Bilkent University

Department of Computer Engineering

# Senior Design Project

*Prelude*

## Low-Level Design Report

Group Members: Samet Özcan, Osman Orhan Uysal, Osman Burak İntişah, Mehmet Alper Karadağ, Ziya Erkoç

Supervisor: Uğur Güdükbay

Innovation Expert: Veysi İşler

Jury Members: Hamdi Dibeklioğlu, Shervin Rahimzadeh Arashloo

Low-Level Design Report

Oct 5, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492/1.

## Table of Contents

# 1. Introduction

## 1.1 Object design trade-offs

We have presented design goals which we strive to reach with our software. Still, we are aware that some of these goals are in conflict with each other, and in this section, we are to mention them.

### 1.1.1 Performance vs. Reliability

The customer textile company demanded a fast detection software from us so that quality control can be completed faster than before. At the same time, it wants a high level of accuracy. Yet, these both requirements are inversely proportional to each other. If we decrease the complexity of the network, detection is faster both less precise. At that point, we aim to put more emphasis on accuracy because the speed of our detection is upper-bounded by the maximum speed of the quality control machine. Hence, we will increase the speed no more than that of a quality control machine, therefore, we will be able to achieve greater accuracy by reducing the speed. Besides, accuracy is much more important for the company because missing defects pose a tremendous threat to the prestige of the company.

### 1.1.2 Usability vs. Reliability

We are going to create our data labeling software in a way that the textile experts can annotate the images quite fast so that they do not lose time. To that end, we will omit some confirmation boxes telling if they are sure about their action for the labeling process to be swift. However, this will hurt the quality of the defect labels. That in turn will decrease reliability because bad annotations will cause wrong training and eventually lead to lower accuracy. Therefore, in that case, we value usability more because the company wants its personnel not to lose much time during this process.

## 1.2 Interface documentation guidelines

We have used naming conventions used in Python programming. Words in a method and variable name are separated by an underscore (_) and all are in lowercase as in *method_name()* and *variable_name*. Our classes are in camel-case with initial character capitalized as in *ClassName*. We are using the below scheme to describe the classes. In the scheme, first, the functionality of the function is described and then variables are listed. Finally, methods of the class are listed along with their aims. Note that although there are no access modifiers or variable types in Python, we added them in documentation according to our intended usage.

| | | |
|---|---|---|
| **Description** | *Description of the class* | |
| **Attributes** | *public attribute_1: Type* | |
| | … | |
| **Methods** | *private method_1() : ReturnType* | *Description of the method* |
| | … | … |

### 1.3 Engineering standards (e.g., UML and IEEE)

We have used UML diagrams to effectively communicate the low-level system including package diagrams. We also make use of the IEEE convention to cite the references.

### 1.4 Definitions, acronyms, and abbreviations

**UML:** Unified Modelling Language. A modeling language for software engineering projects to visualize several aspects of the project in a standardized way [1].

**YOLO:** You Only Look Once. An object detection algorithm we are using to detect defects instead of the RCNN algorithm. [2]

**GUI:** Graphical User Interface. A set of visual elements such that users interact with to perform actions in or receive information from the computer.

## 2. Packages

Prelude’s low-level system is composed of 4 main subsystems, which are User Interface Layer, Machine Learning Layer, Report Layer, and Data Layer.

## 2.1 User Interface Layer

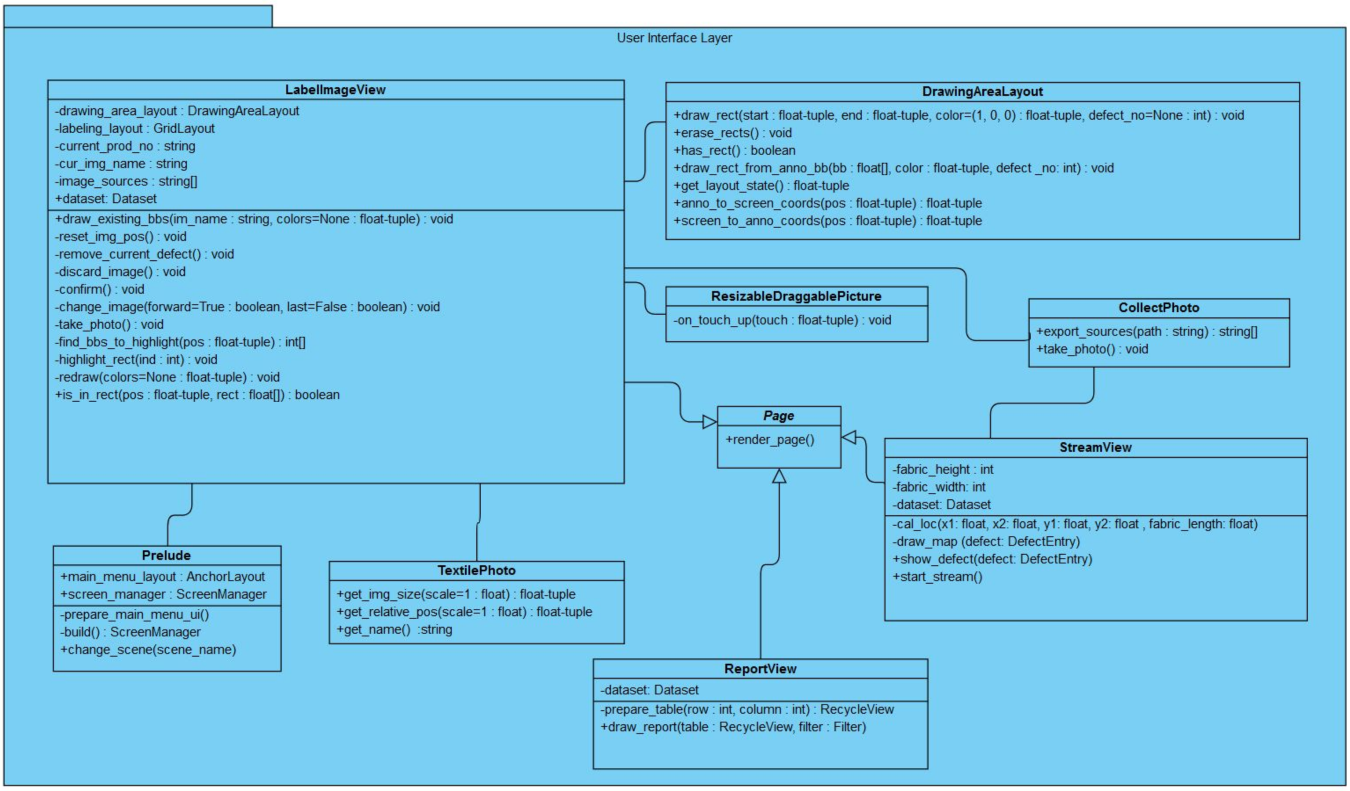
User Interface Layer
LabelImageView
-drawing_area_layout : DrawingAreaLayout
-labeling_layout : GridLayout
-current_prod_no : string
-cur_img_name : string
-image_sources : string[]
+dataset: Dataset
+draw_existing_bbs(im_name : string, colors=None : float-tuple) : void
-reset_img_pos() : void
-remove_current_defect() : void
-discard_image() : void
-confirm() : void
-change_image(forward=True : boolean, last=False : boolean) : void
-take_photo() : void
-find_bbs_to_highlight(pos : float-tuple) : int[]
-highlight_rect(ind : int) : void
-redraw(colors=None : float-tuple) : void
+is_in_rect(pos : float-tuple, rect : float[]) : boolean
DrawingAreaLayout
+draw_rect(start : float-tuple, end : float-tuple, color=(1, 0, 0) : float-tuple, defect_no=None : int) : void
+erase_rects() : void
+has_rect() : boolean
+draw_rect_from_anno_bb(bb : float[], color : float-tuple, defect _no: int) : void
+get_layout_state() : float-tuple
+anno_to_screen_coords(pos : float-tuple) : float-tuple
+screen_to_anno_coords(pos : float-tuple) : float-tuple
ResizableDraggablePicture
-on_touch_up(touch : float-tuple) : void
CollectPhoto
+export_sources(path : string) : string[]
+take_photo() : void
Page
+render_page()
StreamView
-fabric_height : int
-fabric_width: int
-dataset: Dataset
-cal_loc(x1: float, x2: float, y1: float, y2: float , fabric_length: float)
-draw_map (defect: DefectEntry)
+show_defect(defect: DefectEntry)
+start_stream()
Prelude
+main_menu_layout : AnchorLayout
+screen_manager : ScreenManager
-prepare_main_menu_ui()
-build() : ScreenManager
+change_scene(scene_name)
TextilePhoto
+get_img_size(scale=1 : float) : float-tuple
+get_relative_pos(scale=1 : float) : float-tuple
+get_name() :string
ReportView
-dataset: Dataset
-prepare_table(row : int, column : int) : RecycleView
+draw_report(table : RecycleView, filter : Filter)

Figure 1: Subsystem decomposition of User Interface Layer

User Interface Layer consists of 3 subsystems: Report View, Label Image View, Defect View. Views in this layer are responsible for the presentation of statistics and visualization of data. Also, Label Image View provides a feasible and appropriate way to label new data.

### Prelude

This class manages the scenes, prepares, and displays the main menu of the application to the user.

### Page

This abstract class has one abstract function called render_page() which is implemented in child classes.

### Label Image View

This class is responsible for providing a data labeling interface in which users can add, delete, and edit dataset elements (Image-annotation pairs).

### Drawing Area Layout

This class is an extension to the GridLayout of Kivy, a python GUI library. It is responsible for the management of the bounding boxes shown to the user. It also enables users to draw or erase new bounding boxes.

### Resizable Draggable Picture

This class is an extension to the Scatter widget of Kivy. It acts as a container to the image being worked on. It is responsible for the move and scaling operations on the image.

### Textile Photo

This class is an extension to the Image class of Kivy. It contains necessary additional information about the image shown on the screen such as its absolute position on the screen.

### Collect Photo

This class contains methods to utilize camera to take photos and use taken photos.

### Stream View

This class is an extension to the GridLayout of Kivy, a python GUI library. It is responsible for streaming the real-time photos taken from the line scanning camera, displaying the map which shows the relative positions of the defects on the fabric at a smaller scale.

### Report View

This class is responsible for preparing and showing statistical reports to the user. For instance, hole type defects in the last batch of fabric.

## 2.2 Machine Learning Layer

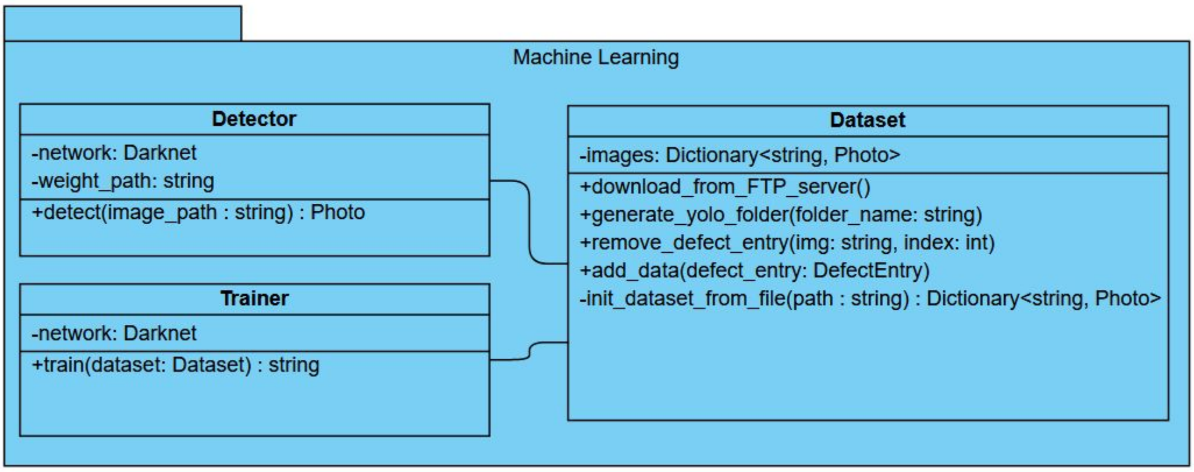
Machine Learning
Detector
-network: Darknet
-weight_path: string
+detect(image_path : string) : Photo
Trainer
-network: Darknet
+train(dataset: Dataset) : string
Dataset
-images: Dictionary<string, Photo>
+download_from_FTP_server()
+generate_yolo_folder(folder_name: string)
+remove_defect_entry(img: string, index: int)
+add_data(defect_entry: DefectEntry)
-init_dataset_from_file(path : string) : Dictionary<string, Photo>

Figure 2: Subsystem decomposition of Machine Learning Layer

The Machine Learning layer is responsible for encapsulating the Deep Learning algorithm and providing training, prediction, and dataset services to the other layers.

### Detector

This class contains the Darknet/YoloV4 algorithm's object instance.  It provides prediction functionality.

### Trainer

This class too includes a Darknet instance and is used for training the deep learning model.

### Dataset

This class encapsulates the images and provides auxiliary functions to synchronize with annotation text files and the textile company's servers.

## 2.3 Report Layer

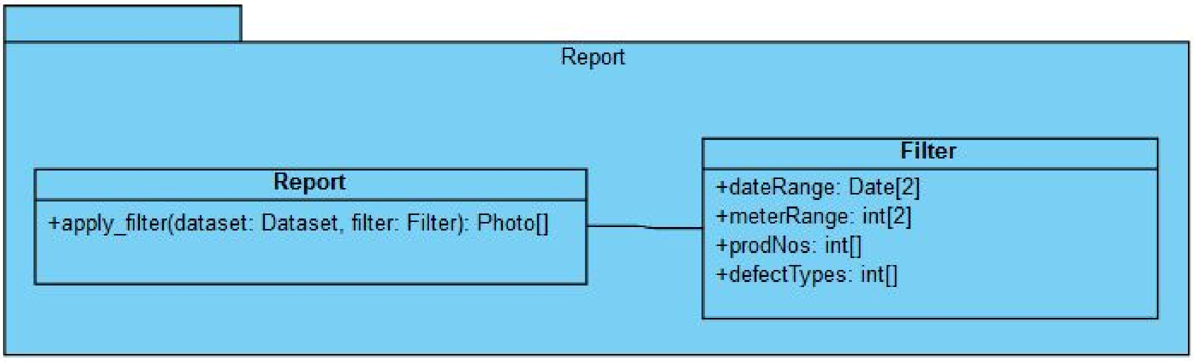

Figure 3: Subsystem decomposition of Report Layer

This layer consists of two classes; one for reporting the statistical results and one for mini-map for visualisation related to the defects in the current (in real time) or past fabric(s).

### Report

This class contains the method used for filtering the whole image dataset.

### Filter

This class includes data for creating reports and filtering photos.

## 2.4 Data Layer

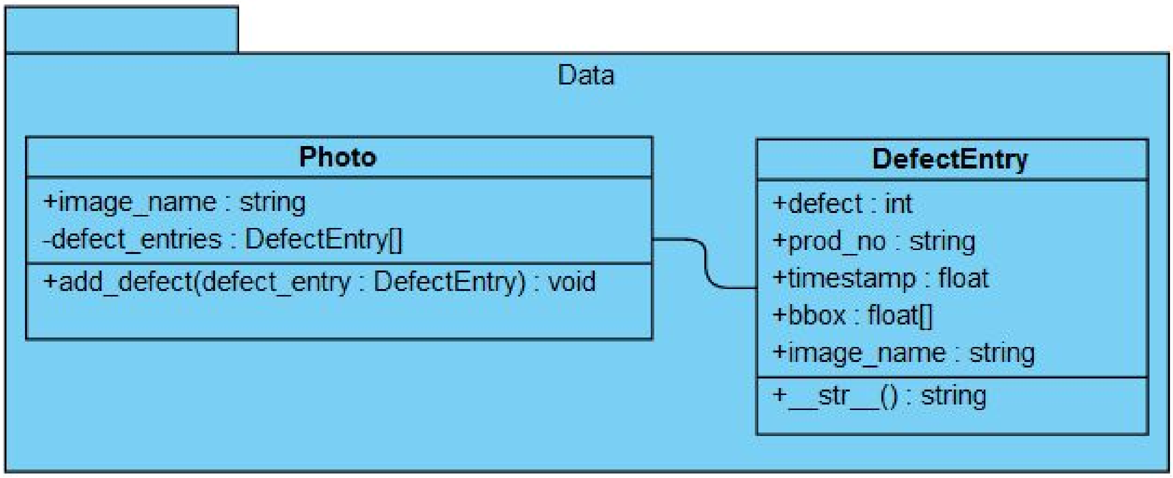

Figure 4: Subsystem decomposition of Data Layer

Data Layer is responsible for providing a fixed way of keeping the data entries throughout the application.

### Photo

This class is responsible for defining a way to keep the image data uniform across the application.

### Defect Entry

This class is responsible for defining a way to keep the defect data uniform across the application.

# 3. Class Interfaces

## 3.1 User Interface Layer

### Prelude

| | | |
|---|---|---|
| **Description** | This class manages the scenes, prepares, and displays the main menu of the application to the user. | |
| **Attributes** | public main_menu_layout : AnchorLayout | |
| | public screen_manager: ScreenManager | |
| **Methods** | private prepare_main_menu_ui() : void | Sets up the main menu. |
| | private build() : ScreenManager | Implementation of Kivy build a function that is necessary to build an application. |
| | public change_scene(scene_name : string) : void | Changes scene between the main menu and views in User Interface Layer. |

### Page

| | | |
|---|---|---|
| **Description** | This abstract class has one abstract function called render_page() which is implemented in child classes. | |
| **Methods** | public render_page() : void | An abstract method which will be implemented by child classes to place user interface |

| | | |
|---|---|---|
| | | elements and do drawing operations. |

### Label Image View

| | | |
|---|---|---|
| **Description** | This class is responsible for providing a data labeling interface in which users can add, delete, and edit dataset elements (Image-annotation pairs). | |
| **Attributes** | public dataset: Dataset | |
| | private drawing_area_layout : DrawingAreaLayout | |
| | private labeling_layout : GridLayout | |
| | private current_prod_no: string | |
| | private cur_img_name: string | |
| | private image_sources : string[] | |
| **Methods** | public draw_existing_bbs(im_name : string, colors=None : float-tuple) : void | Draws bounding boxes saved before and read into the annotations dictionary variable. |
| | private remove_current_defect() : void | Removes the bounding boxes, which are not saved, from the screen. |
| | private reset_img_pos() : void | Recenters and rescales the current image on the screen. |
| | private discard_image() : void | Removes the current image shown on the screen from the database. |
| | private confirm() : void | Adds the current image to the database. |
| | private change_image(forward=True : boolean, last=False : boolean) : void | Changes the image shown on the screen. |
| | private take_photo() : void | Calls take_photo function of Collect_photo class and |

|  |  |  |
|---|---|---|
|  |  | updates image_sources variable. |
|  | private find_bbs_to_highlight(pos : float-tuple) : int[] | Finds and returns the indexes of bounding boxes surrounding the given position on the screen. |
|  | private highlight_rect(ind : int) : void | Changes the color of the bounding box from red to green |
|  | private redraw() : void | Redraws the bounding boxes on the screen. |
|  | public static is_in_rect(pos : float-tuple, rect : float[]) : boolean | Checks whether the given point is in the given bounding_box and returns the result. |

### Drawing Area Layout

<table>
<tr><td><b>Description</b></td><td colspan="2">This class is an extension to the GridLayout of Kivy, a python GUI library. It is responsible for the management of the bounding boxes shown to the user. It also enables users to draw or erase new bounding boxes.</td></tr>
<tr><td rowspan="5"><b>Methods</b></td><td>public draw_rect(start : float-tuple, end : float-tuple, color=(1, 0, 0) : float-tuple, defect_no=None : int) : void</td><td>Draws a rectangle to screen from start to end.</td></tr>
<tr><td>public erase_rects() : void</td><td>Removes all rectangles from the screen.</td></tr>
<tr><td>public has_rect() : boolean</td><td>Returns whether there is a rectangle on the screen or not.</td></tr>
<tr><td>public draw_rect_from_anno_bb(bb : float[], color : float-tuple, defect _no: int) : void</td><td>Draw bounding box from annotation coordinates.</td></tr>
<tr><td>public get_layout_state() : float-tuple</td><td>Returns layout parameters such as scale and position.</td></tr>
</table>

| | | |
|---|---|---|
| | public anno_to_screen_coords(pos : float-tuple) : float-tuple | Converts annotation coordinates to screen coordinates using layout state parameters. |
| | public screen_to_anno_coords(pos : float-tuple) : float-tuple | Converts screen coordinates to annotation coordinates using layout state parameters. |

### Resizable Draggable Picture

| | | |
|---|---|---|
| **Description** | This class is an extension to the Scatter widget of Kivy. It acts as a container to the image being worked on. It is responsible for the move and scaling operations on the image. | |
| **Methods** | private on_touch_up(touch : float-tuple) : void | Overrides default touch behavior of Scatter widget to zoom on mouse wheel input and move on right button drag. |

### Textile Photo

| | | |
|---|---|---|
| **Description** | This class is an extension to the Image class of Kivy. It contains necessary additional information about the image shown on the screen. | |
| **Methods** | public get_img_size(scale=1 : float) : float-tuple | Returns current absolute image size depending on scale. |
| | public get_relative_pos(scale=1 : float) : float-tuple | Returns position of the image relative to the window. |
| | public get_name() : string | Returns image name. |

### Collect Photo

| | | |
|---|---|---|
| **Description** | This class contains methods to utilize cameras to take photos and use taken photos. | |
| **Methods** | public export_sources(path : string) : string[] | Returns a list of paths of the images taken. |
| | public take_photo() : void | Command camera to take |

| | | |
|---|---|---|
| | | photos. |

### Stream View

| | | |
|---|---|---|
| Description | This class is responsible for streaming the real-time photos taken from the line scanning camera, displaying the map which shows the relative positions of the defects on the fabric. | |
| Attributes | private fabric_height: int | |
| | private fabric_width: int | |
| | private dataset: Dataset | |
| Methods | private cal_loc(x1: float, x2: float, y1: float, y2: float , fabric_length: float) | Computes the location of the defect on the map with the given coordinates and fabric length information. |
| | private draw_map (defect: DefectEntry) | Draw the locations of the defects from annotation coordinates(proportionally). |
| | public show_defect(defect: DefectEntry) | Shows lastly detected defect |
| | public start_stream() | Starts image flow and continuously checks for defects |

### Report View

| | | |
|---|---|---|
| Description | This class is responsible for preparing and showing statistical reports to the user. For instance, hole type defects in the last batch of fabric. | |
| Attributes | private dataset: Dataset | |
| Methods | private prepare_table(row : int, column : int) : RecycleView | Prepares an empty table using Kivy's RecycleView |
| | public draw_report(table : RecycleView, filter : Filter) : void | Fills the given table and draws it on screen after applying the filter on the dataset via the |

|  |  |  |
|---|---|---|
|  |  | apply_filter method of Report class. |

## 3.2 Machine Learning Layer

### Detector

|  |  |  |
|---|---|---|
| **Description** | This class contains the Darknet/YoloV4 algorithm's object instance.  It provides prediction functionality. |  |
| **Attributes** | private network: Darknet |  |
|  | private weight_path: string |  |
| **Methods** | public detect(image_path : string) : Photo) | Detect all the defects on a single image |

### Trainer

|  |  |  |
|---|---|---|
| **Description** | This class too includes a Darknet instance and is used for training the deep learning model. |  |
| **Attributes** | private network: Darknet |  |
| **Methods** | public train(dataset: Dataset): string | Trains all the images located in the dataset and returns the name of the weight file containing weights of the network. |

### Dataset

|  |  |  |
|---|---|---|
| **Description** | This class encapsulates the images and provides auxiliary functions to synchronize with annotation text files and the textile company's servers. |  |
| **Attributes** | private images: Dictionary<string, Photo> |  |
| **Methods** | public download_from_FTP_server() | Fetches images and annotations from the FTP |

|  |  |  |
|---|---|---|
|  |  | server of the textile company |
|  | public  generate_yolo_folder(folder_name: string) | It creates a folder containing images and annotation text files in the format YoloV4 requires. |
|  | public void add_data(defect_entry: DefectEntry): | Adds the defect_entry in the parameter to the dataset and the annotation file. |
|  | private static init_dataset_from_file(path : string) : Dictionary<string, Photo> | Creates and returns a dictionary from the given file. |
|  | private void remove_defect_entry(img_name : string, entry_index : int, all=False: boolean) | Removes the given defect entry from the dataset and the annotation file. |

## 3.3 Report Layer

**Report**

|  |  |  |
|---|---|---|
| **Description** | This class contains the method used for filtering the whole image dataset. |  |
| **Methods** | public static apply_filter(dataset : Dataset, filter: Filter): Photo[] | Applies filters on all the photos in the dataset object and returns matched photos. |

**Filter**

|  |  |
|---|---|
| **Description** | This class includes data for creating reports and filtering photos. |
| **Attributes** | public date_range: Date[2] |
|  | public meter_range: int[2] |
|  | public prod_nos: int[] |
|  | public defect_types: int[] |

## 3.4 Data Layer

### Photo

| | | |
|---|---|---|
| **Description** | This class is responsible for defining a way to keep the image data uniform across the application. | |
| **Attributes** | public image_name : string | |
| | private defect_entries : DefectEntry[] | |
| **Methods** | public add_defect(defect_entry : DefectEntry) : void | Adds defect to the defect_entries list |

### Defect Entry

| | |
|---|---|
| **Description** | This class is responsible for defining a way to keep the defect data uniform across the application. |
| **Attributes** | public defect : int |
| | public prod_no : string |
| | public timestamp : float |
| | public bbox : float[] |
| | public image_name : string |

# 4. Glossary

**Kivy:** GUI library for Python. [3]

**Darknet:** A neural network structure that runs YOLO algorithms. [4]

**GridLayout:** A layout in Kivy that divides the screen into rows and columns.

# 5. References

[1] [Unified Modelling Language]

https://en.wikipedia.org/wiki/Unified_Modeling_Language last accessed 4.10.2020

[2] [Darknet Convolutional Neural Network]

https://github.com/pjreddie/darknet last accessed 4.10.2020

[3] [Kivy GUI Library]

https://kivy.org/ last accessed 4.10.2020

[4] [YOLO Algorithm]

https://pjreddie.com/darknet/yolo/ last accessed 4.10.2020