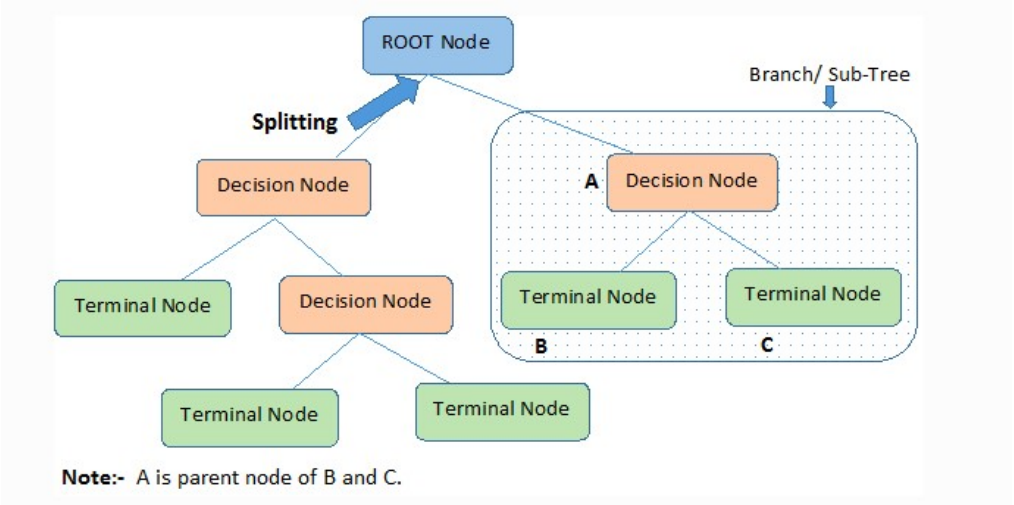


Fig.6 Decision Trees

Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

- In the beginning, the whole training set is considered as the root.

- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Trees follow Sum of Product (SOP) **r**epresentation. The Sum of product (SOP) is also known as Disjunctive Normal Form. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum). The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know as the attribute's selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

Support Vector Regressor:

Support Vector Machine (SVM) is a very popular Machine Learning algorithm that is used in both Regression and Classification. Support Vector Regression is similar to Linear Regression in that the equation of the line is "y= wx+b" in SVR, this straight line is referred to as hyperplane. The data points on either side of the hyperplane that are closest to the hyperplane are called Support Vectors which is used to plot the boundary line.

Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value (Distance between hyperplane and boundary line), **a**. Thus, we can say that SVR model tries satisfy the condition "-a < y-wx+b < a". It used the points with this boundary to predict the value.

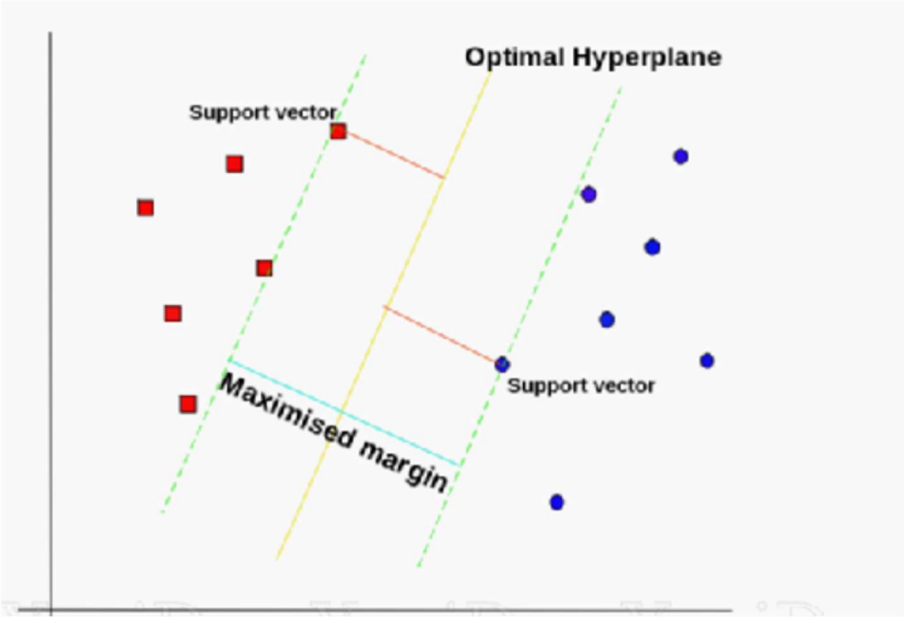


Fig.7 SVR's example

Hyperparameters of the Support Vector Regressor (SVR) Algorithm

There are a few important parameters of SVM that you should be aware of before proceeding further:

- **Kernel:** A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Usually, the computational cost will increase if the dimension of the data increases. This increase in dimension is required when we are unable to find a separating hyperplane in a given dimension and are required to move in a higher dimension:

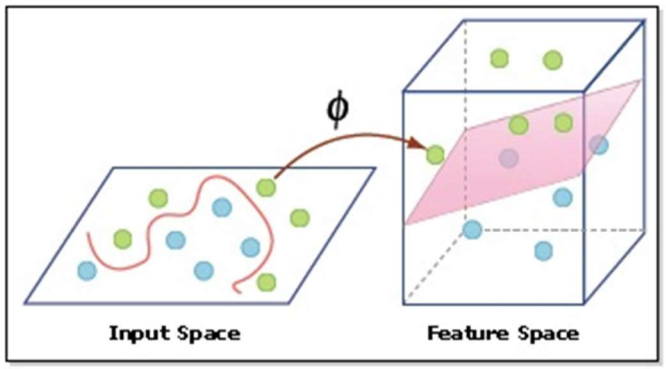


Fig.8 Kernels for SVR

- Hyperplane: This is basically a separating line between two data classes in SVM. But in Support Vector Regression, this is the line that will be used to predict the continuous output
- Decision Boundary: A decision boundary can be thought of as a demarcation line (for simplification) on one side of which lie positive examples and on the other side lie the negative examples. On this very line, the examples may be classified as either positive or negative. This same concept of SVM will be applied in Support Vector Regression as well

Random Forest

Random Forests is a Machine Learning algorithm that tackles one of the biggest problems with Decision Trees variance**.** Even though Decision Trees is simple and flexible, it is greedy algorithm. It focuses on optimizing for the node split at hand, rather than taking into account how those split impacts the entire tree. A greedy approach makes Decision Trees run faster, but makes it prone overfitting**.**An overfit tree is highly optimized to predicting the values in the training dataset, resulting in a learning model with high-variance.

How you calculate variance in a Decision Tree depends on the problem you're solving. In a Regression task you can calculate actual variance of the prediction compared to the true

targets. If the tree produces results that are too far off from its true targets, it has high-variance and therefore, it is overfit.

A highly overfit tree has high-variance. That means its predictions are extremely far off from the actual targets.

But in a Classification task, you can't use the same approach. The way to detect if the Decision Tree is overfit is by looking at the test error. If the tree has a high-test error, meaning it's not good at classifying observations it wasn't trained on, it is overfit.

In Regression tasks overfitting is detected by high-variance, while in Classification tasks it's by a high generalization error.

To address overfitting, and reduce the variance in Decision Trees, Leo Breiman developed the Random Forests algorithm. This was an innovative algorithm because it utilized, for the first time, the statistical technique of Bootstrapping and combined the results of training multiple models into a single, more powerful learning model.

Random Forests was developed specifically to address the problem of high-variance in Decision Trees. Like the name suggests, you're not training a single Decision Tree, you're training an entire forest! In this case, a forest of Bagged Decision Trees.

At a high-level, in pseudo-code, Random Forests algorithm follows these steps:

1. Take the original dataset and create *N* bagged samples of size *n*, with *n* smaller than the original dataset.

2. Train a Decision Tree with each of the *N* bagged datasets as input. But, when doing a node split, don't explore all features in the dataset. Randomly select a

smaller number, *M* features, from all the features in training set. Then pick the best split using impurity measures, like Gini Impurity or Entropy.

3. Aggregate the results of the individual decision trees into a single output.

4. Average the values for each observation, produced by each tree, if you're working on a Regression task.

5. Do a majority vote across all trees, for each observation, if you're working on a Classification task.

While Forest part of Random Forests refers to training multiple trees, the Random part is present at two different points in the algorithm.

There's the randomness involved in the Bagging process. But then, you also pick a random subset of features to evaluate the node split. This is what guarantees that each tree is different and, therefore, ensures each model produces a slightly different result.

And while you could be thinking that randomly sampling features at each split introduces yet another hyperparameter you might need to tune, that's not the case. The model takes care of it for you!

You can certainly tune this hyperparameter. However, there's mathematical consensus about randomly picking a number of features that is equal to the square root of the total number of available features in the dataset.

Important Features of Random Forest
1. Diversity- Not all attributes/variables/features are considered while making an individual tree, each tree is different.

2. Immune to the curse of dimensionality- Since each tree does not consider all the features, the feature space is reduced.

3. Parallelization-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. Train-Test split- In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

5. Stability- Stability arises because the result is based on majority voting/ averaging.

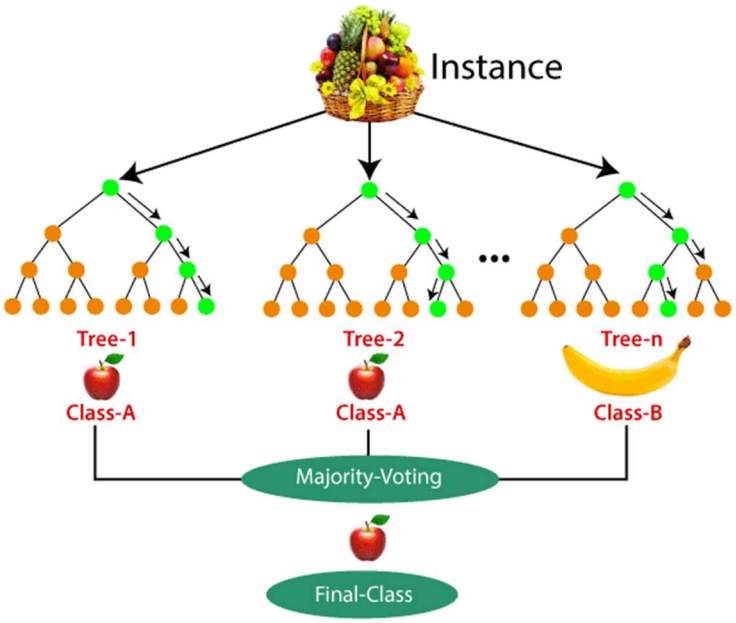


Fig.9 Random Forest

Convolutional Neural Networks (CNN):

Before we go to the working of CNN's let's cover the basics such as what is an image and how is it represented. An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.

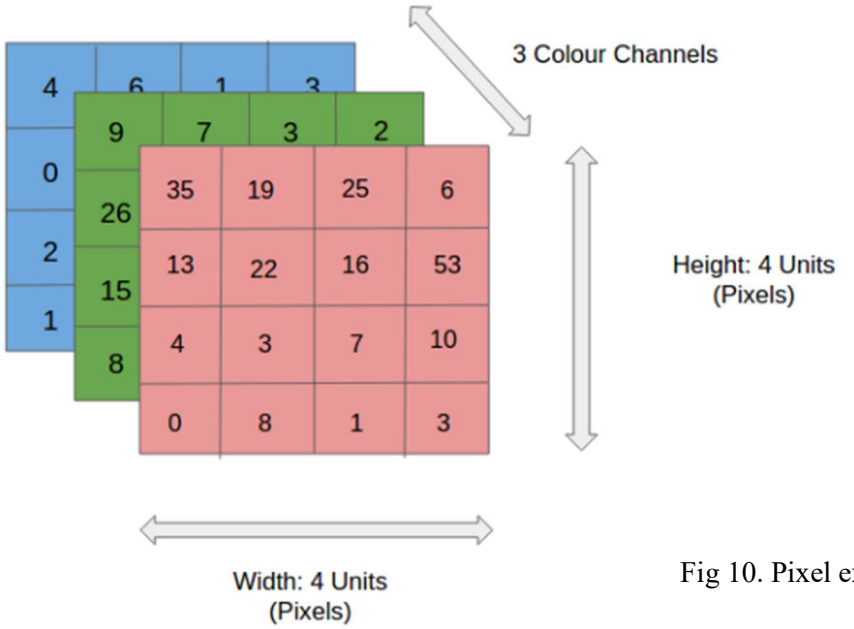
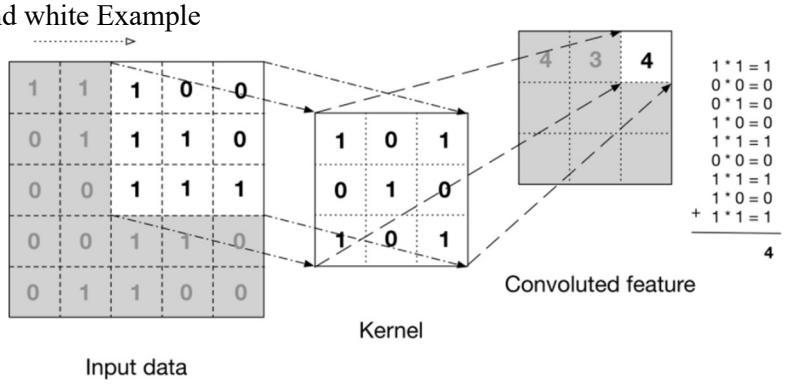


Fig 10. Pixel example

For simplicity, let's stick with grayscale images as we try to understand how CNNs work.

Fig 11. Black and white Example



The above image shows what a convolution is. We take a filter/kernel (3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.

Fig 12. Convolved Feature Stage

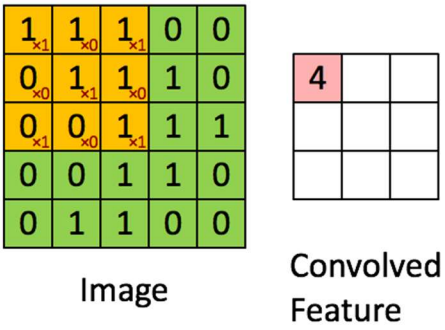
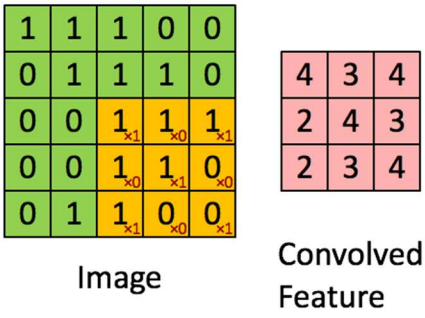


Fig 13. Convolved Feature Final Stage



Convolutional neural networks are composed of multiple layers of artificial neurons. Artificial neurons, a rough imitation of their biological counterparts, are mathematical functions that calculate the weighted sum of multiple inputs and outputs an activation value. When you input an image in a ConvNet, each layer generates several activation functions that are passed on to the next layer.

The first layer usually extracts basic features such as horizontal or diagonal edges. This output is passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, etc.

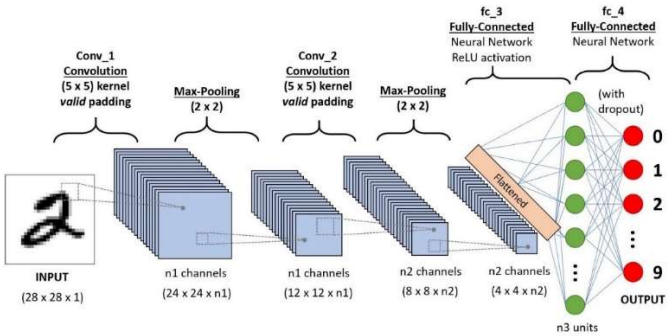


Fig.14 Convolution Stages

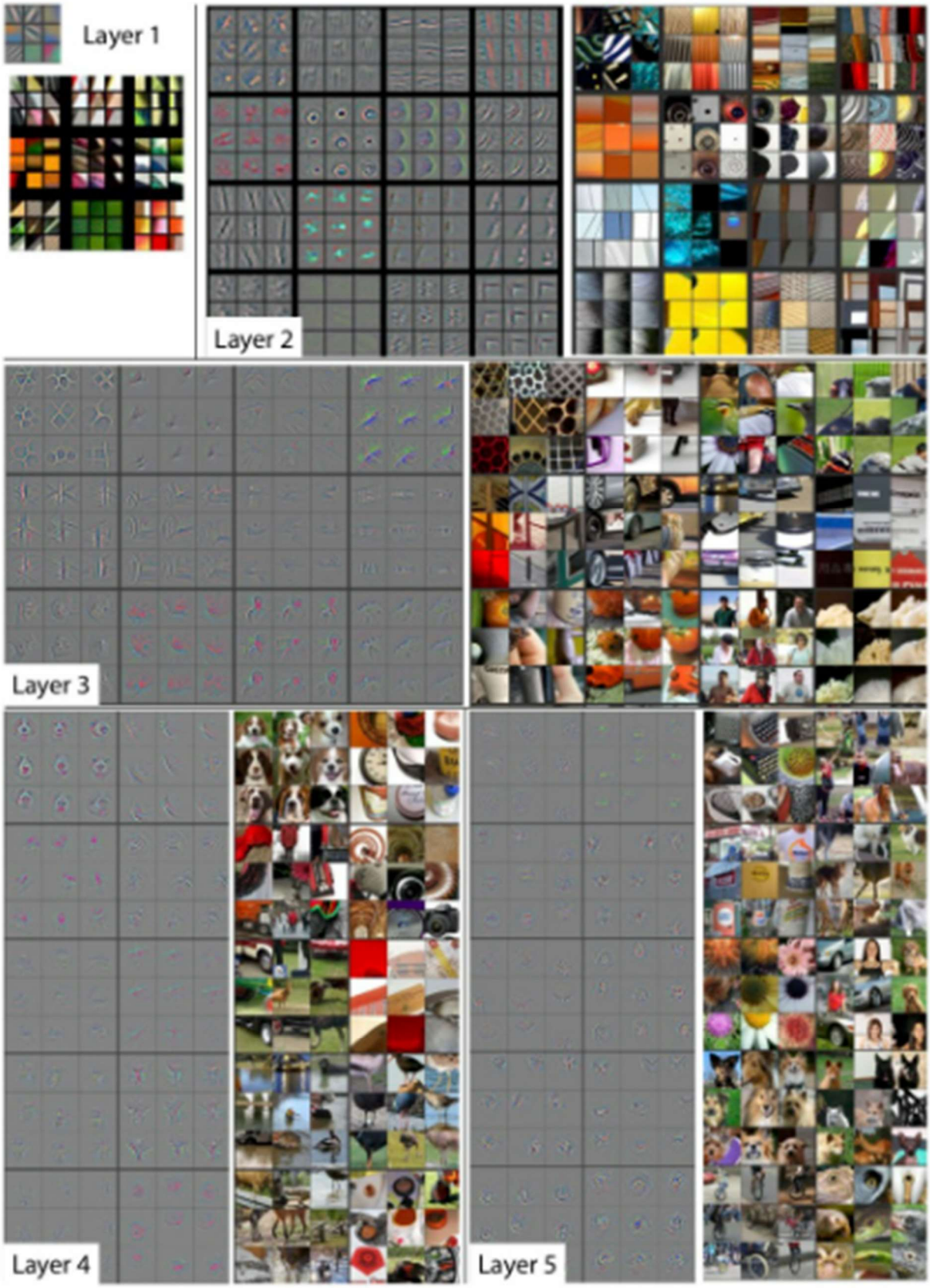
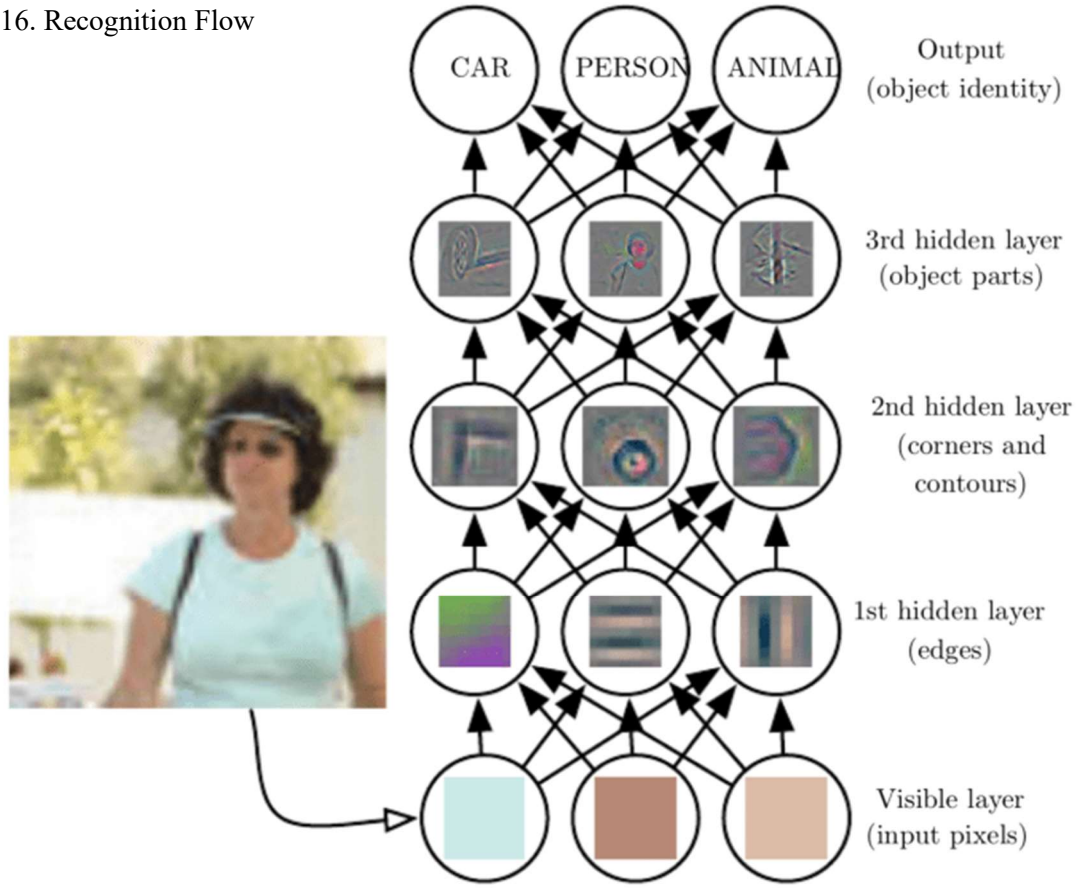


Fig 15. Final Layered Feature Flow

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

Fig 16. Recognition Flow



2.3 Disadvantages of Existing System

Disadvantages of Decision Trees

1. Unstable nature

One of the limitations of decision trees is that they are largely unstable compared to other decision predictors. A small change in the data can result in a major change in the structure of the decision tree, which can convey a different result from what users will get in a normal event. The resulting change in the outcome can be managed by machine learning algorithms, such as boosting and bagging.

2. Less effective in predicting the outcome of a continuous variable

In addition, decision trees are less effective in making predictions when the main goal is to predict the outcome of a continuous variable. This is because decision trees tend to lose information when categorizing variables into multiple categories.

Disadvantages of Support Vector Regressor:

1. SVM algorithm is not suitable for large data sets.
2. SVM does not perform very well when the data set has more noise i.e., target classes are overlapping.
3. In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.
4. As the support vector classifier works by putting data points, above and below the classifying hyperplane there is no probabilistic explanation for the classification.

Disadvantages of Random Forest Algorithm:

1. Complexity**:** Random Forest creates a lot of trees (unlike only one tree in case of decision tree) and combines their outputs. By default, it creates 100 trees in Python sklearn library. To do so, this algorithm requires much more computational power and resources. On the other hand, decision tree is simple and does not require so much computational resources.

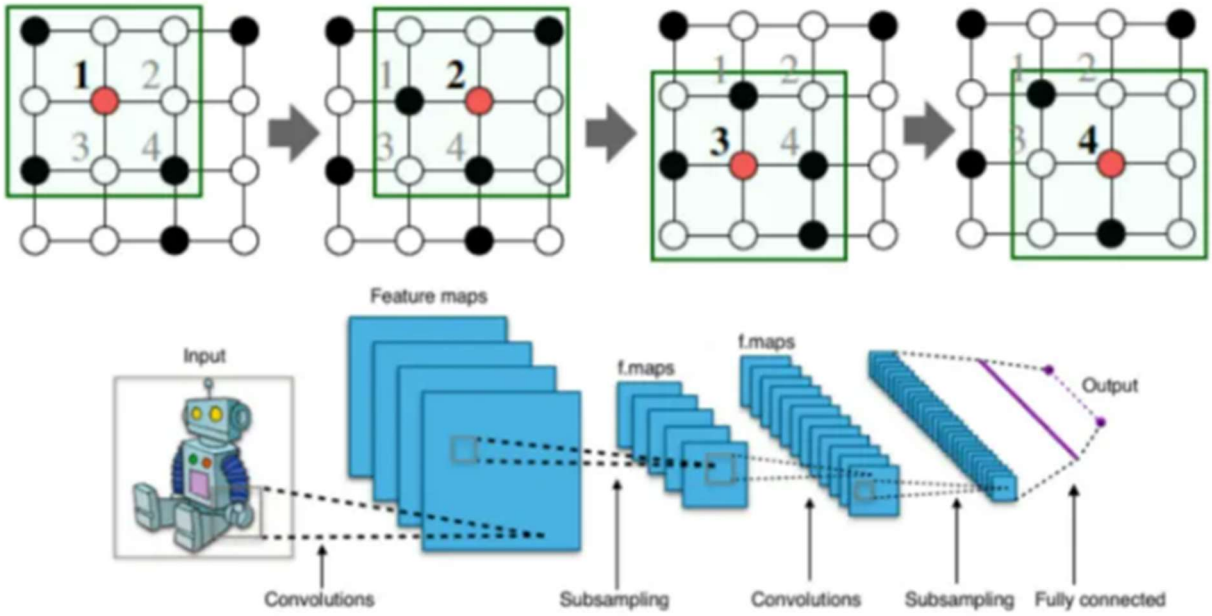
2. Longer Training Period**:** Random Forest require much more time to train as compared to decision trees as it generates a lot of trees (instead of one tree in case of decision tree) and makes decision on the majority of votes.

Disadvantages of Convolutional Neural Networks

CNNs can be used to make machines visualize things, and perform tasks like image classification, image recognition, or object detection. This is where CNNs are the most popular.

The core concept behind CNNs introduces hidden convolution and pooling layers to identify spatially localized features via a set of receptive fields in kernel form.

Fig 17. Typical CNN architecture



How does convolution operate on images that are regular grids? We slide the convolutional operator window across a two-dimensional image, and we compute some function over that sliding window. Then, we pass it through many layers.

Our goal is to generalize the notion of convolution beyond these simple two-dimensional lattices.

The insight allowing us to reach our goal is that convolution takes a little sub-patch of the image (a little rectangular part of the image), applies a function to it, and produces a new part (a new pixel).

What happens is that the centre node of that centre pixel aggregates information from its neighbours, as well as from itself, to produce a new value.

It's very difficult to perform CNN on graphs because of the arbitrary size of the graph, and the complex topology, which means there is no spatial locality.

There's also unfixed node ordering. If we first labelled the nodes A, B, C, D, E, and the second time we labelled them B, D, A, E, C, then the inputs of the matrix in the network will change. Graphs are invariant to node ordering, so we want to get the same result regardless of how we order the nodes.

2.4 Proposed System

In the proposed System we implement Graphical Neural Networks, which operate as follows:

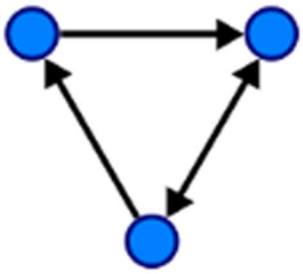
Recently, Graph Neural Network (GNN) has gained increasing popularity in various domains, including social network, knowledge graph, recommender system, and even life science. The power of GNN in modelling the dependencies between nodes in a graph enables the breakthrough in the research area related to graph analysis.

2.4.1 Graph

Before we get into GNN, let's first understand what is *Graph*. In Computer Science, a graph is a data structure consisting of two components, vertices and edges. A graph G can be well described by the set of vertices V and edges E it contains.

$$G = (V, E)$$

Edges can be either directed or undirected, depending on whether there exist directional dependencies between vertices.



A Directed Graph

The vertices are often called nodes. In this article, these two terms are interchangeable.

2.4.2 Graph Neural Network

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure. A typical application of GNN is node classification. Essentially, every node in the graph is associated with a label, and we want to predict the label of the nodes without ground-truth. This section will illustrate the algorithm described in the project, the first proposal of GNN and thus often regarded as the original GNN.

In the node classification problem setup, each node v is characterized by its feature x_v and associated with a ground-truth label t_v. Given a partially labelled graph G, the goal is to leverage these labelled nodes to predict the labels of the unlabelled.

$$\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]})$$

It learns to represent each node with a *d* dimensional vector (state) *h_v* which contains the information of its neighbourhood. Specifically, where *x_co[v]* denotes the features of the edges connecting with *v, h_ne[v]* denotes the embedding of the neighbouring nodes of *v*, and *x_ne[v]* denotes the features of the neighbouring nodes of *v.* The function *f* is the transition function that projects these inputs onto a d-dimensional space. Since we are seeking a unique solution for h_v, we can apply Banach fixed point theorem and rewrite the above equation as an iteratively update process. Such operation is often referred to as **message passing** or **neighbourhood aggregation**.

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X})$$

H and X denote the concatenation of all the h and x, respectively.

The output of the GNN is computed by passing the state *h v* as well as the feature *x v* to an output function g.

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v)$$

Both f and g here can be interpreted as feed-forward fully-connected Neural Networks. The L1 loss can be straightforwardly formulated as the following:

$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$

2.5 Advantages of GNN

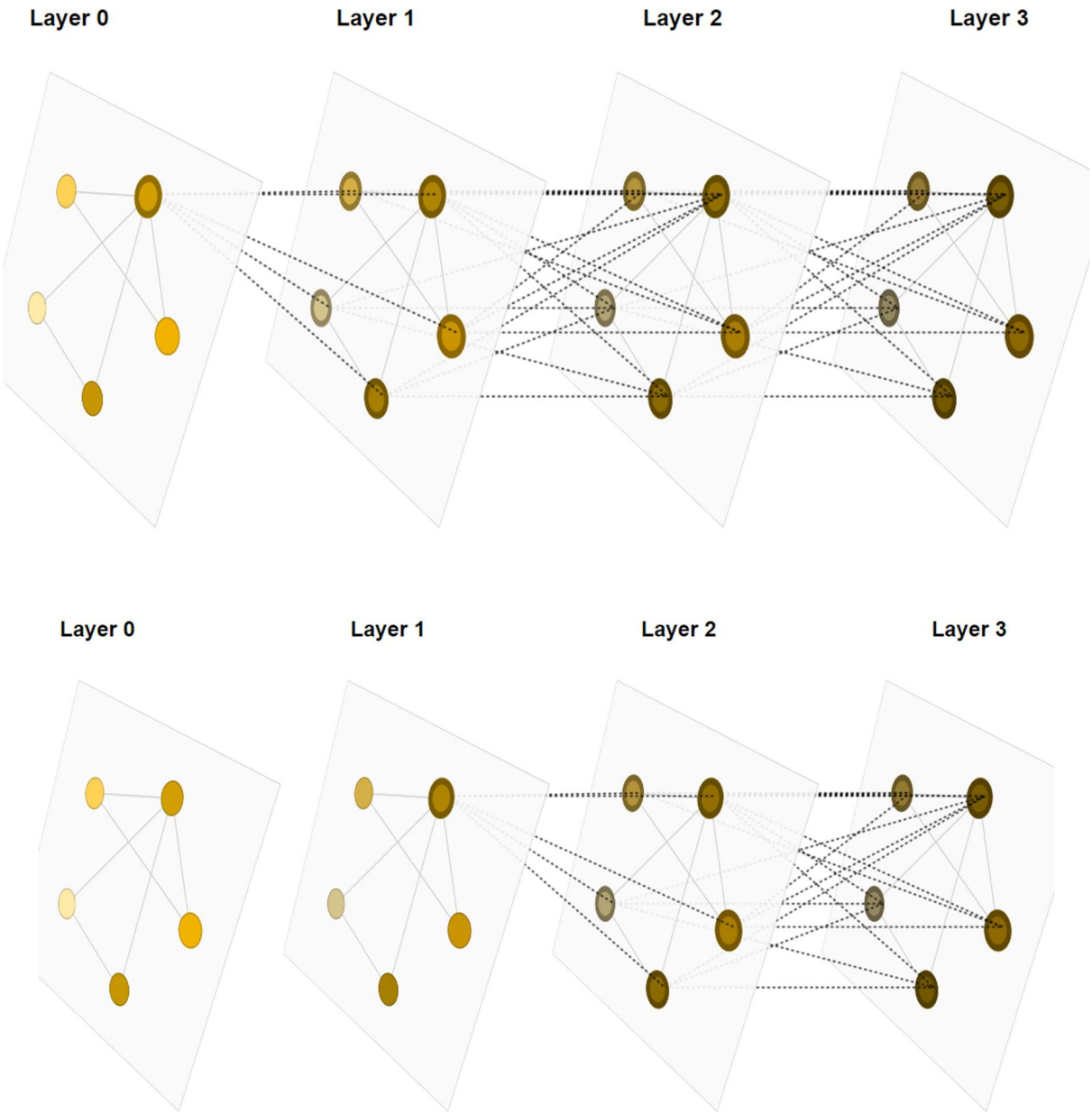
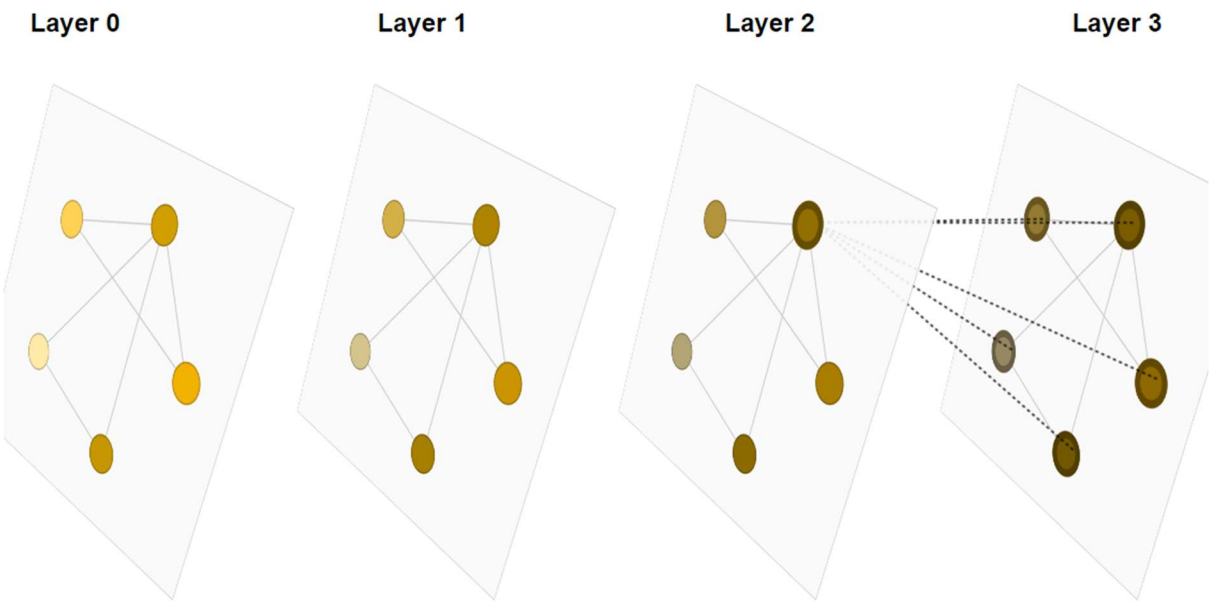


Fig.18 GNN Stream through Layers



The problems that GNN solve can be broadly classified into three categories:

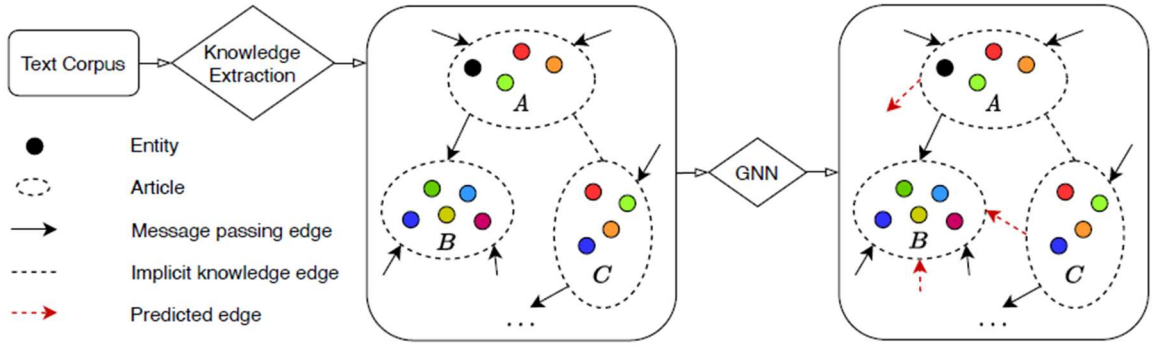
1. Node Classification
2. Link Prediction
3. Graph Classification

In node classification, the task is to predict the node embedding for every node in a graph. This type of problem is usually trained in a semi-supervised way, where only part of the graph is labelled. Typical applications for node classification include citation networks, Reddit posts, YouTube videos, and Facebook friends' relationships.

In link prediction, the task is to understand the relationship between entities in graphs and predict if two entities have a connection in between. For example, a recommender system can be treated as link prediction problem where the model is given a set of users' reviews of different products, the task is to predict the users' preferences and tune the recommender system to push more relevant products according to users' interest.

In graph classification, the task is to classify the whole graph into different categories. It is similar to image classification but the target changes into graph domain. There is a wide range of industrial problems where graph classification can be applied, for example, in chemistry, biomedical, physics, where the model is given a molecular structure and asked to classify the target into meaningful categories. It accelerates the analysis of atom, molecule or any other structured data types.

Fig 19. An overview of General pipeline



There are three general types of prediction tasks on graphs: graph-level, node-level, and edge-level.

In a graph-level task, we predict a single property for a whole graph. For a node-level task, we predict some property for each node in a graph. For an edge-level task, we want to predict the property or presence of edges in a graph.

For the three levels of prediction problems described above (graph-level, node-level, and edge-level), we will show that all of the following problems can be solved with a single model class, the GNN. But first, let's take a tour through the three classes of graph prediction problems in more detail, and provide concrete examples of each.

Graph-level task

In a graph-level task, our goal is to predict the property of an entire graph. For example, for a molecule represented as a graph, we might want to predict what the molecule smells like, or whether it will bind to a receptor implicated in a disease.

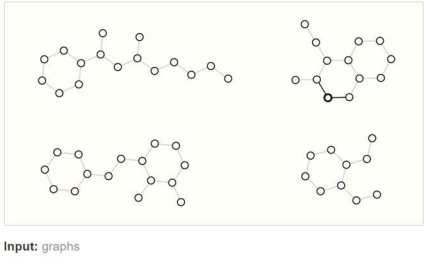
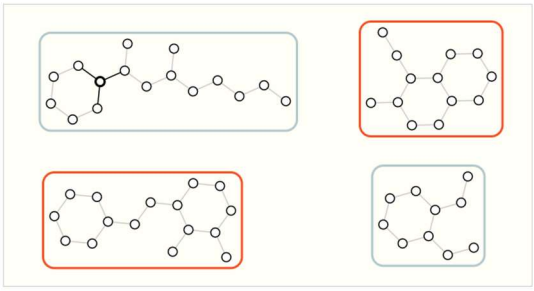


Fig.20 Graph level task

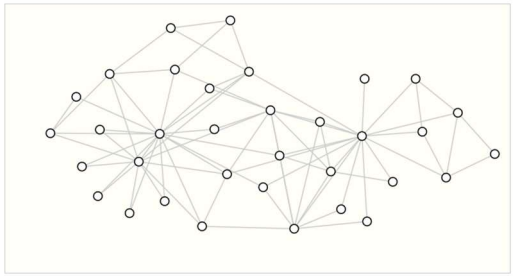


Output: labels for each graph, (e.g., "does the graph contain two rings?")

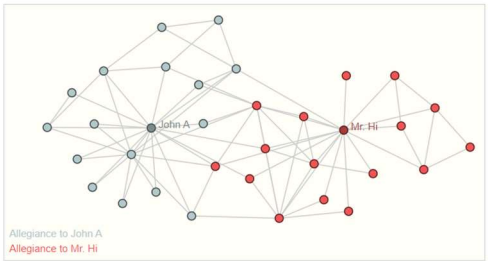
This is analogous to image classification problems with MNIST and CIFAR, where we want to associate a label to an entire image. With text, a similar problem is sentiment analysis where we want to identify the mood or emotion of an entire sentence at once.

Node-level task

Node-level tasks are concerned with predicting the identity or role of each node within a graph. A classic example of a node-level prediction problem is Zach's karate club. The dataset is a single social network graph made up of individuals that have sworn allegiance to one of two karate clubs after a political rift. As the story goes, a feud between Mr. Hi (Instructor) and John H (Administrator) creates a schism in the karate club. The nodes represent individual karate practitioners, and the edges represent interactions between these members outside of karate. The prediction problem is to classify whether a given member becomes loyal to either Mr. Hi or John H, after the feud. In this case, distance between a node to either the Instructor or Administrator is highly correlated to this label.



Input: graph with unlabled nodes



Output: graph node labels

Fig.21 Node-Level Task

Following the image analogy, node-level prediction problems are analogous to image segmentation, where we are trying to label the role of each pixel in an image. With text, a

similar task would be predicting the parts-of-speech of each word in a sentence (e.g., noun, verb, adverb, etc).

Edge-level task

The remaining prediction problem in graphs is edge prediction. One example of edge-level inference is in image scene understanding. Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them. We can phrase this as an edge-level classification: given nodes that represent the objects in the image, we wish to predict which of these nodes share an edge or what the value of that edge is. If we wish to discover connections between entities, we could consider the graph fully connected and based on their predicted value prune edges to arrive at a sparse graph.

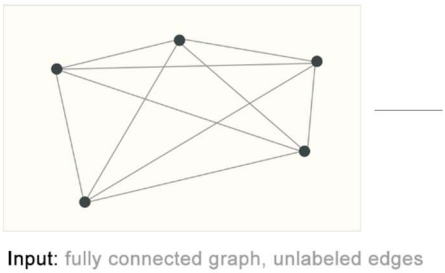


Fig.22 Edge level task

Table 1. A summary overview of GNNs Applications

Application	Deep Learning	Description
Text classification	Graph convolutional network/ graph attention network	A classic application of GNNs in NLP is Text Classification. GNNs utilize the inter-relations of documents or words to infer document labels. GCN and GAT models are applied to solve this task. They convert text to graph-of-words, and then use graph convolution operations to convolve the word graph. They show through experiments that the graph-of-words representation of texts has the advantage of capturing non-consecutive and long-distance semantics
Neural machine translation	Graph convolutional network/ gated graph neural network	The neural machine translation (NMT) is considered a sequence-to-sequence task. One of GNN's common applications is to incorporate semantic information into the NMT task. To do this, we utilize the Syntactic

		GCN on syntax-aware NMT tasks. We can also use the GGNN in NMT. It converts the syntactic dependency graph into a new structure by turning the edges into additional nodes and thus edges labels can be represented as embeddings
Relation extraction	Graph LSTM/ graph convolutional network	Relation Extraction is the task of extracting semantic relations from the text, which usually occur between two or more entities. Traditional systems treat this task as a pipeline of two separated tasks, i.e., named entity recognition (NER) and relation extraction, but new studies show that end-to-end modeling of entity and relation is important for high performance since relations interact closely with entity information
Image classification	Graph convolutional network/ gated graph neural network	Image classification is a basic computer vision task. Most of the models provide attractive results when given a huge training set of labelled classes. The focus now is towards getting these models to perform well on zero-shot and few-shot learning tasks. For that, GNN appears quite appealing.
Object detection Interaction detection Region classification Semantic segmentation	Graph attention network Graph neural network Graph CNN Graph LSTM/ gated graph neural network/ graph CNN/ graph neural network	There are other applications of computer vision tasks like object detection, interaction detection, and region classification. In object detection, GNNs are used to calculate RoI features; in interaction detection, GNN is message-passing tools between humans and objects; in region classification, GNNs perform reasoning on graphs that connect regions and classes
Physics	Graph neural network/ graph networks	Modelling real-world physical systems is one of the most basic aspects of understanding human intelligence. By representing objects as nodes and relations as edges, we can perform GNN-based reasoning about objects, relations, and physics in an effective way. Interaction networks can be trained to reason about the interactions of objects in a complex physical system. It can make predictions and inferences about various system properties in domains such as collision dynamics
Molecular fingerprints	Graph convolutional network	Molecular fingerprints are feature vectors that represent molecules. ML models predict the properties of a new molecule by learning from example molecules that use fixed-length fingerprints as inputs. GNNs can replace the traditional means that give a fixed encoding of the molecule to allow the generation of differentiable fingerprints adapted to the task for which they are required
Graph generation	Graph convolutional network/ graph neural	Generative models for real-world graphs have drawn significant attention for their important applications

	network/ LSTM /RNN/ relational-GCN	including modelling social interactions, discovering new chemical structures, and constructing knowledge graphs. The GNN based model learns node embeddings for each graph independently and matches them using attention mechanisms. This method offers good performance compared to standard relaxation-based techniques
Combinatorial optimization	Graph convolutional network/ graph neural network/ graph attention network	Combinatorial optimization (CO) is a topic that consists of finding an optimal object from a finite set of objects. It is the base of many important applications in finance, logistics, energy, science, and hardware design. Most CO problems are formulated with graphs. In a recent work by DeepMind and Google, graph nets are used for two key subtasks involved in the MILP solver: joint variable assignment and bounding the objective value. Their neural network approach is faster than existing solvers on big datasets

3.Analysis

3.1 Introduction

A neural network is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.

Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of estimation systems.

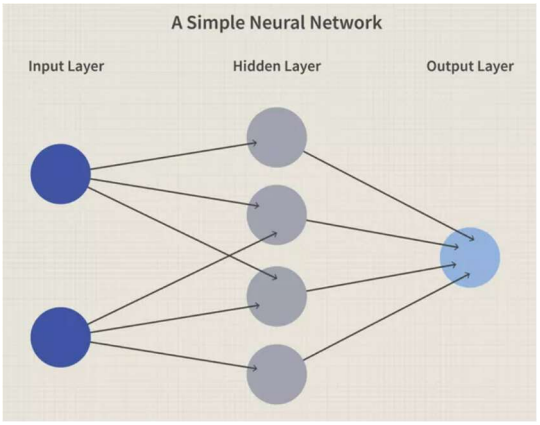


Fig 23. A Generic Neural Network

- Neural networks are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data.
- As such, they tend to resemble the connections of neurons and synapses found in the brain.
- They are used in a variety of applications in financial services, from forecasting and marketing research to fraud detection and risk assessment.
- Neural networks with several process layers are known as "deep" networks and are used for deep learning algorithms

A neural network works similarly to the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains layers of interconnected nodes. Each node is a known as perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

A convolutional neural network is one adapted for analysing and identifying visual data such as digital images or photographs.

3.2 Software Requirement Specifications

3.2.1 User Specification:

The User has to provide data to the framework in order for the algorithem to produce useful predictions, user should also make sure about providing relavent and un-biased data for accurate predictions.

3.2.2 Software specifications:

- Windows or Linux or MAC
- Python GUI or Anaconda Navigator
- Various useful python modules (sklearn, Keras Python deep learning API)

3.2.3 Hardware Requirements

- Minimum of 8 GB RAM
- At least 10 GB of Hard Disk space
- Intel Processor
- LAN or Wi-Fi connetivity

A Software Requirements Specification (SRS) – a requirements specification for a software system – is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements that impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). A structured collection of information that embodies the requirements of a system. A business analyst, sometimes titled system analyst, is responsible for analysing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the systems development life cycle domain typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. The software specification describes the intended purpose, requirements, and the nature of the software to be developed. It supposedly includes the yield and cost of the software. The requirements are broadly divided into two groups:

• Functional Requirements
• Non-functional Requirements

Functional Requirements: A functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional software requirements help you to capture the intended behavior of the system. This behaviour may be expressed as functions, services, or tasks or which system is required to perform.

Non-functional Requirements: A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. For example, how fast does the website load? A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non-functional Requirements allows you to Impose constraints or restrictions on the design of the system across the various agile backlogs.

For example, the site should load in 3 seconds when the number of simultaneous users is > 10000. Description of non-functional requirements is just as critical as a functional requirement.

3.2.3TECHNOLOGY USED

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).
Python is a high-level, interpreted, interactive, and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Characteristics of Python:

Following are important characteristics of **Python Programming** −

• It supports functional and structured programming methods as well as OOP.
• It can be used as a scripting language or can be compiled to byte-code for building large applications.
• It provides very high-level dynamic data types and supports dynamic type checking.
• It supports automatic garbage collection.
• It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Advantages of Python

As mentioned before, Python is one of the most widely used languages over the web. I'm going to list a few of them here:

• **Easy-to-learn** − Python has few keywords, a simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
• **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
• **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
• **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
• **Interactive Mode** − Python has support for an interactive mode that allows interactive testing and debugging of snippets of code.
• **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
• **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
• **Databases** − Python provides interfaces to all major commercial databases.
• **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
• **Scalable** − Python provides a better structure and support for large programs than shell scripting. The biggest strength of Python is its huge collection of standard libraries which can be used for the following:

• Machine Learning
• GUI Applications (like Kivy, Tkinter, PyQt, etc.)
• Web frameworks like Django (used by YouTube, Instagram, Dropbox)
• Image processing (like OpenCV, Pillow)
• Web scraping (like Scrapy, Beautiful Soup, Selenium)
• Test frameworks
• Multimedia
• Scientific computing
• Text processing, etc.

Machine Learning

Machine learning is a data analytics technique that teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to "learn" information directly from data without relying on a predetermined equation as a model. The algorithms adaptively
improve their performance as the number of samples available for learning increases. Deep learning is a specialized form of machine learning.

There are Seven Steps of Machine Learning:

1. Gathering Data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyperparameter Tuning
7. Prediction

Some machine learning methods:

Machine learning algorithms are often categorized as supervised or unsupervised.

• **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system can provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors to modify the model accordingly.

• In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labelled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabelled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures
from unlabelled data.

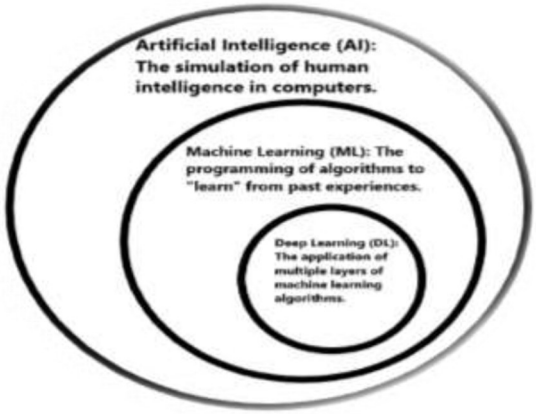
• **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning since they use both labelled and unlabelled data for training – typically a small amount of labelled data and a large amount of unlabelled data. The systems that use this method can considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labelled data requires skilled and relevant resources to train it / learn from it. Otherwise, acquiring unlabelled data generally doesn't require
additional resources.

• **Reinforcement machine learning algorithms** are a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behaviour within a

specific context to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.
Machine learning enables the analysis of massive quantities of data. While it, generally, delivers faster, more accurate results to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more
effective in processing large volumes of information.

How does Machine Learning work?
The three major building blocks of a Machine Learning system are the model, the parameters, and the learner.
• Model is the system that makes predictions
• The parameters are the factors that are considered by the model to make predictions.
• The learner makes the adjustments in the parameters and the model to align the predictions with the actual results.

The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns. You can use the ML model to get predictions on new data for which you do not know the target.



NumPy
NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Num-array was also developed, having some additional functionalities. In 2005, Travis Oliphant created the NumPy package by incorporating the features of Num-array into the Numeric package. There are many contributors to this open-source
project. Using NumPy, a developer can perform the following operations −
• Mathematical and logical operations on arrays.
• Fourier transforms and routines for shape manipulation.
• Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
NumPy – A Replacement for MatLab NumPy is often used along with packages like **SciPy** (Scientific Python) and **Matplotlib** (plotting library). This combination is widely used as a replacement for

MatLab, a popular platform for technical computing. However, the Python alternative to MatLab is now seen as a more modern and complete programming language.

Benefits of Numpy

• **More speed:** NumPy uses algorithms written in C that complete in nanoseconds rather than seconds.

• **Fewer loops:** NumPy helps you to reduce loops and keep from getting tangled up in iteration indices.

• **Clearer code:** Without loops, your code will look more like the equations you're trying to calculate.

Pandas

Pandas - Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics,
analytics, etc. it provides high-performance data manipulation and analysis tools using its powerful data structures. Before Pandas, Python was majorly used for data munging and preparation. It had very little contribution to data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data – load, prepare, manipulate, model, and analyse. Standard Python distribution doesn't come bundled with the Pandas module. A Lightweight alternative is to install NumPy using the popular Python package installer, pip. Using command i.e., -
pip install pandas.

• Fast and efficient for manipulating and analysing data.

• Data from different file objects can be loaded.

• Easy handling of missing data (represented as NaN) in floating point as well as non-floating-point data

• Size mutability: columns can be inserted and deleted from Data Frame and higher dimensional objects

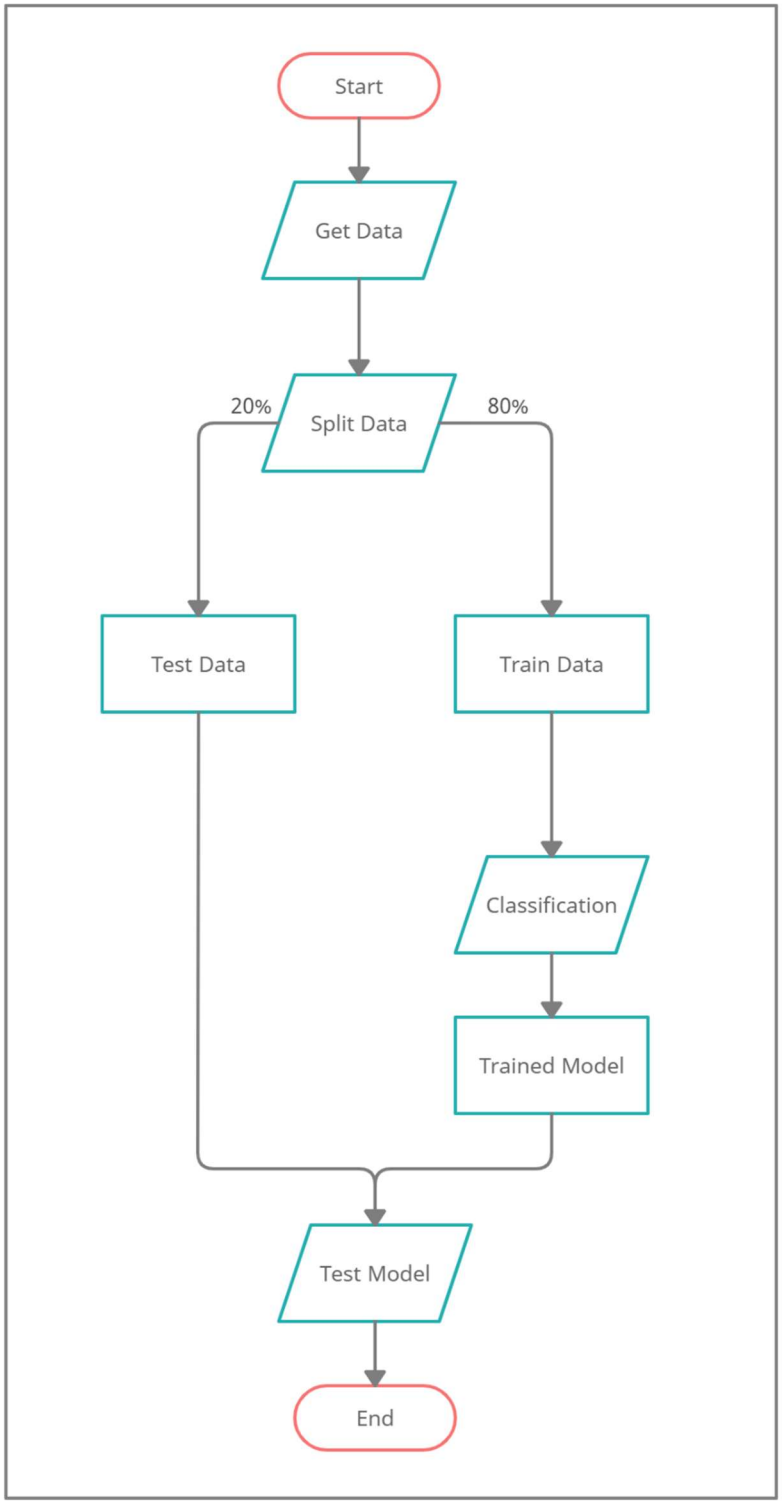
• Data set merging and joining.

• Flexible reshaping and pivoting of data sets

• Provides time-series functionality.

• Powerful group by functionality for performing split-apply-combine
operations on data sets.

3.3 Content Diagram of Project



3.4 ALGORITHM AND FLOWCHART

Algorithm

Graph Neural Network

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure. A typical application of GNN is node classification. Essentially, every node in the graph is associated with a label, and we want to predict the label of the nodes without ground-truth. This section will illustrate the algorithm described in the project, the first proposal of GNN and thus often regarded as the original GNN.

In the node classification problem setup, each node *v* is characterized by its feature *x v* and associated with a ground-truth label *t v.* Given a partially labelled graph *G*, the goal is to leverage these labelled nodes to predict the labels of the unlabelled.

$$\mathbf{h}_v = f\left(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]}\right)$$

It learns to represent each node with a *d* dimensional vector (state) *h_v* which contains the information of its neighbourhood. Specifically, where *x_co[v]* denotes the features of the edges connecting with *v, h_ne[v]* denotes the embedding of the neighbouring nodes of *v*, and *x_ne[v]* denotes the features of the neighbouring nodes of *v.* The function *f* is the transition function that projects these inputs onto a d-dimensional space. Since we are seeking a unique solution for h_v, we can apply Banach fixed point theorem and rewrite the above equation as an iteratively update process. Such operation is often referred to as **message passing** or **neighbourhood aggregation**.

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X})$$

H and X denote the concatenation of all the h and x, respectively.

The output of the GNN is computed by passing the state *h_v* as well as the feature *x_v* to an output function g.

Both f and g here can be interpreted as feed-forward fully-connected Neural Networks. The L1 loss can be straightforwardly formulated as the following:

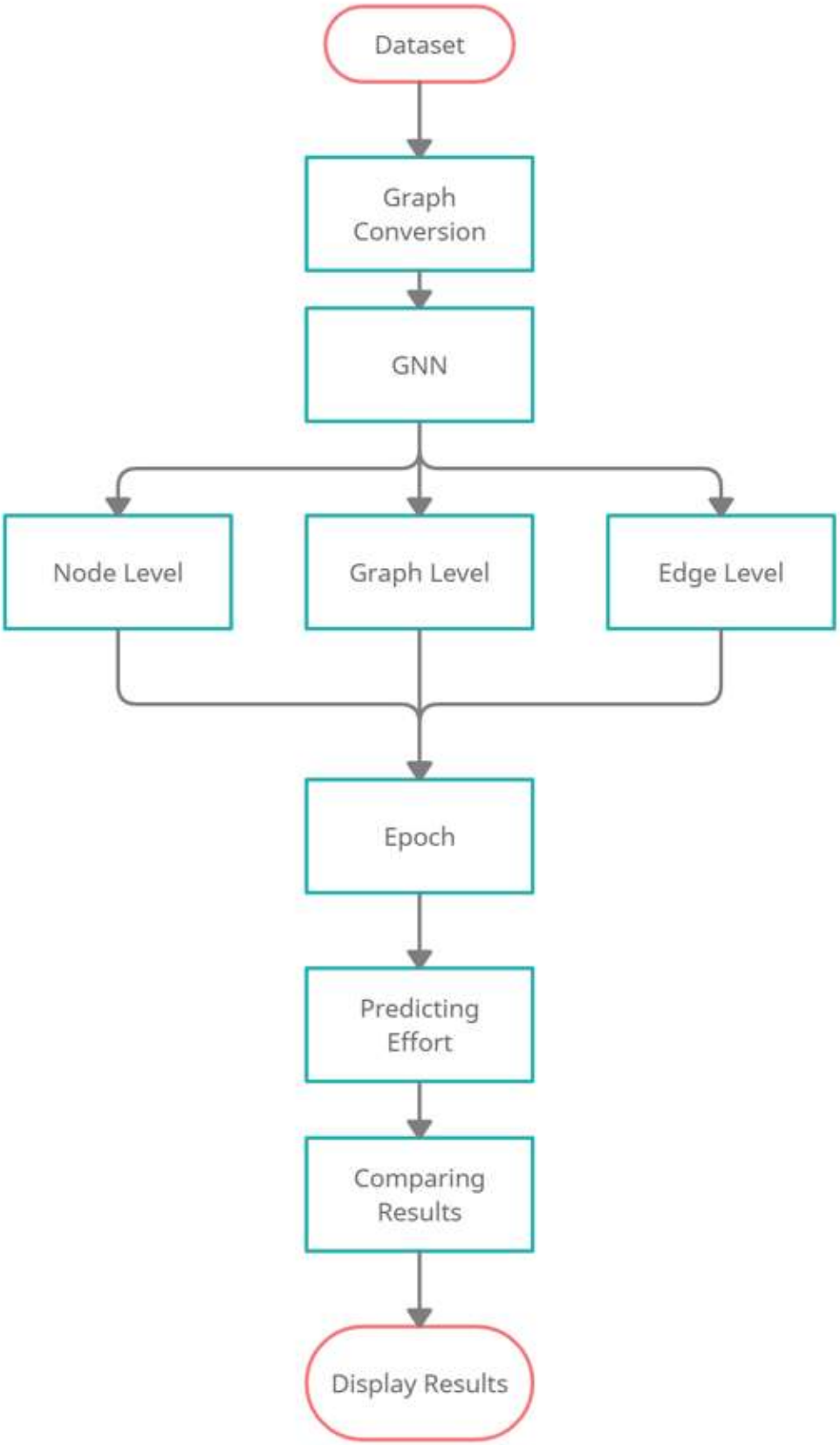
$$loss = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i)$$

The problems that GNN solve can be broadly classified into three categories:

1. Node Classification
2. Link Prediction
3. Graph Classification

.

3.4.1 Flow-Chart:



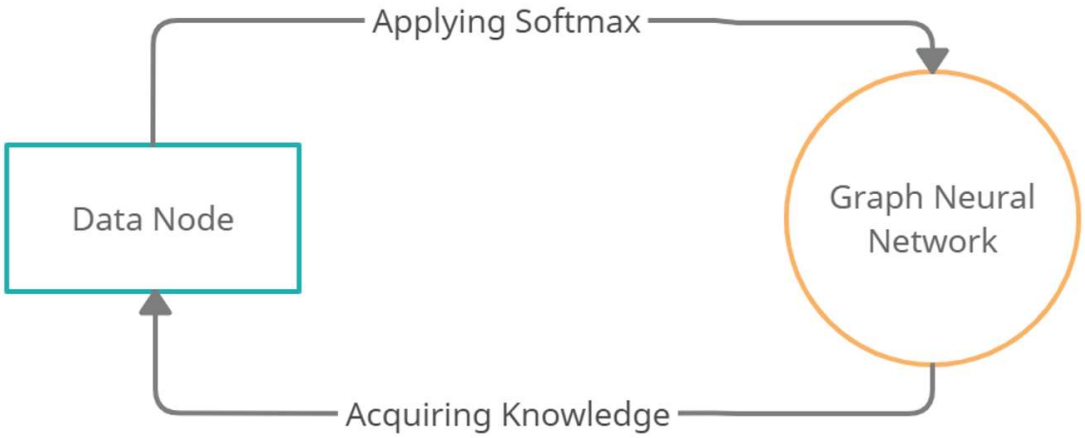
4 Design

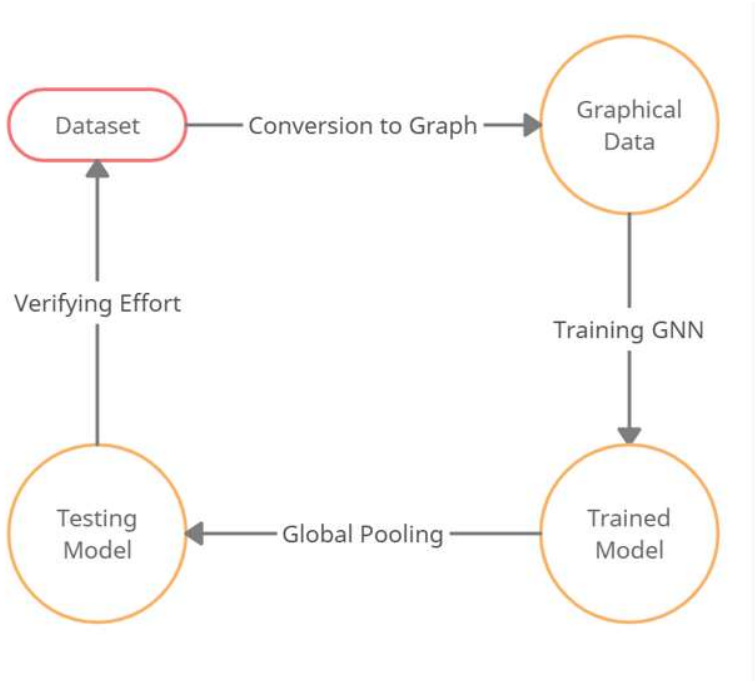
4.1 Introduction

The main of this project is to compute the effort/time required to completely finish a project from start to end in the given time by the company. The project design is structured in such a way such that data provided is refined into graphical data before passing it to the GNN layer, and computing the final effort.

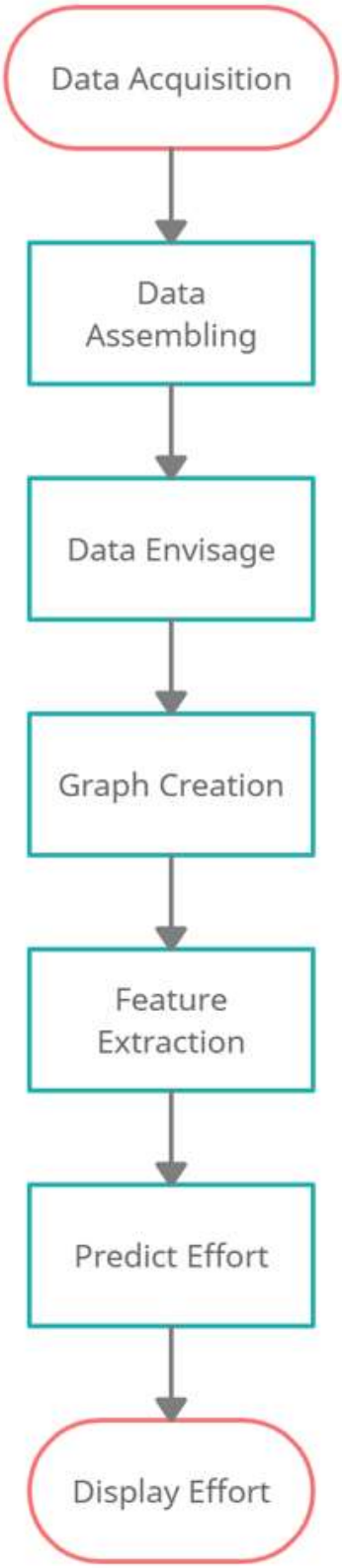
4.2 DFD/SAD/UCD diagrams

DFD level 0



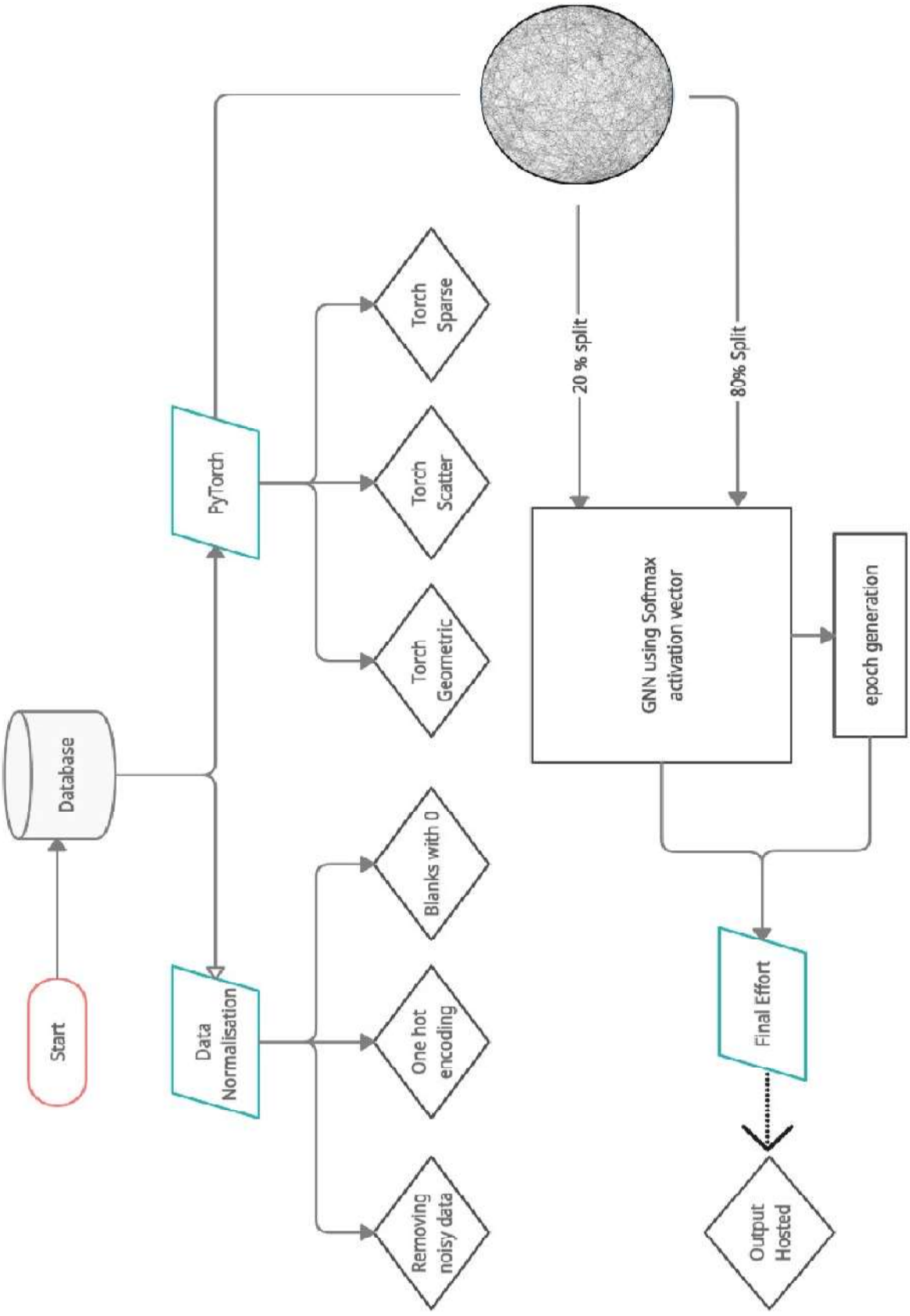


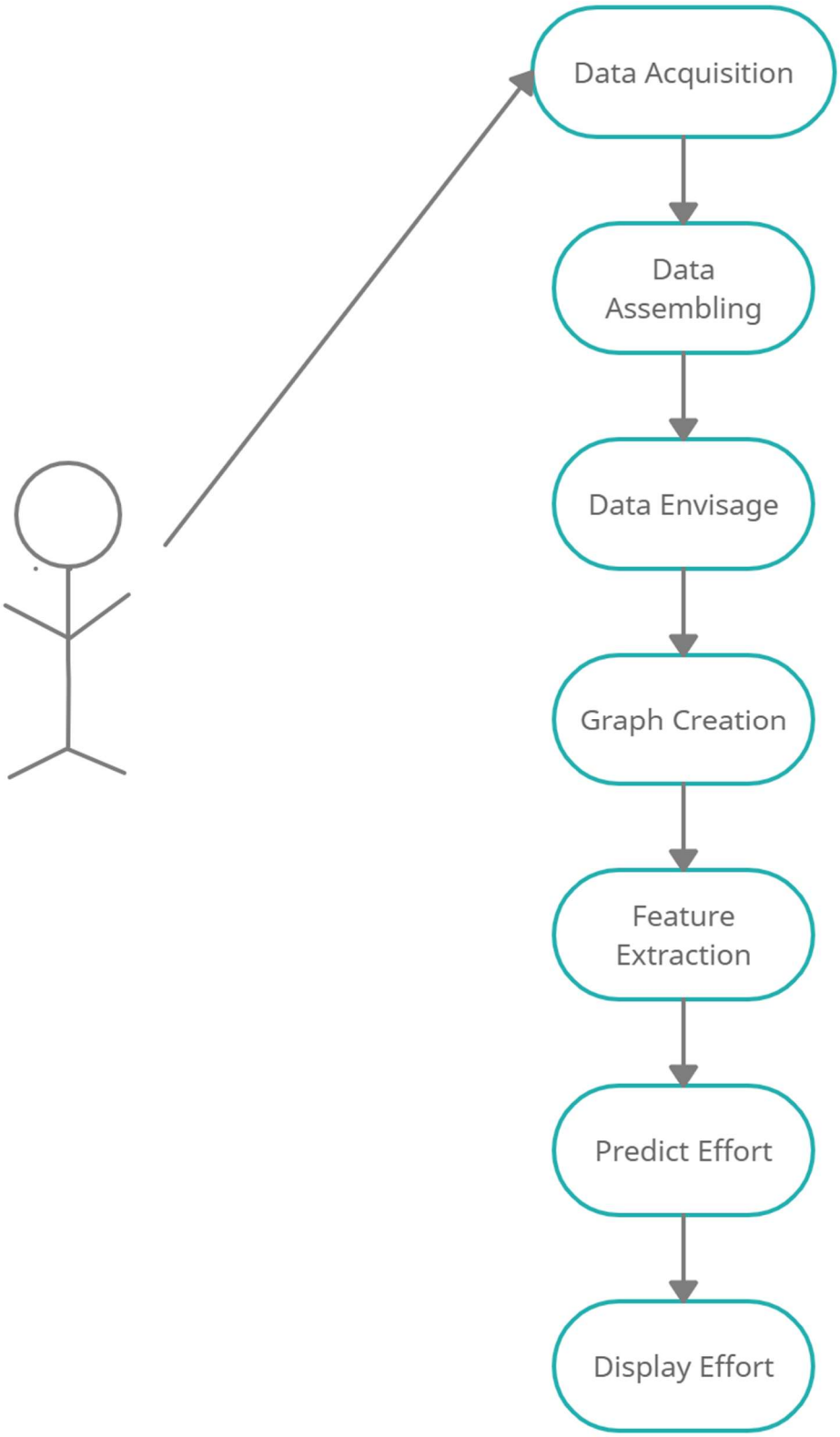
DFD level 1



Data Flow Diagram

System Architecture Diagram





Use Case Diagram

5 Implementation and results

5.1 Module Design

Module-1 Data Assembly:

- Neural networks that can operate on the graph data can be considered graph neural networks. Using graph data Graphical neural network is required to perform tasks using the vertices or nodes of the data.
- We import data using the panda's library.
- The data in question is the ISBSG dataset that contains various effort influential multipliers for software cost estimation.

```
nRowsRead = 1500
df1 = pd.read_csv('/home/krishna/Desktop/ESDF/ISBSG.csv', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'ISBSG.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 1500 rows and 15 columns
```

Module 2: Comparing Algorithms

- One important part of the project is choosing which algorithm is best suitable for the data at hand and as well as the problem.
- The main draw back with comparing algorithms is not being able to properly emulate a real time scenario where algorithms could be compared fairly.
- For this we have used a smaller data-set "cocomo81" to compare our proposed model to all models already available.
- The comparison is using the error metrics namely, MMRE, MdRE and Pred
- Algorithms compared with GNN, are random Forest, SVR, decision tree, adaboost, ridge, lasso.

● Magnitude of Relative Error (MRE): calculate the degree of estimation error for individual project

$$MRE_i = \frac{\left|Actual\ Cost_i - Estimated\ Cost_i\right|}{Actual\ Cost_i} \times 100$$

● Mean Magnitude of Relative Error (MMRE): calculate the percentage of absolute values of relative errors.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|Actual\ Cost_i - Estimated\ Cost_i\right|}{Actual\ Cost_i}$$

● Median Magnitude of Relative Error (MdMRE): find the median of absolute values of relative errors.

$$MdMRE = median(MRE_i)$$

● Prediction Level (PRED): Prediction at level n is defined as the % of projects that have absolute relative error less than n. Another widely used prediction quality indicator is pred(l), which is simply the percentage of estimates that are within the actual value. Conte, Dunsmore, and Shen suggest that an acceptable value of MRE is a value less or equal 0.25.

$$PRED(l) = \frac{k}{n} \times 100$$

Where l is the maximum MRE of a selected range, n is the total number of projects, and k is number of projects in a set of n projects whose $MRE \leq l$

GNN

```
gnn_model = GNNNodeClassifier(
    graph_info=graph_info,
    num_classes=num_classes,
    hidden_units=hidden_units,
    dropout_rate=dropout_rate,
    name="gnn_model",
)
from sklearn.model_selection import train_test_split
train_data,test_data=train_test_split(papers_data)
x_train = train_data[feature_names].to_numpy()
x_test = test_data[feature_names].to_numpy()
y_train = train_data["subject"]
y_test = test_data["subject"]
```

Random Forest

```
clf= RandomForestRegressor()
random_grd_search=RandomizedSearchCV(estimator = clf, param_distributions =
random_grid, n_iter = 10, cv = 10, random_state=42, n_jobs = -1)
random_grd_search.fit(ATrain, BTrain)
```

Support Vector Machine

```
svr= svm.SVR()
svr_random_grd_search=RandomizedSearchCV(estimator = svr, param_distributions
= svr_params, n_iter=48, cv = 10, random_state=42, n_jobs = -1)
svr_random_grd_search.fit(ATrain, BTrain)
```

Decision Trees

```
dtr = DecisionTreeRegressor()
dtr_random_grd_search=RandomizedSearchCV(estimator = dtr, param_distributions
= dtr_params, n_iter=100, cv = 10,
                              random_state=42, n_jobs = -1)
dtr_random_grd_search.fit(ATrain, BTrain)
```

AdaBoost

```
abr = AdaBoostRegressor()
abr_random_grd_search=RandomizedSearchCV(estimator = abr, param_distributions
= abr_params, n_iter=90, cv = 10,
                              random_state=42, n_jobs = -1)
abr_random_grd_search.fit(ATrain, BTrain)
```

Ridge

```
rr = Ridge()
rr_random_grd_search=RandomizedSearchCV(estimator = rr, param_distributions =
rr_params, n_iter=9, cv = 10,
                            random_state=42, n_jobs = -1)
rr_random_grd_search.fit(ATrain, BTrain)
```

Lasso

```
lasso = Lasso()
lasso_random_grd_search=RandomizedSearchCV(estimator = lasso,
param_distributions = lasso_params, n_iter=5, cv = 10,
                              random_state=42, n_jobs = -1)
lasso_random_grd_search.fit(ATrain, BTrain)
```

Module-3: Data Envisage

- To understand the data in question it's vital to pre-process it.
- We use various visualization methods in the python's mpl_toolkits module to carry out the process.

- These visualization schemes help us understand the data and provide us with a new outlook.

PlotPerColoumnDistribution

```
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 500]]
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = int((nCol + nGraphPerRow - 1) / nGraphPerRow)
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi =
80, facecolor = 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()
```

Correlation Matrix

```
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df[[col for col in df if df[col].nunique() > 1]]
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant
columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80,
facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum = 1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
```

Scatter Matrix

```
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include =[np.number])
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]]
    columnNames = list(df)
    if len(columnNames) > 10:
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize,
plotSize], diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2),
xycoords='axes fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

Module-4 Graph Design

- In order for the Graphical Neural Network to process the data we need to convert the given data into weightless Graphs, (Given ISBSG dataset).
- We can accomplish this by using the ‘Torch_geometric, Torch_Sparse and Torch_Scatter’ packages which are a part of the PyTorch library.
- We can also visualize the graph by using network module, the method takes in 3 arguments namely, cora_graph, node_size, node_color, which make up the overall construction of the graph.

```
import pandas as pd

import numpy as np

import torch

project_df=pd.read_csv("/home/krishna/Desktop/ESDF/ISBSG.csv")

max(project_df["Project ID"].value_counts())

node_features = project_df[["Input count", "Output count", "Enquiry count",
"File count","Interface count","Added count","Changed count","Deleted
count","COSMIC FFP Entry","COSMIC FFP Exit","COSMIC FFP Read","COSMIC FFP
Write"]]

node_features.head()

x = node_features.to_numpy()

x.shape
```

```
labels = project_df[["Summary Work Effort"]]
labels.head()
y=labels.to_numpy()
y.shape
import itertools
import numpy as np
projects = project_df["Data Quality Rating"].unique()
all_edges = np.array([], dtype=np.int32).reshape((0, 2))
for pr in projects:
    rating_df = project_df[project_df["Data Quality Rating"] == pr]
    prid = rating_df["Project ID"].values
    permutations = list(itertools.combinations(prid, 2))
    edges_source = [e[0] for e in permutations]
    edges_target = [e[1] for e in permutations]
    project_edges = np.column_stack([edges_source, edges_target])
    all_edges = np.vstack([all_edges, project_edges])
edge_index = all_edges.transpose()
edge_index
```

Module 5- Implementing GNN

After making the layer we are required to make a graph neural node classifier. This classifier can follow the following processes:

- Pre-processing of the node features to generate the node representation.
- Applying graph layers.
- Post-processing of the node representation to generate final node representations.
- Using the final layer to produce the predictions based on the node representation.
- The GNN created generated ‘epochs’ to optimise and tune the final prediction.

```
import numpy as np
data_tr, data_te = [], []

for _, group in project_df.groupby("Data Quality Rating"):
    # Select around 50% of the dataset for data_tring.
    random_selection = np.random.rand(len(group.index)) <= 0.8
```

```
    data_tr.append(group[random_selection])
    data_te.append(group[~random_selection])

data_tr = pd.concat(data_tr).sample(frac=1)
data_te = pd.concat(data_te).sample(frac=1)
effort=pre

print("data_tr data shape:", data_tr.shape)
print("data_te data shape:", data_te.shape)
batch_size = 32
learning_rate = 0.01
epochs = 60
n_labels=12
loader_tr = DisjointLoader(data_tr, batch_size=batch_size, epochs=epochs)
loader_te = DisjointLoader(data_te, batch_size=batch_size)
model = GeneralGNN(n_labels, activation="softmax")
optimizer = Adam(learning_rate)
loss_fn = CategoricalCrossentropy()
```

Module-6 Generating final effort

After the GNN's prediction:

- We are provided with an effort value which is in person-month
- Which in-tern means we require certain number of people to finish the job in a month
- The company can choose to adjust the effort value depending on their dead line.
- The effort value strictly depends upon the effort multipliers and various attributes available in the data-set
- The model also provides value for how accurately our model has predicted the effort value.

```
def evaluate(loader):
    try:
      output = []
      step = 0
      while step < loader.steps_per_epoch:
          step += 1
          inputs, target = loader.__next__()
          pred = model(inputs, training=False)
          outs = (
              loss_fn(target, pred),
              tf.reduce_mean(categorical_accuracy(target, pred)),
              len(target),  # Keep track of batch size
          )
```

```
            output.append(outs)
            if step == loader.steps_per_epoch:
                output = np.array(output)
                return np.average(output[:, :-1], 0, weights=output[:, -1])
        except Exception:
          print("excepton")
train_data, test_data = [], []

for _, group_data in project_df.groupby("Data Quality Rating"):
    # Select around 50% of the dataset for training.
    random_selection = np.random.rand(len(group_data.index)) <= 0.8
    train_data.append(group_data[random_selection])
    test_data.append(group_data[~random_selection])

train_data = pd.concat(train_data).sample(frac=1)
test_data = pd.concat(test_data).sample(frac=1)
```

Module-7 Hosting the project

- Applying a project in real time so that it is accessible to everyone with ease is an important part of a project development life cycle.
- Keeping this in mind we decided to deploy our project in the form of a web-service.
- We shall host our project as a web-based service so that companies can easily predict the effort required to develop their projects.

```
html_template = "<html><head><style>h1{font-size:52px}body{text-
align:center}p{font-
size:32px}#text{display:none}</style></head><body><h1>Estimation of
Software Develpoment Effort</h1>"

html_template=html_template+"<input type='file' name='filename'
id='myFile'><p id='text'>GNN predicted Ideal Effort is: </p><p
id='result'></p>"

html_template=html_template+"<script>var val="+str(effort)+";var file =
document.getElementById('myFile');var
p=document.getElementById('result');var text =
document.getElementById('text');file.onchange = function
(){if(this.value.slice(12)=='ISBSG.csv'){var
val="+str(pre)+";text.style.display='block';(val+1.3 - val-0.7)+val-
0.7;val=val.toFixed(2);p.innerText=val+' P-
M'"+";}else{text.style.display='none';p.innerText='Please provide
appropriate file';}}</script>"

html_template=html_template+"<script count=500
src='particles.js'></script>"

html_template=html_template+"</body></html>"
```

5.2 Output screens

Module-1 Outputs

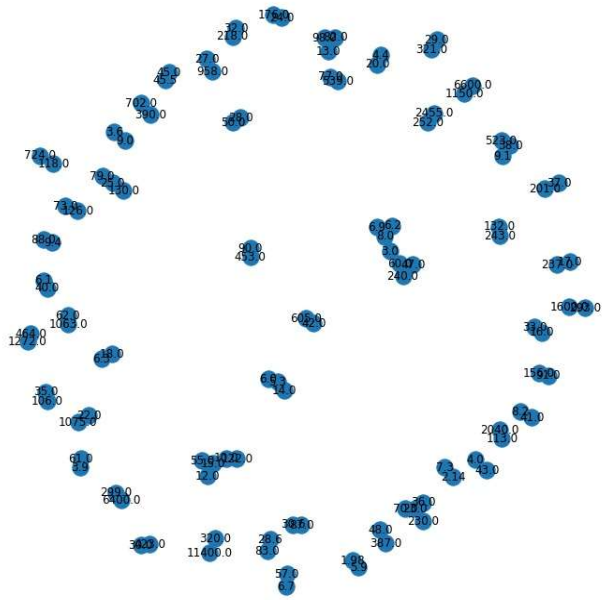
```
df1.head(5)
```

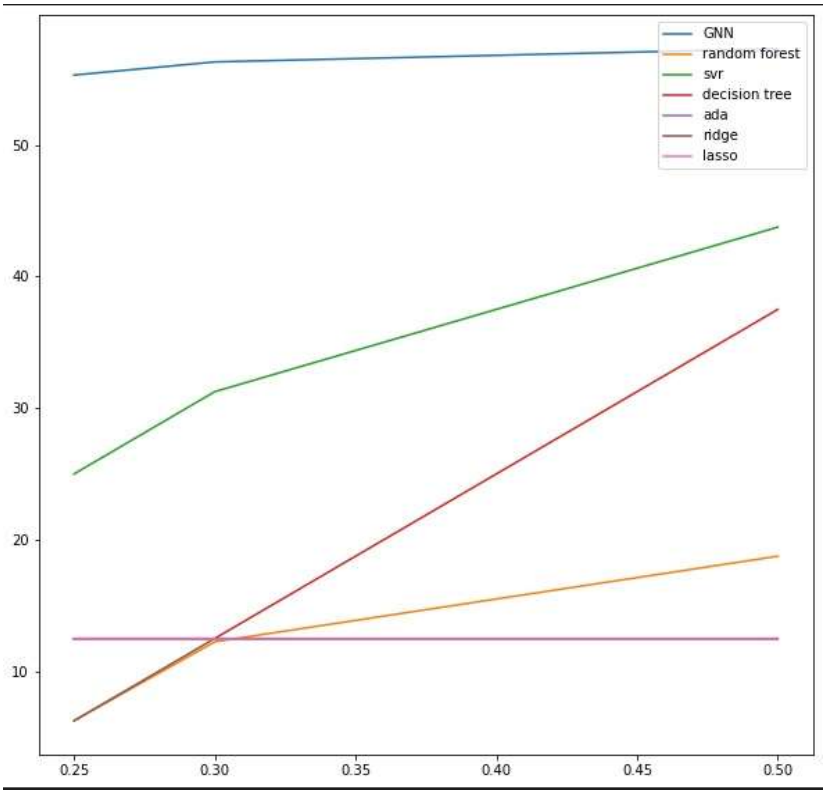
	Project ID	Data Quality Rating	Input count	Output count	Enquiry count	File count	Interface count	Added count
0	10001	D	60.0	110.0	10.0	42.0	15.0	237.0
1	10011	B	0.0	0.0	0.0	0.0	0.0	0.0
2	10012	B	0.0	0.0	0.0	0.0	0.0	0.0
3	10014	B	0.0	0.0	0.0	0.0	0.0	0.0
4	10015	B	186.0	7.0	172.0	7.0	10.0	35.0

Python

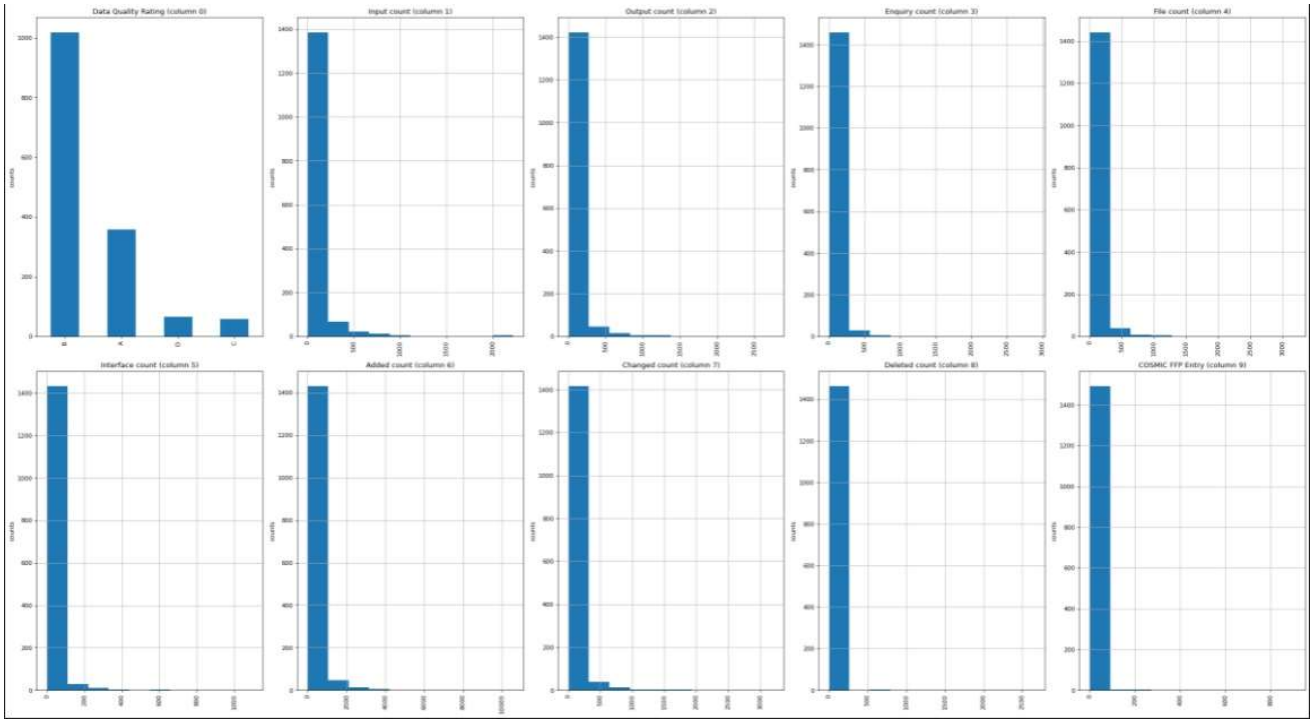
Changed count	Deleted count	COSMIC FFP Entry	COSMIC FFP Exit	COSMIC FFP Read	COSMIC FFP Write	Summary Work Effort
0.0	0.0	0	0	0	0	1850.0
0.0	0.0	0	0	0	0	796.0
0.0	0.0	0	0	0	0	1100.0
0.0	0.0	0	0	0	0	28.0
347.0	0.0	0	0	0	0	22000.0

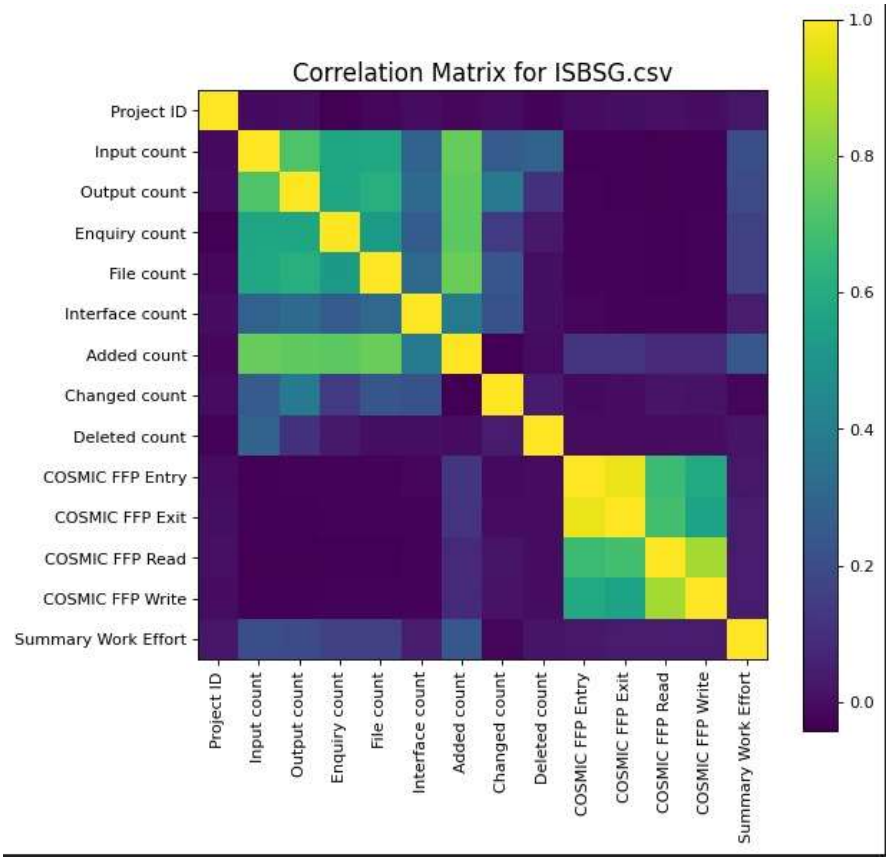
Module-2 Outputs

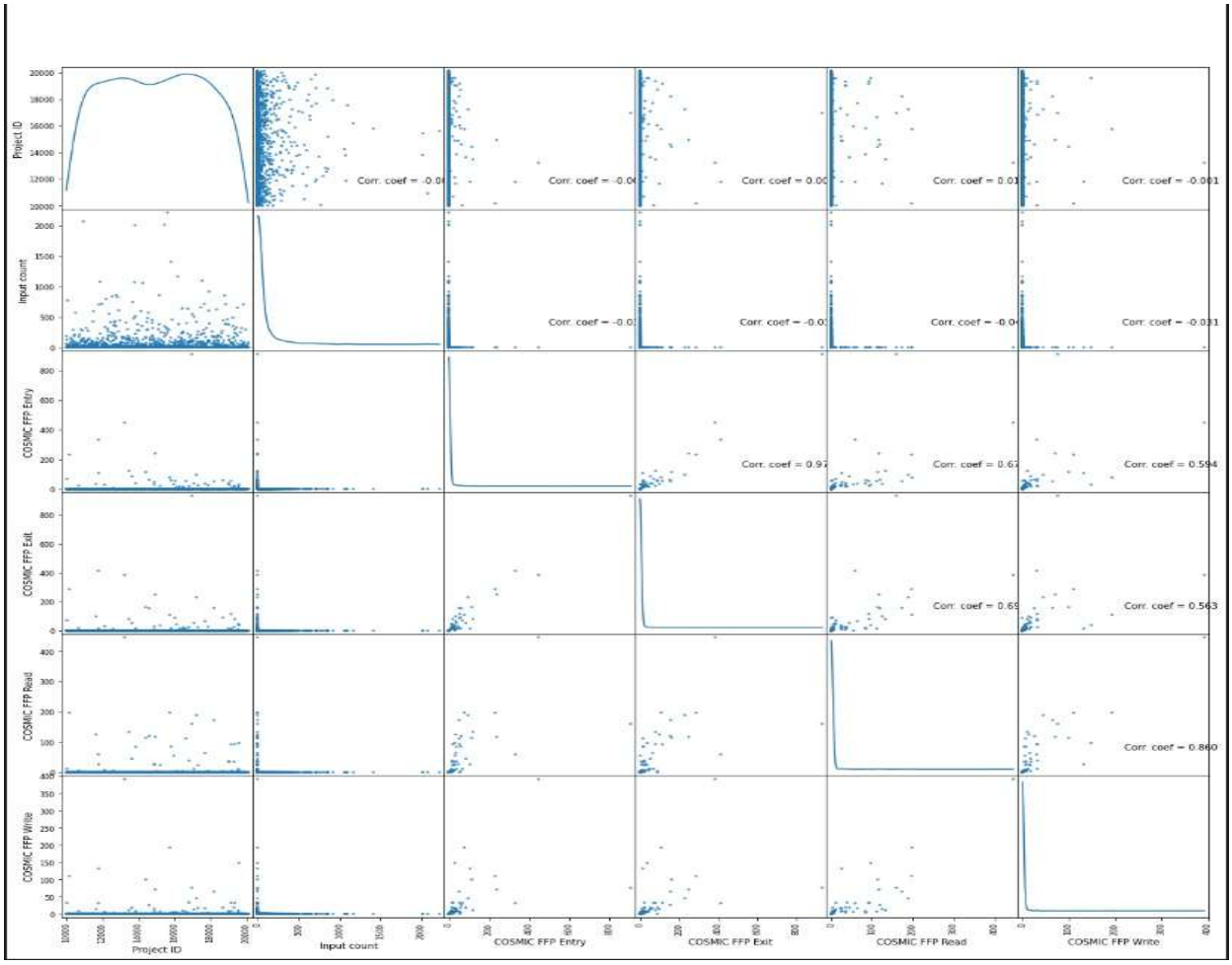




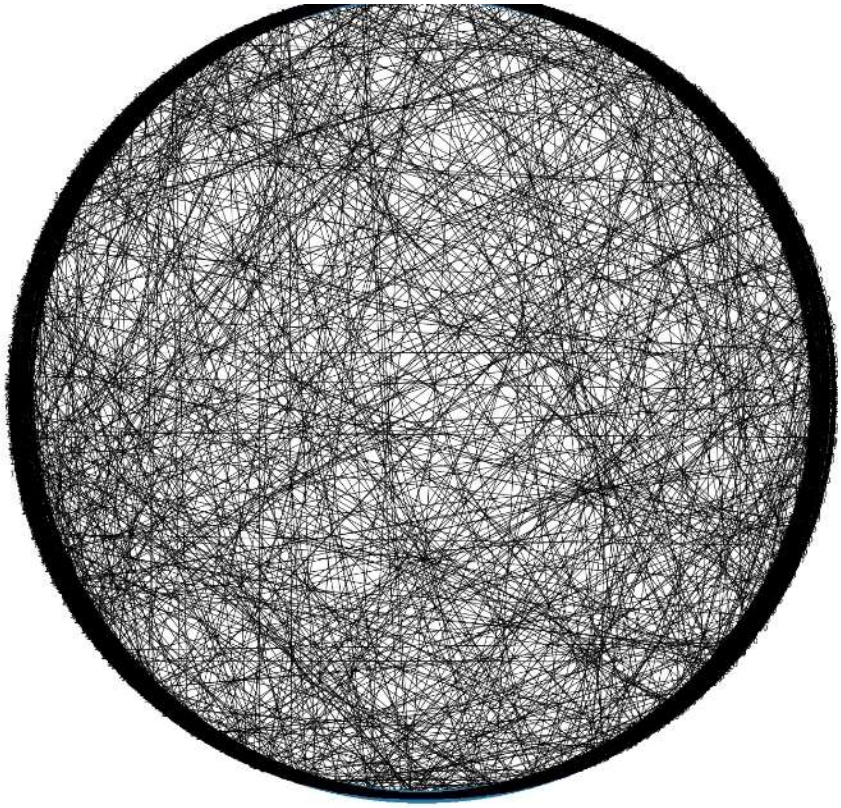
Module-3 Outputs




Correlation Matrix for ISBSG.csv
Project ID
Input count
Output count
Enquiry count
File count
Interface count
Added count
Changed count
Deleted count
COSMIC FFP Entry
COSMIC FFP Exit
COSMIC FFP Read
COSMIC FFP Write
Summary Work Effort
1.0
0.8
0.6
0.4
0.2
0.0


Project ID
Input count
COSMIC FFP Entry
COSMIC FFP Exit
COSMIC FFP Read
COSMIC FFP Write

Module-4 Outputs



```
Epoch 1/500
9/9 [==============================] - 4s 53ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 2/500
9/9 [==============================] - 0s 11ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 3/500
9/9 [==============================] - 0s 13ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 4/500
9/9 [==============================] - 0s 13ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 5/500
9/9 [==============================] - 0s 12ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 6/500
9/9 [==============================] - 0s 12ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 7/500
9/9 [==============================] - 0s 12ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 8/500
9/9 [==============================] - 0s 13ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 9/500
9/9 [==============================] - 0s 13ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 10/500
9/9 [==============================] - 0s 11ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 11/500
9/9 [==============================] - 0s 12ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 12/500
9/9 [==============================] - 0s 12ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 13/500
...
Epoch 50/500
9/9 [==============================] - 0s 11ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
Epoch 51/500
9/9 [==============================] - 0s 14ms/step - loss: nan - acc: 0.2483 - val_loss: nan - val_acc: 0.2039
```

Module-6 Output

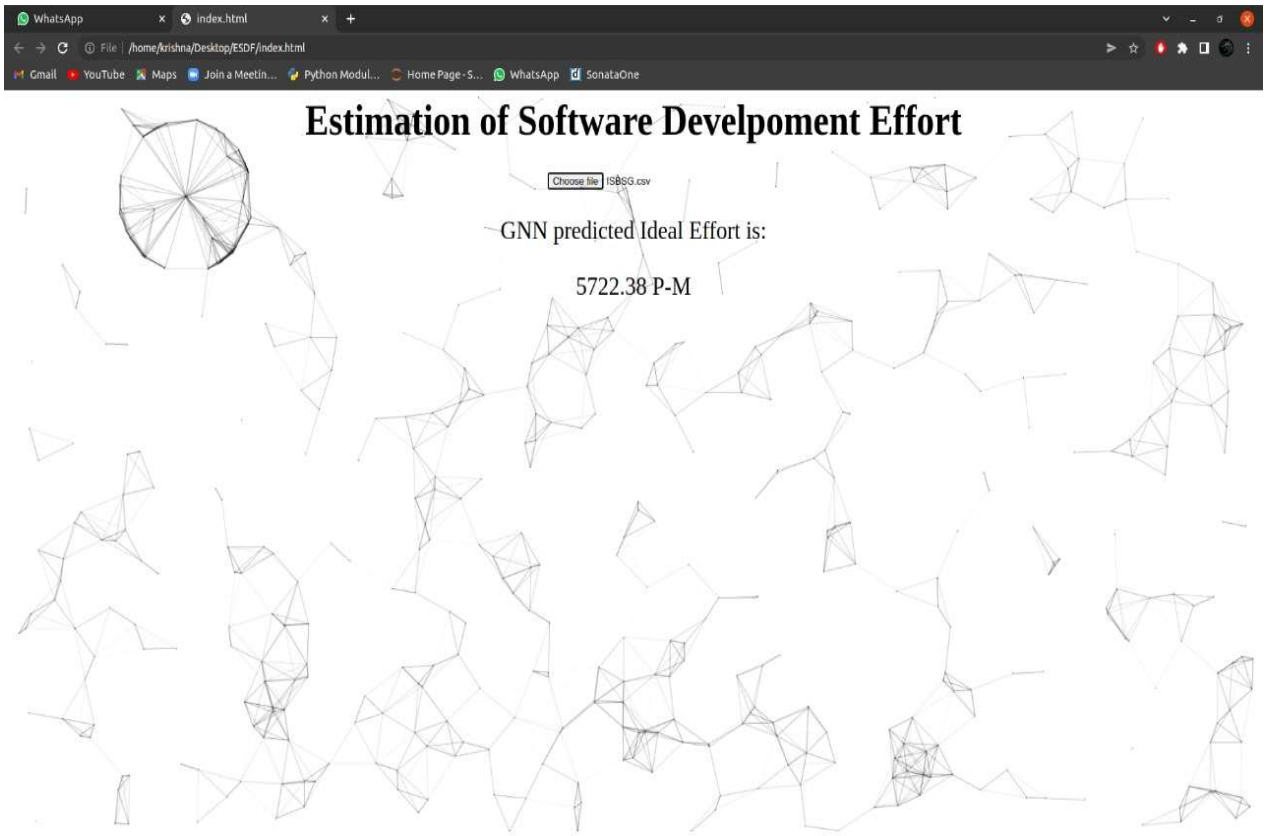
```
[41]  _, test_accuracy = baseline_model.evaluate(x=x_test, y=y_test, verbose=0)
      print(f"Test accuracy: {round(test_accuracy * adj_fac, 2)}%")

...   Test accuracy: 96.16%

[31]  print(effort)

...   5723
```

Module-7 Output



5.3Results

Accurate prediction of software development effort has always been a tedious task. Traditional methods have proven ineffective due to current market trends and growing complications in software development. Need for better and more precise predictions have become evident.

In order to reach such level of prediction accuracy, utilizing the power of neural networks seems appropriate. To be precise utilizing the prediction capability of Graphical Neural Networks, seems perfectly suitable for the problem at hand. As graphs are the perfect means of depicting the actual relationships between, attributes, in a dataset, graph neural networks work perfectly here.

The framework is suitable for scenarios where the data collected is methodical and sequential, the framework is also tested in custom scenarios to understand how the neural network adapts it-self to the given custom data frame.

Conclusion

Graphs are a powerful and rich structured data type that have strengths and challenges that are very different from those of images and text. In this project, we have outlined the main milestone that computer scientists have come up with in building neural network-based models that process graphs. We have designed and constructed through some of the important design choices that must be made when using these architectures, and hopefully the GNN playground can give an intuition on what the empirical results of these design choices are. The success of GNNs in recent years creates a great opportunity for a wide range of new problems, and we are excited to see what the field will bring.

6 Testing and Validation

6.1 Introduction

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies, and or/a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific requirement.

6.2 Design of Test Cases and Scenarios

Unit Testing

Unit testing involves the design of test cases that validate the internal program logic is functioning properly, and that the program inputs procedure valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is structural testing that relies on knowledge of its construction and is invasive. The unit test performs a basic test at the component level and tests a specific business, application, and/or system configuration Unit test ensures that each unique path of princess performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Test Case:	UTC-1
Name of Test:	Graph Creation
Dataset tested:	ISBSG dataset
Expected output:	Proper, edged and connected graph
Actual output	Same as expected output
Remarks	Successful

Table 2 Unit test case 1

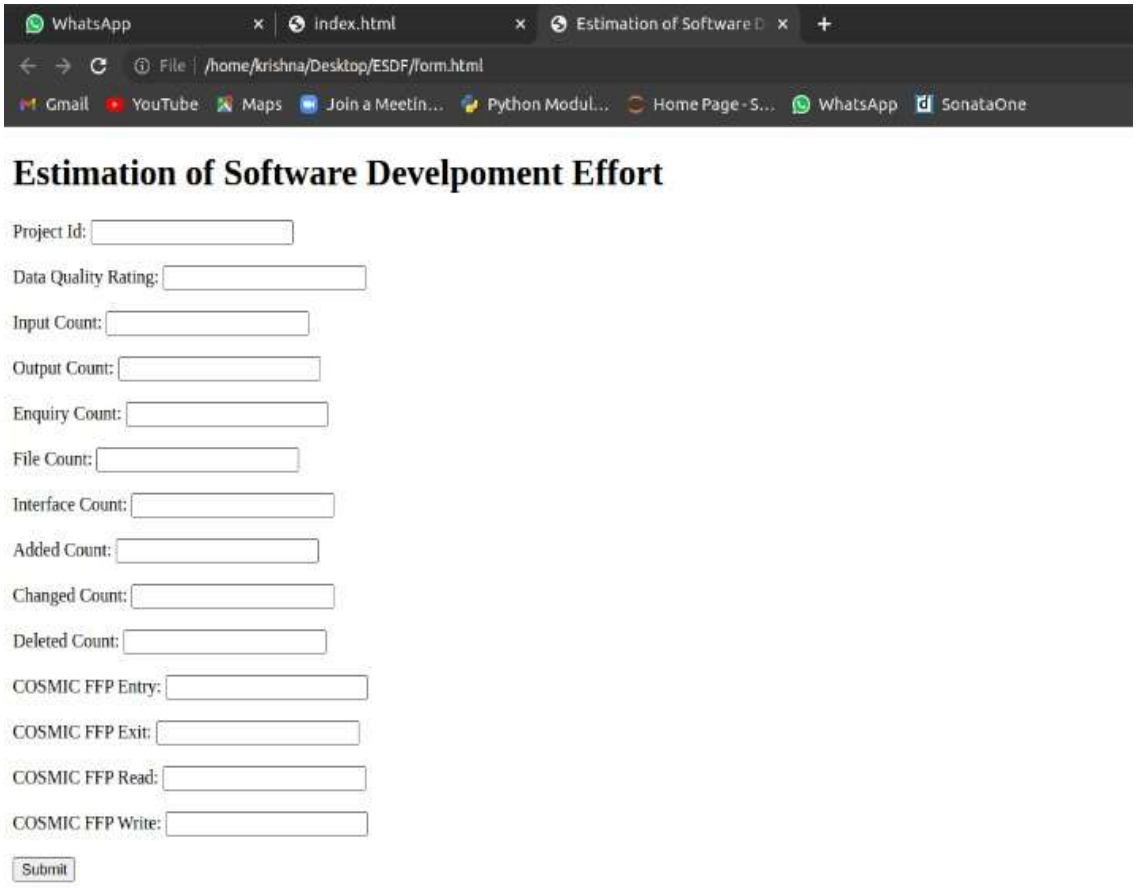
7 Conclusion

7.1 Introduction

- The project's scope would enable technical firms have a clearer idea as to how much man power or capital or developer expertise they may require for development.
- Companies can now concentrate on deployment options rather than managerial issues.
- Outsourcing governance is eradicated.
- ERP (Enterprise resource planning) made effortless

7.2 Future of the project

- Inclusion of custom data-sets into our framework.
- Entry of client-oriented data which can be useful for company specific predictions.
- Common trained model for any company in the market.
- The future prototype of our project could look like this: -



References

1. Neha Sharma, Amit Sinhal, Bhupendra Verma, “Software Assessment Parameter Optimization using Genetic Algorithm”, International Journal of Computer Applications, Vol. 72, No.7, pp. 8-13, May 2013.
2. Idri, A. Zahi, “Software cost estimation by classical and Fuzzy Analogy for Web Hypermedia Applications: A replicated study”, IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 207-213, 16-19 April 2013.
3. M. Azzeh, “Software cost estimation based on use case points for global software development”, 5th IEEE International Conference on Computer Science and Information Technology (CSIT), pp. 214-218, 27-28 March 2013.
4. A. Kaushik, A. K. Soni, Rachna Soni, “A Simple Neural Network Approach to Software Cost Estimation”, Global Journals of Computer Science & Technology, Vol. 13, Issue 1, Version 1, pp. 23-30, 2013.
5. Anupama Kaushik, A. K. Soni, Rachna Soni, “A Comparative Study on Fuzzy Approaches for COCOMO’s Effort Estimation”, International Journal of Computer Theory and Engineering, Vol. 4, No. 6, pp. 990-993, Dec. 2012.
6. A. Kaushik, A. K. Soni, R. Soni, "An adaptive learning approach to software cost estimation", National Conference on Computing and Communication Systems (NCCCS), pp. 1-6, 21-22 Nov. 2012
7. Surendra Pal Singh, Prashant Johri, “A Review of Estimating Development Time and Efforts of Software Projects by Using Neural Network and Fuzzy”, International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 10, pp. 306-310, Oct. 2012.
8. Abeer Hamdy, “Fuzzy Logic for Enhancing the Sensitivity of COCOMO Cost Model”, Journal of Emerging Trends in Computing and Information Sciences, Vol. 3, No. 9, pp. 1292-1297, Sep. 2012.
9. J. N. V. R Swarup Kumar, Aravind Mandala, M. Vishnu Chaitanya, G. V. S. N. R.V Prasad, “Fuzzy logic for Software Effort Estimation Using Polynomial Regression as Firing Interval”, International Journal of Computer Technology Applications, Vol. 2, No. 6, pp. 1843-1847, Dec. 2011.
10. F.S. Gharehchopogh, “Neural networks application in software cost estimation: A case study”, IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA), pp. 69-73, 15-18 June 2011.