

St. Cloud State University

## The Repository at St. Cloud State

---

Culminating Projects in Computer Science and  
Information Technology

Department of Computer Science and  
Information Technology

---

4-2022

### Parsing Structural and Textual Information from UML Class diagrams to assist in verification of requirement specifications

Sandip Gautam

Follow this and additional works at: [https://repository.stcloudstate.edu/csit\\_etds](https://repository.stcloudstate.edu/csit_etds)

---

#### Recommended Citation

Gautam, Sandip, "Parsing Structural and Textual Information from UML Class diagrams to assist in verification of requirement specifications" (2022). *Culminating Projects in Computer Science and Information Technology*. 38.

[https://repository.stcloudstate.edu/csit\\_etds/38](https://repository.stcloudstate.edu/csit_etds/38)

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at The Repository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of The Repository at St. Cloud State. For more information, please contact [tdsteman@stcloudstate.edu](mailto:tdsteman@stcloudstate.edu).

**Parsing Structural and Textual Information from UML Class diagrams to assist in  
verification of requirement specifications**

by

Sandip Gautam

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfilment of the Requirements

for the Degree of

Master of Science

in Computer Science

May, 2022

Starred Paper Committee:

Maninder Singh, Chairperson

Andrew A. Anda

Aleksandar Tomovic

## Abstract

Unified Modeling Language (UML) class diagrams are widely used throughout software design lifecycle to model the Software Requirement Specifications in developing any software. In many cases these class diagrams are initially drawn, as well as subsequently revised using hand in a piece of paper, or a whiteboard. Although these hand-drawn class diagrams capture most of the specifications, they need a lot of revision and visual inspection by the software architects for verification of the captured requirements, of the system being modeled. Manually verifying the correctness and completeness of the class diagrams involves a lot of redundant work, and can raise issues due to human errors, for e.g., diagrams not drawn to scale, typeface issues, unclear handwriting, and even missing requirements etc. In this paper, we propose a state-of-the-art technique that pipelines the object detection, text extraction and replication phase to parse requirements incorporated within the user-drawn class diagrams. The parsed output from the proposed system, can be used to keyword-match the requirements in the Software Requirement Specification (SRS) document. We show that the proposed system can localize the UML classes and relationships in the diagram with 100% accuracy and can identify the localized objects of each type with maximum mean Average Precision of 0.8584. We also show that the text can be efficiently parsed from the diagrams with the character error rate of 0.3043.

### **Acknowledgement**

I would like to acknowledge and give my sincerest thanks to my advisor Dr. Maninder Singh, whose expertise in the subject matter and unparalleled guidance made this work possible. I'd also like to thank the members of committee, Dr. Andrew A. Anda, and Dr. Aleksandar Tomovic for providing their invaluable suggestions and support throughout the course of this work. All the committee members have had tremendous positive impact in offering timely response, best practices for writing paper, recommendations and much more than what can be brought into light.

A debt of gratitude is also owed to the dean of Computer Science Department, Dr. Ramnath Sarnath, and all the faculty members of Computer Science Department at St. Cloud State University, who has contributed to my academic development and competency. St. Cloud State University, along with the Department of Graduate Studies undoubtedly deserves a recognition for providing me the platform to learn and succeed in my studies. A special thanks to Mr. Cliff Moran, for making the operational work such as scheduling meetings, registering classes etc. smoother, and most of all setting me on the track to graduate by providing all the information about the guidelines of the department, and the university.

It goes without saying, but above all I am grateful to my parents Dr. Ishwori Prasad Gautam and Mrs. Amrita Gautam, and all my family members for their everlasting support. I would like to thank my friends, for helping to me to collect the data and for being there for me.

## Table of Contents

	Page
List of Figures .....	7
Chapter	
1. Introduction .....	9
1.1 Problem Statement .....	9
1.2 Proposed Solution .....	10
1.3 Key Concepts .....	10
1.3.1 Unified Modeling Language (UML) Class Diagram .....	10
1.3.2 Neural Networks .....	14
1.3.3 Computer Vision .....	16
2. Related Work .....	18
2.1 Convolutional Neural Networks (CNN) .....	18
2.2 Region Proposal Convolutional Neural Network (RCNN) .....	19
2.3 Faster Region Proposal Convolutional Neural Network (Faster-RCNN) .....	21
2.4 Character Region Awareness for Text Detection (CRAFT) .....	22
2.5 Convolutional Recurrent Neural Network (CRNN) .....	23
3. Design .....	24
3.1 Research Questions .....	25

Chapter	Page
3.2 Experiment Design .....	26
3.2.1 Data Collection.....	26
3.2.2 Label Images .....	27
3.2.3 Train Faster RCNN Detector.....	27
3.2.4 CRNN based scene text recognition model.....	27
3.2.5 Replicate Image .....	28
3.3 Evaluation Metrics .....	28
3.3.1 Faster-RCNN Evaluation Metrics .....	28
3.3.2 CRNN based Scene Text Evaluation Metrics.....	29
4. Implementation.....	31
4.1 Environment Setup.....	31
4.2 Libraries Installation .....	31
4.3 Input Data Set Preparation .....	32
4.3.1 Data Set for Training.....	32
4.3.2 Data Set for Testing.....	34
4.4 Train Object detection model.....	36
4.4.1 Accuracy thresholding for predictions. ....	37
4.4.2 Non-max suppression for overlapping predictions. ....	39

Chapter	Page
4.5 Evaluate object detection model .....	41
4.6 Segment UML Class .....	43
4.7 Text Extraction from segmented UML Class .....	48
4.8 Evaluate text extraction and recognition model.....	50
5. Results .....	52
6. Conclusion and Future Work .....	54
References .....	56

## List of Figures

Figure	Page
1. Sample UML class.....	11
2. Sample class diagram for course registration system in university .....	13
3. Neural Network with a single neuron [9] .....	15
4. Deep neural network with multiple hidden layers [14] .....	16
5. Convolution with padding and no strides [13].....	17
6. Architecture of CNN [15] .....	19
7. Architecture of Region Proposal CNN [2].....	20
8. Faster-RCNN Architecture [4].....	21
9. Ground Truth generation process of CRAFT architecture [8].....	23
10. Proposed Architecture for parsing structural and textual information from hand-drawn UML Class Diagrams .....	24
11. Experiment Design.....	26
12. Sample image (1) of the training dataset with color coded annotations .....	33
13. Sample image (2) of the training dataset with color coded annotations .....	33
14. Test image (1) drawn using computer- graphics.....	34
15. Test image (2) drawn using hand.....	35
16. Test image (3) drawn using smart ink.....	35
17. Unfiltered predictions for the hand-drawn test diagram. ....	38
18. Filtered predictions for the hand-drawn test diagram with respect to the accuracy score	39
19. Filtered predictions after non-max suppression of overlapping bounding boxes. ....	41



Figure	Page
20. Performance metric of object detection model, for hand-drawn test diagram.....	43
21. Straight lines (green) detected in test hand drawn image for UML class Dog (left) and, Person (right) .....	47
22. Straight lines (green) detected in test hand drawn image for UML class Retriever (left), Shepherd (middle) and, Family (right) .....	47
23. Segmented compartments for detected UML classes .....	48
24. Recognized text for all the class segments of each detected UML class .....	50
25. Word error rate and Character error rate for detected and recognized text .....	51
26. Comparison of Average Precision and Average Recall for all the test diagrams. ....	52

## Chapter 1: Introduction

In this proposal, we will extend on the applications of highly developed field of Computer Vision, and Neural Networks to develop an idea and implement the process of parsing a hand-drawn UML class diagram to obtain the requirement specifications captured within the diagram, for validating the completeness of software requirement specification. We intend to take several samples of hand-drawn class diagrams —i.e., image— and try to efficiently parse and convert the diagram into their digital equivalent and output the analysis of requirements contained within the diagram.

### 1.1 Problem Statement

UML is a broadly accepted mechanism for modeling software system, at present. However, with the ability to model complex systems, also comes the vast specifications that is needed to represent these models. Software architects usually model the software systems in multiple pass, and this also requires updating the UML diagram consistently throughout the modeling. As the software design cycle progresses from architecture design, to detailed design, and finally to development phase, there would be numerous changes that needs to be accommodated in all these design artifacts (such as specifications, use cases, uml diagrams, test cases etc.). While all these design artifacts are always documented and stored in a digital format, it is usually first done using hand-written/ hand-drawn artifacts. The requirements that the hand-drawn diagram depicts may not be analogous to the original design specifications, mainly due to human errors in drawing these draft of class diagrams. Moreover, all these artifacts need to be digitized, and continually adding revisions can prove to be arduous and time-consuming task. It may seem that a simple image of

these artifacts be sufficiently able in digitizing it, however there are still problems that can rise from unclear writings, partially drawn structures, typeface issues — bold, italic, and underlines — etc. that can result in unclear documentations.

## **1.2 Proposed Solution**

Recent developments in neural networks have enabled us to identify and localize objects, as well as to parse textual information in an image. However, validating both the structural as well as textual information in an image is observed as a standalone process till date. To verify the completeness of UML diagrams in capturing the software requirement specifications, we propose a workflow-based system that consists of different states such as preprocessing, object detection, object segmentation, and text extraction. Each of these states have their own logic to perform a specific task and can additionally have child workflows to accommodate for any new changes and decoupling sub-tasks. Although, the proposed system is distributed (states corresponds with subsequent states directly), a centralized workflow controller can also be added to manage complexities in future work and enhancements.

## **1.3 Key Concepts**

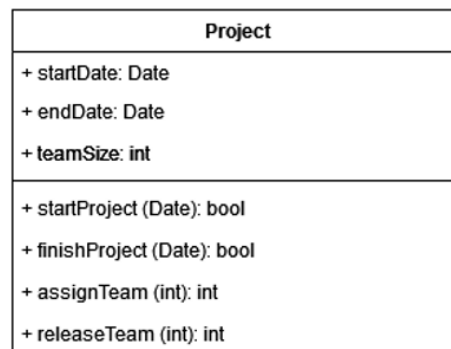
We discuss few key concepts which are required for understanding of the problem statement as well as for following the implementation of proposed solution in this paper.

### **1.3.1 Unified Modeling Language (UML) Class Diagram**

Unified Modeling Language (UML) is the standard in modeling the software artifacts using various set of diagrams that helps software and system engineers to specify, visualize, and implement these artifacts, mainly in the object-oriented development paradigm. UML is used to

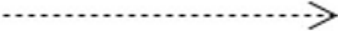
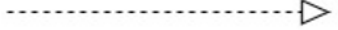





efficiently model large and complex systems and is based on the best engineering practices. UML defines a set of diagrams that can be broadly classified into structural and behavioral diagrams. All these diagrams are made up of components (*i.e.*, *classifiers*), that represent conceptual or physical aspects of the model. Few examples of classifiers include class, interface, association types etc. Classifiers are represented by a solid-outline rectangles containing the classifier's name, and optionally with compartments separated by horizontal lines containing features, or other members of the classifiers.

A UML class is a classifier, that describes a set of objects with common features (attributes and methods), constraints, and semantics. A sample Project class is as shown below in Figure 1. If a class is represented with 3 different compartments inside a rectangle, the top compartment holds the class name — centered text, bold face with the first letter of class name capitalized —, middle compartment holds the attributes of class with their respective types, and the bottom compartment holds the methods of class with input parameters' type, and respective return types of these methods. Each attribute and method of class can be preceded by a visibility operator which can be one of +, -, or # that represents public, private, and protected visibility respectively.



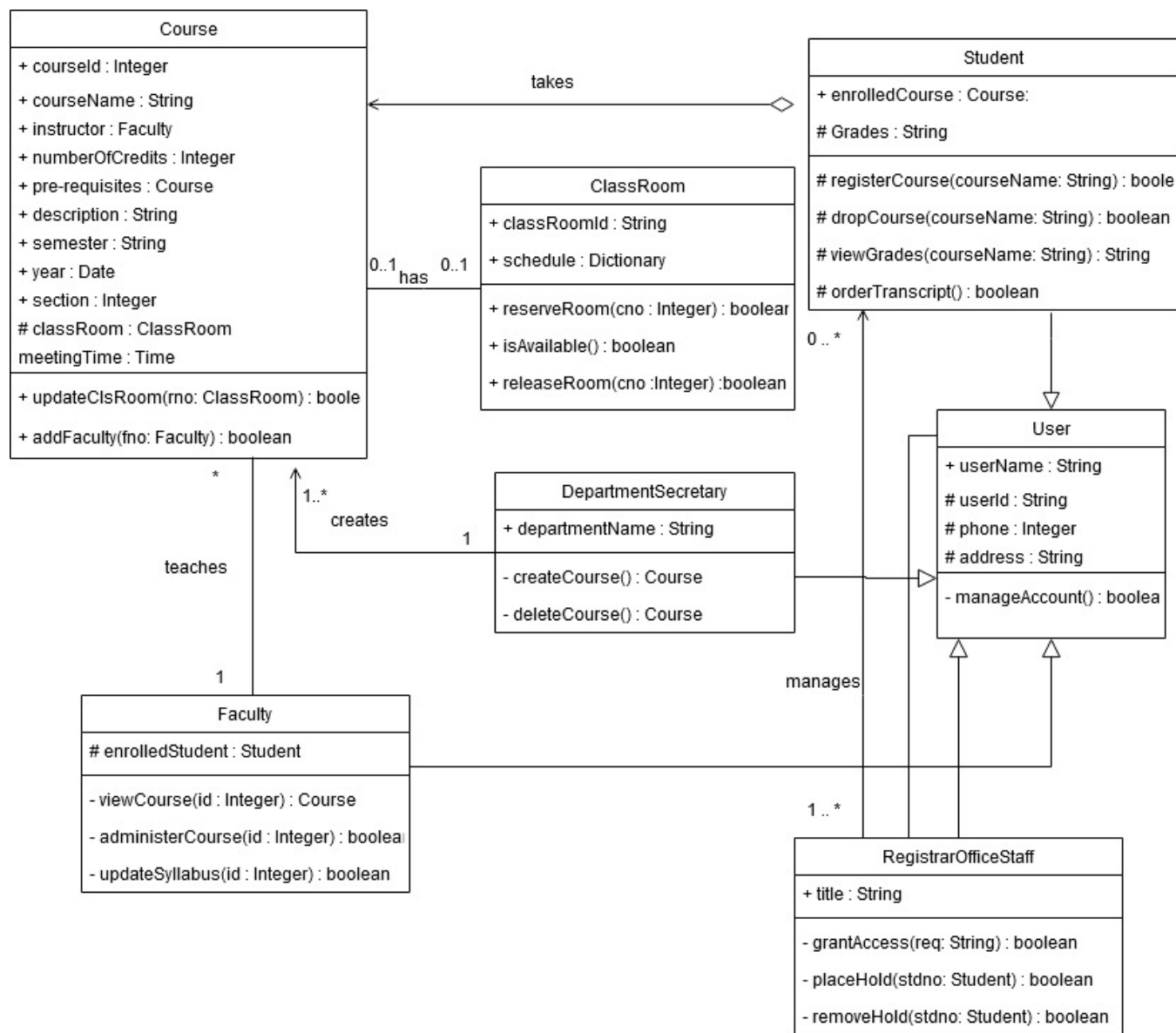
**Figure 1.** Sample UML class

**Table 1.** UML Relationships

	Dependency
	Realization
	Association
	Aggregation
	Composition
	Generalization
	Containment

A UML relationship is an abstract element that intends to depict dependency between the UML elements. Different types of relationships can exist between classes, as listed in Table 1 above. Further, relationships also establish associated multiplicity between classes which are related with each other (usually written on each end of arrows).

A class diagram is a static structural diagram, which is used for the conceptual modeling of an application to describe classes and their relationships. Class diagram promotes abstraction — one of the many principles of Object-Oriented Modeling and Design — by enabling us to design the system at high level, while hiding the implementation details. A sample class diagram for course registration system in university is as shown in Figure 2 below.



**Figure 2.** Sample class diagram for course registration system in university

### 1.3.2 Neural Networks

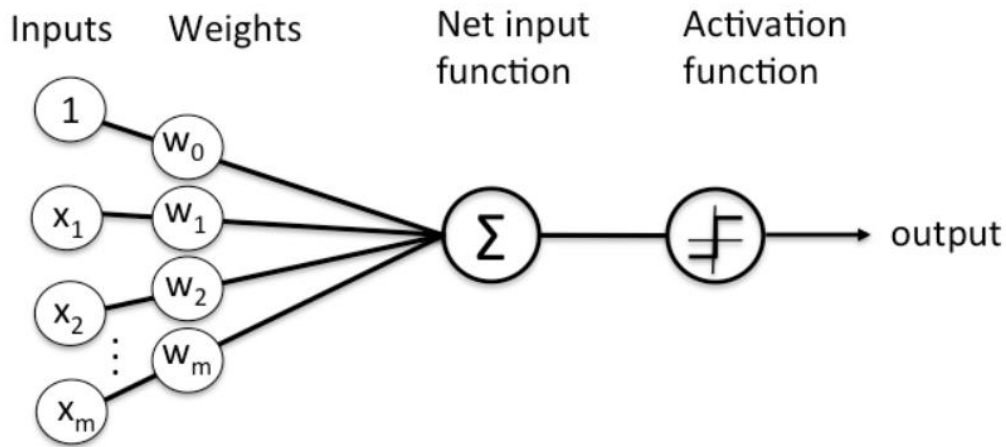
Neural Networks, also known as Artificial Neural Networks (ANNs) are the subset of Machine Learning Algorithms, that are modeled loosely after the human brain. These algorithms are powerful in recognizing the underlying relationships in a set of data. Neural Networks are widely used today for classification (binary classification, multiclass classification as well as regression), and clustering of data.

A neuron –building block of Artificial Neural Network–, is a mathematical function that takes a set of inputs and multiplies them with their corresponding weights. These input-weights products are then summed, and the final value is passed through some non-linear activation function [1]. The purpose of using non-linear activation function is to approximate complex function by introducing non-linearities in our decision boundary [2]. If the output of the activation function is greater than some threshold  $0 < t \leq 1$ , then the neuron is said to be fired or activated. For a single input with  $\mathbf{n}$  features (attributes), this can be represented by the equation,

$$y = \varphi(\sum_{i=1}^n w_i x_i + b) \quad (1)$$

where  $y$  is the output,  $x_i$  is the  $i^{\text{th}}$  input feature,  $w_i$  is the weight for corresponding feature,  $b$  is the bias, and  $\varphi$  is non-linear activation function (sigmoid [20], RELU [19] etc.).

For multiple real-valued inputs (training examples), the inputs can be arranged in a 2D matrix where each column represents a single input with  $n$  features, and there are total  $m$  columns for all the training examples. For e.g.



**Figure 3.** Neural Network with a single neuron [9]

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}$$

Then the output of a single neuron for multiple training examples can be written as,

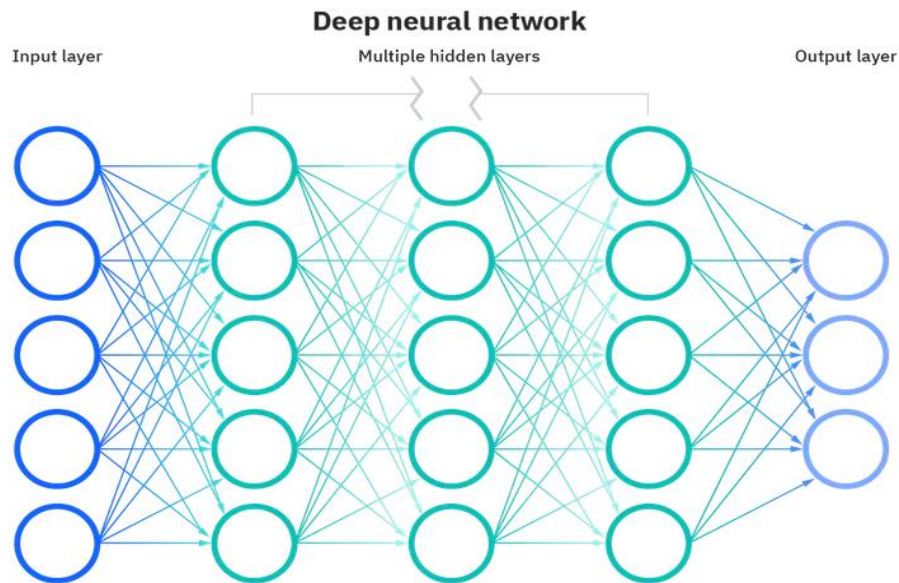
$$y = \varphi(\mathbf{W}^T \mathbf{X} + b) \quad (2)$$

where,  $\mathbf{W}^T$  is the transpose of weight matrix  $\mathbf{W}$ ,  $\mathbf{X}$  is the input matrix and  $\varphi$  is the activation function.

A neural network is a network of several neurons stacked together in multiple layers. The network consists of a single input layer, single or multiple output layers and several hidden layers. If the number of hidden layers is more than 3, which is usually the case in modern applications to solve specific problem, it is termed as Deep Neural Networks. In this type of stacked neural



network, the output of each layer is the input to subsequent layers. Training a neural network means



**Figure 4.** Deep neural network with multiple hidden layers [14]

continuously readjusting the weights and biases of input features, usually through backpropagation [10], by calculating the gradient of the error function  $E(X, \theta)$  where  $\theta$  collectively denotes the weights  $\mathbf{W}$  and, bias  $\mathbf{b}$  of all the inputs in the network. A sample diagram of neural network is as shown in Figure 4.

### 1.3.3 Computer Vision

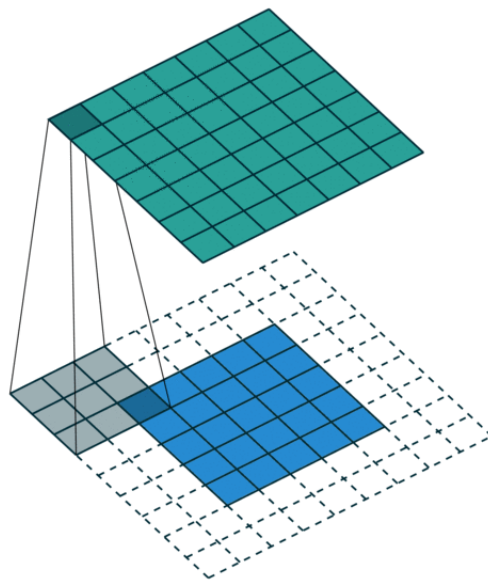
Computer vision is a field of Artificial Intelligence (AI), that helps us derive high-level understanding and meaningful information from digital artifacts such as images and, videos. While Image processing is mainly related to performing transformations in a digital image to enhance or simplify a given image(pre-processing), computer vision is primarily concerned with image

understanding that includes a wide range of applications such as object detection, 3D modeling, surveillance, optical character recognition, human computer interaction and so on. One key concept that plays a major role in both Image Processing as well as computer vision is Convolution.

In mathematical terms, Convolution is an operator that takes two functions  $f$  and  $g$ , and outputs a third function ( $f * g$ ), that expresses how the shape of one is modified by the other. The term *convolution* refers to both the result function and the process of computing it [11]. Convolution for two functions  $f(a)$  and  $g(b)$  at point  $c$  is given by the equation,

$$(f * g)(c) = \sum^a f(a) * g(c - a) \quad (3)$$

In computer vision's perspective, Convolution simply refers to sliding a kernel (small matrix) over an image to achieve different effects in image such as sharpening, blurring, edge detection etc. This is shown in the Figure 5 below.



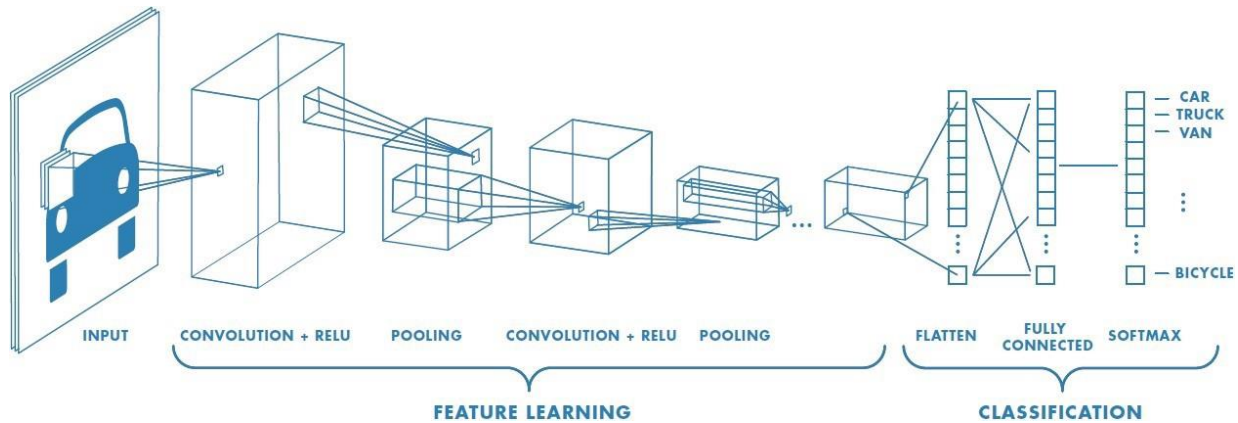
**Figure 5.** Convolution with padding and no strides [13]

## Chapter 2: Related Work

Object detection has existed for a while and can be done using either traditional approach of classification using Support Vector Machines (SVMs) or a modern Neural- Network Architecture to automatically detect and learn objects in given image. It is very important that the input image is free of noise and have clearly defined edges for us to detect the contours. Image preprocessing steps in [5], discusses algorithms for histogram equalization, edge detection, edge thinning, and Hough transform. These algorithms for pre-processing of image can be used out-of-box from open-source libraries such as OpenCV, ImageJ etc. The algorithm in [17], provides efficient way of detecting the contours in an image, by finding fewer points (i.e., pixels), that represents the input curve and approximating different kind of Polygons which is of interest to the users that can be validated using Cross co-relation or Correlation coefficient. The modern way of object detection, however, uses a Region Proposal Convolutional Neural Networks (RCNN) [4] or Faster RCNN [3] to approximate a region such as rectangles, ovals, or any different kind of polygons.

### 2.1 Convolutional Neural Networks (CNN)

A Convolutional Neural Network is a deep neural network in which the hidden layers perform convolution (a dot product of the convolution kernel with layer's input matrix), by sliding the convolution kernel along the input matrix to generate a feature map, which is usually passed as input to the next layer [12]. Pre-processing on CNN doesn't require hand-engineered kernel (or, filters) as with traditional classification algorithms, and these kernels are automatically learned while training the network.

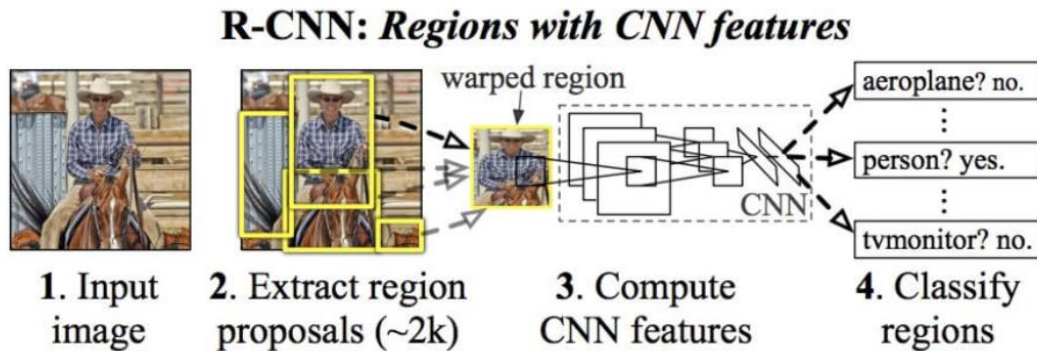


**Figure 6.** Architecture of CNN [15]

Input to CNN is a tensor of shape  $no\_of\_inputs * input\_height * input\_width * no\_channels$ . The ability of convolving the input tensor without having to flatten the image into a single dimension –as with the case in feedforward neural networks–, helps capture the spatial and temporal dependencies in the image, through the application of relevant filters. The convolution layer enables us to extract lower-level features in given image such as edges, gradient orientation, color etc. With subsequent convolution layers, the network can learn high level features, thus giving us full understanding of image. This convolution layer is then followed by pooling layers, fully connected layers, and normalization layers. The architecture of Convolutional Neural Network is as shown in Figure 6.

## 2.2 Region Proposal Convolutional Neural Network (RCNN)

CNNs are widely used for classification problems. However, object detection requires localization of objects in given image. Traditional way of object detection using CNNs, usually have 2 convolution layers and pooling layers to maintain high spatial resolution.

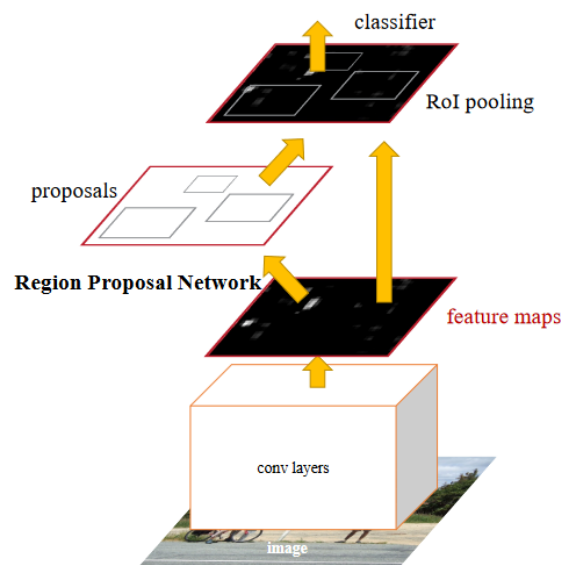


**Figure 7.** Architecture of Region Proposal CNN [2]

CNNs are usually combined with sliding window detectors to detect the portions of image that constitute the object, typically on constrained object categories such as pedestrians and faces. This method of localizing objects uses exhaustive search to search in all possible windows even for a smaller image, which is computationally very expensive. To solve the localization problem in traditional CNN, RCNN [2] uses a paradigm known as “recognition using regions” that are proven successful for object recognition and semantic segmentation [1]. RCNN uses three modules to detect the objects within an image. The first module generates category-independent region proposals (2000 approximately), which define the set of candidate detections available to detector. The regions proposals on RCNN are generated using a technique known as selective search where an image is initially sub-segmented into many regions, and recursively merged using a greedy approach based on similarity in color, texture, size and fill. This is followed by affine image warping to compute fixed-size CNN input from each region’s proposal. The second module is a large convolutional neural network – 5 convolution layers with large receptive fields (195 \* 195 pixels) and strides (32 \* 32 pixels) – that extracts a fixed length feature vector from each region, and the third module classifies each region with category-specific linear SVMs. The architecture of RCNN is as shown in Figure 7.

### 2.3 Faster Region Proposal Convolutional Neural Network (Faster-RCNN)

Fast-RCNN [3] (not, Faster-RCNN), address two main downsides of RCNN that affects the speed and accuracy of object detection; All the region proposals must be processed for object localization; and candidate object location only provides rough localization and needs to be refined to output the precise localization. Fast-RCNN works by feeding the input image to CNN to generate a convolutional feature map, as opposed to traditional RCNN which feeds the regional proposals to CNN layers. This convolutional feature map is then used to identify the regions of proposal, and with the help of ROI pooling layer –uses max pooling to reduce the feature inside any valid regions of interest into a smaller feature map–, the feature map is reshaped into a fixed size so that it can be fed into a fully connected layer. This process avoids the network to process all  $\approx 2000$  region proposals, and thus saves the speed.

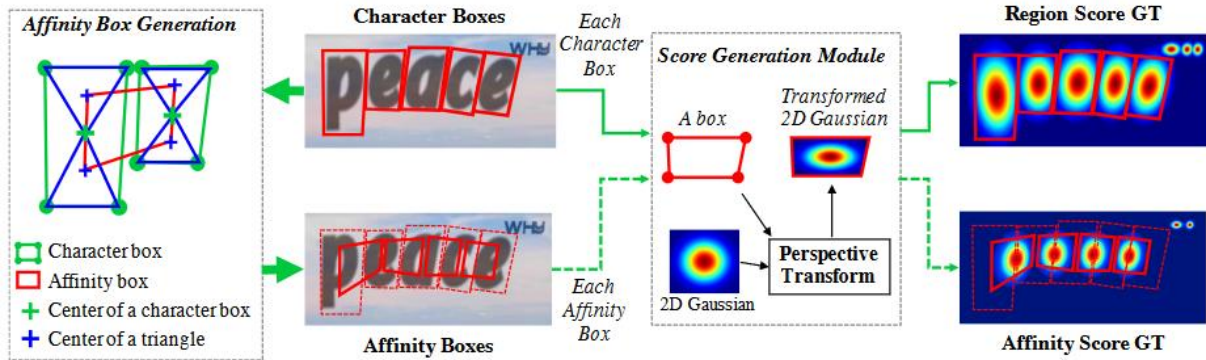


**Figure 8.** Faster-RCNN Architecture [4]

Faster RCNN [4] on the other hand, improves upon both RCNN, and Fast-RCNN in that it doesn't use selective search to generate region proposals. Instead, a separate region proposal network (RPN) is used to predict the region proposals, each with an objectness score. Faster-RCNN shares the same convolutional feature map with RPN and the object detector network (Fast-RCNN) thereby enabling nearly cost-free region proposals. Faster-RCNN develops on the idea of attention networks and let RPN module tell the Fast-RCNN module where to look for region proposals. The regions proposals are then used by ROI pooling layer to reshape into fixed size regions. The regions are classified, and offsets values for the bounding box are calculated. The architecture of Faster- RCNN is as shown in Figure 8 where the Region Proposal Network acts as the “attention” of the unified network.

#### **2.4 Character Region Awareness for Text Detection (CRAFT)**

Identifying text in an image, requires us to detect text regions before text could be retrieved and parsed from given image. Since text can be curved, lengthy and deformed which can be difficult to detect at word-level within the scope of single bounding box, the Character Region Awareness for Text Detection (CRAFT) [8] framework, proposes a character-awareness text detection method. CRAFT works by detecting individual character regions and links the detected characters to a text instance. It does so by training a convolutional neural network in weakly-supervised manner to localize individual character in the image using region scores –probability that the given pixel is the center of character– and group each character into single instance using affinity score –center probability of the space between adjacent character–. CRAFT uses heatmap representation to learn both region scores, and affinity scores due to its high flexibility when



**Figure 9.** Ground Truth generation process of CRAFT architecture [8]

dealing with ground-truth regions that are not rigidly bounded. The ground truth generation process of CRAFT framework is as shown in Figure 9.

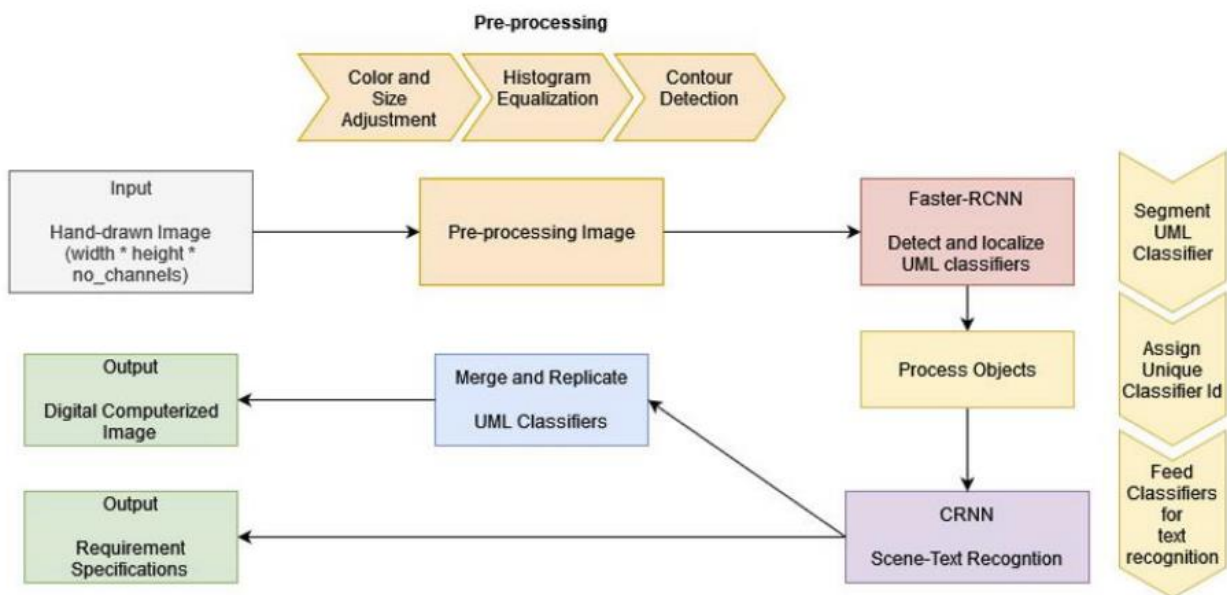
## 2.5 Convolutional Recurrent Neural Network (CRNN)

After suitable text regions are detected for given image, scene-text-recognition for the identified regions of interest can be performed with very high accuracy using a CRNN architecture [5], by training the neural networks to learn different numbers of filters representing hierarchical feature maps, which is then combined with bi-direction Long Short-Term Memory [6], and a dense neural network layer to output a probability of each word in the input scene — a small sub-part in our image —, using Connectionist Temporal Classifier (CTC) as a loss function. The detected contour for a region-of-interest and the true region are evaluated using evaluation metrics such as Intersection of Union (IOU) [16], and the Structural Similarity Index (SSI) [7].



### Chapter 3: Design

To efficiently identify and replicate classifiers –UML classes and Relationships– using a software system, we use the techniques in computer vision to read and, pre-process the image. The pre-processing, among many things, includes size adjustment, brightness and contrast adjustments, noise removal, and contour detection. The image is then passed to Faster-RCNN detector to detect and localize the classifiers present in the image. After all relevant classifiers has been identified, the next step is to parse the text, present inside those classifiers. Parsing textual information in given image can be best executed, with the help of Convolutional Recurrent Neural Networks (CRNN) with very high accuracy. The final step would be to replicate the same structure and text, in a digital format using a graphics library. The proposed architecture for this design is as shown in Figure 10.



**Figure 10.** Proposed Architecture for parsing structural and textual information from hand-drawn UML Class Diagrams

### 3.1 Research Questions

The following research questions are studied and evaluated in this research.

- a) **RQ-1:** How accurately does our model identify the UML classes in given input images?

*Description about RQ-1:* This RQ is aimed at evaluating our proposed methods on how accurately they can perform object detection and localization from different image samples. The performance of an object detector is typically evaluated using mean Average Precision (mAP) over classes, which is based on ranking of detection scores of each class. The area under the precision-recall (PR) curve gives the average precision for each object class. What is the mAP for our classifier objects?

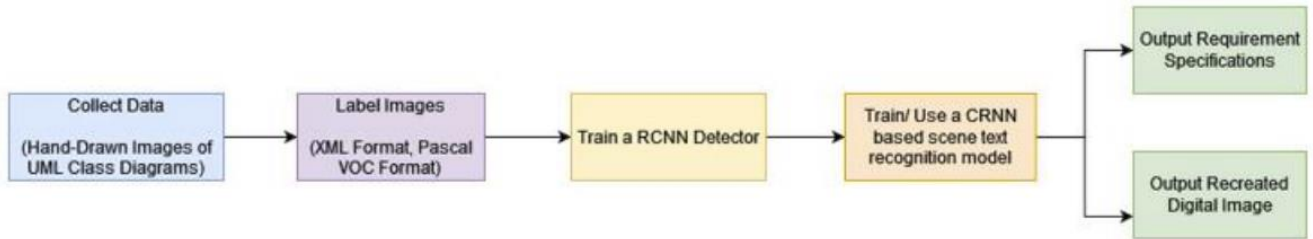
- b) **RQ-2:** How does our proposed model performs in detecting UML class diagrams when drawn by computer tool, by hand, and by smart inking?

*Description about RQ-2:* What would be the mAP difference between feeding hand-drawn, smart-inked and computer graphics version of UML class diagrams to Faster-RCNN architecture?

- c) **RQ-3:** How does the CRNN based architecture performs for text detection inside a class diagram?

*Description about RQ-3:* How accurately does the scene text detection and recognition model detects text-regions and recognizes the text inside the UML Class diagram? What is the word error rate(WRR) and character error rate(CRR) between the original text and the detected text?

## 3.2 Experiment Design



**Figure 11.** Experiment Design

The experiments consist of series of steps such as data collection, data annotation, training an object detector and using a text recognition model, each of which are described in detail below.

### 3.2.1 Data Collection

Data collection involves acquiring images of hand-drawn samples of UML class diagrams, drawn either in paper with ink, or a digital version using a stylus. The collected samples are tested for their readability, quality, and correctness. To reduce the complexity in preparing data for labelling, several design considerations are proposed to verify the correctness of the input image, which are listed below.

- a) Two different types of images are collected for a single class diagram, one with just the outline of the class diagram without any content inside or around the classifiers, and one with all the contents.
- b) All the UML relationships are drawn in disjoint fashion i.e., single arrow extending to more than two classes are drawn separately and are not merged in any way.

- c) All classifier and relationship are drawn without overlapping any other classifier or relationships such that each of these objects are individually selectable using a rectangular bounding-box.

### **3.2.2 Label Images**

The collected images of UML class diagrams are labelled by selecting them using a rectangular selection box and tagging their respective names. The outputs of labelled images are stored in XML format, that records the position of each of these selections using xmin, ymin, xmax and ymax.

### **3.2.3 Train Faster RCNN Detector**

Both the image file, as well as XML file which has the ground truth label for the position of objects in given image, are passed into the Faster-RCNN Model to train for object recognition and localization. Each image, and corresponding XML file are a single training example for the model. If we consider all the images, then there would be total of  $m$  training examples. After a model has converged, the output of the model would be a list that has the xmin, ymin, xmax and ymax co-ordinates for each bounding box and their respective labels, that the model detected and localized as one of the UML classifier and relationship.

### **3.2.4 CRNN based scene text recognition model**

Each identified objects from RCNN detector are given a unique object id. In case the classifier can be segmented into compartments (for e.g., UML class can be segmented into 3 compartments, namely the className compartment, attributes compartment, and methods

compartment), then each of these compartments are tagged with segment names. The identified regions for the UML class and relationship are then passed into text detector and CRNN based scene text recognizer, which would output the text present inside those classifiers.

### 3.2.5 Replicate Image

After all the classes and relationships are identified with all the texts inside and around them, the next step is to re-create the image using Graphics Library.

## 3.3 Evaluation Metrics

We use the industry-wide adopted metrics to evaluate the performance of the Faster-RCNN model on our dataset for object detection as well as for our CRNN based scene text recognizer. The metrics are described in detail below.

### 3.3.1 Faster-RCNN Evaluation Metrics

For object detection, the commonly used metric to compare the model's predicted bounding box to that of ground-truth bounding box, is Intersection over Union (IOU). IOU evaluates the overlap between two bounding boxes and is calculated as overlapping area between the ground-truth and predicted bounding box divided by the area of union between them. This is shown in Equation 4.

$$IOU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \quad (4)$$

IOU is then used to measure the number of,

- a) True Positive (TP): Correct detection indicated with  $IOU \geq threshold (0.5)$
- b) False Positive (FP): Incorrect detection indicated with  $IOU < threshold (0.5)$
- c) False Negative (FN): A ground truth that is not detected.

Using the values of TP, FP and FN we can calculate Precision, which gives the percentage of the prediction which are correct. Recall gives the percentage of True Positives detected among all relevant ground-truth. Then, the Average Precision (AP) is the area under the Precision-Recall curve (PR curve). Finally, the mean Average precision (mAP) can be calculated by taking the mean AP over all classes.

$$Precision = \frac{TP}{TP + FP} \quad (5)$$

$$Recall = \frac{TP}{TP + FN} \quad (6)$$

$$Average\ Precision = \frac{1}{11} \sum_{Recall(i)} Precision * Recall(i) \quad (7)$$

### 3.3.2 CRNN based Scene Text Evaluation Metrics

CRNN based Scene Text Recognition model uses two main metrics to evaluate the performance of the model: Character Error Rate (CER) and Word Error Rate (WER). The character error rate and word error rate measure the amount of text in the given handwriting, that the model did not read correctly. The lower the CER and WER value, the better the performance of the model in recognizing hand-written text.

$$CER = \frac{S + D + I}{S + D + C} \quad (8)$$

where,  $S$  is the number of Substitutions,  $D$  is the number of deletions,  $I$  is the number of insertions, and  $C$  is the number of correct characters.

In addition to that, we use Levenshtein distance [18] — edit distance —, between original text string  $a$ , and predicted text string  $b$  to measure the rate of correctly predicted characters in our input image.

$$Lev(a, b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ lev(\text{tail}(a), \text{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} lev(\text{tail}(a), b) \\ lev(a, \text{tail}(b)) \\ lev(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise} \end{cases} \quad (9)$$

where,  $\text{tail}(x)$  for some string  $x$  is the string of all characters of  $x$ , except for the first character.

## Chapter 4: Implementation

### 4.1 Environment Setup

Training a neural network for custom object detection (UML classifiers, in our case) requires significant resources in terms of Disk, Memory, CPU, GPU and TPU. It is also crucial to have provision, for interactive inspection of model output as well as the code. We used the well-known Jupyter Notebook, in Google colab platform to do all our developments. Python (v. 3.7) is used as the programming language of choice, because of its exceptional computational ability, out of box code optimizations and, broader open-source community along with availability of many libraries and frameworks.

```
# Setup and check python version in Google Colab
!type python
!python --version
```

```
python is /usr/local/bin/python
Python 3.7.12
```

### 4.2 Libraries Installation

We used detecto framework for training out custom Faster-RCNN model, to detect UML classes and relationships. easyocr framework is used for text recognition and detection. Both frameworks are written in Pytorch — an open-source machine learning framework based on Torch library, developed by Meta—, and promotes speedy development. We used Opencv library, for performing several image transformations and matplotlib for visualization of results.



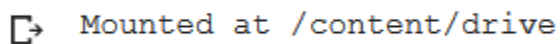
```
# Install "detecto" framework to train our Faster-RCNN model
import sys
sys.executable
!{sys.executable} -m pip install --quiet detecto

# Install all other the required libraries
! pip install --quiet "torch==1.10.0" "torchmetrics" "lightning-
bolts" "pytorch-lightning" "torchvision" "opencv-python-
headless==4.1.2.30" "easyocr"
```

### 4.3 Input Data Set Preparation

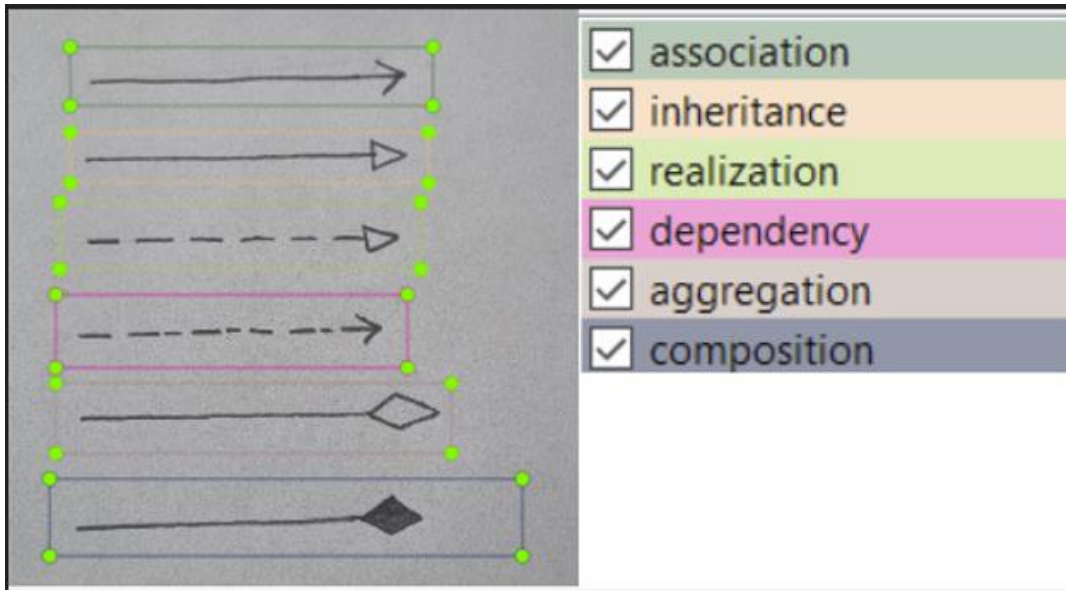
The training and testing dataset (images) are hosted in Google Drive. Both datasets are annotated with their respective class labels using labelImg annotation tool.

```
from google.colab import drive
drive.mount('/content/drive')
```

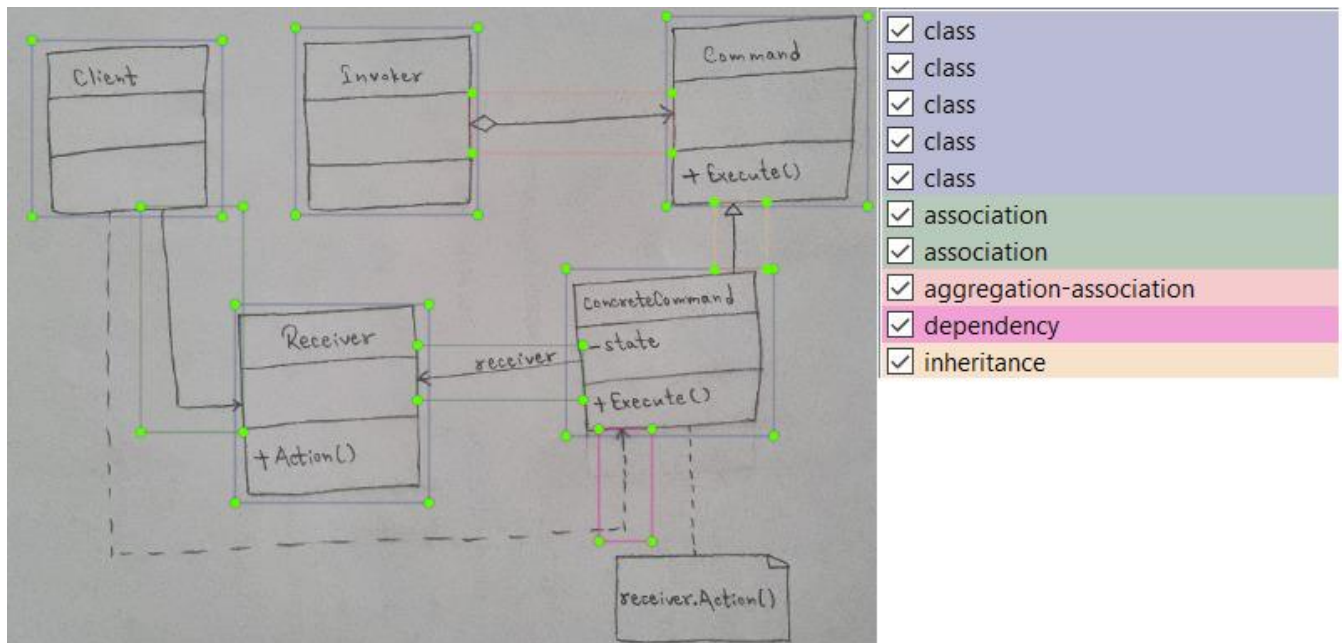
A screenshot of a terminal window showing the output of the drive.mount command. It displays a folder icon followed by the text "Mounted at /content/drive".

#### 4.3.1 Data Set for Training

Training dataset consists of total 68 input images, with their respective 68 annotations on PascalVOC format.



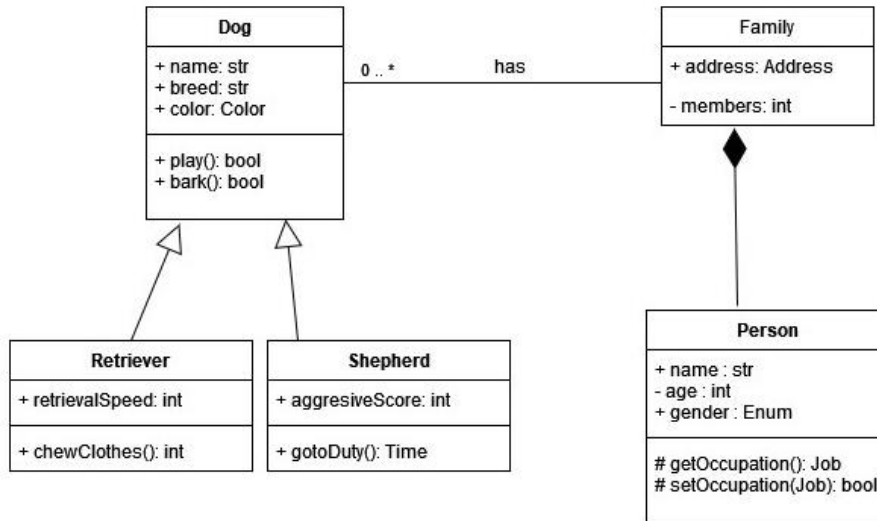
**Figure 12.** Sample image (1) of the training dataset with color coded annotations



**Figure 13.** Sample image (2) of the training dataset with color coded annotations

### 4.3.2 Data Set for Testing

Testing dataset consists of 3 identical UML class diagrams drawn using computer graphics, hand, and smart ink respectively.



**Figure 14.** Test image (1) drawn using computer- graphics

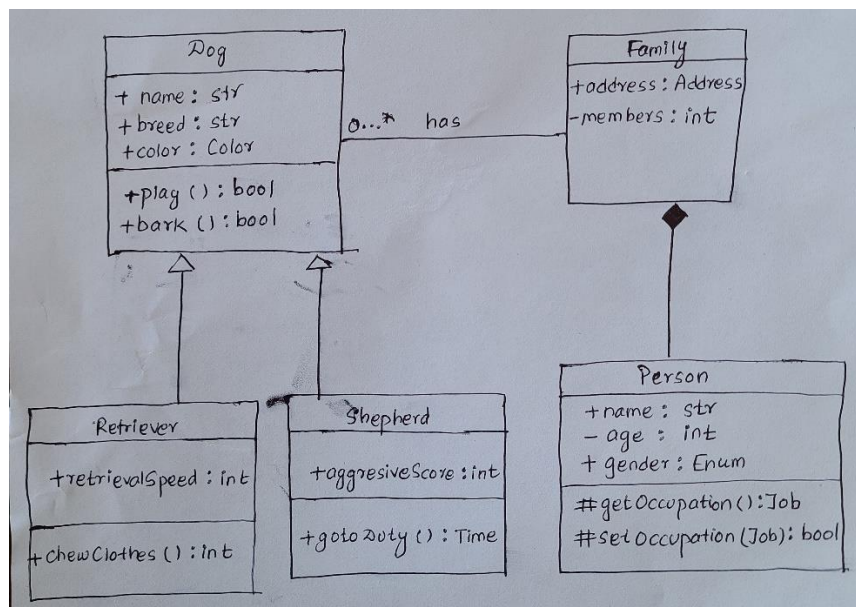


Figure 15. Test image (2) drawn using hand

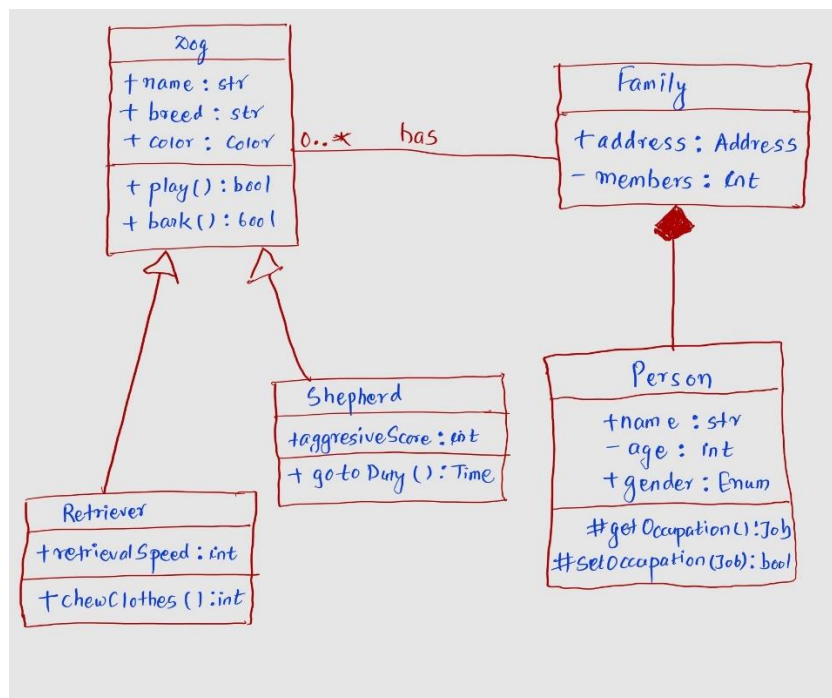


Figure 16. Test image (3) drawn using smart ink

#### 4.4 Train Object detection model

We trained the object detection model using detecto —which internally uses state-of-art Faster RCNN model for training object detectors—, with 12 different class labels. We did not use validation set to compare performance between different models trained using varying hyperparameters. The default values for hyperparameters that were used to train the model, are as shown below.

```

epochs                10
learning rate         0.005
momentum              0.9
weight_decay          0.0005
gamma                 0.1
lr_step_size          3

from detecto import core, visualize, utils
from detecto.core import Dataset, DataLoader

train_dataset = core.Dataset('/content/drive/My Drive/Train')
test_dataset = Dataset('/content/drive/My Drive/Test')

# Use the validation dataset, to tune in the hyperparamaters of the model
# and get more control over the model prediction
# validation_dataset = core.Dataset('/content/drive/My Drive/Validation')

loader = core.DataLoader(train_dataset, batch_size= 5, shuffle= True)

# label '0' always represents the background class
labels_map = { 'class': 1,
               'association': 2,
               'aggregation': 3,
               'realization': 4,
               'dependency': 5,
               'u-association': 6,
               'inheritance': 7,
               'composition': 8,
               'agg-depy-agg': 9,

```

```

        'aggregation-association': 10,
        'box': 11,
        'oval': 12
    }
model = core.Model([*labels_map.keys()])
model.fit(train_dataset)

# Training while specifying the hyperparameters(epochs, learning rate, and
# val. dataset) results in CUDA out of memory, when training on limited
# GPU on colab.
# Use the code below to train your model if you have sufficient GPU.

# model.fit(loader, verbose= True)
# losses = model.fit(loader, validation_dataset, epochs= 15,
#                     learning_rate= 0.001, verbose= True)
# plt.plot(losses)
# model.save("uml_weights.pth")
# uml_model = model.get_internal_model()
# print(type(uml_model))

model.save("uml_weights.pth")

```

#### 4.4.1 Accuracy thresholding for predictions.

The output predictions from trained model were filtered using `accuracy_score > 0.25`. The initial predicted bounding boxes, as well as filtered bounded boxes using the accuracy score, are shown in Figure 17 and Figure 18.

```

import numpy as np
image=utils.read_image('/content/drive/My Drive/Test/test_handwritten.jpg'
)
print("Image Shape: ", image.shape)

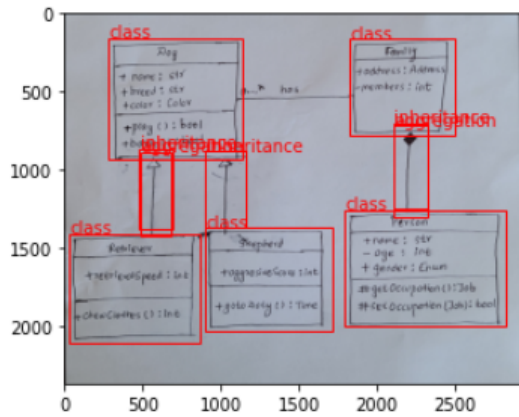
predictions = model.predict(image)
labels, boxes, scores = predictions

# Show all the predictions, without filtering

```



↳ Filtered predictions after thresholding, i.e, accuracy > 0.25.  
Total number of predictions: 10



```

tensor([ 272.7581, 159.6465, 1138.8734, 937.0354]) tensor(0.9977) class
tensor([1795.3662, 1257.4847, 2824.4614, 1996.1605]) tensor(0.9977) class
tensor([ 32.4517, 1409.9215, 866.4893, 2108.2705]) tensor(0.9974) class
tensor([ 896.1554, 1372.4366, 1720.2419, 2032.8591]) tensor(0.9965) class
tensor([1821.7087, 163.7551, 2496.3604, 776.5922]) tensor(0.9962) class
tensor([2102.9976, 700.2303, 2320.7363, 1251.2357]) tensor(0.3909) inheritance
tensor([ 476.1547, 865.1815, 688.2950, 1376.3400]) tensor(0.3409) inheritance
tensor([2110.0176, 718.4122, 2327.9536, 1300.6497]) tensor(0.2975) aggregation
tensor([ 900.6353, 880.7171, 1155.8413, 1365.6949]) tensor(0.2913) inheritance
tensor([ 483.1385, 884.2556, 681.4133, 1424.1674]) tensor(0.2742) aggregation

```

**Figure 18.** Filtered prediction for the hand-drawn test diagram with respect to the accuracy score

#### 4.4.2 Non-max suppression for overlapping predictions.

To eliminate overlapping bounding boxes with same or different prediction labels, we used non-max suppression algorithm —calculates the area of overlap between overlapping bounding boxes and merges the bounding boxes into a single box if the amount of overlap is greater than given threshold value—, with the Intersection over Union (IOU) threshold of 0.5. The results after suppressing overlapping boxes are shown in Figure 19.



```

import torchvision
import torch
from pprint import pprint
from torchmetrics.detection.map import MAP

# "ops.nms" returns the index (dec. ord of scores) of all bboxes that we
# should keep after performing the non-max suppression.
#
# Batched NMS does not suppress overlapping boxes of different classes.
# Usually, best practice is to not suppress boxes with different class
# labels and instead classify them as False positives (FP).

# Discards all overlapping boxes with IoU > iou_threshold

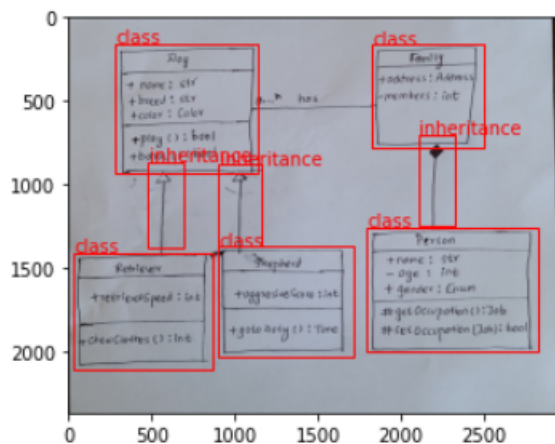
int_labels = torch.tensor(list(map(labels_map.get, labels)), dtype= torch.
int)
keep = torchvision.ops.nms(boxes, scores, iou_threshold= 0.5)

filtered_boxes, filtered_scores = boxes[keep], scores[keep]
filtered_labels = [labels[idx] for idx in keep]

print("Filtered predictions after non-
max suppression of overlapping bboxes, IOU > 0.5")
print("Total number of predictions: ", len(filtered_boxes), "\n")
visualize.show_labeled_image(image, filtered_boxes, filtered_labels)
print("\n")
for fb, fl, fs in zip(filtered_boxes, filtered_labels, filtered_scores):
    print(fb, fl, fs)

```

↳ Filtered predictions after non-max suppression of overlapping bboxes, IOU > 0.5  
Total number of predictions: 8



```

tensor([ 272.7581,  159.6465, 1138.8734,  937.0354]) class tensor(0.9977)
tensor([1795.3662, 1257.4847, 2824.4614, 1996.1605]) class tensor(0.9977)
tensor([  32.4517, 1409.9215,  866.4893, 2108.2705]) class tensor(0.9974)
tensor([ 896.1554, 1372.4366, 1720.2419, 2032.8591]) class tensor(0.9965)
tensor([1821.7087,  163.7551, 2496.3604,  776.5922]) class tensor(0.9962)
tensor([2102.9976,  700.2303, 2320.7363, 1251.2357]) inheritance tensor(0.3909)
tensor([ 476.1547,  865.1815,  688.2950, 1376.3400]) inheritance tensor(0.3409)
tensor([ 900.6353,  880.7171, 1155.8413, 1365.6949]) inheritance tensor(0.2913)

```

**Figure 19.** Filtered predictions after non-max suppression of overlapping bounding boxes.

#### 4.5 Evaluate object detection model

The final predictions obtained after accuracy thresholding and non-max suppression of bounding boxes are used to calculate mean Average Precision (mAP) and mean Average Recall (mAR) for the trained object detection model. The Average Precision (AP) is calculated by comparing ground truth values with prediction results, with respect to varying IOU threshold range (for eg., mAP<sub>25</sub>, mAP<sub>50</sub> etc.) for all individual classes. The Average Precision over all the classes are then averaged to calculate the mAP. The calculated metric to test performance our object detection model is as shown in Figure 20.

```

# Generates the mean Average Precision of the Object Detection over all
# the classes.
# Prediction labels should be '0' indexed whereas target should be '1'
# indexed.
#
# https://torchmetrics.readthedocs.io/en/latest/references/modules.html#ma
p

# If we require MAP for multiple test images, add a new entry to pred and
# target list. The number of entries in pred and target need to be same.
preds = [
    dict(
        boxes= filtered_boxes,
        scores= filtered_scores,
        labels= torch.tensor(list(map(lambda x: labels_map.get(x), filtered
_labels))), dtype= torch.int)
    )
]

# im_num is the index position of the test image in the test_dataset.
# For different test_image change imnum constant accordingly
imnum = 1
_, ground_truth = test_dataset[imnum]
gt_boxes, gt_labels = ground_truth["boxes"], ground_truth["labels"]

target = [
    dict(
        boxes= gt_boxes,
        labels= torch.tensor(list(map(labels_map.get, gt_labels))), dtype= t
orch.int)
    )
]

# print(preds, target)
# Mean Average Precision of detected classes
metric = MAP(class_metrics= True)
metric.update(preds, target)
pprint(metric.compute())

```

```

↳ {'map': tensor(0.2547),
   'map_50': tensor(0.4167),
   'map_75': tensor(0.2500),
   'map_large': tensor(0.2547),
   'map_medium': tensor(-1.),
   'map_per_class': tensor([0.8762, 0.0000, 0.1424, 0.0000]),
   'map_small': tensor(-1.),
   'mar_1': tensor(0.0450),
   'mar_10': tensor(0.2825),
   'mar_100': tensor(0.2825),
   'mar_100_per_class': tensor([0.8800, 0.0000, 0.2500, 0.0000]),
   'mar_large': tensor(0.2825),
   'mar_medium': tensor(-1.),
   'mar_small': tensor(-1.)}

```

**Figure 20.** Performance metric of object detection model, for hand-drawn test diagram

#### 4.6 Segment UML Class

Each predicted bounding box that are labelled as class are further segmented into 3 compartments that holds class name, attributes, and methods of that UML class, respectively. We used morphological operations to detect all separators (straight horizontal lines that run across the UML class), and iteratively merged area bounded by 2 separators at different y-position, into a single compartment. To detect lines, which are otherwise not perfectly straight as in the case of hand-drawn figures, we define the width of desired straight line (`str_element_width`) to be 7% of the total width of class. This reduction in width of structuring element, can result in multiple detected lines for a single horizontal separator. If there are multiple straight lines that are detected for a single horizontal separator, the lines are merged as a single line given that the distance between any 2 lines is within a `vertical_threshold` of 30%.

```

import cv2
import matplotlib.pyplot as plt
import time

def segment_uml_class(image, coordinates):

    xmin, ymin, xmax, ymax = map(int, coordinates)
    img = image.copy()[ymin: ymax, xmin: xmax]

    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(img_gray, 127, 255,
                           cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1])

    # Structuring element dimensions (width * height).
    # We take 7 % width for our mentioned bbox co-ordinate
    # Str El. Width can vastly differentiate the number of st.lines,
    # we detect.
    # More zigzag lines means, width should be kept low.

    temp = int(0.5 * len(img[0]))
    str_element_width = int(0.07 * len(img[0]))
    str_element_height = 1

    # Detect horizontal lines

    # MORPH_OPEN work by first eroding the image(removing the small blobs)
    # and then dilation of the image(adding small blobs)
    horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
                                                  (str_element_width, str_element_height))
    detect_horizontal = cv2.morphologyEx(thresh, cv2.MORPH_OPEN,
                                         horizontal_kernel, iterations= 1)
    cnts = cv2.findContours(detect_horizontal, cv2.RETR_EXTERNAL,
                           cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]

    # 'cnts' is a numpy.ndarray with shape N * 1 * 2
    # -> where N is the number of points
    # We need to merge the contours that is near each other in
    # vertical axis.
    # We represent the vertical threshold, as difference between 2

```

```

# contours, expressed a percentage

vertical_threshold = .3
count = 0
reduced_cnts = []

for c in cnts:
    cv2.drawContours(img, [c], -1, (12, 255, 36), 1)
    x, y, w, h = cv2.boundingRect(c)
    y = y + (h // 2)

    if not reduced_cnts:
        reduced_cnts.append(y)
    else:
        if abs(reduced_cnts[-1] - y) / 100 <= vertical_threshold:
            reduced_cnts[-1] = (reduced_cnts[-1] + y) // 2
        else:
            reduced_cnts.append(y)

imgplot = plt.imshow(img)
plt.show()

print(reduced_cnts)
return reduced_cnts

# For each of the bboxes that are labelled as class, segment the UML class
# Index the bboxes segments, with their respective (xmin, xmax)

bbox_segments = {}
captured_reln = []

for fb, fl in zip(filtered_boxes, filtered_labels):
    if fl == 'class':
        # fb has data in order [xmin, ymin, xmax, ymax]
        key = (int(fb[0]), int(fb[2]))
        y_diff = int(fb[1])

        # Rescale the y-coordinates to the original image co-ordinates
        y_pts = list(map(lambda x : x + y_diff, segment_uml_class(image, fb)))
        class_segment = []

```

```

for i in range(1, len(y_pts)):
    class_segment.append([y_pts[i-1], y_pts[i]])

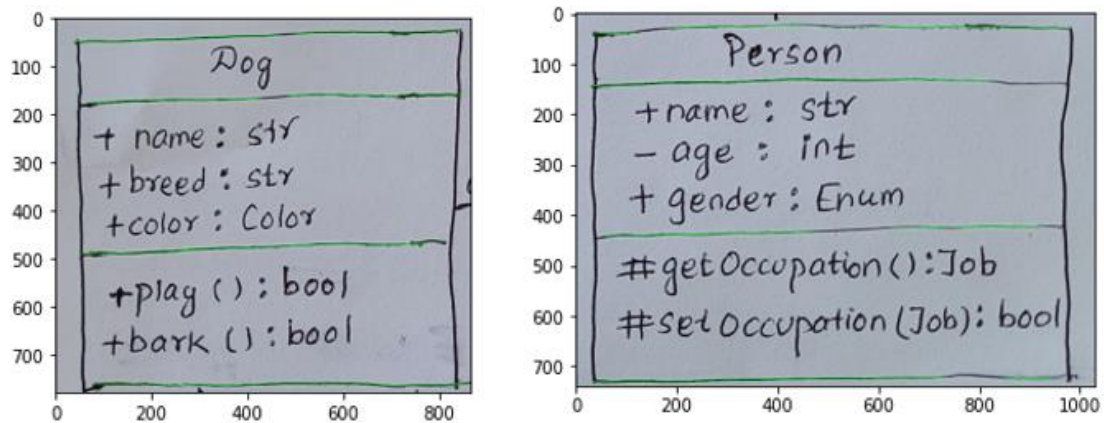
bbox_segments[key] = class_segment

else:
    captured_reln.append(fl)

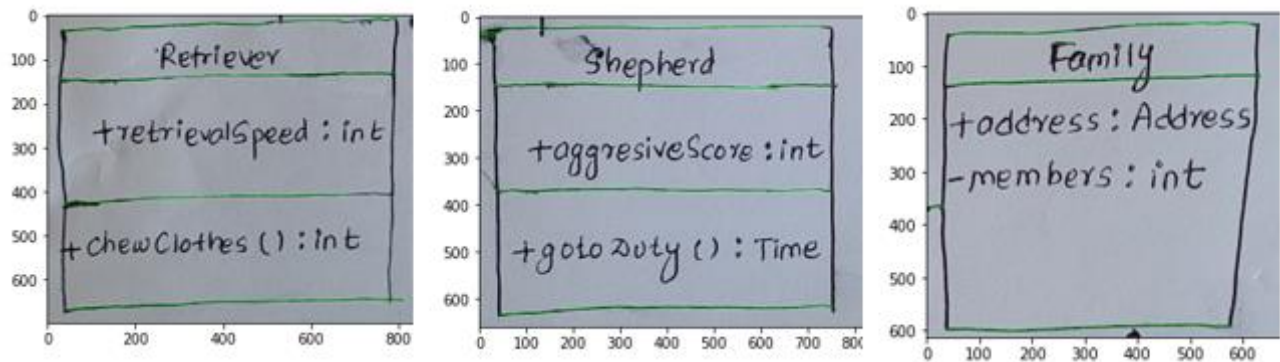
# Catch Exception for IndexError because, contours may not have length 4
# for every detected class, and classes that are falsely identified

# Plot all the class segments for each of the detected classes in image.
for k, v in bbox_segments.items():
    fig, axs = plt.subplots(1, 3)
    try:
        for i in range(len(v)):
            axs[i].imshow(
                image[v[i][1]: v[i][0], k[0]: k[1]]
            )
    except IndexError:
        print("Exception occurred in parsing UML class. \n \
            Reasons could be, \n \
                [wrong identification,\n \
                threshold for merging contours,\n \
                missing attributes or methods for class]\n \
            ")
    pass

```

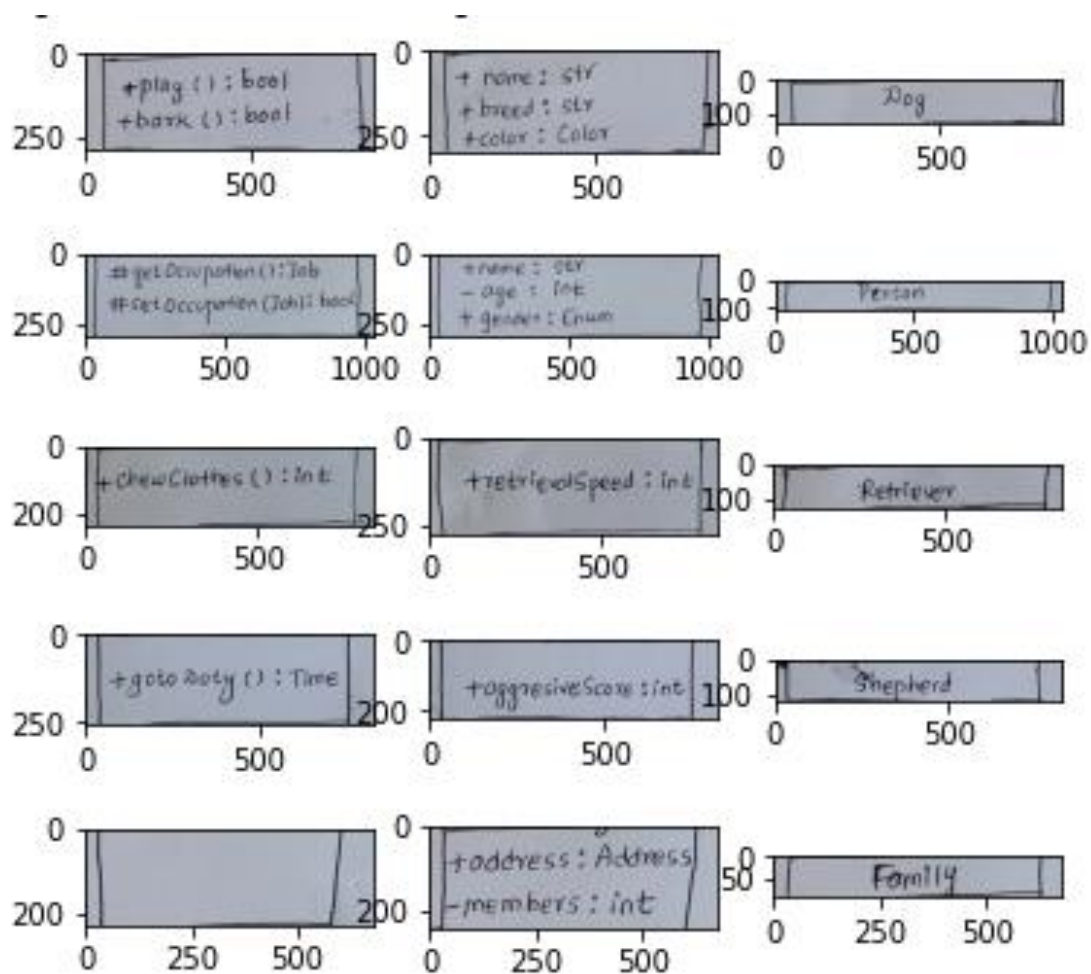


**Figure 21.** Straight lines (green) detected in test hand drawn image for UML class Dog (left) and, Person (right)



**Figure 22.** Straight lines (green) detected in test hand drawn image for UML class Retriever (left), Shepherd (middle) and, Family (right)





**Figure 23.** Segmented compartments for detected UML classes

#### 4.7 Text Extraction from segmented UML Class

All UML class segments are then passed through the text extraction framework easyocr, which uses CRAFT [8] to detect text regions in each of these segments, and CRNN [5] to recognize the text present inside the text regions. We used the pre-trained text recognition model — automatically downloaded, when invoking the Reader method—, to recognize the texts. The output text shown in Figure 24, are in the order of class methods, class attributes and class name for each

UML class. Further, all identified relationships between the UML classes are also depicted in

Figure 24.

```
import easyocr
from torchmetrics.text.wer import WER
from torchmetrics.text.cer import CharErrorRate

text_ground_truth = ['+ play(): bool + bark(): bool',
                     '+ name: str + breed: str + color: Color',
                     'Dog', '# getOccupation(): Job # setOccupation(Job):
bool',
                     '+ name: str - age: int + gender: Enum',
                     'Person',
                     '+ chewClothes(): int',
                     '+ retrievalSpeed: int',
                     'Retriever',
                     '+ gotoDuty(): Time',
                     '+ aggressiveScore: int', 'Shepherd',
                     '+ address: Address - members: int',
                     'Family']

# This needs to run only once to load the model into memory
# Subsequent calls to "Reader" method would not download pretrained model
# again.
reader = easyocr.Reader(['en'])
print("Text-prediction in detected objects: \n")
text_predictions = []
for k, v in bbox_segments.items():
    text_in_each_class = []
    for i in range(len(v)):
        temp = reader.readtext(image[v[i][1]: v[i][0], k[0]: k[1]],
                               detail = 0, x_ths = 10, paragraph = True)

        print(temp)

        text_in_each_class.extend(temp)

    print("\n")

    text_predictions.extend(text_in_each_class)

print("Captured Relationships: ", captured_reln, "\n\n")
```

```

↳ Text-prediction in detected objects:

['tPlag () ; boo | tbark () : boo/']
['t name : Sfy breed : stv tcoloy Coloy']
['Pod']

['#getOccpation (): Job #sel occpotion (Job): booll']
['tname : Str age iot + gender Enum']
['Person']

['Chew Clothes ( ) : in&']
['tvetrievlS peed : inl']
['"Retriever"']

['tgoto Att ( ; Time']
['todgresivescove :int']
['Shepherd']

[]
['taddyess Address ~members : it']
['Family']

Captured Relationships: ['inheritance', 'inheritance', 'inheritance']

```

**Figure 24.** Recognized text for all the class segments of each detected UML class

#### 4.8 Evaluate text extraction and recognition model

We computed the Character Error Rate (CRR) and Word Error Rate (WRR) for all recognized text, by comparing it with the ground truth text, in the input image. The results are as shown in Figure 25.

```
# Compute the word error rate, and char error rate of the handwriting
# recognition
word_metric = WER()
word_error = word_metric(text_predictions, text_ground_truth)
print("Word error rate: ", word_error)

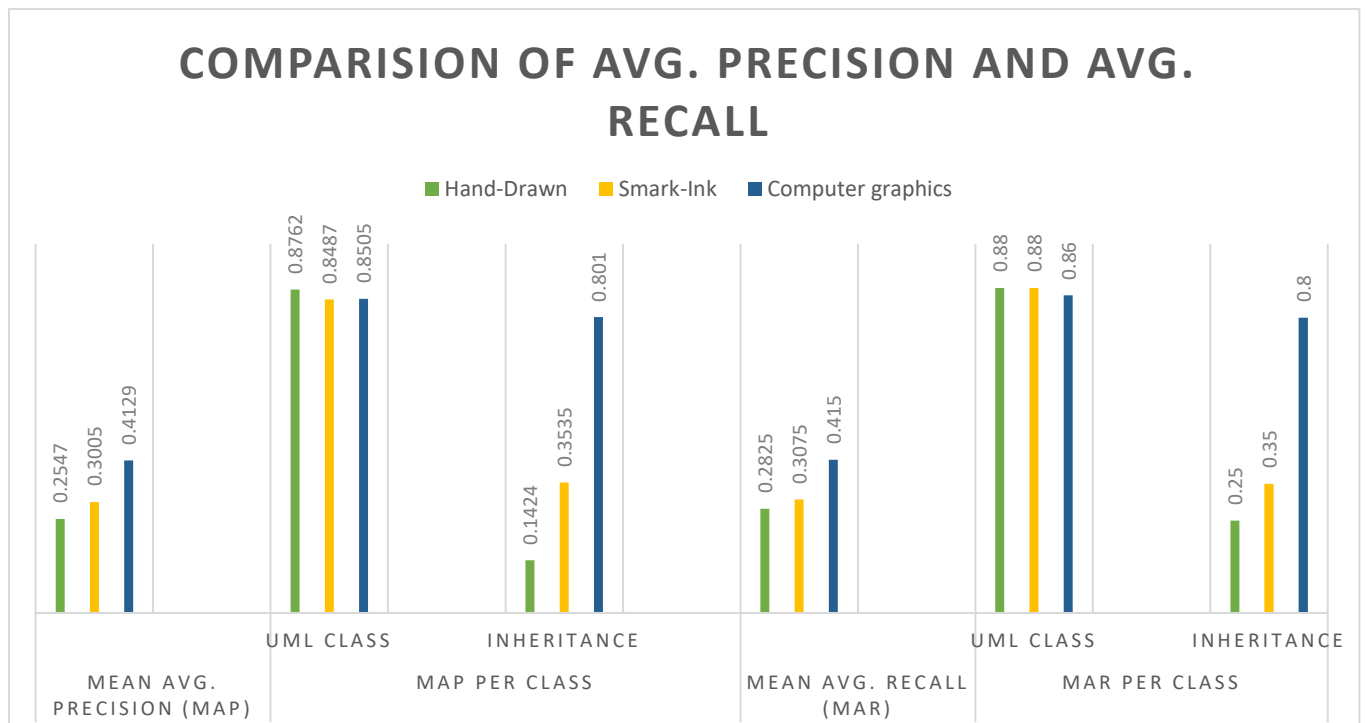
char_metric = CharErrorRate()
char_error = char_metric(text_predictions, text_ground_truth)
print("Character error rate: ", char_error)
```

```
Word error rate: tensor(1.0377)
Character error rate: tensor(0.3043)
```

**Figure 25.** Word error rate and Character error rate for detected and recognized text

## Chapter 5: Results

We evaluated the performance of our object detection model using mean Average Precision (mAP) and mean Average Recall (mAR). These metrics are computed as aggregate for entire model, as well as for each detected class type on our input data. Further, we also compared the model performance between 3 types of inputs i.e., hand drawn, smart-ink and, computer graphics generated UML class diagrams. To evaluate the performance of pre-trained text recognition model for detecting and recognizing text in our input, we calculated the Character Error Rate (CER) and Word Error Rate (WER). The model performance for each of these metrics on different sources of test data, is shown in Figure 26.



**Figure 26.** Comparison of Average Precision and Average Recall for all the test diagrams.

As labelled in the diagram, the green, orange, and blue legend represents the corresponding metric for hand-drawn, smart-ink and computer graphics generated diagrams respectively. We observed that the mean average precision for computer-generated image i.e., 0.4129, is much greater than the hand-drawn image (0.2547) and the smart-ink image (0.3005). This means that the trained model performed very well in determining accurate class labels out of all predicted class labels, for the computer graphics generated image as compared to the hand-drawn image. Although, any of these scores may not be good enough to be perceived as an average model performance, this is largely due to the small data samples which skewed the results. We can justify this by looking at the results of `MAP Per Class` for `UML class`, which is greater than 0.84 for all 3 types of inputs. The MAP for `INHERITANCE` class (especially the score of hand-drawn, 0.1424), is the primary contributor to the lower score for the overall MAP for the given type of input image. We can also see that, the Faster-RCNN trained model was able to accurately filter out correct class labels out of all real class labels present in image, by looking at `MAR Per Class` for `UML class` having score greater than 0.86 for all types of inputs. The lower score for overall MAR can be justified with same reasoning as with the overall MAP for the corresponding inputs.

We also calculated the CER and WER of the text recognition model to be 0.3043 and 1.0377 respectively. The lower the value of CER, the better the performance of the recognition system. In our case, the character error rate of 0.3043 can be observed as a good result. However, the word error rate of 1.0377 being greater than 1, signifies that the recognition system did not accurately predicted the words at all. This is because unlike characters, words are of different lengths and do not allow a clear comparison (a word is incorrectly recognized, if just one characters fails to match). Hence, Word Error Rate can be avoided to characterize the value of a model.

## Chapter 6: Conclusion and Future Work

Training a neural network for any sort of learning task requires large amount of data. The network can learn correct and distinctive features in the dataset, as well as identify co-relations between learned features. However, our dataset for training object detection model is significantly less than what is required for high precision identification of UML classifiers. Despite this, the trained model performed very well in terms of detecting most of the model classes (UML classes and inheritance). To improve model accuracy, in addition to collecting more data from disparate sources, we can perform data augmentation to populate new data points and pre-process our input to remove noise, enhance handwritten text and contours, scale image data to appropriate resolution etc. Further, we can leverage image processing techniques to approximate contours in the dataset and match those contours with output predictions from the model for high accuracy output and eliminate false positives. Our use case of parsing information from hand-drawn UML class diagrams involves complications such as, the need to segment and differentiate between similar UML classifiers and handle highly unstructured erroneous data. But, given that UML is a standard specification, it should be feasible to develop logic to handle most of these cases.

The pre-trained CRAFT + CRNN based text recognition model performed well to recognize text, in the test diagram. Since the model was trained on natural language rather than structured language (UML in our case) it incurred added complexity of trying to predict next character/word in the sequence by establishing dependency between current character/word in the sequence, which does not serve our use case very well. This is because, UML has specific standards on how to write attributes and behavior inside the UML class (For eg., access specifier (+ / - / #) attribute/ behavior: data type/ return type) which contrasts with free-flowing natural

language. Therefore, we could have an additional text recognition model trained specific to this use case, to consider only minimal or entirely no state specific information flowing from previous state, by manipulating the state propagation variable.

The ability to parse structural and textual information has lot of applications in requirement engineering. The object and text data can be converted into natural language and keyword-matched with software requirement specification document, to determine if the requirements are captured correctly in the design diagram. The parsed information can also be used, to compile automatic computer-graphics generated UML diagrams for documentation and future edits, using graphics library tools such as Graphviz. Moreover, this information can also be used to generate code templates in any desired programming language to speed up the development process and, reduce the complexity to manually verify the relationships between classes in same or different code modules.



## References

- [1] C. Gu, J. J. Lim, P. Arbelaez, and J. Malik, “Recognition using regions,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 1030–1037. doi: [10.1109/CVPR.2009.5206727](https://doi.org/10.1109/CVPR.2009.5206727).
- [2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587. doi: [10.1109/CVPR.2014.81](https://doi.org/10.1109/CVPR.2014.81).
- [3] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448. doi: [10.1109/ICCV.2015.169](https://doi.org/10.1109/ICCV.2015.169).
- [4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv:1506.01497 [cs]*, Jan. 2016.
- [5] B. Shi, X. Bai, and C. Yao, “An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition,” *arXiv:1507.05717 [cs]*, Jul. 2015.
- [6] H. Sak, A. Senior, and F. Beaufays, “Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition,” *arXiv:1402.1128 [cs, stat]*, Feb. 2014.
- [7] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: [10.1109/TIP.2003.819861](https://doi.org/10.1109/TIP.2003.819861).
- [8] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee, “Character Region Awareness for Text Detection,” *arXiv:1904.01941 [cs]*, Apr. 2019.
- [9] “A Beginner’s Guide to Neural Networks and Deep Learning,” *Pathmind*.  
<http://wiki.pathmind.com/neural-network>
- [10] “Backpropagation | Brilliant Math & Science Wiki.”  
<https://brilliant.org/wiki/backpropagation/>
- [11] “Convolution,” *Wikipedia*. Sep. 25, 2021.  
<https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1046465635>
- [12] “Convolutional neural network,” *Wikipedia*. Sep. 29, 2021.  
[https://en.wikipedia.org/w/index.php?title=Convolutional\\_neural\\_network&oldid=1047161725](https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1047161725)

- [13] vdumoulin, *Convolution arithmetic*. 2022. [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)
- [14] “What are Neural Networks?,” Aug. 03, 2021. <https://www.ibm.com/cloud/learn/neural-networks>
- [15] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” *Medium*, Dec. 17, 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- [16] “Jaccard index,” *Wikipedia*. Apr. 16, 2021. [https://en.wikipedia.org/w/index.php?title=Jaccard\\_index&oldid=1018153019](https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1018153019)
- [17] “Ramer–Douglas–Peucker algorithm,” *Wikipedia*. Nov. 30, 2020. [https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker\\_algorithm&oldid=991615121](https://en.wikipedia.org/w/index.php?title=Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm&oldid=991615121)
- [18] “Levenshtein distance,” *Wikipedia*. Sep. 20, 2021. [https://en.wikipedia.org/w/index.php?title=Levenshtein\\_distance&oldid=1045391132](https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=1045391132)
- [19] “Rectifier (neural networks),” *Wikipedia*. Jul. 22, 2021. [https://en.wikipedia.org/w/index.php?title=Rectifier\\_\(neural\\_networks\)&oldid=1034977791](https://en.wikipedia.org/w/index.php?title=Rectifier_(neural_networks)&oldid=1034977791)
- [20] “Sigmoid function,” *Wikipedia*. Sep. 17, 2021. [https://en.wikipedia.org/w/index.php?title=Sigmoid\\_function&oldid=1044853270](https://en.wikipedia.org/w/index.php?title=Sigmoid_function&oldid=1044853270)