Low Level Design (HLD)

Mask Wear Detector with Computer Vision

**Document Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 20 October 2023 | 1 | Initial LLD – V1.0 | Nitin Udmale |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

1. Introduction………………………………………………………..…………………………………...4

1.1 Why this Low-Level Design Document ? ………………………………….……...………4

1.2 Scope………………………………………………………………………………………….4

2. Architecture…………….……………………………………………………………………………….5

3. Architecture Description……………………………………………………………………………….6

3.1 Data Description…..………………………………………………………………...……….6

3.2 Data Preprocessing...……………………………………………………………………….6

3.3 Model Trainer…………..……………………………………………………………………..6

3.4 Model Evaluation……………………………………………………………………………..6

3.5 Data From User...…………………………………………………………………………….6

4. Unit Test Cases………………………………………………………………………………………...7

1. **Introduction**

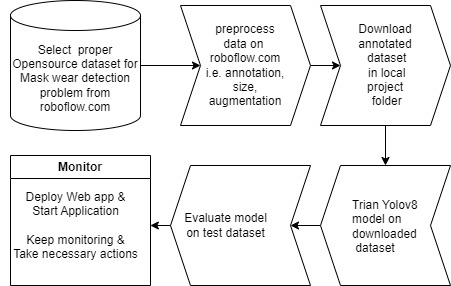
**1.1. What is Low-Level design document?**

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code for Mask wear detector System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

## **1.2. Scope**

Low-level design (LLD) is a component-level design process that follows a step-by step [refinement](https://en.wikipedia.org/wiki/Refinement_(computing)) process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work

1. **Architecture**



1. **Architecture Description**

## **3.1 Data Description**

We will be using Roboflow.com for data collection, annotation , preprocessing in this project. We will be using opensource annotated datasets for face mask detection having three classes “Proper Mask”, “No Mask” and “Improper Mask”. Dataset contains 5841 images( train set : 5098-87% , Valid set : 495-8% , test set: 248-4%)

Dataset Link - <https://universe.roboflow.com/new-workspace-2cnfr/mask-ecop7/dataset/2>

## **3.2 Data Preprocessing**

In this process, we will resize images to 640x640. Apply augmentation if needed. Check whether all the images are annotated properly

## **3.3 Model Trainer**

In this step, train yolov8 model on jupyter notebook as per instruction from ultralytics model training documents. Following below link :

<https://docs.ultralytics.com/modes/train/>

## **3.4 Model Evaluation**

Trained model will be evaluated on test dataset. Trained model from previous step will be compared with model already in production, whichever is the best model, it will forwarded for model pusher component.

## **3.5 Data from User**

Here we will create web page application, using streamlit framework. User will give input to this page, data from page will be used for prediction purpose. Prediction will be done by model in production which will handle preprocessing of data and prediction. Result we be shown on the webapp itself containing bounding boxes and labels.

**4.Unit Test Cases**

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| Verify whether the Application URL is accessible to the user | 1. Application URL should be defined | Application URL should be accessible to the user |
| Verify whether the Application loads completely for the user when the URL is accessed | 1. Application URL is accessible 2. Application is deployed | The Application should load completely for the user when the URL is accessed |
| Verify whether the User is able to sign up in the application | 1. Application is accessible | The User should be able to sign up in the application |
| Verify whether user is able to successfully login to the application | 1. Application is accessible 2. User is signed up to the application | User should be able to successfully login to the application |
| Verify whether user is able to see input fields on logging in | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be able to see input fields on logging in |
| Verify whether user is able to edit all input fields | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be able to edit all input fields |
| Verify whether user gets Predict button to submit the inputs | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should get Submit button to submit the inputs |
| Verify whether user is presented with recommended results on clicking submit | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | User should be presented with recommended results on clicking submit |
| Verify whether the recommended results are in accordance to the selections user made | 1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application | The recommended results should be in accordance to the selections user made |

## 