

# AI-BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMA

A.THENNARASU(TL)  
R.SENTHILKUMAR  
S.SANJAYKUMAR  
P.M.REVANTHKUMAR  
S.DEEPAK

# Table of Contents

Chapter No.	Title	Page No.
<b>1</b>	<b>INTRODUCTION</b>	
1.1	Project Overview	5
1.2	Purpose	5
<b>2</b>	<b>LITERATURE SURVEY</b>	
2.1	Existing problem	6
2.2	References	6
2.3	Problem Statement Definition	6
<b>3</b>	<b>IDEATION &amp; PROPOSED SOLUTION</b>	
3.1	Empathy Map Canvas	7
3.2	Ideation & Brainstorming	7
3.3	Proposed Solution	10
3.4	Problem Solution fit	12
<b>4</b>	<b>REQUIREMENT ANALYSIS</b>	
4.1	Functional requirement	13
4.2	Non-Functional requirements	13
	Non-Functional requirements	13
<b>5</b>	<b>PROJECT DESIGN</b>	
5.1	Data Flow Diagrams	14
5.2	Solution & Technical Architecture	14

5.3	User Stories	15
<b>6</b>	<b>PROJECT PLANNING &amp; SCHEDULING</b>	
6.1	Sprint Planning & Estimation	17
6.2	Sprint Delivery Schedule	18
6.3	Reports from JIRA	20
<b>7</b>	<b>CODING &amp; SOLUTIONING</b>	
7.1	Feature 1	22
7.2	Feature 2	25
7.3	Database Schema	36
<b>8</b>	<b>TESTING</b>	
8.1	Test Cases	38
8.2	User Acceptance Testing	39
<b>9</b>	<b>RESULTS</b>	
9.1	Performance Metrics	40
<b>10</b>	<b>ADVANTAGES &amp; DISADVANTAGES</b>	41
<b>11</b>	<b>CONCLUSION</b>	42
<b>12</b>	<b>FUTURE SCOPE</b>	42
<b>13</b>	<b>APPENDIX</b>	
13.1	Source Code	43
13.2	GitHub & Project Demo Link	59

## List of Figures

Figure	Name of the Figure	Page number
1	Empathy Map Canvas	7
2	Ideation and Brainstorming	8
3	Problem Solution Fit	12
4	Data Flow Diagrams	14
5	Solution Architecture Diagram	15
6	Technical Architecture	15
7	Burndown Chart	19
8	Roadmap	20
9	Backlog	21
10	Board	21

# INTRODUCTION

## 1.1 Overview

Erythema is the redness of the skin or mucous membranes, caused by hyperaemia in the superficial capillaries. If these diseases are not treated at an early stage, they can cause complications in the body, including the spread of infection from one person to another. The skin diseases can be prevented by investigating the infected region at an early stage. The characteristics of the skin images are diversified, so it is a challenging job to devise an efficient and robust algorithm for the automatic detection of skin disease and its severity. Skin tone and skin colour play an important role in skin disease detection. The colour and coarseness of skin are visually different. Automatic processing of such images for skin analysis requires a quantitative discriminator to differentiate the diseases.

## 1.2 Purpose

To overcome the above problem, we are building an AI-based model that is used for the prevention and early detection of erythema. Basically, skin disease diagnosis depends on different characteristics like colour, shape, texture, etc. Here, the user can capture images of their skin, which are then sent to the trained model, where the information is processed using image processing techniques and then extracted for machine interpretation. The pixels in the image can be manipulated to achieve any desired density and contrast. Finally, the model generates a result and determines whether or not the person has skin disease. Image processing technologies significantly reduce the time spent on a specific activity by the customer. Hence, it is a time- and money-saving process.

# LITRATURE SURVEY

## 2.1 Existing Problem

Although computer-aided diagnosis (CAD) is used to improve the quality of diagnosis in various medical fields such as mammography and colonography, it is not used in dermatology, where non-invasive screening tests are performed only with the naked eye, and avoidable inaccuracies may exist. This study shows that CAD may also be a viable option in dermatology by presenting a novel method to sequentially combine accurate segmentation and classification models. Given an image of the skin, we decompose the image to normalize and extract highlevel features. Using a neural network-based segmentation model to create a segmented map of the image, we then cluster sections of abnormal skin and pass this information to a classification model. We classify each cluster into different common skin diseases using another neural network model. Our segmentation model achieves better performance compared to previous studies, and also achieves a near-perfect sensitivity score in unfavourable conditions. Our classification model is more accurate than a baseline model trained without segmentation, while also being able to classify multiple diseases within a single image. This improved performance may be sufficient to use CAD in the field of dermatology.

## 2.2 References

<https://link.springer.com/article/10.1007/s11042-021-11823-x>

[https://link.springer.com/chapter/10.1007/978-981-19-0863-7\\_10](https://link.springer.com/chapter/10.1007/978-981-19-0863-7_10)

<https://iopscience.iop.org/article/10.1088/1757-899X/1076/1/012045>

<https://onlinelibrary.wiley.com/doi/full/10.1002/ski2.81>

<https://www.sciencedirect.com/science/article/pii/S1877050919321295>

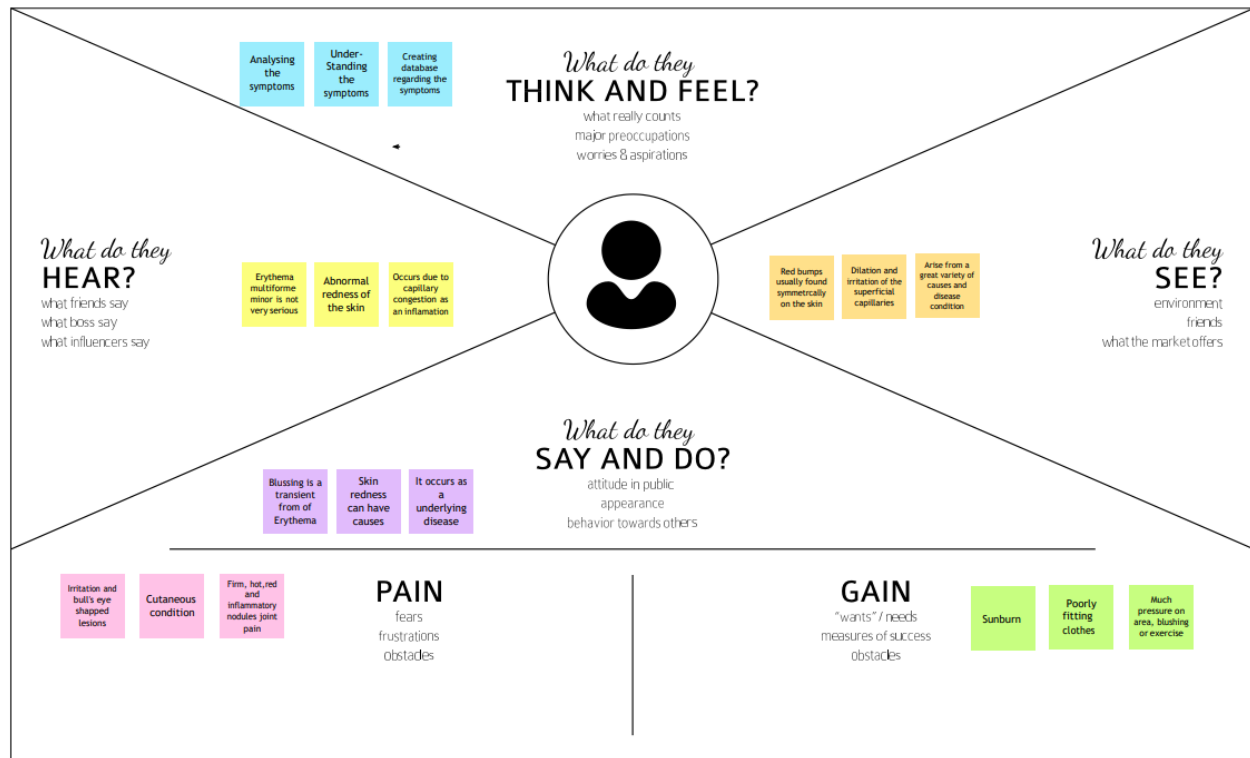
## 2.3 Problem Statement Definition

Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love. A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

# **IDEATION & PROPOSED SOLUTION**

## 3.1 Empathy Map Canvas

Empathy Map Canvas: An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes. It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.



### 3.2 Ideation & Brainstorming


Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.



Step-1:

## Team Gathering, Collaboration and Select the Problem Statement

Template



# Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

🕒 10 minutes to prepare  
🕒 1 hour to collaborate  
👤 2-8 people recommended

DESIGNED BY :

A.Thennarasu (TL)  
S.Deepak  
S.Sanjaykumar  
R.Senthilkumar  
P.M.Revanthkumar

[Share template feedback](#)

➔

### Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 10 minutes

📋

**Team gathering**  
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

📋

**Set the goal**  
Think about the problem you'll be focusing on solving in the brainstorming session.

📋

**Learn how to use the facilitation tools**  
Use the Facilitation Superpower to run a happy and productive session.

[Open article](#) ➔

1

### Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

🕒 5 minutes

📌

🎯

To build a web application powered with AI-based localization and classification of skin diseases with anythema

🧠

### Key rules of brainstorming

To run a smooth and productive session

🗣️ Stay in topic.

💡 Encourage wild ideas.

⏸️ Defer judgment.

👂 Listen to others.

🗣️ Go for volume.

👁️ If possible, be visual.

## Step-2:

### Brainstorm, Idea Listing and Grouping

2

#### Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes



3

#### Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes



Step-3:

## Idea Prioritization

4

## Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



### 3.3 Proposed Solution

To overcome the problems due to Erythema, we are building an AI- model that is used for the early detection and prevention of Erythema by investigating the infected region.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	User is a busy worker who needs an immediate result with more accuracy for his/her skin problem but he/she has no time to visit dermatologists in-person.
2.	Idea / Solution description	The person can capture the images of skin and then the image will be sent to the trained model. The model analyses the image and detects whether the person is having skin disease or not.
3.	Novelty / Uniqueness	Images with noise have also been taken and are enhanced with effective algorithms for predicting the diseases.
4.	Social Impact / Customer Satisfaction	By just uploading the images various skin diseases can be diagnosed and this system is very efficient which serves civilians to detect the diseases earlier.
5.	Business Model (Revenue Model)	As we are planning to design a proprietary product as a solution and distribute it to users, this will serve as our return on investment.
6.	Scalability of the Solution	This system is more scalable because it takes any type of images regardless of its resolution and it provides high performance irrespective of the environment.

## 3.4 Problem Solution Fit

Define CS, fit into CC	<div>1. CUSTOMER SEGMENT(S)<div>CS</div></div> <div>Who is your customer? i.e. working parents of 0-5 y.o. kids</div> <div>All the patients including child, adult and old age people.</div>	<div>6. CUSTOMER CONSTRAINTS<div>CC</div></div> <div>What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.</div> <div>The cost and budget aspects constraints a patient to take necessary action.</div>	<div>5. AVAILABLE SOLUTIONS<div>AS</div></div> <div>Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros &amp; cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking</div> <div>A portal or chat window ( basically a computer program) can help in making a platform for conversation between patient and doctor to solve their concerns.</div>	Explore AS, differentiate
	<div>2. JOBS-TO-BE-DONE / PROBLEMS<div>J&amp;P</div></div> <div>Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides.</div> <div>Delayed test reports or vague reports on the diagnosis can be considered as a problem.</div>	<div>9. PROBLEM ROOT CAUSE<div></div></div> <div>What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations.</div> <div>Even though a patient can consult a doctor in-person and gets analysis on his conditions, it generally takes quite a lot of time and physical work.</div>	<div>7. BEHAVIOUR<div></div></div> <div>What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)</div> <div>A chatbot which can interpret a lot of intents that are being provided by a patient and be able to prescribe medications based on the diagnosis. These chatbots have to be supporting 24 X 7 and should provide a quick response, irrespective of the number of patients ping the system.</div>	
Focus on J&P, tap into BE, understand RC				
Identify strong TR & EM	<div>3. TRIGGERS<div>TR</div></div> <div>What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news.</div> <div>The ability to diagnose a disease real quick and get a quick response from the hospital.</div>	<div>10. YOUR SOLUTION<div>SL</div></div> <div>If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour.</div> <div>Patients should be made aware of the solutions that are being provided to solve their issues.</div>	<div>8. CHANNELS OF BEHAVIOUR<div>CH</div></div> <div>8.1 ONLINE What kind of actions do customers take online? Extract online channels from #7</div> <div>Quick approach to the online portals or chatbots.</div>	Identify strong TR & EM
	<div>4. EMOTIONS: BEFORE / AFTER<div>EM</div></div> <div>How do customers feel when they face a problem or a job and afterwards? i.e. lost, insecure &gt; confident, in control - use it in your communication strategy &amp; design.</div> <div>It makes a patient to feel depressed and worried before and it makes him/her to feel confident and hospitalized after.</div>		<div>8.2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.</div> <div>Try to reach the hospital and get clarified on their queries.</div>	

## REQUIREMENT ANALYSIS

### 4.1 Functional Requirements

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	<b>User Registration process</b>	Registration through Mobile Number Registration through Google Account Registration through Facebook
FR-2	<b>User Confirmation</b>	Confirmation via E-mail Confirmation via Call Confirmation via One Time Password
FR-3	<b>Patient Image Capturing Process</b>	Provide Access to Capture Image Through Camera Provide Access to Upload Image Through Gallery
FR-4	<b>Patient Medicine Reminder</b>	Remind the Patients to take their Medicines/ointments At right time through remaindering alarm.
FR-5	<b>Suggestion Box</b>	Patients can take suggestions from the Doctors through Chats.
FR-6	<b>Flareup Cycles</b>	Patients can know their medicine level from doctors Through message.

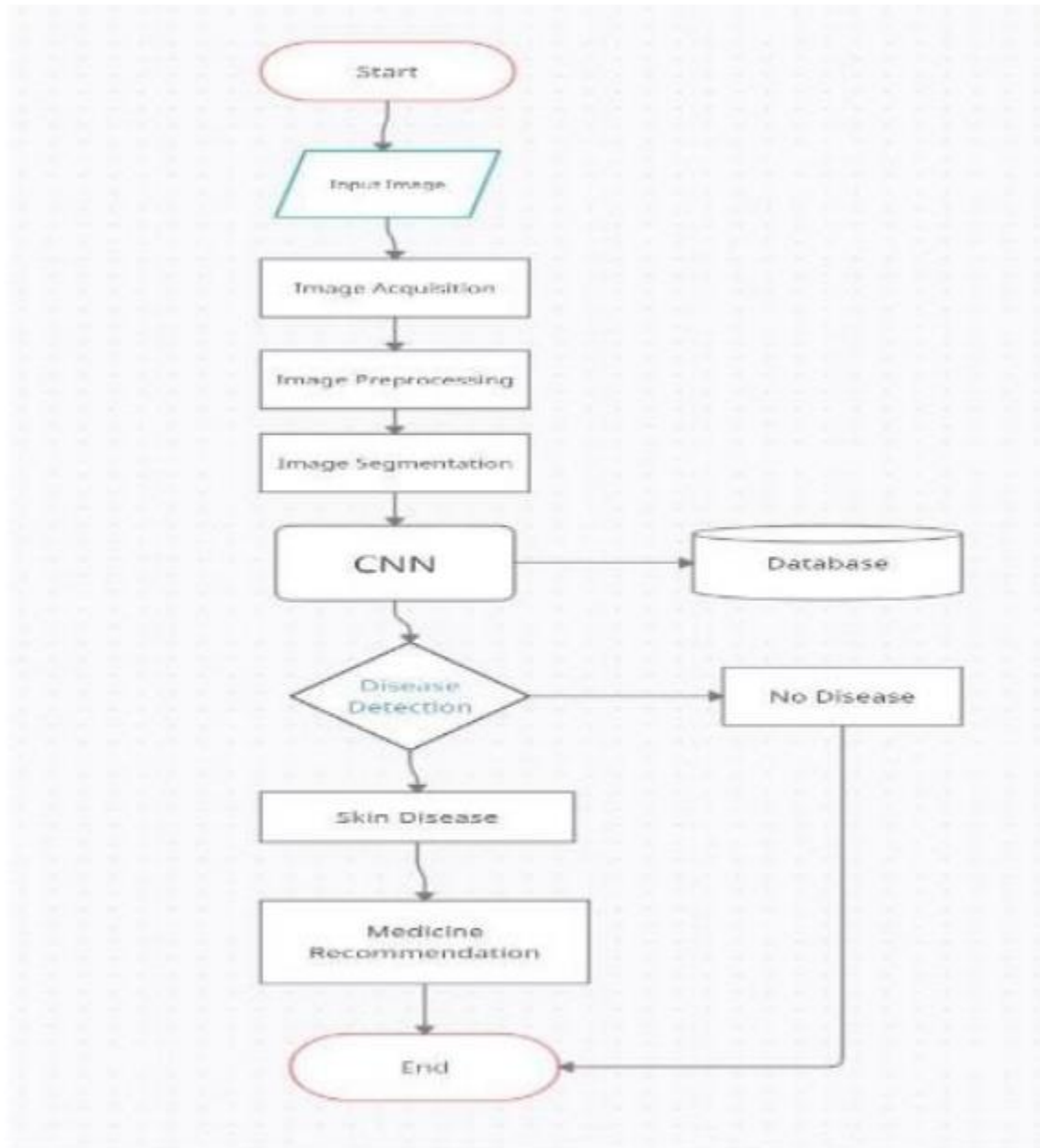
## 4.2 Non-functional Requirements

FR No.	Non-Functional Requirement	Description
NFR-1	<b>Usability</b>	It doesn't specify parts of the system functionality, only how that functionality is to be perceived by the user
NFR-2	<b>Security</b>	Data privacy and security practices may vary based on users and their age
NFR-3	<b>Reliability</b>	Extent to which the software system consistently performs the specified functions without failure.
NFR-4	<b>Performance</b>	The website's load time should not be more than one second for users
NFR-5	<b>Availability</b>	How likely the system is accessible to a user at a given point in time
NFR-6	<b>Scalability</b>	The ability to appropriately handle increasing (and decreasing) workloads.

# PROJECT DESIGN

## 5.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



## 5.2 Solution Architecture:

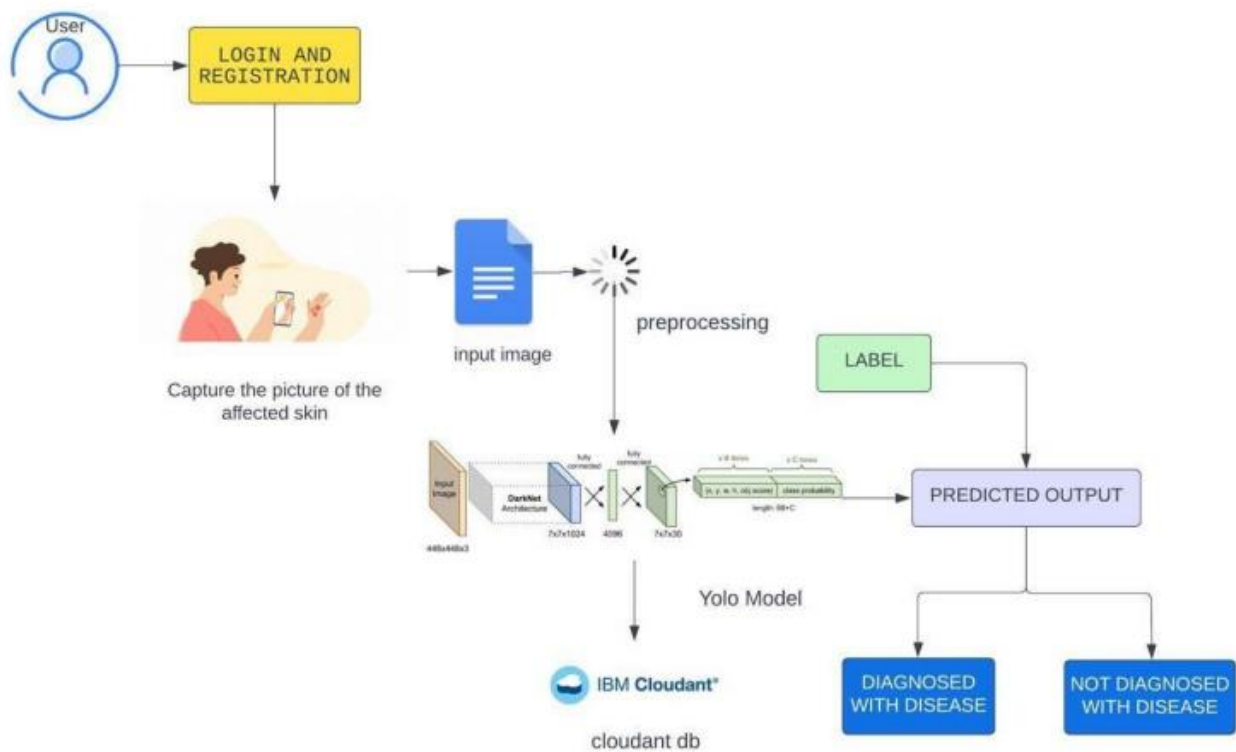
Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

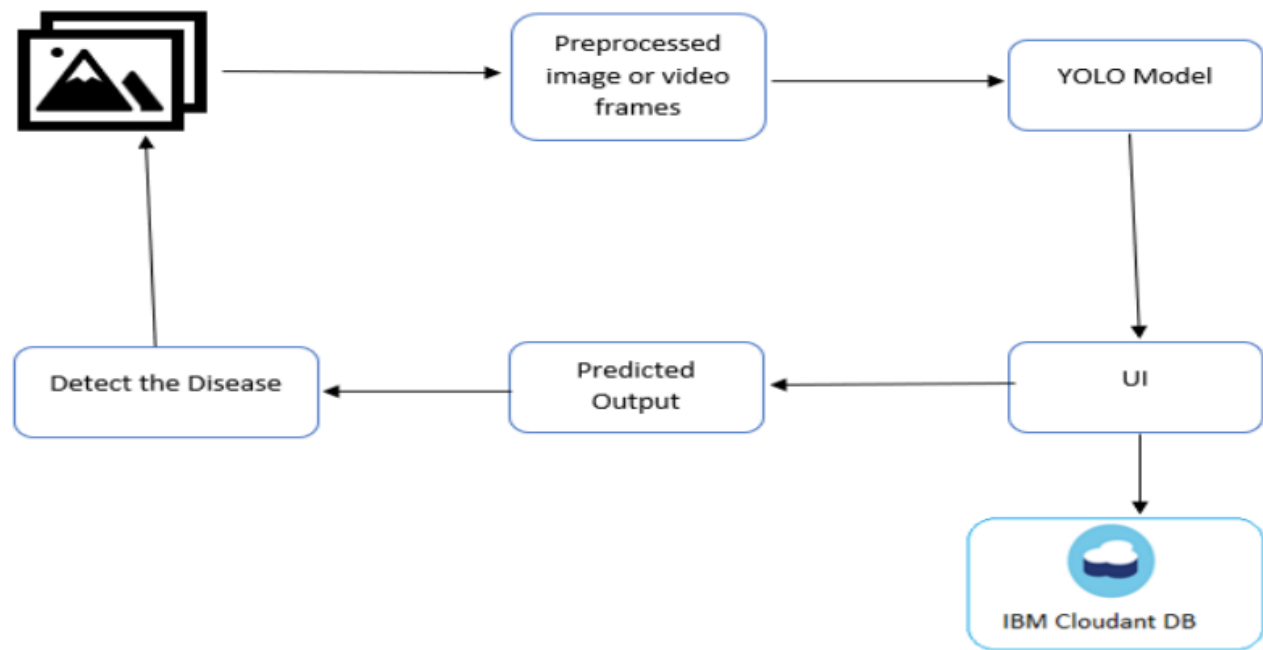


- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

### Solution Architecture Diagram:



## Technical Architecture:



## 5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account/ dashboard.	High	Sprint-1
	Confirmation	USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1

	Login	USN-3	As a user, I can login for the application through Gmail	I can access my account/ dashboard	medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can access my account/ dashboard	high	Sprint-1
	Dashboard	USN-5	As a user, I can see the my profile, medical history, upload image, getting report services provided by the application	I can get into one of the services and use	medium	Sprint-2
	Data input	USN-6	As a user, I can upload the images of the affected skin area	I can submit it to the application	high	Sprint-2
Administrator	Train model	USN-7	As a administrator , I can train a model to compare the images uploaded with the images in the database to detect the disease	I can test the model whether it meets the criteria	high	Sprint-3
Trained model	Image processing	USN-8	By comparing the images the disease will be detected with the given datasets	All the necessary operation performed and information extracted.	high	Sprint-3
	Report generation	USN-9	Based on the detection of disease, report generated	The results will be shown on the screen to the patients.	high	Sprint-4

## PROJECT PLANNING & SCHEDULING

### 6.1 Sprint Planning & Estimation

Product Backlog, Sprint Schedule, and Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register and login into application by entering my email, password, and confirming my password.	7	High	Senthil kumar R Deepak S Thennarasu A
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application and correct details about medical report	6	High	Senthil kumar R Deepak S
Sprint-2	Register/Screening	USN-3	As a user, I can register for the application through mobile and find the method more efficient and accurate	1 2	High	Thennarasu A Sanjay kumar S
Sprint-1		USN-4	As a user, I can use it physical interaction with device	3	Medium	Senthil kumar R
Sprint-4	Login	USN-5	As a user, I can use the database and software installed I can log into the application by entering login credentials	7	High	Deepak S Revanth kumar P M
Sprint-2		USN-6	As a user, find the light weight	8	high	Revanth kumar P M
Sprint-3	Safety/Testing	USN-7	As a user, I can be safe as the detection method and undergo testing without fear of pain	6	Medium	Senthil kumar R Sanjay kumar S
Sprint-3	Dashboard	USN-8	As a user, I can upload my images and get my details of skin diseases	5	High	Sanjay kumar S Deepak S
Sprint-1	logout	USN-7	As a user, I can logout successfully	4	Medium	Deepak S Thennarasu A
Sprint-4	Feedback	USN-8	As a user/customer care executive, I can be able to interact with all and get their feedback from them which can be used to enhance the scope of projects	7	Medium	Deepak S Thennarasu A
Sprint-3		USN-9	As a user, I can conduct the awareness among people to undergo frequent medical check up	4	Medium	Sanjay kumar S Senthil kumar R
Sprint-3		USN-10	As a user, I can get the results immediately after screening process	5	Medium	Deepak S
Sprint-4		USN-11	Based on the prediction of skin diseases, the health care report generated to provide feedbacks	6	Medium	Thennarasu A

## 6.2 Sprint Delivery Schedule

Project Tracker, Velocity & Burndown Chart:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	07 Nov 2022	20	07 Nov 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	14 Nov 2022
Sprint-4	20	6 Days	07 Nov 2022	19 Nov 2022	20	19 Nov 2022

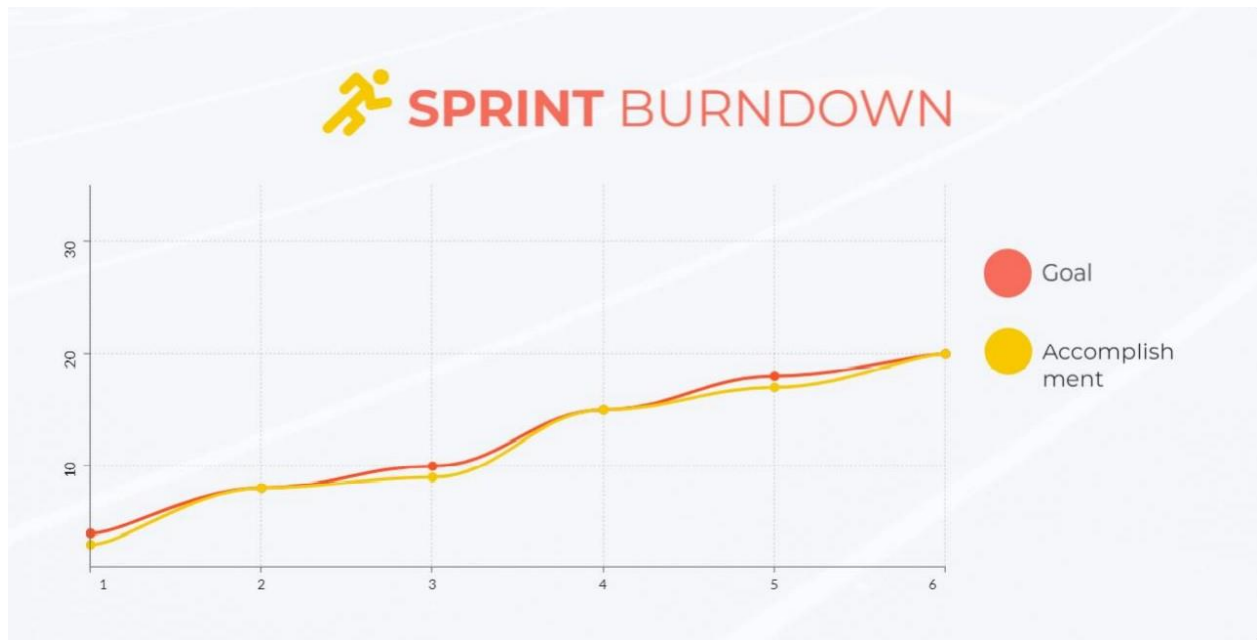
### Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{\text{sprint duration}}{\text{velocity}} = \frac{20}{10} = 2$$

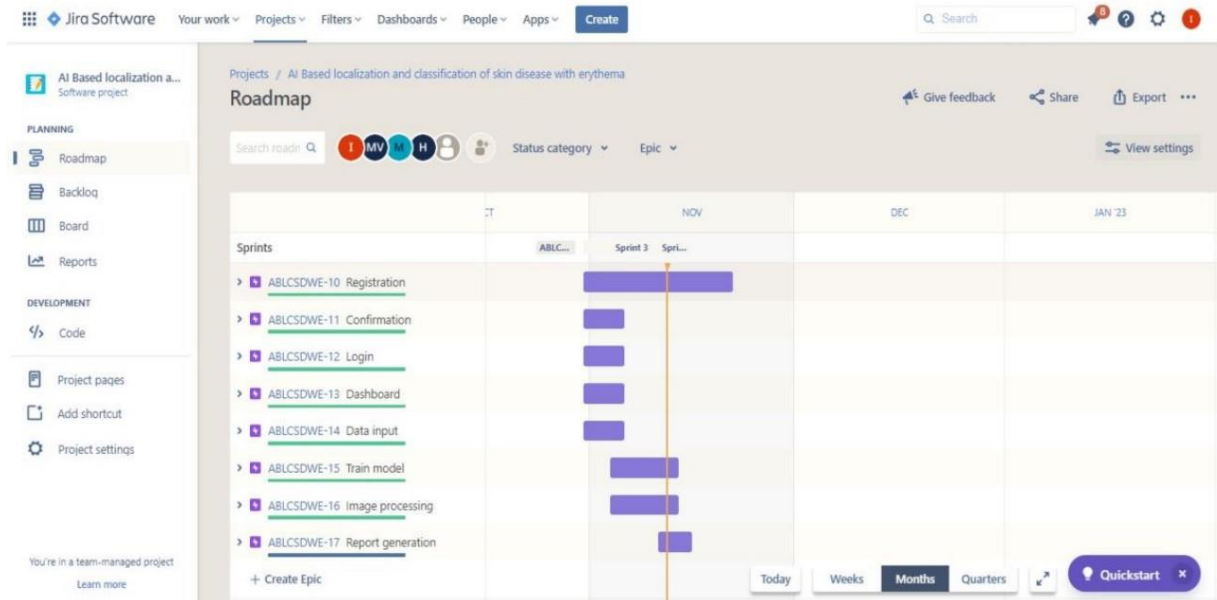
## Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

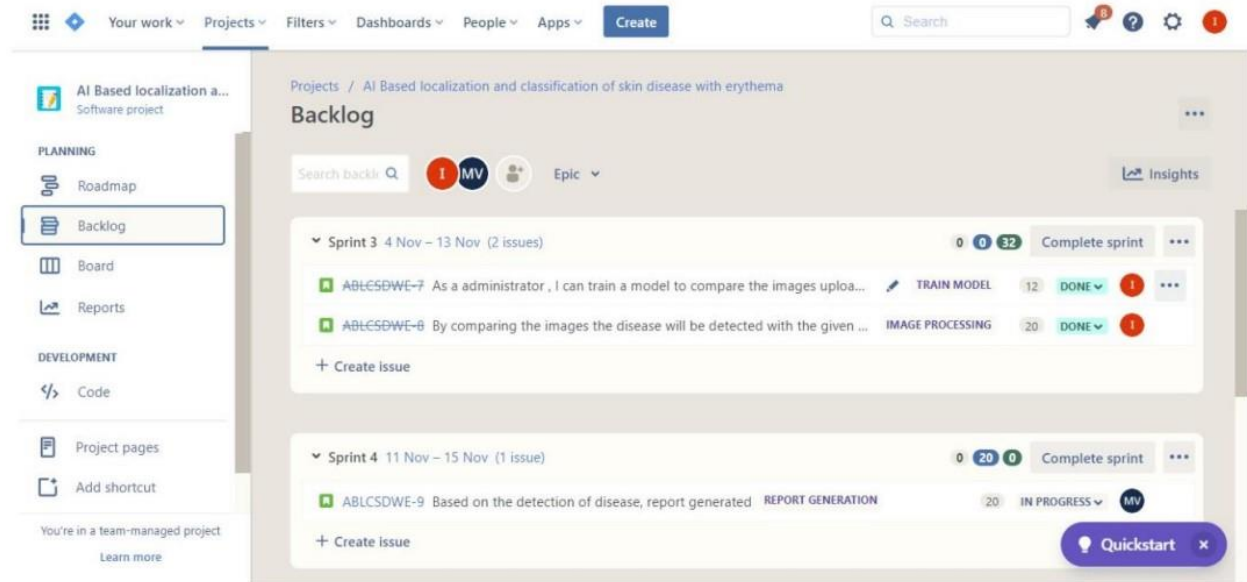


## 6.4 Reports from JIRA

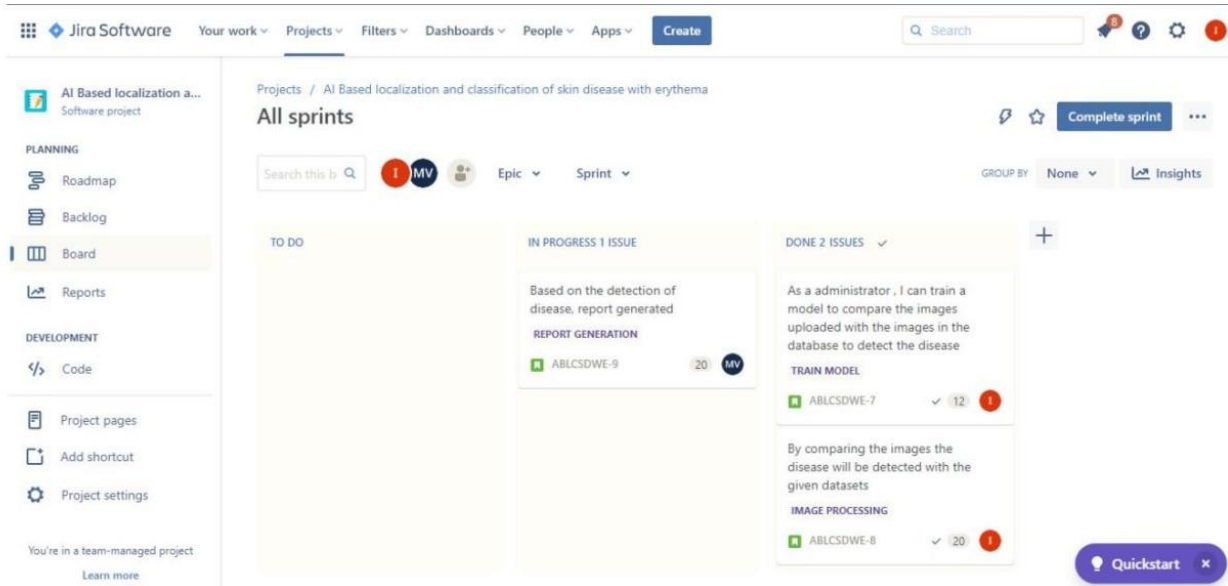
Roadmap:



Backlog:



Board:



## CODING & SOLUTIONING

### 7.1 Feature 1

Annotate Images Our detector needs some high-quality training examples before it can start learning. The images in our training folder are manually labelled using Microsoft's Visual Object Tagging Tool (VOTT). At least 100 images should be annotated for each category to get respectable results. The VOTT csv formatted annotation data is converted to YOLOv3 format by Convert\_to\_YOLO\_format.py file.

Code:

```
from PIL import Image
from os import path, makedirs
import os
import re
import pandas as pd
import sys
import argparse
```

```
def get_parent_dir(n=1):
```

```
    """ returns the n-the parent directory of the current
```



```

working directory """
current_path = os.path.dirname(os.path.abspath(__file__))
for k in range(n):
    current_path = os.path.dirname(current_path)
return current_path

sys.path.append(os.path.join(get_parent_dir(1), "Utils")) from
Convert_Format import convert_vott_csv_to_yolo
Data_Folder = os.path.join(get_parent_dir(1), "Data")
VoTT_Folder = os.path.join(
    Data_Folder, "Source_Images", "Training_Images", "vott-csv-export"
)
VoTT_csv = os.path.join(VoTT_Folder, "Annotations-export.csv")
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")

model_folder = os.path.join(Data_Folder, "Model_Weights")
classes_filename = os.path.join(model_folder, "data_classes.txt")

if __name__ == "__main__":
    # surpress any inhereted default values
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """
    parser.add_argument(
        "--VoTT_Folder",

```

```
type=str,

default=VoTT_Folder,
help="Absolute path to the exported files from the image tagging step with
VoTT. Default is "
+ VoTT_Folder,
)

parser.add_argument(
    "--VoTT_csv",
    type=str,
    default=VoTT_csv,
    help="Absolute path to the *.csv file exported from VoTT. Default is "
+ VoTT_csv,
)

parser.add_argument(
    "--YOLO_filename",
    type=str,
    default=YOLO_filename,
    help="Absolute path to the file where the annotations in YOLO format should be
saved. Default is "
+ YOLO_filename,
)

FLAGS = parser.parse_args()
```

```

# Prepare the dataset for YOLO
multi_df = pd.read_csv(FLAGS.VoTT_csv)
labels = multi_df["label"].unique()
labeldict = dict(zip(labels, range(len(labels))))
multi_df.drop_duplicates(subset=None, keep="first", inplace=True)
train_path = FLAGS.VoTT_Folder
convert_vott_csv_to_yolo(
    multi_df, labeldict, path=train_path, target_name=FLAGS.YOLO_filename
)

# Make classes file
file = open(classes_filename, "w")

# Sort Dict by Values
SortedLabelDict = sorted(labeldict.items(), key=lambda x: x[1])
for elem in SortedLabelDict:
    file.write(elem[0] + "\n")
file.close()

```

## 7.2 Feature 2

### Training Yolo

To prepare for the training process, convert the YOLOv3 model to the Keras format. The YOLOv3 Detector can then be trained by Train\_YOLO.py file.

Code:

```

import os
import sys
import argparse
import warnings

def get_parent_dir(n=1):
    """ returns the n-th parent directory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path

src_path = os.path.join(get_parent_dir(0), "src")
sys.path.append(src_path)

utils_path = os.path.join(get_parent_dir(1), "Utils")
sys.path.append(utils_path)

import numpy as np
import keras.backend as K from keras.layers
import Input, Lambda from
keras.models import Model from
keras.optimizers import Adam from
keras.callbacks import (
    TensorBoard,
    ModelCheckpoint,
    ReduceLROnPlateau,
    EarlyStopping,
)
from keras_yolo3.yolo3.model import (
    preprocess_true_boxes,

```

```
yolo_body,  
tiny_yolo_body,  
yolo_loss,  
)  
from keras_yolo3.yolo3.utils import get_random_data  
from PIL import Image from time import time  
  
import tensorflow.compat.v1 as tf  
import pickle  
  
from Train_Utils import (  
    get_classes,  
    get_anchors,  
  
    create_model,  
    create_tiny_model,  
    data_generator,  
    data_generator_wrapper,  
    ChangeToOtherMachine,  
)  
  
keras_path = os.path.join(src_path, "keras_yolo3")  
Data_Folder = os.path.join(get_parent_dir(1), "Data")  
Image_Folder = os.path.join(Data_Folder, "Source_Images", "Training_Images")  
VoTT_Folder = os.path.join(Image_Folder, "vott-csv-export")  
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")
```

```
Model_Folder = os.path.join(Data_Folder, "Model_Weights")
YOLO_classname = os.path.join(Model_Folder, "data_classes.txt")
```

```
log_dir = Model_Folder
anchors_path = os.path.join(keras_path, "model_data", "yolo_anchors.txt")
weights_path = os.path.join(keras_path, "yolo.h5")
```

```
FLAGS = None
```

```
if __name__ == "__main__":
    # Delete all default flags
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
    """
    Command line options
    """

    parser.add_argument(
        "--annotation_file",
        type=str,
        default=YOLO_filename,
        help="Path to annotation file for Yolo. Default is " + YOLO_filename,
    )
    parser.add_argument(
        "--classes_file",
        type=str,
```

```
        default=YOLO_classname,
        help="Path to YOLO classnames. Default is " + YOLO_classname,
    )

    parser.add_argument(
        "--log_dir",
        type=str,
        default=log_dir,
        help="Folder to save training logs and trained weights to. Default is "
        + log_dir,
    )

    parser.add_argument(
        "--anchors_path",
        type=str,
        default=anchors_path,

        help="Path to YOLO anchors. Default is " + anchors_path,
    )

    parser.add_argument(
        "--weights_path",
        type=str,
        default=weights_path,
        help="Path to pre-trained YOLO weights. Default is " + weights_path,
    )

    parser.add_argument(
```

```
--val_split",
type=float,
default=0.1,
help="Percentage of training set to be used for validation. Default is 10%.",
)
parser.add_argument(
    "--is_tiny",
    default=False,
    action="store_true",
    help="Use the tiny Yolo version for better performance and less accuracy.
Default is False.",
)
parser.add_argument(
    "--random_seed",
    type=float,
    default=None,
    help="Random seed value to make script deterministic. Default is 'None', i.e.
non-deterministic.",
)
parser.add_argument(
    "--epochs",
    type=float,
    default=51,
    help="Number of epochs for training last layers and number of epochs for
finetuning layers. Default is 51.",
)
parser.add_argument(
```



```
--warnings",  
    default=False,  
    action="store_true",  
    help="Display warning messages. Default is False.",  
)
```

```
FLAGS = parser.parse_args()
```

```
if not FLAGS.warnings:
```

```
    tf.logging.set_verbosity(tf.logging.ERROR)  
    os.environ['TF_CPP_MIN_LOG_LEVEL']='3'  
    warnings.filterwarnings("ignore")
```

```
np.random.seed(FLAGS.random_seed)
```

```
log_dir = FLAGS.log_dir
```

```
class_names = get_classes(FLAGS.classes_file)
```

```
num_classes = len(class_names)
```

```
anchors = get_anchors(FLAGS.anchors_path)
```

```
weights_path = FLAGS.weights_path
```

```
input_shape = (416, 416) # multiple of 32, height, width
```

```
epoch1, epoch2 = FLAGS.epochs, FLAGS.epochs
```

```
is_tiny_version = len(anchors) == 6 # default setting
```

```
if FLAGS.is_tiny:
```

```

        model = create_tiny_model(
            input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path
        )
    else:
        model = create_model(
            input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path

        ) # make sure you know what you freeze

log_dir_time = os.path.join(log_dir, "{}".format(int(time())))
logging = TensorBoard(log_dir=log_dir_time)
checkpoint = ModelCheckpoint(
    os.path.join(log_dir, "checkpoint.h5"),
    monitor="val_loss",
    save_weights_only=True,
    save_best_only=True,
    period=5,
)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3,
verbose=1)
early_stopping = EarlyStopping(
    monitor="val_loss", min_delta=0, patience=10, verbose=1
)

val_split = FLAGS.val_split

```

```
with open(FLAGS.annotation_file) as f:
```

```
    lines = f.readlines()
```

```
    # This step makes sure that the path names correspond to the local machine
```

```
    # This is important if annotation and training are done on different machines (e.g.  
training on AWS)
```

```
    lines = ChangeToOtherMachine(lines, remote_machine="")
```

```
    np.random.shuffle(lines)
```

```
    num_val = int(len(lines) * val_split)
```

```
    num_train = len(lines) - num_val
```

```
    # Train with frozen layers first, to get a stable loss.
```

```
    # Adjust num epochs to your dataset. This step is enough to obtain a decent model.
```

```
    if True:
```

```
        model.compile(
```

```
            optimizer=Adam(lr=1e-3),
```

```
            loss={
```

```
                # use custom yolo_loss Lambda layer.
```

```
                "yolo_loss": lambda y_true, y_pred: y_pred
```

```
            },
```

```
        )
```

```
        batch_size = 32
```

```
        print(
```

```
            "Train on { } samples, val on { } samples, with batch size { }".format(  
num_train, num_val, batch_size
```

```
        )
```

```

)
history = model.fit_generator(
    data_generator_wrapper(
        lines[:num_train], batch_size, input_shape, anchors, num_classes
    ),
    steps_per_epoch=max(1, num_train // batch_size),
    validation_data=data_generator_wrapper(
        lines[num_train:], batch_size, input_shape, anchors, num_classes
    ),
    validation_steps=max(1, num_val // batch_size),
    epochs=epoch1,
    initial_epoch=0,
    callbacks=[logging, checkpoint],
)
model.save_weights(os.path.join(log_dir, "trained_weights_stage_1.h5"))

step1_train_loss = history.history["loss"]

file = open(os.path.join(log_dir_time, "step1_loss.npy"), "w")
with open(os.path.join(log_dir_time, "step1_loss.npy"), "w") as f:
    for item in step1_train_loss:
        f.write("%s\n" % item)
file.close()

step1_val_loss = np.array(history.history["val_loss"])

```

```

file = open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w")
with open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w") as f:
    for item in step1_val_loss:
        f.write("%s\n" % item)
file.close()

# Unfreeze and continue training, to fine-tune.
# Train longer if the result is unsatisfactory.
if True:
    for i in range(len(model.layers)):
        model.layers[i].trainable = True
    model.compile(
        optimizer=Adam(lr=1e-4), loss={"yolo_loss": lambda y_true, y_pred: y_pred}
    ) # recompile to apply the change
    print("Unfreeze all layers.")

    batch_size = (
        4 # note that more GPU memory is required after unfreezing the body
    )
    print(
        "Train on {} samples, val on {} samples, with batch size {}".format(
num_train, num_val, batch_size
        )
    )
    history = model.fit_generator(
        data_generator_wrapper(
            lines[:num_train], batch_size, input_shape, anchors, num_classes

```

```

    ),
    steps_per_epoch=max(1, num_train // batch_size),
    validation_data=data_generator_wrapper(
        lines[num_train:], batch_size, input_shape, anchors, num_classes
    ),
    validation_steps=max(1, num_val // batch_size),
    epochs=epoch1 + epoch2,
    initial_epoch=epoch1,
    callbacks=[logging, checkpoint, reduce_lr, early_stopping],
)
model.save_weights(os.path.join(log_dir, "trained_weights_final.h5"))
step2_train_loss = history.history["loss"]

file = open(os.path.join(log_dir_time, "step2_loss.npy"), "w")
with open(os.path.join(log_dir_time, "step2_loss.npy"), "w") as f:
    for item in step2_train_loss:
        f.write("%s\n" % item)
file.close()

step2_val_loss = np.array(history.history["val_loss"])

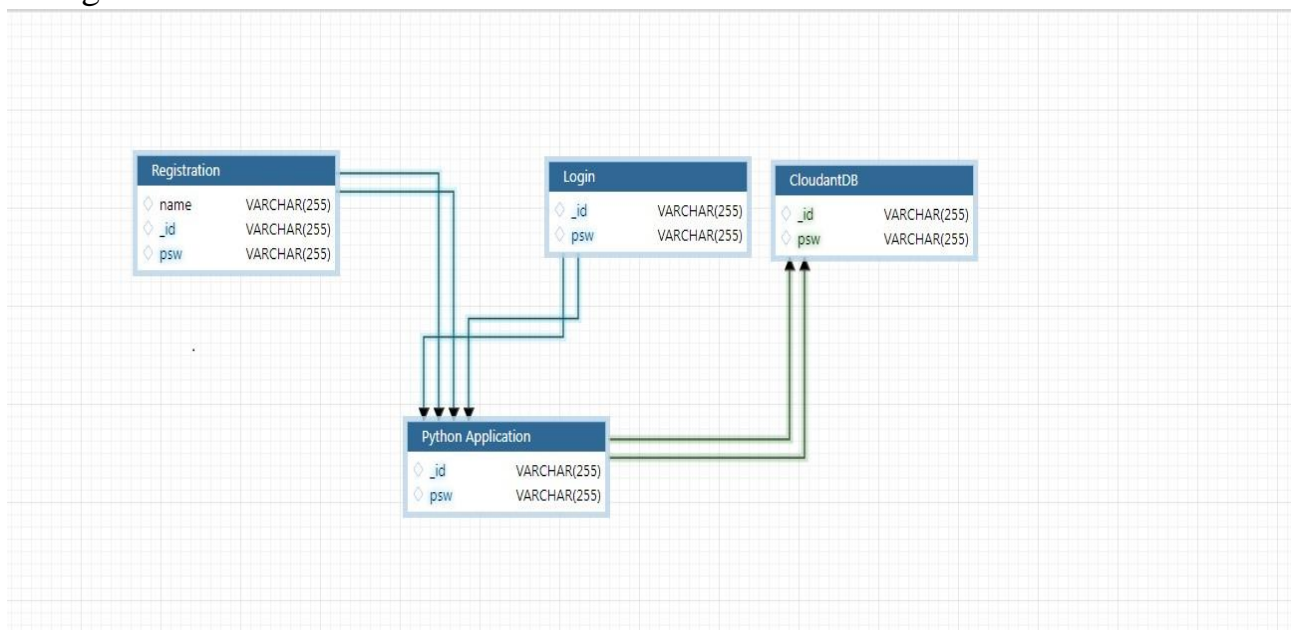
file = open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w")
with open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w") as f:
    for item in step2_val_loss:
        f.write("%s\n" % item)
file.close()

```

### 7.3 Database Schema

- Registration: When a new user registers, the backend connects to the IBM Cloudant and stores the user's credentials in the database.
- Login: To check if a user is already registered, the backend connects to Cloudant when they attempt to log in. They are an invalid user if they are not already registered.
- IBM cloudant: Stores the data which is registered.
- app.py: Connects both Frontend and the cloudant for the verification of user credentials

Diagram:



## TESTING

### 8.1 Test Case

Test Case No.	Action	Expected Output	Actual Output	Result
---------------	--------	-----------------	---------------	--------

1	Register for the website	Stores name, email, and password in Database	Stores name, email, and password in Database	Pass
2	Login to the website	Giving the right credentials, results in a successful login.	Giving the right credentials, results in a successful login.	Pass
3	Detecting the disease	It should predict the disease	It should predict the disease	Pass

## 8.2 User Acceptance Testing

Section	Total Cases	Not Tested	Fail	Pass
Registration	9	0	0	9
Login	40	0	0	40
Security	2	0	0	2
Disease Detection	10	0	0	10
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2



## RESULTS

### 9.1 Performance Metrics

S.No.	Parameter	Values
1.	Model Summary	To evaluate object detec on models like R-CNN and YOLO, the mean average precision (mAP) is used. The mAP compares the ground-truth bounding box to the detected box and returns a score.
2.	Accuracy	Training Accuracy – 89% Valida on Accuracy – 95%
3.	Confidence Score (Only Yolo Projects)	Class Detected – 93% Confidence Score – 90%

## ADVANTAGES & DISADVANTAGES

### Advantages:

- Image processing technology has enabled more efficient and accurate treatment plans.
- It is time and money-saving process.
- Performance of the model will be good even with the higher user traffic.
- In Image processing, the pixels in the image can be manipulated to any desired density and contrast.
- Since high pixel quality is generated, easy classification of skin disease is possible

Disadvantages:

- AI-Models are Susceptible to security risks.
- Inaccuracies are still possible.
- Although AI has come a long way, human surveillance is still essential.

## **CONCLUSION**

Even without a large dataset and high-quality images, it is possible to achieve sufficient accuracy rates in this AI model. With accurate segmentation, we gain knowledge of the location of the disease, which is useful in the pre-processing of data used in classification as it allows the YOLO model to focus on the area of interest. Our method provides a solution to classifying multiple diseases with higher quality and a larger quantity of data. With the assistance of our AI-based methods, it saves time and money for patients.

## **FUTURE SCOPE**

The future of AI in detecting skin diseases could include tasks that range from simple to complex—everything from answering the phone to medical record review, reading radiology images, making clinical diagnoses and treatment plans, and even talking with patients. AI is already at work, increasing convenience and efficiency, reducing costs and errors, and generally making it easier for more patients to receive the health care they need. While AI is being used in health care, it will become increasingly important for its potential to enhance patient engagement in their own care and streamline patient access to care.

## APPENDIX

### SOURCE CODE

```
import re
import numpy as np
import os
from flask import Flask, app, request, render_template
import sys
from flask import Flask, request, render_template, redirect, url_for
import argparse
from tensorflow import keras
from PIL import Image
from time import time
import default_timer as timer
import test
import pandas as pd
import numpy as np
import random

def get_parent_dir(n=1):
    """ returns the n-th parent directory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))

    for k in range(n):
        current_path = os.path.dirname(current_path)

    return current_path

src_path
=r'C:\Users\MadhuVasanth1606\Desktop\yolo_structure\2_Training\src'
print(src_path)
utils_path =
r'C:\Users\MadhuVasanth1606\Desktop\yolo_structure\Utils'
print(utils_path)

sys.path.append(src_path)
sys.path.append(utils_path)
```

```
import argparse from keras_yolo3.yolo import YOLO, detect_video from PIL
import Image from timeit import default_timer as timer from utils import
load_extractor_model, load_features, parse_input, detect_object import test
import utils import pandas as pd import numpy as np from Get_File_Paths
import GetFileList import random os.environ["TF_CPP_MIN_LOG_LEVEL"] =
"3"

# Set up folder names for default values data_folder =
os.path.join(get_parent_dir(n=1), "yolo_structure", "Data") image_folder
= os.path.join(data_folder, "Source_Images") image_test_folder =
os.path.join(image_folder, "Test_Images")

detection_results_folder = os.path.join(image_folder,
"Test_Image_Detection_Results")
detection_results_file = os.path.join(detection_results_folder,
"Detection_Results.csv") model_folder =
os.path.join(data_folder, "Model_Weights")

model_weights = os.path.join(model_folder, "trained_weights_final.h5")
model_classes = os.path.join(model_folder, "data_classes.txt")

anchors_path = os.path.join(src_path, "keras_yolo3", "model_data",
"yolo_anchors.txt")

FLAGS = None
```

```
from cloudant.client import Cloudant
```

```
# Authenticate using an IAM API key
```

```
client = Cloudant.iam('5b73f72f-2449-4298-88e8-  
3f887f8bbd2dbluemix','t3wXXORf8KoIMLzYFX2sk4e22uluSBKhM9-K4Q5b1zuK',  
connect=True)
```

```
# Create a database using an initialized client my_database
```

```
= client.create_database('skindisease')
```

```
app=Flask(__name__)
```

```
#default home page or route
```

```
@app.route('/') def index():
```

```
    return render_template('index.html')
```

```
@app.route('/index.html') def
```

```
home():
```

```
    return render_template("index.html")
```

```
#registration page
```

```
@app.route('/register') def
```

```
register():
```

```
    return render_template('register.html')
```

```
@app.route('/afterreg', methods=['POST']) def
afterreg():
    x = [x for x in request.form.values()]
    print(x)

    data = {
        '_id': x[1], # Setting _id is optional    'name': x[0],
        'psw':x[2]
    }
    print(data)

    query = {'_id': {'$eq': data['_id']}}

    docs = my_database.get_query_result(query)
    print(docs)

    print(len(docs.all()))

    if(len(docs.all())==0):
        url = my_database.create_document(data)
        #response = requests.get(url)

        return render_template('register.html', pred="Registration Successful, please
login using your details")
    else:
        return render_template('register.html', pred="You are already a member, please
login using your details")
```

```
#login page
```

```
@app.route('/login') def
```

```
login():
```

```
    return render_template('login.html')
```

```
@app.route('/afterlogin',methods=['POST']) def
```

```
afterlogin():
```

```
    user = request.form['_id']
```

```
    passw = request.form['psw']
```

```
    print(user,passw)
```

```
    query = {'_id': {'$eq': user}}
```

```
    docs = my_database.get_query_result(query)
```

```
    print(docs)
```

```
    print(len(docs.all()))
```

```
    if(len(docs.all())==0):
```

```
        return render_template('login.html', pred="The username is not found.")
```

```
    else:
```

```
        if((user==docs[0][0]['_id'] and passw==docs[0][0]['psw'])):
```

```
            return redirect(url_for('prediction'))
```

```
        else:
```

```
print('Invalid User')
```

```
@app.route('/logout') def
```

```
logout():
```

```
    return render_template('logout.html')
```

```
@app.route('/prediction') def
```

```
prediction():
```

```
    return render_template('prediction.html')
```

```
@app.route('/result',methods=["GET","POST"])
```

```
def res():
```

```
    # Delete all default flags
```

```
    parser = argparse.ArgumentParser(argument_default=argparse.SUPPRESS)
```

```
    """
```

```
    Command line options
```

```
    """
```

```
    parser.add_argument(
```

```
        "--input_path",
```

```
        type=str,
```

```
        default=image_test_folder,
```

```
        help="Path to image/video directory. All subdirectories will be included. Default  
is "
```



```
+ image_test_folder,  
)
```

```
parser.add_argument(  
    "--output",  
    type=str,  
    default=detection_results_folder,  
    help="Output path for detection results. Default is "  
    + detection_results_folder,  
)
```

```
parser.add_argument(  
    "--no_save_img",  
    default=False,  
    action="store_true",  
    help="Only save bounding box coordinates but do not save output images with  
annotated boxes. Default is False.",  
)
```

```
parser.add_argument(  
    "--file_types",  
    "--names-list",  
    nargs="*",  
    default=[],  
    help="Specify list of file types to include. Default is --file_types .jpg .jpeg .png  
.mp4",  
)
```

```
parser.add_argument(  
    "--yolo_model",  
    type=str,  
    dest="model_path",  
    default=model_weights,  
    help="Path to pre-trained weight files. Default is " + model_weights,  
)
```

```
parser.add_argument(  
    "--anchors",  
    type=str,  
    dest="anchors_path",  
    default=anchors_path,  
    help="Path to YOLO anchors. Default is " + anchors_path,  
)
```

```
parser.add_argument(  
    "--classes",  
    type=str,  
    dest="classes_path",  
    default=model_classes,  
    help="Path to YOLO class specifications. Default is " + model_classes,  
)
```

```
parser.add_argument(  

```

```
    "--gpu_num", type=int, default=1, help="Number of GPU to use. Default is 1"  
)
```

```
parser.add_argument(  
    "--confidence",  
    type=float,  
    dest="score",  
    default=0.25,  
    help="Threshold for YOLO object confidence score to show predictions. Default  
is 0.25.",  
)
```

```
parser.add_argument(  
    "--box_file",  
    type=str,  
    dest="box",  
    default=detection_results_file,  
    help="File to save bounding box results to. Default is "  
+ detection_results_file,  
)
```

```
parser.add_argument(  
    "--postfix",  
    type=str,  
    dest="postfix",  
    default="_disease",  
    help='Specify the postfix for images with bounding boxes. Default is "_disease",
```

)

FLAGS = parser.parse\_args()

save\_img = not FLAGS.no\_save\_img

file\_types = FLAGS.file\_types

#print(input\_path)

if file\_types:

    input\_paths = GetFileList(FLAGS.input\_path, endings=file\_types)

    print(input\_paths)

else:

    input\_paths = GetFileList(FLAGS.input\_path)

    print(input\_paths)

# Split images and videos

img\_endings = (".jpg", ".jpeg", ".png")

vid\_endings = (".mp4", ".mpeg", ".mpg", ".avi")

input\_image\_paths = []

input\_video\_paths = []

for item in input\_paths:

    if item.endswith(img\_endings):

input\_image\_paths.append(item)        elif

item.endswith(vid\_endings):

```
        input_video_paths.append(item)

output_path = FLAGS.output
if not os.path.exists(output_path):
    os.makedirs(output_path)

# define YOLO detector
yolo = YOLO(
    **{
        "model_path": FLAGS.model_path,
        "anchors_path": FLAGS.anchors_path,
        "classes_path": FLAGS.classes_path,
        "score": FLAGS.score,
        "gpu_num": FLAGS.gpu_num,
        "model_image_size": (416, 416),
    }
)

# Make a dataframe for the prediction outputs
out_df = pd.DataFrame(
    columns=[
        "image",
        "image_path",
        "xmin",
        "ymin",
        "xmax",
```

```
        "ymax",
        "label",
        "confidence",
        "x_size",
    "y_size",
    ]
)
```

```
# labels to draw on images
```

```
class_file = open(FLAGS.classes_path, "r")
```

```
input_labels = [line.rstrip("\n") for line in class_file.readlines()]
```

```
print("Found { } input labels: { } ...".format(len(input_labels), input_labels))
```

```
if input_image_paths:
```

```
    print(
```

```
        "Found { } input images: { } ...".format(
```

```
            len(input_image_paths),
```

```
            [os.path.basename(f) for f in input_image_paths[:5]],
```

```
        )
```

```
    )
```

```
    start = timer()
```

```
    text_out = ""
```

```
# This is for images
```

```
for i, img_path in enumerate(input_image_paths):
```

```
    print(img_path)
```

```

prediction, image,lat,lon= detect_object(
    yolo,
    img_path,
    save_img=save_img,
    save_img_path=FLAGS.output,
    postfix=FLAGS.postfix,
)
print(lat,lon)
y_size, x_size, _ = np.array(image).shape
for single_prediction in prediction:
    out_df = out_df.append(
        pd.DataFrame(
            [
                [
                    os.path.basename(img_path.rstrip("\n")),
                    img_path.rstrip("\n"),
                ]
                + single_prediction
                + [x_size, y_size]
            ],
            columns=[
                "image",
                "image_path",
                "xmin",
                "ymin",
                "xmax",
            ]
        )
    )

```

```

        "ymax",
        "label",
        "confidence",
        "x_size",
        "y_size",
    ],
)
)
end = timer()
print(
    "Processed { } images in {:.1f}sec - {:.1f}FPS".format(
        len(input_image_paths),
        end - start,
        len(input_image_paths) / (end - start),
    )
)
out_df.to_csv(FLAGS.box, index=False)

# This is for videos
if input_video_paths:
    print(
        "Found { } input videos: { } ...".format(
            len(input_video_paths),
            [os.path.basename(f) for f in input_video_paths[:5]],
        )
    )

```



```

start = timer()
for i, vid_path in enumerate(input_video_paths):
    output_path = os.path.join(
        FLAGS.output,
        os.path.basename(vid_path).replace(".", FLAGS.postfix + "."),
    )
    detect_video(yolo, vid_path, output_path=output_path)

end = timer()
print(
    "Processed { } videos in {:.1f}sec".format(
        len(input_video_paths), end - start
    )
)

# Close the current yolo session
yolo.close_session()
return render_template('prediction.html')

""" Running our application """ if
__name__ == "__main__":
    app.run(debug=True)

```

GitHub & Project Demo Link

Github: <https://github.com/IBM-EPBL/IBM-Project-43585-1660718295>

Project Demo Link: <https://youtube.com/watch?v=87x1zJFO4V8&feature=share>