Extracting Flowchart Features into a Structured Representation

A Project Report

Submitted by

**Aditya Chakraborti, Aaditya Naik,
Aditya Pansare, Utkarsh Pant**

Under the Guidance of

Dr. Prachi Natu

in partial fulfillment for the award of the degree of

BACHELORS OF TECHNOLOGY
COMPUTER ENGINEERING
At

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT AND ENGINEERING

MARCH 2020

DECLARATION

I, Aditya Chakraborti, Roll No. E003 B.Tech (Computer Engineering), VIII semester understand that plagiarism is defined as anyone or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.

2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)

3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source:IEEE, The institute, Dec. 2004)

4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the Student:

Name: Aditya Chakraborti

Roll No.: E003

Place: Mumbai

Date:

DECLARATION

I, Aaditya Naik, Roll No. E015 B.Tech (Computer Engineering), VIII semester understand that plagiarism is defined as anyone or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.

2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)

3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source:IEEE, The institute, Dec. 2004)

4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the Student:

Name: Aaditya Naik

Roll No.: E015

Place: Mumbai

Date:

DECLARATION

I, Aditya Pansare, Roll No. E019 B.Tech (Computer Engineering), VIII semester understand that plagiarism is defined as anyone or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.

2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)

3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source:IEEE, The institute, Dec. 2004)

4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the Student:

Name: Aditya Pansare

Roll No.: E019

Place: Mumbai

Date:

DECLARATION

I, Utkarsh Pant, Roll No. E020 B.Tech (Computer Engineering), VIII semester understand that plagiarism is defined as anyone or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.

2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)

3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source:IEEE, The institute, Dec. 2004)

4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been clearly identified using quotation marks.

5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the Student:

Name: Utkarsh Pant

Roll No.: E020

Place: Mumbai

Date:

CERTIFICATE

This is to certify that the project entitled "Extracting Flowchart Features into a Structured Representation" is the bonafide work carried out by Aditya Chakraborti (E003), Aaditya Naik (E015), Aditya Pansare (E019) and Utkarsh Pant (E020) of B.Tech (Computer Engineering), MPSTME (NMIMS), Mumbai, during the VIII semester of the academic year 2019-2020, in partial fulfillment of the requirements for the award of the Degree of Bachelors of Technology as per the norms prescribed by NMIMS. The project work has been assessed and found to be satisfactory.

Dr. Prachi Natu

Internal Mentor

Examiner 1

Examiner 2

Dean, MPSTME, NMIMS University

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Codes

Abbreviations

Sr. No.	Full Form	Abbreviation
1.	Graph Model	GM
2.	Flowchart Graph	FG
3.	Optical Character Recognition	OCR
4.	Natural Language Processing	NLP
5.	Long Short Term Memory	LSTM

Abstract

Flowcharts form an essential part of a problem’s implementation, serving as a blueprint for all the steps to be followed, and thus chalking out control flow, information transfer and the processes involved. Most flowcharts, especially ones drawn by hand, are only available as images making them difficult to analyse. Features of these flowcharts, including the individual instructions and control flows can be extracted to create a sizable knowledge base that helps us understand the semantic implication of these steps or processes. The project focuses on extracting the said features using a variety of techniques for each kind of feature. This system uses object detection using deep learning to detect shapes, lines and arrows, whereas uses optical character recognition to detect all the text present in these flowcharts. Later, a Graph Models (GM) is built for each of these nodes and a graphical representation is constructed, mapping all the nodes in the flowchart, followed by constructing a Flowchart Graph (FG), a data structure, which gives the parse trees for each of these nodes, that are connected as per the GM. This FG can be used for multiple applications. One could involve assisting the conversion of flowchart into source code, which could further lead to the development of a mobile application that facilitates efficient code generation where the target user would have to upload only the flowchart image as an input. This comprehensive report explores the particulars of the system through all its literature survey, implementation and further discusses the results and future scope for the same.

Chapter 1

Introduction

1.1. Motivation

The heart of problem-solving lies in thinking effectively. In most college-level programming education, a student is inducted into the world of programming by first educating them on the art of organizing their information, assimilating it and developing a solution to logical problems that are posed to them. As such, they are initially encouraged to forgo efficiency of their solutions, focusing instead on arriving at one using logic and very often, the creativity of thought.

The first step in this journey is to understand the elements that constitute logic in solutions. This is when students are introduced to the most well-known tool to organising and communicating logic - flowcharts. By introducing conventions in symbolism to the process - such as enclosing actions in rectangles, decisions in diamonds and desired states in ovals - students are habituated to a standard way of organising and communicating. This ensures that their solution is expressed clearly and is easily understood by other individuals examining it - without once worrying about syntax and semantics of code, which can vary widely between languages. These differences arise from the fact that while many programming languages may be used to solve a problem, the inherent purpose of each programming language may be different and consequently their syntax and semantics differ too; adding unnecessary complexity to the problem-solving process.

The aim of this project, therefore, is to develop a tool that parses a flowchart by extracting information about its structure, the body of the solution and its logical elements, into a common, structured representation. With an optimistic outlook, further processing of the same in the future allows to generate syntactically and semantically correct programs.

1.2. Defining the Representation

The structured representation is defined as a Graphical Representation of Nodes and their Control Flows, with each node containing a corresponding parse tree. This is referred to as a Flowchart Graph (FG). The FG consists of a set of nodes, each node representing a node (or shape) of the flowchart, and a set of control-flows, which define the edges between the nodes. Each node will furthermore contain the specifics of the node and more importantly, a traversable parse tree corresponding to the text contents of that node.

Note that a Graph Model (GM) is also referred to, which serves as a template for the FG in the sense that the GM does not contain the parse trees or text content for each node.

This representation is aimed at providing the user with a good idea of what the flowchart's control flow looks like while also providing an easy way to analyze and further process the contents of each node of the flowchart by means of the parse trees. Each parse tree can be used to get a good idea of the semantic features of the contents while still remaining relatively general and high-level for the user to leverage it better.

An example flowchart is shown to diagrammatically describe the graphical representation. This is represented in Figure 1.1. Here, the flowchart (left) starts with an ellipse, containing the command "Start" and then it's followed by 3 rectangles, each containing a certain expression, and eventually terminated by a "Stop" ellipse. Thus, an FG is a data structure (right) where each node's parse tree is included and arranged in the same control flow as the original flowchart. This gives the semantic information about each node while maintaining the logical flow of the nodes.

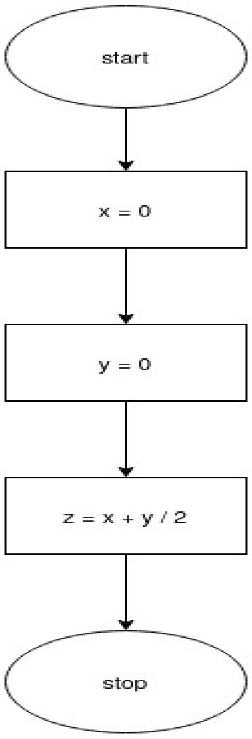


Figure 1.1. An example of a flowchart

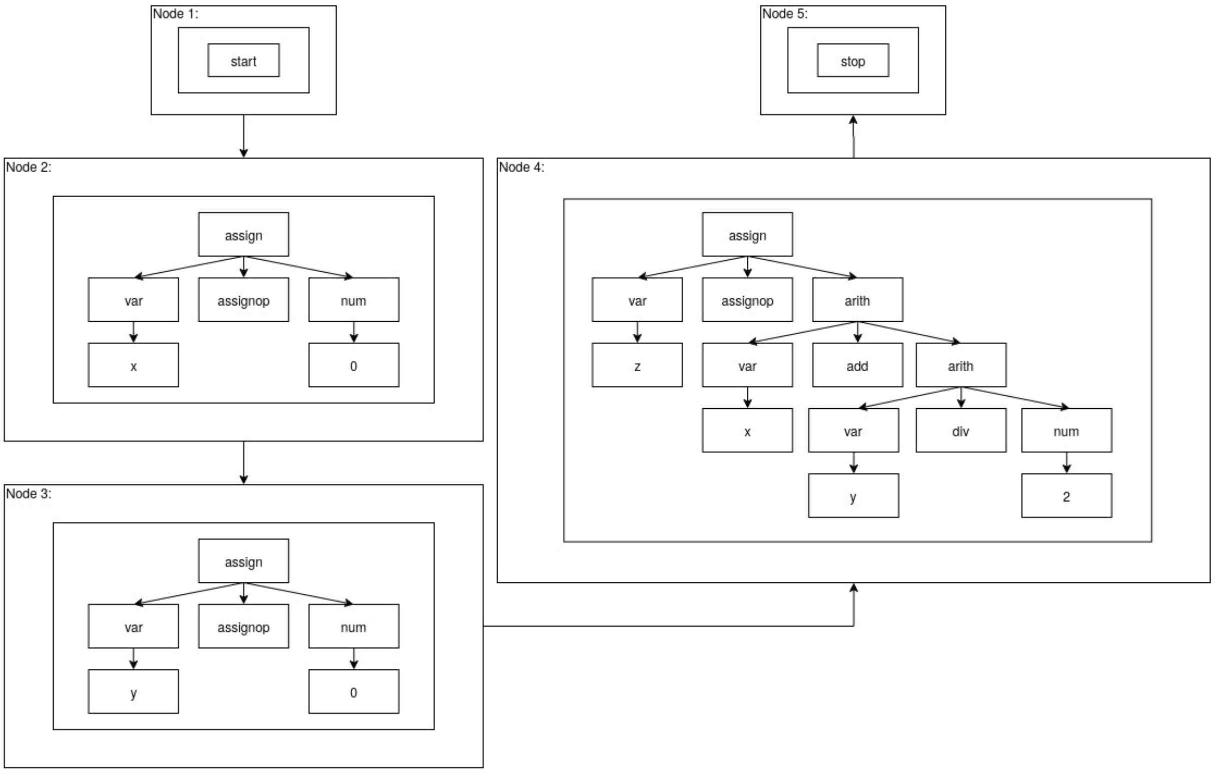


Figure 1.2. The diagrammatic representation of the FG for the flowchart in Figure 1.1

1.3. Real-Life Application

Representing the flowchart in a structured representation has many benefits- it allows for the semantic structure of a flowchart in an image, which is an unstructured form, to be represented in a system which can be easily parsed and interpreted. This structured representation of the flowchart can be used for various purposes, from performing analysis on the flowchart to actually translating the flowchart into a language. We intend to release the system as an open-source library which will allow others to easily work with the intermediate representation of the flowchart and release their own tools to make use of the same.

1.4. Problem Statement

“Accept an image of a flowchart, and extract its features into a common knowledge basis. This image can be processed and represented in a structured graphical representation.”

To be more specific, the system takes an image from the user, processes the image, derives all the semantic features, information and structure, and using all this knowledge, in creating a structured representation of this flowchart.

Chapter 2

Literature Review

The Literature Survey conducted in the duration of the project consisted of papers on various aspects of the project, ranging from validating the need for the project, the components and their architectures to tying all of them together.

2.1. The Importance of Flowcharts

The project started with the most fundamental step of reviewing literature that explored the concept, structure and the notations used in a flowchart. This study helped establish and finalize important guidelines and also in understanding why flowcharts are essential in developing a proper logical flow of one’s implementation plan thus demarcating and stating all the information and control flow within a blueprint.

Hooshyar, D. et al. [1] explored the importance of flowcharts in the learning and problem-solving process. The system described therein interprets the logical meaning of the textual description and attempts to draw its flowchart.

Xinogalos, S. [2] describes the importance of using flowcharts as a tool to think and communicate logic and ideas, along with a description of prevalent methods of generating flowcharts and how certain software workflows incorporate flowcharts into the development process.

2.2. Flowchart Processing Systems

A preliminary survey of the literature was conducted to identify efforts aimed at similar problem statements and to explore existing solutions proposed in current literature.

Supaartagorn, C. [3] delves deep into the implementation of a web-based code generation system by describing 3 types of structures in any flowchart - namely sequences, selections and iterations. In this system, the representation of the flowchart is centered around these structures and is represented as a JSON Object which can later be parsed easily using easily available tools.

Also, Vasudevan, B. G. et al. [4] Introduces methods to extract knowledge from flowcharts by processing specific flowchart features using image processing techniques. In this system, the image is converted into a binary image to aid shape recognition and lines are traced pixel by pixel from the arrowhead. Shape recognition is achieved by extracting shape vectors.

In Hooshyar, D. et al. [5], the proposed system was designed to achieve text-to-flowchart conversion. But naturally, the system employs semantic and syntactic analysis of the text that can be achieved using NLP Modules. Two knowledge bases are applied, namely - FlowchartNet and ActionNet A dialogue-based tutoring system is applied & a crawler is tasked to search the Internet for the new programming problems which are used to update FlowchartNet.

In Chen, Q. et al. [6], methods are proposed with regards to how multiple components of the flowchart and thus the intended system in this project, may tie in together in the future. The system described herein groups and classifies flowchart features and analyses the inferred structure, basis which a graph grammar is formalised.

2.3. Object Detection

Object Detection in the present system refers to the detection of shapes, lines and arrows - elements that form the structure of flowcharts and give it a semblance of logical flow. The aim of the literature survey of Object Detection methods was to explore the various methods of object detection that currently exist and choose the most appropriate method for the proposed system, considering scale, processing capability and other resources unique to the proposed use case.

Miyao, H. et al. [7], for instance is an exploration of a method that performs detection of flowchart components as they are being drawn. Lines and shapes are determined by detecting closing loops and curves. Further, shapes are classified using a Support Vector Machine.

Bresler, M. et al. [8] proposes the process to apply offline detection and processing techniques on images to gain information on the structure of the flowchart. The described system reconstructs strokes on images of flowcharts and then applies online recognition techniques. Strokes are reconstructed using masks to identify the endpoints, midpoints and ambiguous points. Shapes are recognized by determining if strokes form a loop, and further processing to classify the shape.

Jorge, J. et al. [9] Focuses on describing an approach to recognising shapes by analysing their geometric properties and using decision trees to make a classification. Renuse, S. et al. [10] also introduces the use of the C4.5 algorithm in feature detection/classification using decision trees.

Jian-ning, H. et al. [11] Describes invariant moments and how these remain constant regardless of the orientation/scale of the input image. These properties can be used to identify patterns in images which may not all satisfy a fixed orientation/scale constraint.

2.4 Deep Learning Frameworks

The aim of the literature survey on deep learning algorithms was to review those deep learning tools and frameworks that can effectively perform object detection while meeting the requirement for accuracy and performance, in view of the challenges posed by the differences between hand-drawn and computer-generated flowcharts. The below literature was sifted through.

In Becker, G [12] the author investigated approaches to feature detection and classification that relied more heavily than others on Machine Learning. The contrast between rule-based and machine-learning-based approaches to shape detection was introduced. 6 Steps were outlined to shape detection and were used to formalise machine learning based approaches to achieve the same. It was learned that generally, feature extraction processes are “manual” or decided on on a case by case basis, while machine learning can be used to verify the accuracy achieved in these systems. In a similar vein, Phanikrishna, C. et al. [13] introduces a method using Neural Networks to practically identify objects/shapes/patterns in input images.

Redmon, J. et al. [14], Liu, W. et al. [15] and Redmon, J. et al. [16] introduce the workings/architecture of YOLO v1 and v3 as well as the SSD architecture for neural networks, along with a comparison in terms of effectivity of one model over another.

2.5 Character Recognition

In the context of Character Recognition, the aim of the literature survey was to understand the challenges posed by the task of character recognition and to explore how existing solutions tide over them. The goal was ultimately to select the best method for our system considering the criticality of the character recognition task in the system. The following represents a summary of the survey that was undertaken.

Kajale, R. et al. [17] collates the various approaches to Character Recognition using Machine Learning techniques. Various features were trained using classification algorithms. All data contains a label, obtained after various tests. A feature set and a label set is obtained. The objective lies in mapping a particular feature set (attributes) to a label.

In Bartz, C. et al. [18] the authors suggest the use of a Spatial Transformer Network to extract all information from the image at once. This is done in two stages: a) Text Detection and b) Text Recognition. In (a), a function is computed by a localization network that takes the feature map as input, and outputs the parameters θ of the transformation that creates a sampling grid which is used to define which part of the input features should be mapped to the output feature map. In (b), the N different regions which are produced by (a) are processed independently of each other.

In Zhang, H. et al. [19] the authors suggest a segmentation-free OCR system where they formalize the OCR as a sequence to sequence mapping problem, which treats the input word image as a sequence of image frames and treats the output word as a sequence of letters. The recognition model consists of two Long Short-term Memories (LSTMs). The first one is an encoder network to consume the input image frame sequences. The second one is a decoder network to generate output texts. Attention connections are added from the decoder to the encoder.

Chapter 3

Software Requirement Specification

3.1 Analysis

This project is a research focused project, and as such a major aspect of it involved experimenting with various implementations and architectures. Making a prototype to demonstrate the capabilities of the system was the primary focus and the architecture is still in the prototype phase.

That being said, there are two major goals for this system. One is to release a suite of tools, most probably as a Python package, to allow for others to use the system easily, and effectively use the Flowchart Graph as they feel fit. Since the output is a structured representation of the flowchart, one such application could be to assist in conversion of the flowchart into into source code which could then be executed and this could potentially lead to development of an end-to-end system that could be used on a mobile device which makes it easier to be accessed by the target users to develop prototypes.

Considering these long-term goals, the general architecture is planned as follows. The system is divided into 4 phases, which are detailed later. These phases operate like a manufacturing line would - sequentially and dependent on the previous module for optimum functioning. The general architecture is described later in this literature. For the purposes of this project and to match closely to the straightforward architecture, the Prototyping Model was chosen to be the SDLC (Software Development Life Cycle). A class diagram (Figure 3.1) and a use-case diagram (Figure 3.2) was developed to support the analysis of the system.

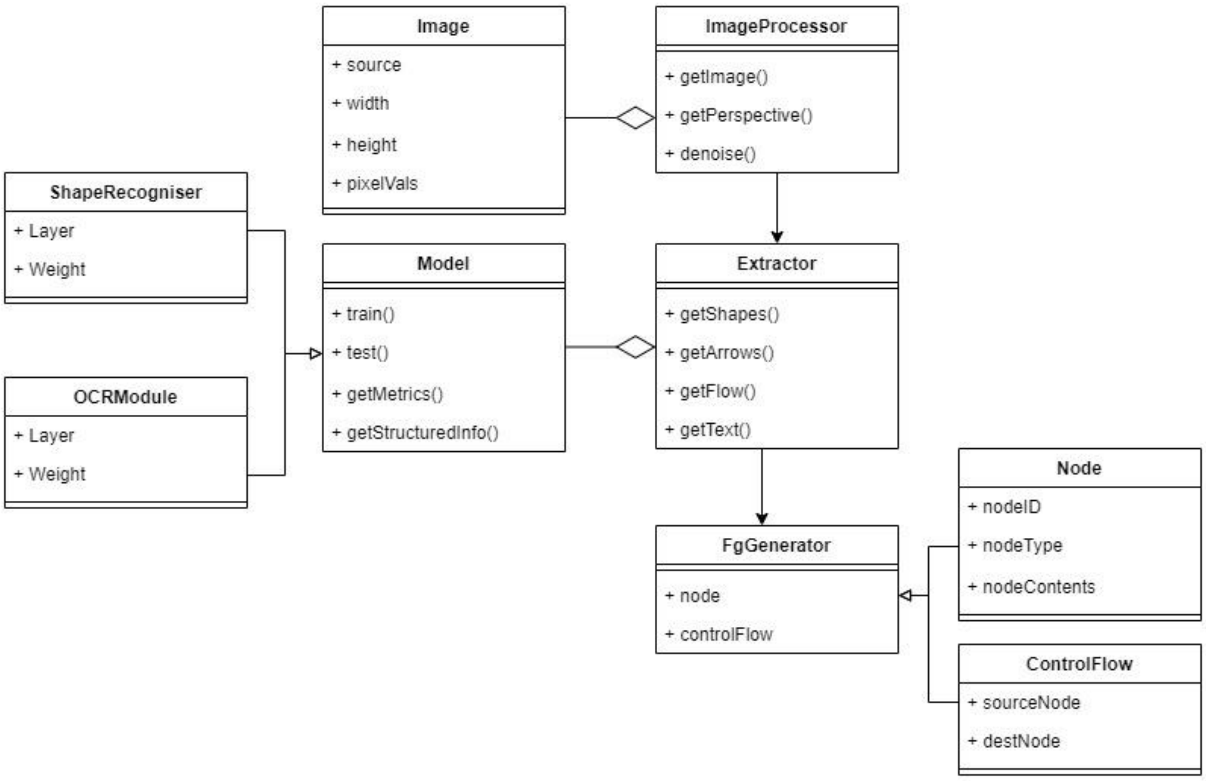


Figure 3.1. The Class Diagram for the proposed system

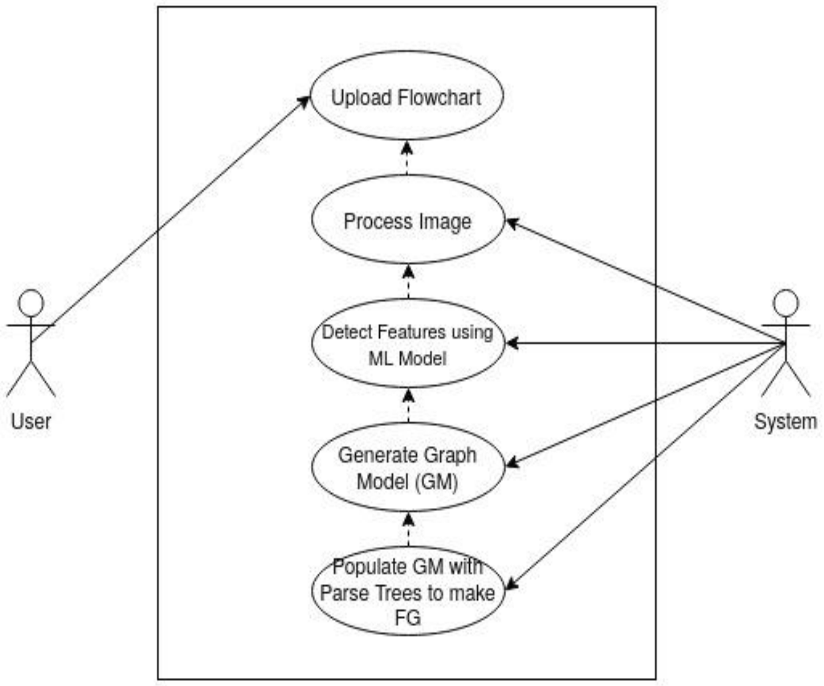

Upload Flowchart
Process Image
Detect Features using ML Model
Generate Graph Model (GM)
Populate GM with Parse Trees to make FG
User
System

Figure 3.2. The Use Case Diagram for the proposed system

This choice was made due to several reasons, primarily that this system is to be a proof-of-concept prototype who's main purpose was, as mentioned earlier, to demonstrate the capabilities of the novel approach to analyzing flowcharts. There was a good idea of the design and the requirements, both functional and non-functional, but several revisions were expected to enhance the performance and the accuracy of the output. There are many more revisions expected to the implementation before the end-to-end implementation is finalized in accordance to the set goals. This makes the Prototype Model a good choice for the system.

3.2. Requirements

3.2.1. Functional Requirements

The functional requirements of a software system, by convention, are defined as processes or tasks that the system can perform. The current system is an amalgamation of multiple subsystems that must work in conjunction to ingest some input, process it and generate the requisite output - thus, the following non-functional requirements may be defined for it:

1. **Detect shapes, lines and arrows**
   The primary descriptors of flowcharts are the shapes, the lines and the arrows, since it is these elements that lend meaning and logical flow to the program. Thus, detecting these is the primary requirement from the system and is achieved using Deep Learning models trained specifically for that purpose.

2. **Detect text**
   This is to detect the text contained in an image and perform optical character recognition on it to obtain text that can be read, modified and manipulated. Since the variability and diversity of handwritten text poses a challenge to machine learning models as well, it was decided that Google Cloud Vision's OCR API [20] should be utilised for this purpose, given the extensive computing power and resources that can be availed.

3. **Arrange the shapes in a logical graph**
   The next functional requirement for the system is to tie all of the various modules together into a cohesive unit, which includes ensuring that the outputs of each subsystem are compatible with the inputs of the next subsystem in the logical chain. Doing so

enables the system to effectively detect shapes, detect the text within, map the detected text to the appropriate shapes, then map the flow of control between each part of the flowchart and thus arrive at a logical directed graph-style representation of the flowchart and its features.

4. **Generate Parse Trees**
   Once a logical structure for each flowchart has been identified/constructed, the system must parse the text using a grammar that is predefined as per the needs of the current use-case. In the present system, this is accomplished using the LARK Parser [21] that accepts a grammar to parse a given input against. The Parse tree thus obtained, contains information about the variables, the operations being performed on them, the commands being executed and the flow of logical control in the program.

5. **Allow user to correct errors**
   Every user-facing software system must account for error. More so when the majority of the input to the system is extracted from a single source image and processed within the system. In the current system, errors may range from incorrectly detected text or even kinds of nodes in the flowchart. To allow for correction of the same, the system must accommodate a module that validates the input and allows the user the flexibility to change it if required.

6. **Display the final graph**
   The ultimate requirement from the system is that the system must accurately and intuitively represent the final graph representation of the flowchart (the final output of the system as it were) to the user.

3.2.2. Non-Functional Requirements

The non-functional requirements of a software system can be understood as being the behaviours a system must exhibit in regular operation. The current system is an amalgamation of multiple subsystems that must work in conjunction to ingest some input, process it and generate the requisite output - thus, the following non-functional requirements may be defined for it:

1. **Accuracy**

   The subsystems responsible for detecting shapes, lines, arrows and text must function with an average accuracy greater than 80% (MAP). A lower accuracy may impair the output of the system and cause errors in multiple modules.

2. **Reliability**

   The reliability of the system stems inherently from factors such as the ability of the system to maintain uptime and sustained quality of service throughout such time periods. These are in turn, aspects that depend on the capacity of individual components (whether locally-run or third-party) to operate efficiently. For example, does Google Cloud Vision crash? What quality of service can be expected in regular operations? If the complexity of the input (size or actual complexity of the input flowchart image) increases, the same should not cause resource-intensive modules such as object-detection to crash.

3. **Real-time constraints**

   The system should be architected keeping in mind future applications, which include lighter platforms of execution such as web-based services and mobile operating systems. Therefore, the latency of the system must be the minimum possible.

4. **Memory efficiency**

   Given that multiple subsystems are expected to function together, along with the fact that the inputs and outputs to each of these subsystems is largely self-generated from a single-source-of-truth input image, it can be expected that the system possesses a large memory footprint. Thus, to meet goals such as time-efficiency and even memory-efficiency, a caching-mechanism must be implemented, among other memory-management mechanisms.

5. **Performance efficiency**

   The system's performance benchmark must be constant and reliably maintained for the majority of the period that the system is operational and a degradation in the same is largely unexpected.

6. **Network efficiency**

   Given that the system employs third-party APIs such as Google Cloud Vision, it is imperative that the number of requests made to these services is minimised. This must be done with a view to reduce network traffic (processing a more complex image should not equate to issuing more network-level requests) and also to save on associated costs.

3.3 Construction

This application was a well stitched amalgamation of many elements, and thus resulted in the use of a variety of platforms, languages and external tools, varying from phase to phase. The architecture consists of 4 phases -

1) Detecting Shapes Lines and Arrows
2) Detecting Text
3) Graph Building
4) Extract Parse Trees

For the first phase, which is the detection of lines, arrows and shapes, we used the technique of Object Detection using deep learning. This was followed by the second phase where we detect the text on the flowchart through the Optical Character Recognition process. In the third phase, we integrate all this information and thus create Graph Models (GM) which is a data structure that represents a logical grouping of such data for each node. And later, we move on to the final, fourth phase, we logically arrange these graph models and interpret the data within, like commands, expressions and statements, to create a logical control flow and information flow, represented as Flowchart Graph (FG).

For the first phase, darknet [22] was used to train all our deep learning models, and Linux-based image annotation tool Yolo_mark [23]. For the second phase we used the Google Cloud Vision API [20] for the Optical Character Recognition due to its performance and ease. The third phase uses the Python3.6 programming language and various other Python libraries, and the fourth phase again uses the same language for creating the Flowchart Graph (FG). In general, the whole system is programmed using the Python3.6 language and all the testing and prototyping is performed on the Ubuntu 18.04 Operating System.

3.4 Cost Estimation

The cost for Google Cloud Vision usage is calculated as price per unit, where each unit is a feature applied to one image. For the first 1000 units per month, no charge is incurred, whereas between 1001 to 5,000,000 units per month is charged at $1.50 per 1000 units and beyond that we are charged $0.60 per 1000 units.

In the future, when the system will be deployed on the web as a service, then app engine pricing also comes into play. Beyond the free quotas, which are 28 instance hours for frontend instances and 9 hours for backend instances per day, it would cost $0.058 per hour per instance.

3.5 Extensibility

The proposed system outputs a Flowchart Graph (FG), described in Section 1.3. This FG is a traversable graph with each node containing a corresponding traversable parse tree. This gives a good idea of the semantic properties exhibited by the flowchart. As such, it is extremely easy to traverse and use. This makes extensibility extremely easy, since with little effort, a user can visit each node, access each parse tree and perform the necessary functions.

An important aspect of constructing the parse trees is the grammar used to construct the parse tree. The process is detailed further in the report. We write this grammar in accordance with the guidelines of the LARK parser [21]. Our software is flexible with the grammar, meaning that any grammar compatible with the LARK parser would be successfully parsed by this system. We therefore allow the user to add their custom grammar to a grammar file which we use as the grammar for generating the parse trees. The user can thus use his own grammar in his flowcharts without facing issues from the parser.

Moreover, it is planned to release a Python package which would allow for more modularity and easier extensibility. With such a package, the user would need to simply call a function to traverse the graph and perform the necessary functions on it.

3.6 Maintainability

The entire system is made up of small modules with a very low level of coupling and a moderate amount of cohesion among the modules. This makes the project highly maintainable and also makes it easy to isolate bugs which in turn speeds up the process of patching it.

The OCR module stands out from the rest of the modules in the sense that the maintenance of the module resides with a third party. However, as Cloud Vision [20] is one of Google's premier products and they have put in a lot of resources into developing and maintaining it, it is fair to assume that the OCR module will be maintained for a long term.

3.7 Software Usage

This application can be used in an extremely versatile way and its capabilities can be harnessed by the end user very efficiently. It can be used in detailed analysis of flowcharts since it gives the knowledge about their semantic structure and various other desired data points which can be used by the user. This application thus, can be extended further to make software that uses all this information, and generates a functional code, thus streamlining the process of converting code directly from a flowchart, and truly helping realise implementation from the logic provided. This application can therefore cut down training cost heavily and can help professionals in prototyping and experimentation minimizing the involvement of syntactic knowledge. This can be used by industry professionals, research scholars or even students who can use these features and extend and integrate them however they desire, in order to build mega-systems outside their knowledge domain.

Chapter 4

Analysis and Design

4.1. Working Assumptions

For the purposes of implementing the system, several working assumptions are considered. These assumptions serve as guidelines to how the system is expected to function, what the system can expect as input and what the system must give as an output. These assumptions also outline several necessary restrictions in terms of the expected external parameters.

The system can expect a single image as its input. This image must be an image of a complete flowchart. All the shapes and connecting edges must be clearly visible to the naked eye and not blend into the background. The text in the flowcharts must be legible. No text must lie outside any shapes. Text for any commands apart from START/STOP, loops or branching conditions can be included in rectangles.

The output will be a saved pickle object of the produced FG. This pickled object can be loaded from the output file called graph.pickle and used for further processing. The object loaded from the pickled file will follow the same definition as the FG defined in section 1.3. The same FG will be displayed in a text format once the system has finished processing the flowchart.

The machine where the system will be executed is expected to be running the Ubuntu 18.04 OS with an Nvidia GPU which can support CUDA >= 10.0. The system will be run on a monolithic architecture, meaning that there is no client-server architecture. The machine will therefore be expected to store the trained weights for all Deep Learning Models and run the Deep Learning interface with those weights. The entire operation will be performed on the same system, so it is necessary for the system to be stable and functioning efficiently.

4.2. System Architecture

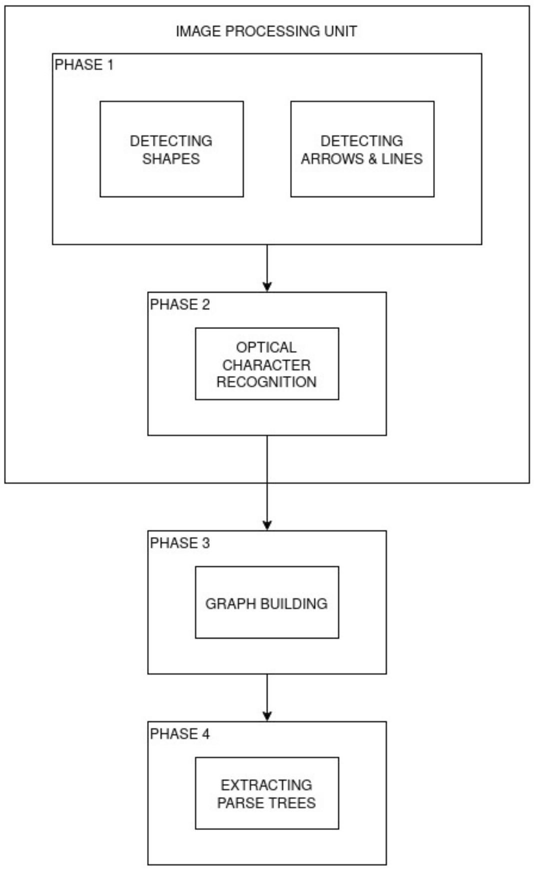


Figure 4.1. The architecture of the proposed system.

As illustrated in Figure 4.1, the system will be built as a sequence of 4 phases. Each phase will be responsible for a separate functionality required by the system as a whole.

The first phase unit will receive an input in the form of an image (.jpg). It performs several steps. First, a 4-point transform is applied to the image to fit the perspective of the image, by scaling and fitting it as per the 4 vertices of the area of interest. Once the image is reduced to the area of interest, Phase 1 identifies two things, which are the shapes that denote the nature of the node in the flowchart, and arrows and lines that denote the flow and interconnectivity of these nodes. To summarize, phase 1 receives input, performs some image preprocessing and then gives the information about the shapes, arrows & lines.

The second phase works on the same input image that's used for Phase 1, but this time it focuses on the actual command, statement or instruction written in the flowchart nodes. It uses Optical Character Recognition to detect the text written in the same flowchart nodes, which then can be used to determine the semantic objective or meaning of the flowchart upon analysis of the text derived.

The third phase collects and analyses all the information received from phases 1 & 2, and generates a Graph Model (GM). This GM is a template for the final FG, and contains a set of empty Nodes and a set of Control Flows. A combination of back-tracing and error-correcting algorithms is used to build the GM.

The fourth phase is the target phase and it is called “Flowchart Graph Generation”. This phase works on the graph model generated in Phase 3 and generates the parse trees for every node in the Graph Model.

4.3. Flowchart Syntax

The syntax of the flowcharts expected by the system is defined in tables 4.1, 4.2 and 4.3. Table 4.1 specifies the different shapes that the system recognizes to be a part of a flowchart. Ellipses with Start/Stop determine the place where the control flow of the flowchart starts or ends. Rectangles are used to depict processes, while diamonds show branching conditions and hexagons demarcate loops. Circles are connectors, which will link different separate parts of the same flowchart together.

Table 4.2 shows the syntax of the statements themselves. These statements may appear inside either rectangles or diamonds. Each token is assigned a meaning and an example of its usage is shown.

Table 4.1. Syntax for shapes.

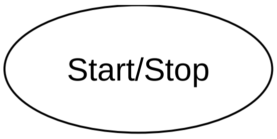
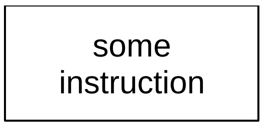
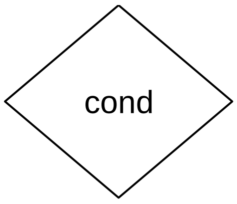
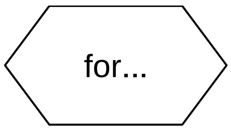
	Begin/End
	Process
	If/else and switch cases (the notations on the arrows can determine the case)
	Loops (for, while, until)

Table 4.2. Syntax for statements.

Token	**Meaning**	**Example**
in	input	in: var1, var2, var3, ...
out	output	out: “the string to print “ + var
=	assignment operator	a <op> b
+, -, *, /, mod	arithmetic operators	
<, >, ==, <=, >=	comparison operators	
and, or	binary operators	
+=, -=, *=, /=, mod=	shorthands	
++, --	unary shorthands	a <op>

Table 4.3 shows the way loops are defined within hexagons. Each loop will have a condition, and depending on the type of loop, the initialization and incrementation may be declared in the same statement itself (in the case of for loops) or may be declared in the loop body (in the case of while loops).

Table 4.3 Syntax for defining loop statements in the flowchart

Loop Name	**Syntax**	**Example**
for	for <init>; <condition>; <incremental statement>	for i = 0; i < 10; i++
while	while <condition>	while i == 0

Chapter 5

Implementation

Figure 5.1 describes the plan of implementation for the whole project. The implementation plan was defined in the phases described in the diagram. Each phase in the diagram represents a major milestone accomplishment, without which we could not proceed to the next phase. The details of the implementation will be discussed further.

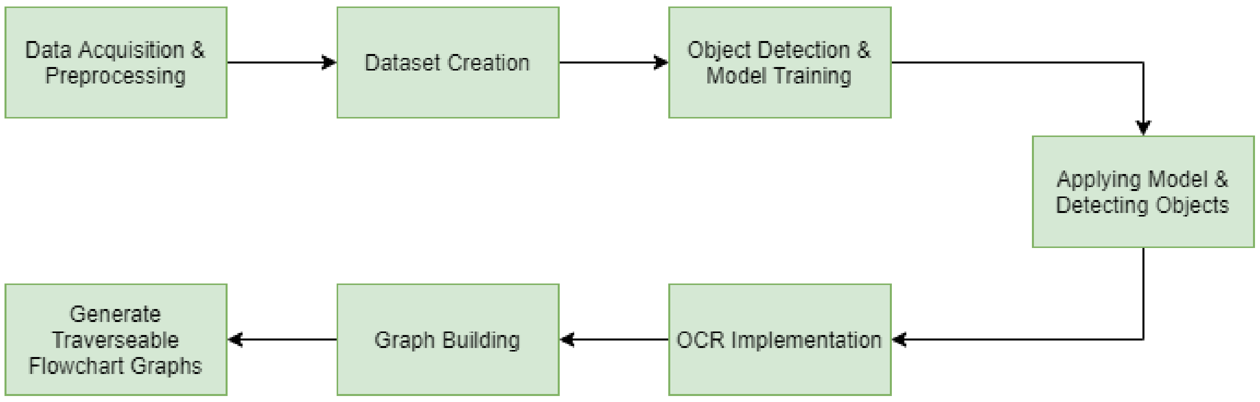

Data Acquisition & Preprocessing
Dataset Creation
Object Detection & Model Training
Applying Model & Detecting Objects
OCR Implementation
Graph Building
Generate Traverseable Flowchart Graphs

Figure 5.1. Planned Implementation Flow

As per the phases, the implementation efforts were divided into the following sections for the sake of discussions-

1. Image Preprocessing, where the images are preprocessed to make it easier for running the object detection models. This is covered in Section 5.1.
2. Collecting Data, where the images are manually annotated to obtain the necessary data for training the models. Section 5.2 details this process and the software used.
3. Object Detection Training (Shapes and Lines/Arrows), where the models are trained using Google Colaboratory [24] on the data from the previous stage. Sections 5.3 and 5.4 discuss more about this.
4. Text Detection, where the text from the entire image is detected and processed. This is discussed in section 5.5.
5. Graph Building, where both the GM and FG are built in accordance with the data from stages 3 and 4. Section 5.6 details this process along with the used algorithms.
6. Error Correction, where any errors are accounted for and corrected by the user. This is shown in Section 5.7.

The final results from the phases and the text format of the final FG produced from a sample flowchart are shown in Section 5.8.

5.1. Image Preprocessing

The image of the flowchart needs to be in a top-down perspective before it is sent for image processing. This is achieved by performing a 4-point perspective transform on an image of the flowchart, as illustrated in Figure 5.2 below. The perspective transform is a mathematical operation which transforms the image with respect to 4 fixed points such that those 4 points define the 4 corners of the new image. In the implementation, the flowchart is required to have a rectangular frame around the flowchart itself. The frame is detected by OpenCV's [25] contour detection algorithm, and the 4 corners of the frame are obtained from the contours. These 4 points are used to perform the 4-point perspective transform. This perspective transform puts the image into a better perspective for both the shape recognition as well as the lines/arrows detection models.

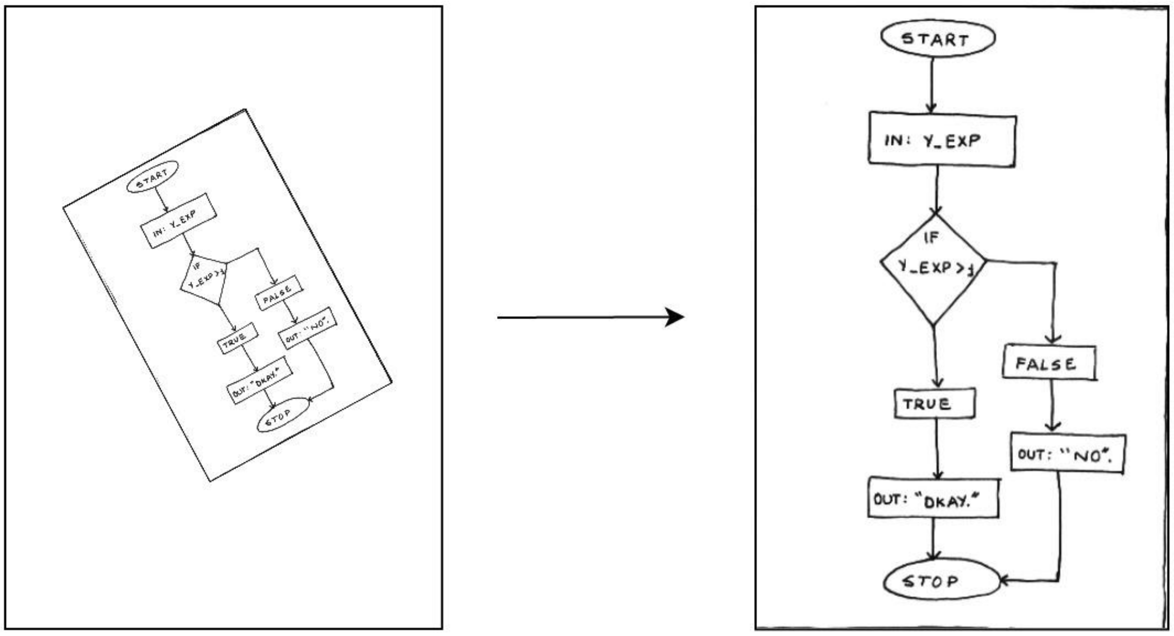
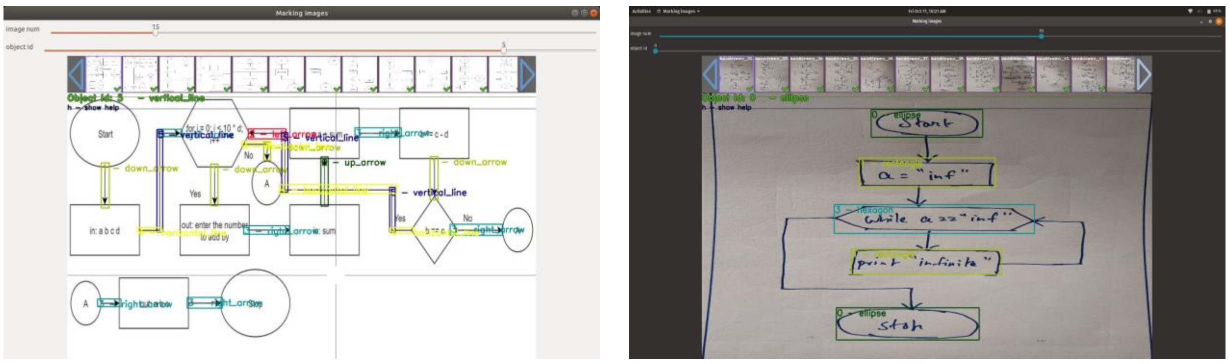


Figure 5.2. Implementation of 4-point Perspective Transform

5.2. Collecting Data

Since the shape recognition and arrow recognition systems work using Deep Learning, a considerable amount of time was spent collecting data. The flowcharts were manually generated, both by computer and hand-drawn, resulting in a collective database of over 161 images of flowcharts, of which 38 were computer generated flowcharts and 123 were hand-drawn. Each of these flowcharts was then annotated using an open source annotation software called Yolo_mark [23]. To annotate the images, Yolo_mark was used to iterate over all the images and manually draw the bounding boxes for each class as shown in Figure 5.3. There are a total of 10 classes considered- ellipse, rectangle, diamond, hexagon, up_arrow, down_arrow, left_arrow, right_arrow, horizontal_line and vertical_line. Table 5.1 describes the total number of each class.



(a) (b)

Figure 5.3. (a) Drawing bounding boxes for lines/arrows recognition (b) Drawing bounding boxes for shape recognition

Table 5.1. Different classes and their total count in the database

Class	Count
ellipse	363
rectangle	497
diamond	118
hexagon	42
up_arrow	83
down_arrow	639
left_arrow	130
right_arrow	203
horizontal_line	321
vertical_line	273

5.3. Shape Recognition

The first step in actual image processing of the flowchart is performing the shape recognition on the transformed flowchart image. This is done with the help of a Deep Neural Network which makes use of Convolutional Neural Networks to detect and classify objects. For this particular application, a model with an existing architecture called Tiny YOLOv3, a smaller version of the YOLOv3 architecture and trained the model using the DarkNet framework. The model was trained to detect and classify four major shapes appearing in our flowcharts- rectangles, diamonds, ellipses and hexagons.

5.4. Lines/Arrows Recognition

After identifying the different shapes in the flowchart, another slightly modified version of YOLOv3 called YOLOv3-SPP, which is the Spatial YOLOv3 model, was used for identifying different arrows and lines. The same images that were used for training the shape detection model were used for training the model for detecting and classifying arrows and lines which appear in the flowchart. The model was trained for 6 classes- up arrow, down arrow, left arrow, right arrow, vertical line and horizontal line.

5.5. Text Detection (Optical Character Recognition)

The next step in the processing of the flowchart is performing text detection on the transformed flowchart image. This is done with the help of Google's Cloud Vision API. Before settling for the cloud vision API, the open-source Tesseract OCR [26] was tested but it proved infeasible as Tesseract OCR performs rather poorly when it comes to handwritten text.

The document_text_detection method was used to perform detection and recognition as input flowcharts to the system can be both computer-aided or handwritten. The post request to Cloud Vision API makes a DOCUMENT_TEXT_DETECTION request and returns a fullTextAnnotation response. This response is converted to JSON by using Google's Protocol Buffer, which formats the hierarchical structure of fullTextAnnotation into the key-pair format of JSON. Although the Cloud Vision API is widely considered to be the best OCR solution available, it is also inaccurate.

5.6. Graph Building

The Graph Building process is where all the detected objects including shapes, lines and arrows as well as the text are put together to form a Graph Model, which will be used to make the final Flowchart Graph (FG).

```
Algorithm build_graph(shapes, lines)
   1. graph <- GraphModel()
   2. graph.nodes <- shapes
   3. graph.connections <- {}
   4. for each node in graph.nodes, do
         a. inedges <- find_inedges(node, lines)
         b. for each inedge in inedges, do
              i.   source_nodes <- trace_back(node, inedge, shapes, lines)
             ii.   for each source_node in source_nodes, append [source_node,
                   node] to graph.connections
   5. return graph
```

Algorithm 5.1. Empty Graph Model Construction

In essence, the general graph building algorithm (as described in Algorithm 5.1) first finds all the shapes, lines and arrows detected in phase 1. Each shape corresponds to a node in the GM. For each node, a list of in-edges (arrows flowing into the node) are determined, and each in-edge is traced back to its source shape. For each such source shape found for the node, a new control flow source_shape -> node is added. This process continues till all the nodes and control flows are accounted for.

```
Algorithm trace_back(node, edge, shapes, edges)
   1. if edge is arrow, then
         a. if source(arrow) in shapes, then return [ source(arrow) ]
         b. else
             i.   prev_edge <- source(arrow)
            ii.   return trace_back(node, prev_edge, shapes, edges)
   2. else
            // edge is a line
         a. if object at opposite end of edge in shapes, then return [ object
            ]
         b. else
             i.   inedges <- find_inedges(edge)
            ii.   src_list = [ ]
           iii.   for inedge in inedges, do
                     1. src_list += trace_back(node, inedge, shapes, edges)
            iv.   return src_list
```

Algorithm 5.2. Tracing Back Edges

Algorithm 5.2 describes the method used to trace back the arrows and lines from the destination node. Simply speaking, for any edge, the source of the edge (the actual definition depends on whether the edge is an arrow or a line) is considered, the same algorithm is recursively called with the source of the edge until a shape is encountered, at which point the shape is returned.

```
Algorithm fill_graph(graph, text)
   1. for each word in text, do
         a. for each node in graph.nodes, do
             i.   if word.bounding_box inside graph.bounding_box, then
                     1. graph.text.append(word)
```

Algorithm 5.3. Populate Graph Model with Node Text

Algorithm 5.3 shows the process of populating each node with the text content detected in phase 2. The text content is mapped to the node inside which its bounding box lies.

```
Algorithm gen_parse_tree(graph, grammar)
   1. for each node in graphs.nodes, do
         a. parse_tree <- parse(node.text, grammar)
         b. node.parse_tree <- simplify(parse_tree)
```

Algorithm 5.4. Generating the Parse Trees for the final FG

Algorithm 5.4 demonstrates the final step of the entire process, which is populating the Graph Model to give the FG. Each node is considered and its text contents parsed by LARK [21], a parser written in Python, to give a parse tree, which is simplified as a nested list representation and stored in the node.

The LARK parser requires a grammar defined for generating the parse tree. In accordance to the guidelines mentioned in Section 4, we design the following grammar-

```
expr : assign
      | io
      | for_loop
      | while_loop
      | bool                        -> bool
      | cond
      | "START"                     -> start
      | "STOP"                      -> stop
      | "start"                     -> start
      | "stop"                      -> stop
      | "Start"                     -> start
      | "Stop"                      -> stop
      | expr "."
      | [ expr ("," expr)+ ]

for_loop : "for" assign ";" cmp ";" assign | "FOR" assign ";" cmp ";"
assign

while_loop : "while" cmp | "WHILE" cmp

cond: "if" cmp | "IF" cmp

bool: "TRUE"                  -> true
      | "true"                      -> true
      | "True"                      -> true
      | "FALSE"                     -> false
      | "false"                     -> false
      | "False"                     -> false

cmp : arith cmpop arith | bool

assign : var assignop arith
```

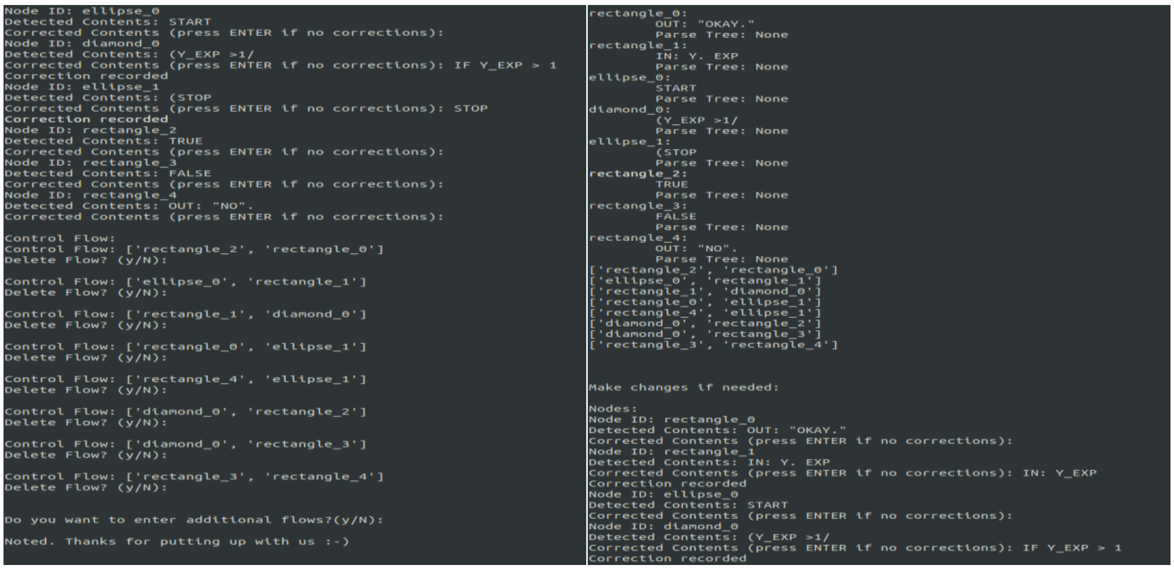
```
      | var shortassignop arith
      | var incr_op

?arith : arith binop arith
      | "(" arith ")"
      | SIGNED_NUMBER               -> num
      | var

io : iofunc ":" [ io_comp ("," io_comp)* ]

?io_comp : ESCAPED_STRING      -> str
      | var
      | SIGNED_NUMBER               -> num

cmpop : "<"                   -> lt
      | ">"                         -> gt
      | "=="                        -> eq
      | ">="                        -> ge
      | "<="                        -> le

assignop : "="

shortassignop : binop"="

binop : "+"                   -> add
      | "-"                         -> min
      | "*"                         -> mul
      | "/"                         -> div
      | "%"                         -> mod

incr_op : "++"                -> add_incr
      | "--"                        -> min_incr

var : CNAME

iofunc : "in"                 -> in
      | "IN"                        -> in
      | "OUT"                       -> out
      | "out"                       -> out

%import common.ESCAPED_STRING
%import common.SIGNED_NUMBER
%import common.CNAME
%import common.WS
%ignore WS
```

Code 5.1. Grammar designed to be used by LARK

Once the text is parsed by LARK in accordance to the grammar in Code 1, the parse tree generated by LARK (which is not easily traversable without designing separate Visitor classes) is simplified into a nested list format for easier usage and traversability.

5.7. Error Corrections

Google Cloud Vision’s OCR is not always accurate, especially for handwriting which may be distorted by noise, lighting or colour differences. Detecting lines and arrows also proves to be difficult. To overcome these issues, the system provides an interactive software which will allow for the user to make any corrections before the final graph is generated and saved. Such an interactive software has two major advantages we intend to leverage in the future. For one, it can easily be made and used, and if made using a good GUI library, the software can be made more appealing. Also, such a software can be easily used to get more data for training, thereby helping our models increase in accuracy.



```
Node ID: ellipse_0
Detected Contents: START
Corrected Contents (press ENTER if no corrections):
Node ID: diamond_0
Detected Contents: (Y_EXP >1/
Corrected Contents (press ENTER if no corrections): IF Y_EXP > 1
Correction recorded
Node ID: ellipse_1
Detected Contents: (STOP
Corrected Contents (press ENTER if no corrections): STOP
Correction recorded
Node ID: rectangle_2
Detected Contents: TRUE
Corrected Contents (press ENTER if no corrections):
Node ID: rectangle_3
Detected Contents: FALSE
Corrected Contents (press ENTER if no corrections):
Node ID: rectangle_4
Detected Contents: OUT: "NO".
Corrected Contents (press ENTER if no corrections):

Control Flow:
Control Flow: ['rectangle_2', 'rectangle_0']
Delete Flow? (y/N):

Control Flow: ['ellipse_0', 'rectangle_1']
Delete Flow? (y/N):

Control Flow: ['rectangle_1', 'diamond_0']
Delete Flow? (y/N):

Control Flow: ['rectangle_0', 'ellipse_1']
Delete Flow? (y/N):

Control Flow: ['rectangle_4', 'ellipse_1']
Delete Flow? (y/N):

Control Flow: ['diamond_0', 'rectangle_2']
Delete Flow? (y/N):

Control Flow: ['diamond_0', 'rectangle_3']
Delete Flow? (y/N):

Control Flow: ['rectangle_3', 'rectangle_4']
Delete Flow? (y/N):


Do you want to enter additional flows?(y/N):

Noted. Thanks for putting up with us :-)
```

```
rectangle_0:
        OUT: "OKAY."
        Parse Tree: None
rectangle_1:
        IN: Y. EXP
        Parse Tree: None
ellipse_0:
        START
        Parse Tree: None
diamond_0:
        (Y_EXP >1/
        Parse Tree: None
ellipse_1:
        (STOP
        Parse Tree: None
rectangle_2:
        TRUE
        Parse Tree: None
rectangle_3:
        FALSE
        Parse Tree: None
rectangle_4:
        OUT: "NO".
        Parse Tree: None
['rectangle_2', 'rectangle_0']
['ellipse_0', 'rectangle_1']
['rectangle_1', 'diamond_0']
['rectangle_0', 'ellipse_1']
['rectangle_4', 'ellipse_1']
['diamond_0', 'rectangle_2']
['diamond_0', 'rectangle_3']
['rectangle_3', 'rectangle_4']


Make changes if needed:

Nodes:
Node ID: rectangle_0
Detected Contents: OUT: "OKAY."
Corrected Contents (press ENTER if no corrections):
Node ID: rectangle_1
Detected Contents: IN: Y. EXP
Corrected Contents (press ENTER if no corrections): IN: Y_EXP
Correction recorded
Node ID: ellipse_0
Detected Contents: START
Corrected Contents (press ENTER if no corrections):
Node ID: diamond_0
Detected Contents: (Y_EXP >1/
Corrected Contents (press ENTER if no corrections): IF Y_EXP > 1
Correction recorded
```

Figure 5.4. An example of the interactive software implemented on the command line

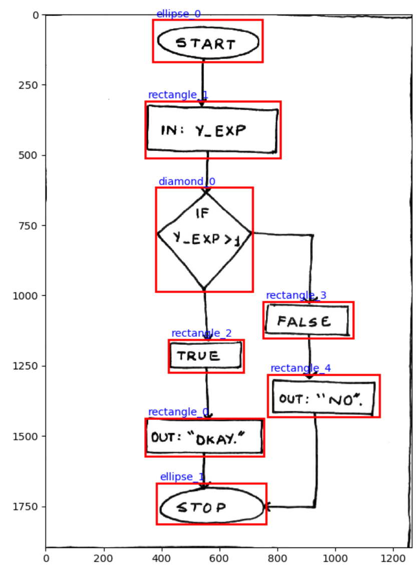


Figure 5.5. A reference to the flowchart’s detected nodes

5.8. Results

For demonstrating our results, the flowchart shown in Figure 5.6 is considered for all the components to effectively show how each phase works on its own and with each other.

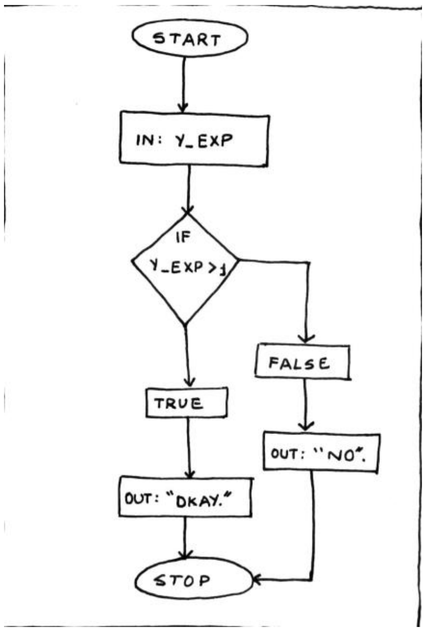


Figure 5.6. The Flowchart used for the following demonstrations

5.8.1. Shape Recognition

The shape recognition model used a Tiny YOLOv3 model with 2 YOLO layers. After training the model on the flowchart images (a 70/30 train to validation ratio was used), the following accuracies were obtained-

```
class_id = 0, name = ellipse, ap = 100.00%        (TP = 119, FP = 4)
class_id = 1, name = rectangle, ap = 99.30%       (TP = 152, FP = 4)
class_id = 2, name = diamond, ap = 100.00%        (TP = 37, FP = 0)
class_id = 3, name = hexagon, ap = 100.00%        (TP = 14, FP = 0)

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.998247, or 99.82 %
```

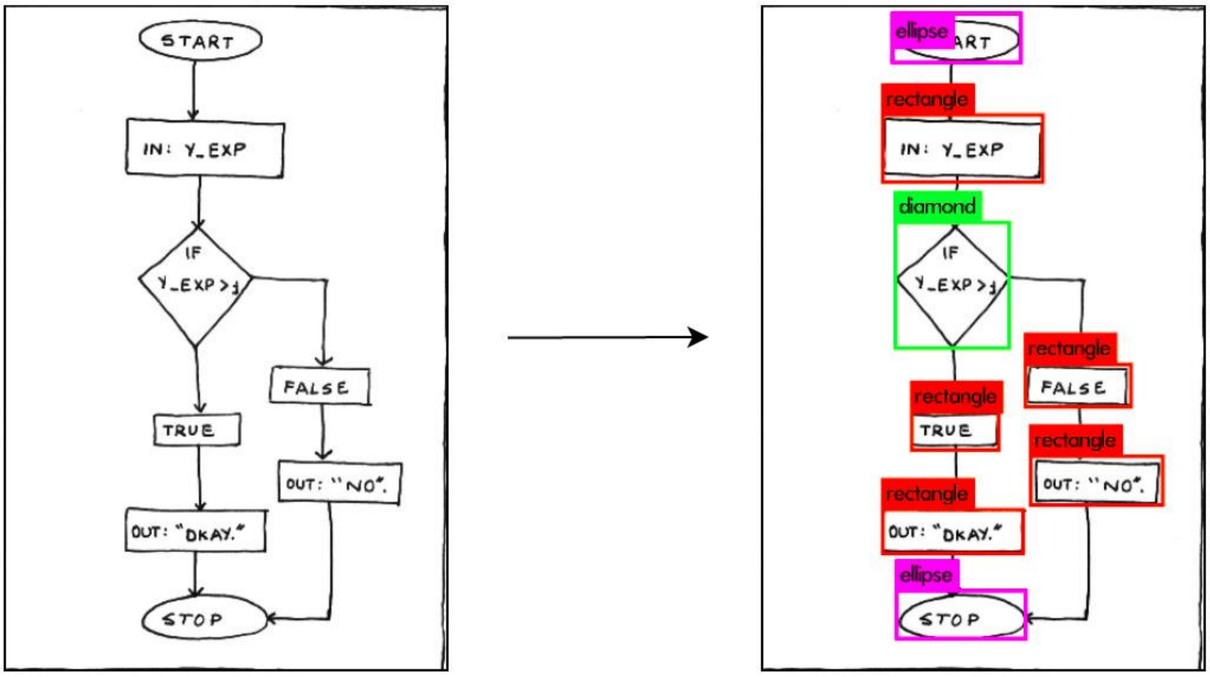


Figure 5.7. Results of the shape recognition module

When the shapes are detected, each shape is added as a graph node in the GM. Each node is assigned a separate ID as we describe in Figure 5.8.

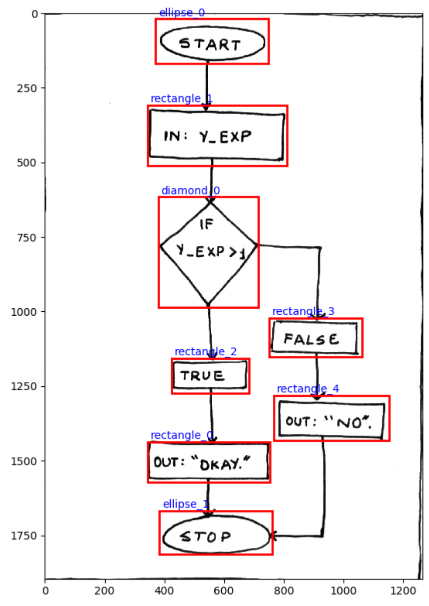


Figure 5.8. The detected shapes and their assigned node IDs

5.8.2. Lines/Arrows Recognition

The lines/arrows recognition model used a modified version of the YOLOv3 model called YOLOv3-SPP. After training the model on the flowchart images (with the same 70/30 train to validation ratio), the following accuracies were obtained-

```
class_id = 0, name = up_arrow, ap = 86.76%          (TP = 20, FP = 1)
class_id = 1, name = down_arrow, ap = 97.28%        (TP = 200, FP = 7)
class_id = 2, name = left_arrow, ap = 92.68%        (TP = 39, FP = 3)
class_id = 3, name = right_arrow, ap = 92.44%       (TP = 57, FP = 7)
class_id = 4, name = horizontal_line, ap = 80.19%    (TP = 81, FP = 12)
class_id = 5, name = vertical_line, ap = 79.47%      (TP = 73, FP = 14)

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
```

`mean average precision (mAP@0.50) = 0.881368, or 88.14 %`

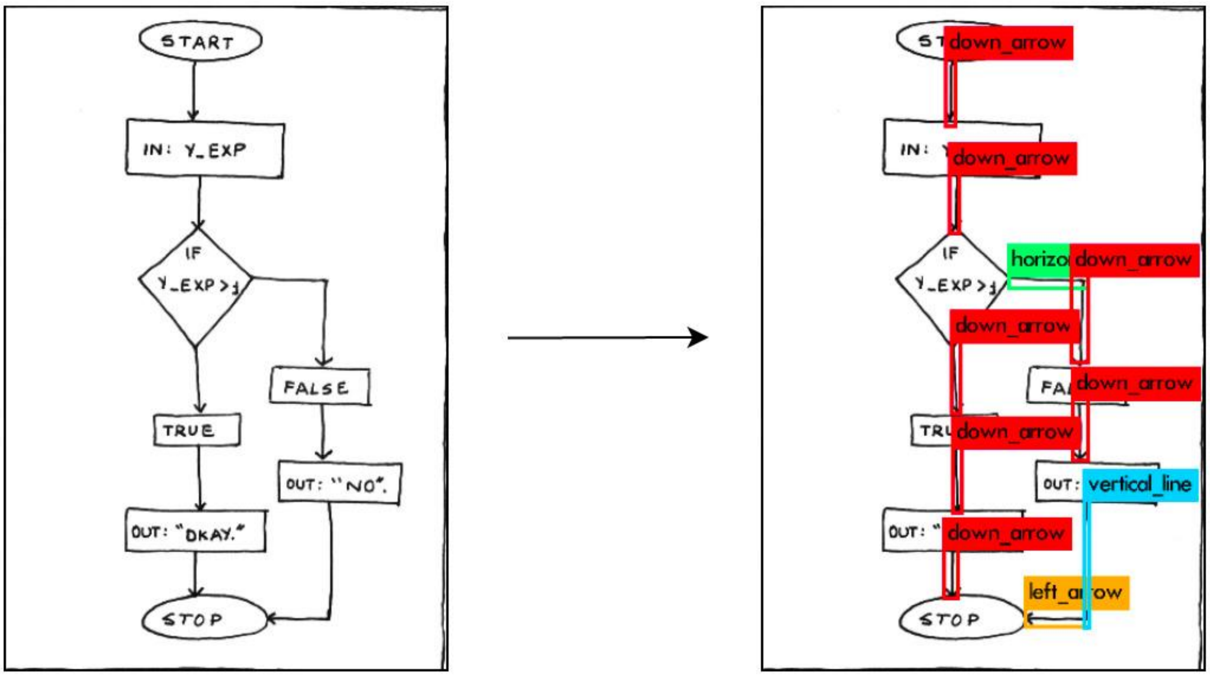


Figure 5.9. Results of the lines/arrows detection module

5.8.3. Text Detection

A snippet of the Google Cloud Vision OCR’s response is shown in Code 5.2.

```
pages {
  property {
    detected_languages {
      language_code: "en"
      confidence: 0.8399999737739563
    }
    detected_languages {
      language_code: "es"
      confidence: 0.1599999964237213
    }
  }
  width: 1265
  height: 1896
  blocks {
.
.
.
        symbols {
          property {
            detected_languages {
              language_code: "en"
            }
          }
          bounding_box {
            vertices {
              x: 592
              y: 378
            }
            vertices {
              x: 618
              y: 378
            }
            vertices {
              x: 618
              y: 440
            }
            vertices {
              x: 592
              y: 440
            }
          }
          text: "E"
          confidence: 0.9599999785423279
        }
.
.
.
    block_type: TEXT
    confidence: 0.949999988079071
  }
}
text:  "START\nIN:  Y.  EXP\n(Y_EXP  >1/\nFALSE\nTRUE\nOUT:  \"NO\".\nOUT:
\"OKAY.\"\n(STOP\n"
```

Code 5.2. A snippet of the response from Google Cloud Vision OCR API for the sample Figure 5.6

5.8.4. The Final Result

The final resulting graph is displayed in text in terms of nodes, node IDs and the control flows shown in Code 5.3 below.

```
rectangle_0:
      OUT: "OKAY."
      Parse Tree: expr: [io: [out, str: ["OKAY."]]]
rectangle_1:
      IN: Y_EXP
      Parse Tree: expr: [io: [in, var: [Y_EXP]]]
ellipse_0:
      START
      Parse Tree: start
diamond_0:
      IF Y_EXP > 1
      Parse Tree: expr: [cond: [cmp: [var: [Y_EXP], gt, num: [1]]]]
ellipse_1:
      STOP
      Parse Tree: stop
rectangle_2:
      TRUE
      Parse Tree: bool: [true]
rectangle_3:
      FALSE
      Parse Tree: bool: [false]
rectangle_4:
      OUT: "NO".
      Parse Tree: expr: [expr: [io: [out, str: ["NO"]]]]
['rectangle_2', 'rectangle_0']
['ellipse_0', 'rectangle_1']
['rectangle_1', 'diamond_0']
['rectangle_0', 'ellipse_1']
['rectangle_4', 'ellipse_1']
['diamond_0', 'rectangle_2']
['diamond_0', 'rectangle_3']
['rectangle_3', 'rectangle_4']
```

Code 5.3. Final Result for the flowchart in Figure 5.6

5.9. Discussion

The accuracies for the shape recognition are reasonable and don't need any boost. However, the accuracy for the arrow/line recognition model can be improved by adding more data (including more images of flowcharts with more instances of the less accurate classes namely left_arrow, horizontal_line and vertical_line) and using more complex models. The OCR, as discussed previously, does not match up to the precision required for an application that is reliant on the correctness of the textual grammar. To overcome this, as well as any other inaccuracies which may come about, the system provides the previously discussed interactive software to help out with error corrections. These corrections can help to contribute towards getting more data for helping with training.

Chapter 6

Conclusion and Future Scope

6.1. Conclusion

Throughout the research, various methods of processing flowcharts were explored. Most image processing systems studied either featured on-line systems, where a sequence of inputs would determine the shapes, or off-line systems where the strokes were reconstructed from images and then analyzed through existing on-line techniques. Despite the success of deep learning in object detection and classification, it was difficult to find literature which harnessed deep learning for extracting the features of flowcharts. The reported statistics between two popular models, SSD and YOLOv3, were compared and determined that YOLOv3 would be a suitable architecture owing to its good performance in complex datasets like COCO and the fact that it has a smaller, less memory-intensive architecture called Tiny YOLOv3.

On running some experiments with the Tiny YOLOv3 model, the conclusion was that object detection using deep learning would be a suitable approach to analyzing flowchart images and extracting their semantic information. Deep learning models can easily identify and detect both shapes as well as directional arrows and lines. Using OCR for handwriting recognition would allow to identify and process the information in each box in the flowchart image. Using the bounding box coordinates from the deep learning models and the text extracted using the OCR model, the structured representation of the flowchart can be easily generated.

6.2. Future Scope

In the near future, it is aimed to release a suite of tools which will support the complete translation of the flowchart into a target language. This will be useful for both students as well as professionals by allowing them to work on common projects without the need of deep knowledge of the syntactic complexities of other languages.

As explained above, in the long run, there are also plans to develop a mobile application that enables any user to simply take pictures of their hand-drawn flowcharts or export their mobile-generated or computer-generated flowcharts and translate them into code.

Another avenue to be explored is the possibility of improving the user interface (UI) for error corrections, either through an app or a web interface, and using the results from the UI provided for improvements in training the models.

References

[1] Hooshyar, D., Ahmad, R. B., Nasir, M. H. N. M., & Mun, W. C. (2014, June). Flowchart-based approach to aid novice programmers: A novel framework. In *2014 International Conference on Computer and Information Sciences (ICCOINS)* (pp. 1-5). IEEE.

[2] Xinogalos, S. (2013, March). Using flowchart-based programming environments for simplifying programming and software engineering processes. In *2013 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1313-1322). IEEE.

[3] Supaartagorn, C. (2017, November). Web application for automatic code generator using a structured flowchart. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (pp. 114-117). IEEE.

[4] Vasudevan, B. G., Dhanapanichkul, S., & Balakrishnan, R. (2008, June). Flowchart knowledge extraction on image processing. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* (pp. 4075-4082). IEEE.

[5] Hooshyar, D., Ahmad, R. B., & Nasir, M. H. N. M. (2014, October). A framework for automatic text-to-flowchart conversion: A novel teaching aid for novice programmers. In *2014 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)* (pp. 7-12). IEEE.

[6] Chen, Q., Shi, D., Feng, G., Zhao, X., & Luo, B. (2015, April). On-line handwritten flowchart recognition based on logical structure and graph grammar. In *2015 5th International Conference on Information Science and Technology (ICIST)* (pp. 424-429). IEEE.

[7] Miyao, H., & Maruyama, R. (2012, September). On-line handwritten flowchart recognition, beautification and editing system. In *2012 International Conference on Frontiers in Handwriting Recognition* (pp. 83-88). IEEE.

[8] Bresler, M., Průša, D., & Hlaváč, V. (2016, October). Recognizing Off-Line Flowcharts by Reconstructing Strokes and Using On-Line Recognition Techniques. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (pp. 48-53). IEEE.

[9] Jorge, J. A., & Fonseca, M. J. (1999, September). A simple approach to recognise geometric shapes interactively. In *International Workshop on Graphics Recognition* (pp. 266-274). Springer, Berlin, Heidelberg.

[10] Renuse, S., & Bogiri, N. (2017, August). Multi label learning and multi feature extraction for automatic image annotation. In *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)* (pp. 1-6). IEEE.

[11] Jian-ning, H., & Ming-quan, W. (2010, March). Research on digital image recognition system based on multiple invariant moments theory and BP neural network. In *2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010)* (Vol. 3, pp. 399-403). IEEE.

[12] Becker, G. (2007, October). Combining rule-based and machine learning approaches for shape recognition. In *36th Applied Imagery Pattern Recognition Workshop (aipr 2007)* (pp. 65-70). IEEE.

[13] Phanikrishna, C., & Reddy, A. V. (2016, October). Contour tracking based knowledge extraction and object recognition using deep learning neural networks. In *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)* (pp. 352-354). IEEE.

[14] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[15] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.

[16] Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

[17] Kajale, R., Das, S., & Medhekar, P. (2017, December). Supervised machine learning in intelligent character recognition of handwritten and printed nameplate. In *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)* (pp. 1-5). IEEE.

[18] Bartz, C., Yang, H., & Meinel, C. (2017). STN-OCR: A single neural network for text detection and text recognition. *arXiv preprint arXiv:1707.08831*.

[19] Zhang, H., Wei, H., Bao, F., & Gao, G. (2017, November). Segmentation-free printed traditional Mongolian OCR using sequence to sequence with attention model. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* (Vol. 1, pp. 585-590). IEEE.

[20] Detect handwriting in images | Cloud Vision API | Google Cloud. (n.d). Retrieved March 26, 2020, from https://cloud.google.com/vision/docs/handwriting

[21] Lark-Parser. lark-parser/lark. (n.d). Retrieved March 26, 2020, from https://github.com/lark-parser/lark

[22] AlexeyAB. (2020, March 23). AlexeyAB/darknet. Retrieved March 26, 2020, from https://github.com/AlexeyAB/darknet

[23] AlexeyAB. (2019, April 24). AlexeyAB/Yolo_mark. Retrieved March 26, 2020, from https://github.com/AlexeyAB/Yolo_mark

[24] Google Colaboratory. (n.d). Retrieved March 26, 2020, from https://colab.research.google.com/

[25] OpenCV. (2020, March 4). Retrieved March 26, 2020, from https://opencv.org/

[26] Tesseract-Ocr. (2020, March 18). tesseract-ocr/tesseract. Retrieved March 26, 2020, from https://github.com/tesseract-ocr/tesseract

[27] Tsang, S.-H. (2019, March 20). Review: YOLOv3 - You Only Look Once (Object Detection). Retrieved March 26, 2020, from https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6.

[28] RAPTOR - Flowchart Interpreter. (n.d.). Retrieved March 27, 2020, from https://raptor.martincarlisle.com/

[29] Flowgorithm - Flowchart Programming Language. (n.d.). Retrieved March 27, 2020, from http://www.flowgorithm.org/

Acknowledgements

First and foremost, we would like to express our gratitude to our mentor, Dr. Prachi Natu, who gave us the chance to do work on this innovative project on the topic “Extracting Flowchart Features into a Structured Representation”. Her invaluable, timely guidance and belief in this project have been of paramount importance to us. It has inspired in us a desire to go beyond our scope of the curriculum and conduct research in order to explore this domain and topic further.

We would also like to thank our parents, friends and various other faculty who offered us unconditional support and assistance, owing to which we were able to complete this project on time gracefully, with the desired results.

We as a team, are indebted to the NMIMS University and the Mukesh Patel School of Technology Management & Engineering for giving us the opportunity, infrastructure and the environment for bringing this mere concept to fruition.

Aditya Chakraborti (E003)
Aaditya Naik (E015)
Aditya Pansare (E019)
Utkarsh Pant (E020)