AI-BASED LOCALIZATION AND CLASSIFICATION OF SKIN
DISEASE WITH ERYTHEMA

Table of Contents

List of Figures

1.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.

2.LITRATURE SURVEY

2.1 Existing Problem

The Yolo v3 detector is the primary method for pre-screening skin lesions and detecting erythema. YOLO is an algorithm that detects and recognizes various objects in real-time pictures. Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images. The YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. The algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction for the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously. Yolo-V3 boasts good performance over a wide range of input resolutions.

2.2 References

[1] J. Kawahara and G. Hamarneh, "Multi-resolution-tract CNN with hybrid pretrained and skin-lesion trained layers," in International Workshop on Machine Learning in Medical Imaging, pp. 164–171, Springer, New York, NY, USA, 2016.

[2] S. Verma, M. A. Razzaque, U. Sangtongdee, C. Arpnikanondt, B. Tassaneetrithep, and A. Hossain, "Digital diagnosis of Hand, Foot, and mouth disease using hybrid deep neural networks," IEEE Access, vol.
9, pp. 143481–143494, 2021.

[3] P. P. Rebouças Filho, S. A. Peixoto, R. V. Medeiros da Nobrega´ et al., "Automatic histologically-closer classification of skin lesions," Computerized Medical Imaging and Graphics, vol. 68, pp. 40–54, 2018.

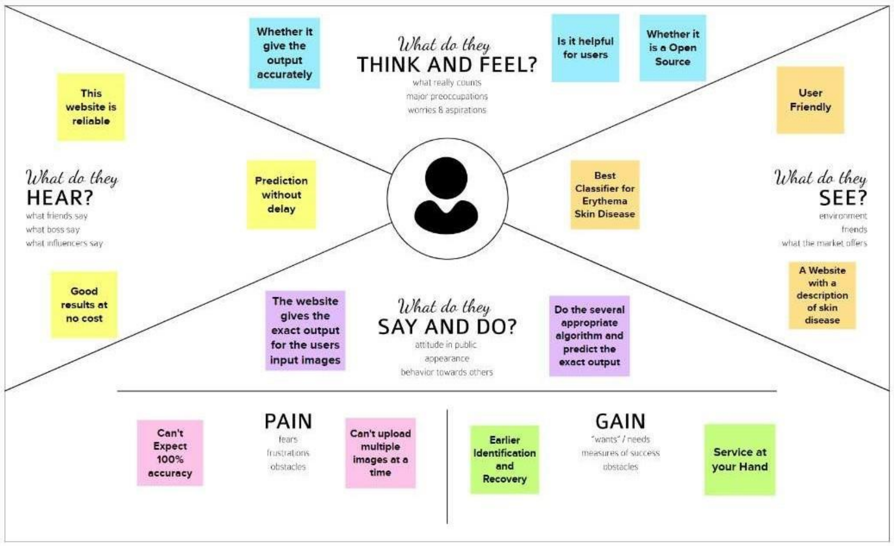
2.3 Problem Statement Definition

To build a model that predicts skin diseases that can be prevented by investigating the infected region. Here, the skin tone and skin colour play an important role in skin disease detection. The person can capture images of their skin, and then the image will be sent to a trained model, which analyses the image and detects whether the person has a skin disease or not.

3.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 helps 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.

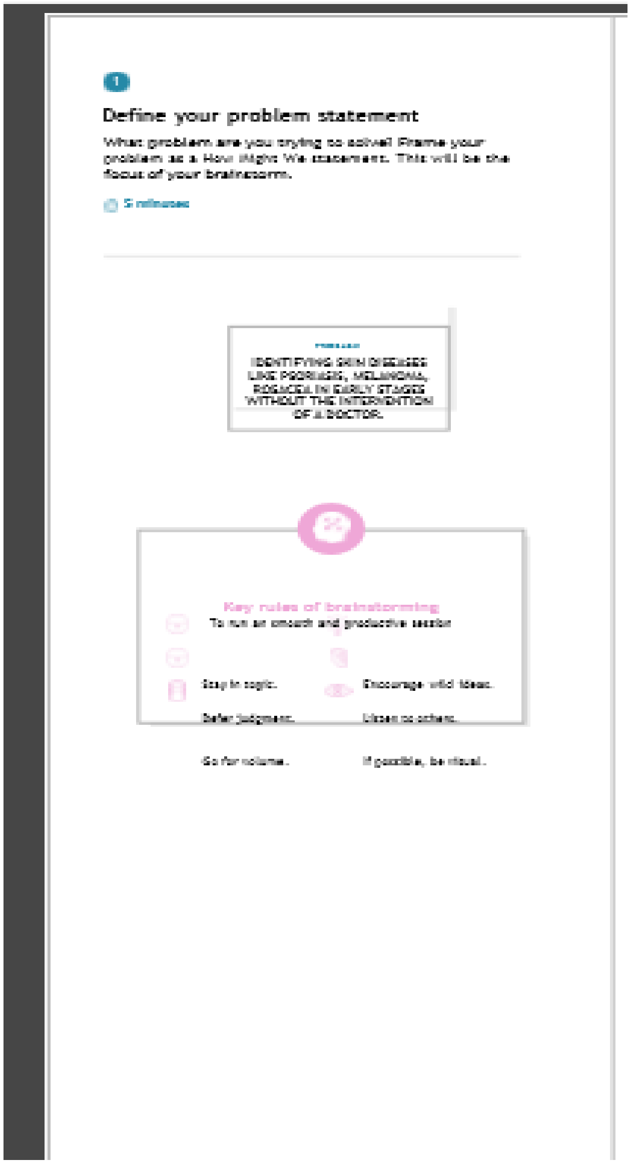


3.2 Ideation & Brainstorming

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

Step-2: Brainstorm, Idea Listing and Grouping



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

PROBLEM

IDENTIFYING SKIN DISEASES LIKE PSORIASIS, MELANOMA, ROSACEA IN EARLY STAGES WITHOUT THE INTERVENTION OF A DOCTOR.

Key rules of brainstorming

To run an smooth and productive session

Stay in topic.

Encourage wild ideas.

Defer judgment.

Listen to others.

Go for volume.

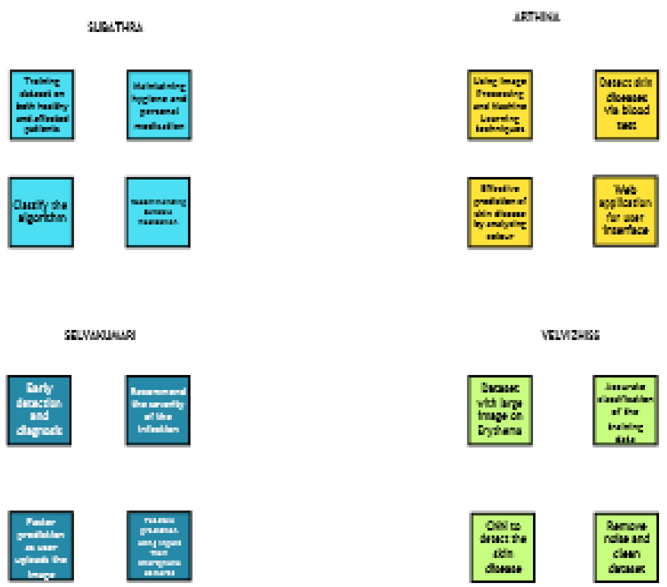
If possible, be visual.

Step-2: Brainstorm, Idea Listing and Grouping

Brainstorm

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

10 minutes



SUBATHRA

Training dataset on both healthy and affected patients

Maintaining hygiene and personal medication

Classify the algorithm

Recommending [illegible] measures

ARTHINA

Using Image Processing and Machine Learning techniques

Detect skin diseases via blood test

Effective prediction of skin disease by analyzing colour

Web application for user interface

SELVAKUMARI

Early detection and diagnosis

Recommend the severity of the infection

Faster prediction as user uploads the image

[illegible]

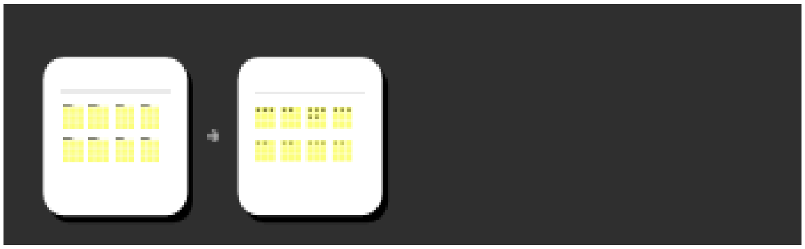
VELVIZHI S

Dataset with large image on Erythema

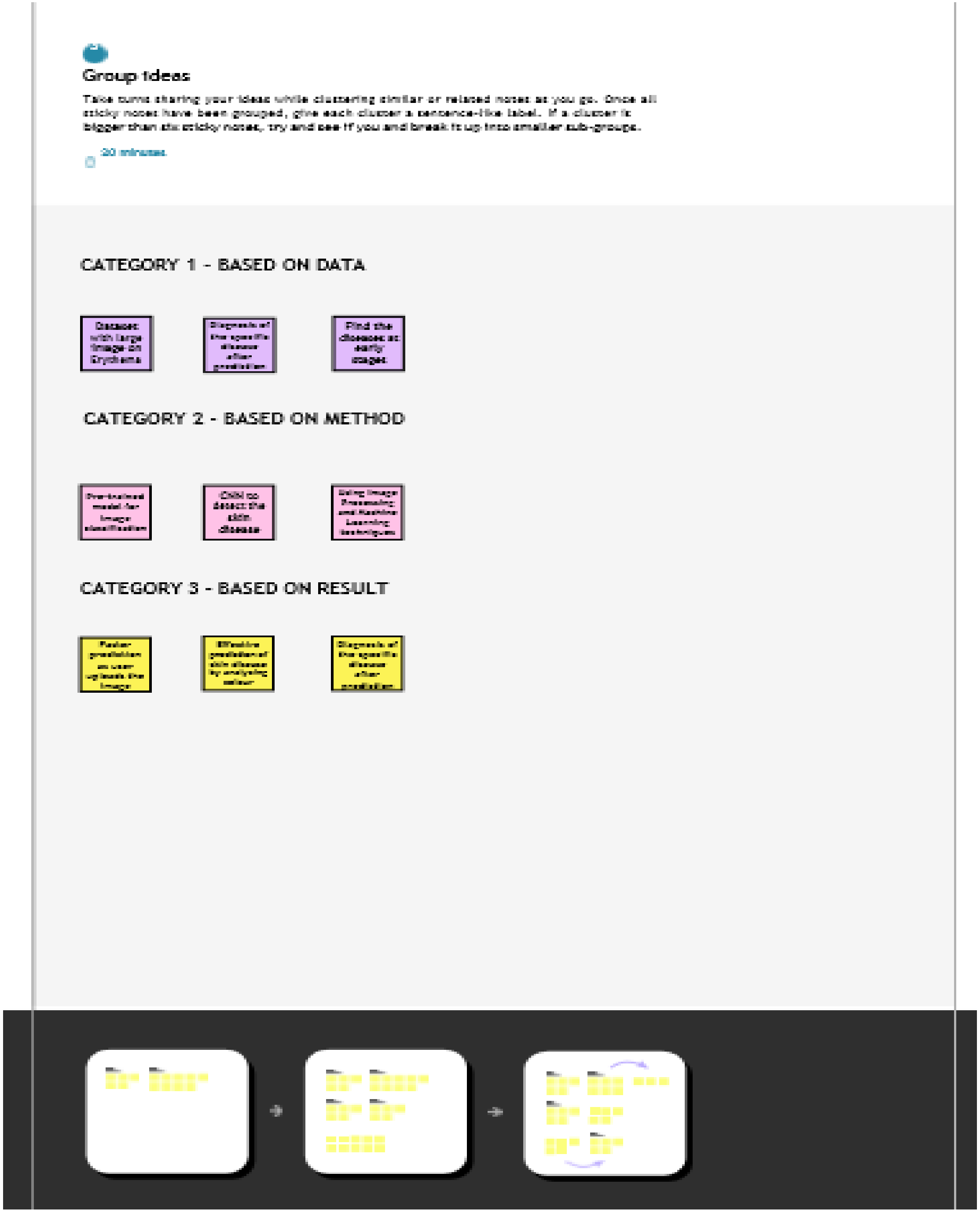
Accurate classification of the training data

CNN to detect the skin disease

Remove noise and clean dataset



Step-3: Idea Prioritization



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 and break it up into smaller sub-groups.

20 minutes

CATEGORY 1 - BASED ON DATA

Dataset with large image on Erythema

Diagnosis of the specific disease after prediction

Find the diseases at early stages

CATEGORY 2 - BASED ON METHOD

Pre-trained model for image classification

CNN to detect the skin disease

Using Image Processing and Machine Learning techniques

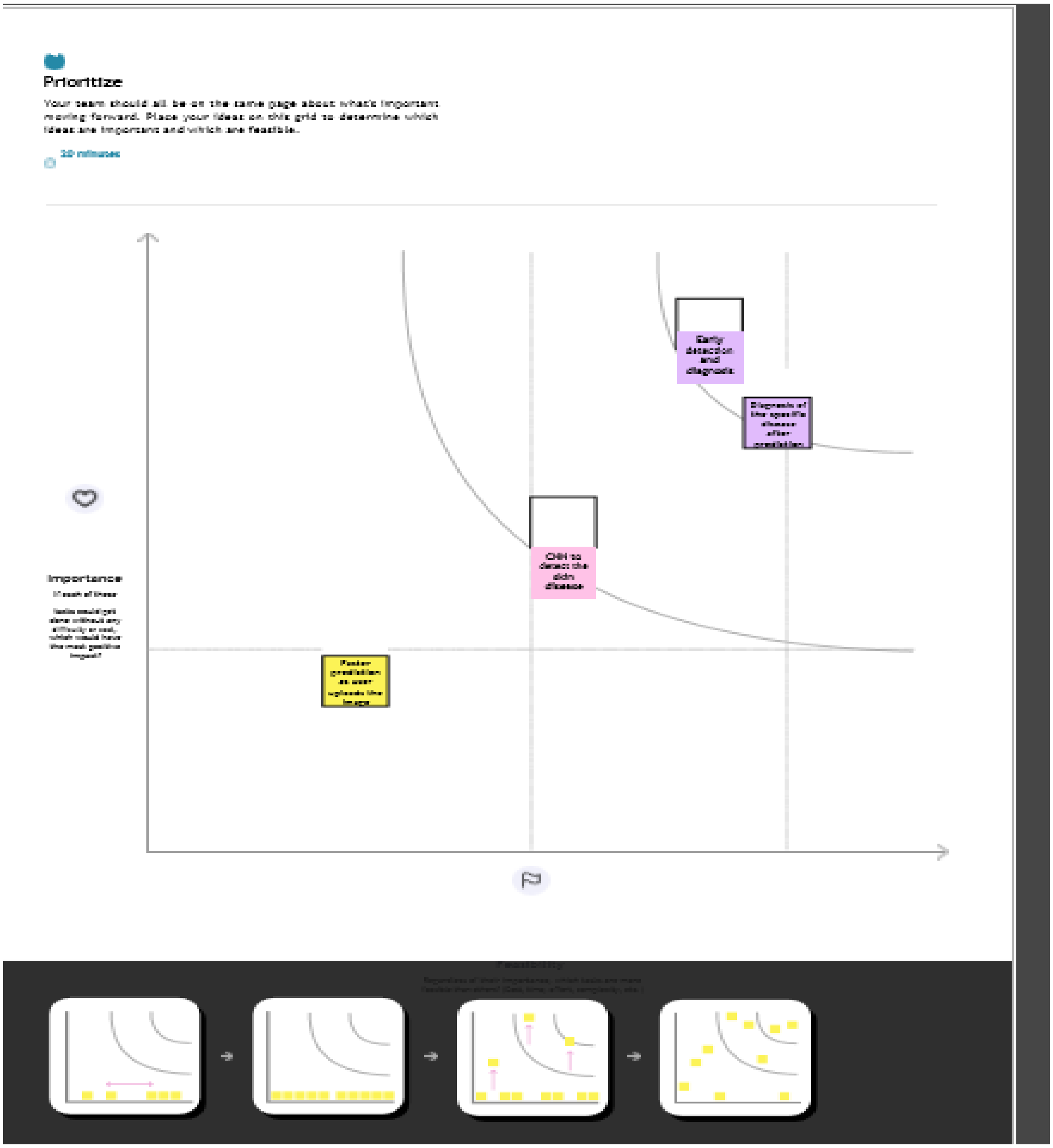
CATEGORY 3 - BASED ON RESULT

Faster prediction as user uploads the image

Effective prediction of skin disease by analysing colour

Diagnosis of the specific disease after prediction

Step4:prtoritize



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

Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

Early detection and diagnosis
Diagnosis of the specific disease after prediction
CNN to detect the skin disease
Faster prediction as user uploads the image

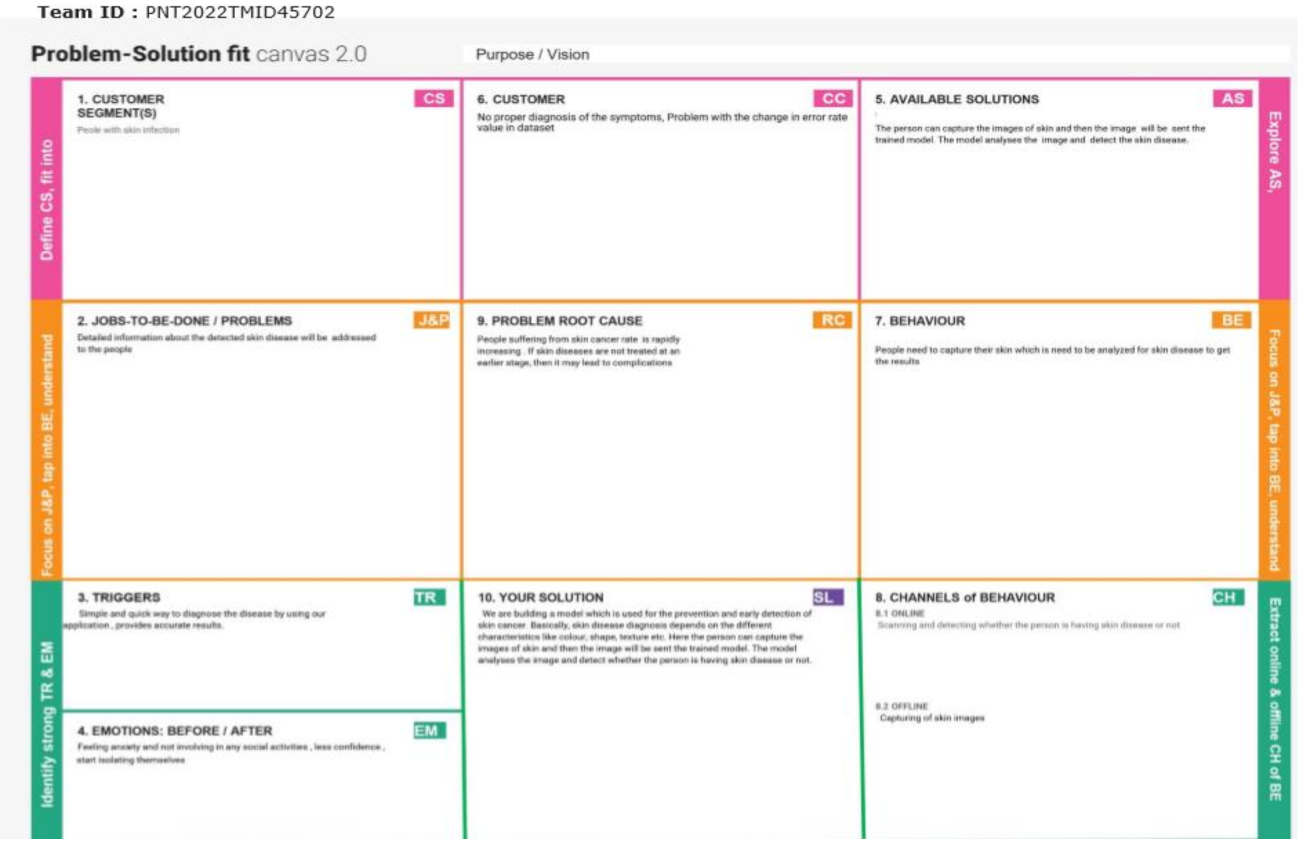
3.3 Proposed Solution

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)	AI-Based Localization and Classification of Skin Diseasewith Erythema
2	Idea / Solution description	To overcome the above problem, we are building a model that is used for the prevention and early detection of Erythema.
3	Novelty / Uniqueness	The model analyses the image and detects whether the person is having a skin disease or not, and if detected, gives a detaileddescription of the disease and treatment suggestions.
4	Social Impact / CustomerSatisfaction	Image processing technologies significantly reduce the time spent on a specific activity by the customer. Hence, it is a time and money-saving process.
5	Business Model (RevenueModel)	Medical sectors are using image recognition to improveimage analysis for identifyingdiseases and predicting the possibilities of health problems.

6	Scalability of the Solution	Image processing technology has enabled more efficient andaccurate treatment plans.

3.4 Problem Solution Fit



Team ID : PNT2022TMID45702

Problem-Solution fit canvas 2.0

Purpose / Vision

1. CUSTOMER SEGMENT(S) CS

Peole with skin infection

6. CUSTOMER CC

No proper diagnosis of the symptoms, Problem with the change in error rate value in dataset

5. AVAILABLE SOLUTIONS AS

The person can capture the images of skin and then the image will be sent the trained model. The model analyses the image and detect the skin disease.

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Detailed information about the detected skin disease will be addressed to the people

9. PROBLEM ROOT CAUSE RC

People suffering from skin cancer rate is rapidly increasing . If skin diseases are not treated at an earlier stage, then it may lead to complications

7. BEHAVIOUR BE

People need to capture their skin which is need to be analyzed for skin disease to get the results

3. TRIGGERS TR

Simple and quick way to diagnose the disease by using our application , provides accurate results.

4. EMOTIONS: BEFORE / AFTER EM

Feeling anxiety and not involving in any social activities , less confidence , start isolating themselves

10. YOUR SOLUTION SL

We are building a model which is used for the prevention and early detection of skin cancer. Basically, skin disease diagnosis depends on the different characteristics like colour, shape, texture etc. Here the person can capture the images of skin and then the image will be sent the trained model. The model analyses the image and detect whether the person is having skin disease or not.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE

Scanning and detecting whether the person is having skin disease or not

8.2 OFFLINE

Capturing of skin images

Define CS, fit into
Explore AS,
Focus on J&P, tap into BE, understand
Focus on J&P, tap into BE, understand
Identify strong TR & EM
Extract online & offline CH of BE

4.REQUIREMENT ANALYSIS

4.1 Functional Requirements

FR.No.	Functional Requirement (Epic)	Sub Requirement (Story /Sub-Task)
1	User Registration	Registration through Form Registration through Gmail
2	User Confirmation	Confirmation via Email Confirmation via OTP
3	User Profile	Users provides their medical history.
4	User Uploads Images(Input)	Upload Images as jpeg Upload Images as png
5	Output Analysis	Output analysed through trained model
6	Provides Description	Gives the detailed description of the skin disease found

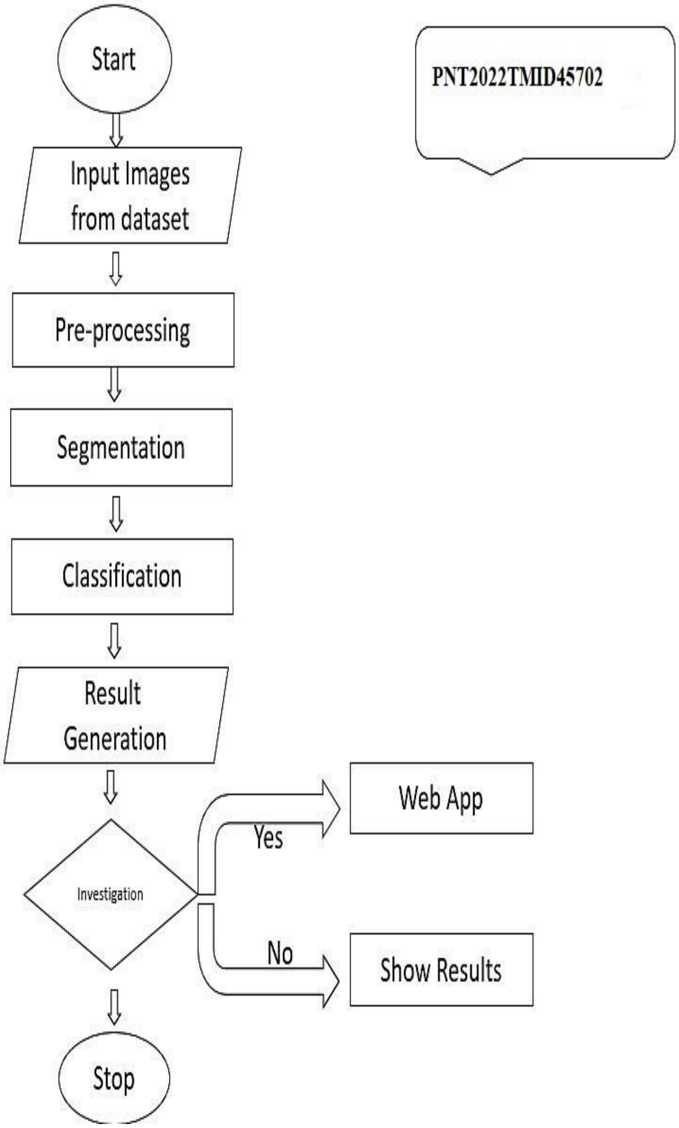
4.2 Non-functional Requirements

NFR No.	Non-Functional Requirement	Description
1	Usability	Used to classify skin diseasewith erythema
2	Security	It offers greater security andprevents unauthorized individuals from accessing user’s data.
3	Reliability	Even with more users, there will be a good performancewithout failure.
4	Performance	With greater accuracy, theperformance is high.
5	Availability	With a good system, all authorized users can accessit.
6	Scalability	Performance will be good even with the higher usertraffic.

5.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.

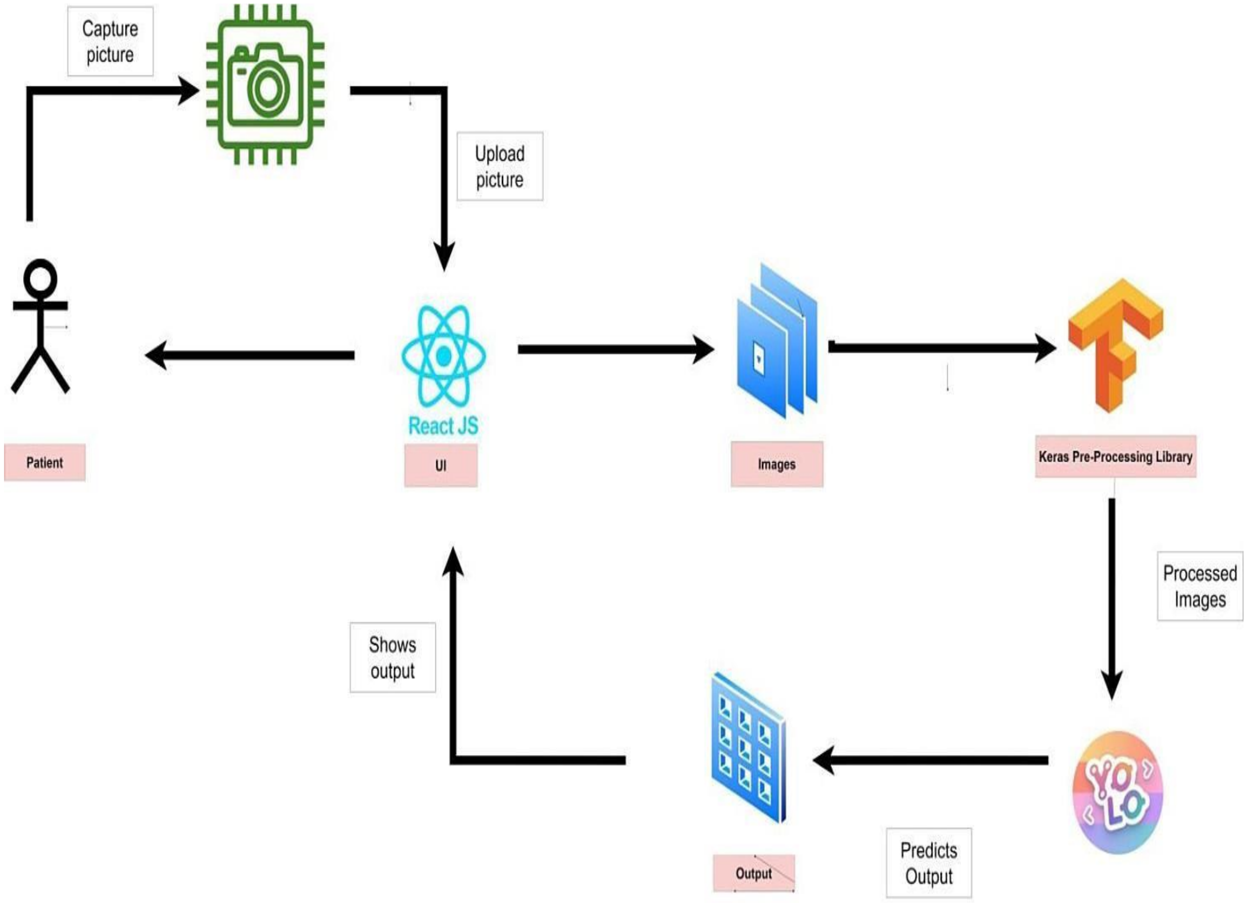


5.2 Solution Architecture

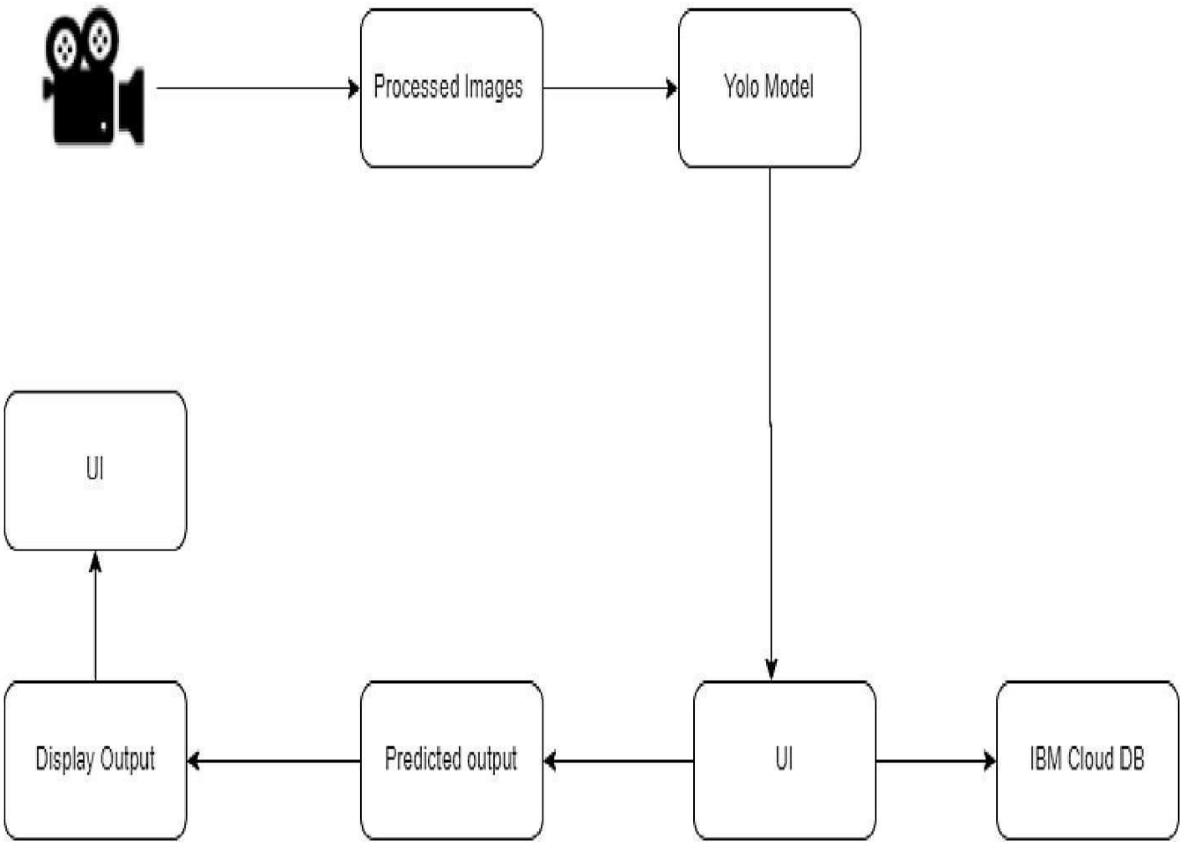
Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, 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

Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Prerequisites	USN-1	Install Python IDE, Python packages, Microsoft Visual Object Tagging Tool, Yolo Structure	3	High
Data Collection	USN-2	Dataset should be collected from google or using a Chrome extension such as Fatkun Batch Downloader	3	High
Annotate Images	USN-3	Create A Project in VOTT (Microsoft's Visual Object Tagging Tool)	2	Medium
Training YOLO	USN-4	train our model using YOLO weights	2	Medium
	USN-5	To Download and Convert Pre-Trained Weights	3	High
	USN-6	To Train YOLOv3 Detector	3	High
Cloudant DB	USN-7	Register & Login to IBM Cloud	3	High
	USN-8	Create Service Instant and Credentials	2	Medium
	USN-9	Launch DB and Create database	3	High
Development Phase	USN-10	To build a web application	3	High
	USN-11	Building HTML pages with python code	2	Medium
	USN-12	To run the application	3	High
Testing Phase	USN-13	As a user login to dashboard	2	Medium
	USN-14	As a user import the images with skin diseases to the software application	2	Medium
	USN-15	YOLO processes the image and give the necessary details	3	High

6.PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Prerequisites	USN-1	Install Python IDE, Python packages, Microsoft Visual Object Tagging Tool, Yolo Structure	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-1	Data Collection	USN-2	Dataset should be collected from google or using a Chrome extension such as Fatkun Batch Downloader	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-1	Annotate Images	USN-3	Create A Project in VOTT (Microsoft's Visual Object Tagging Tool)	2	Medium	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-2	Training YOLO	USN-4	train our model using YOLO weights	2	Medium	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-2		USN-5	To Download and Convert PreTrained Weights	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-2		USN-6	To Train YOLOv3 Detector	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M

Sprint-3	Cloudant DB	USN-7	Register & Login to IBM Cloud	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-3		USN-8	Create Service Instant and Credentials	2	Medium	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-3		USN-9	Launch DB and Create database	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-3	Development Phase	USN-10	To build a web application	3	High	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-3		USN-11	Building HTML pages with python code	2	Medium	Subathra.V Arthina.R Selvakumari.R Velvizhi.M
Sprint-3		USN-12	To run the application	3	High	Subathra.V Arthina.R
Sprint-4	Testing Phase	USN-13	As a user login to dashboard	2	Medium	Selvakumari.R Velvizhi.M
Sprint-4		USN-14	As a user import the images with skin diseases to the software application	2	Medium	Subathra.V Selvakumari.R
Sprint-4		USN-15	YOLO processes the image and give the necessary details	3	High	Arthina.R

Product Backlog, Sprint Schedule, and Estimation:

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	04 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

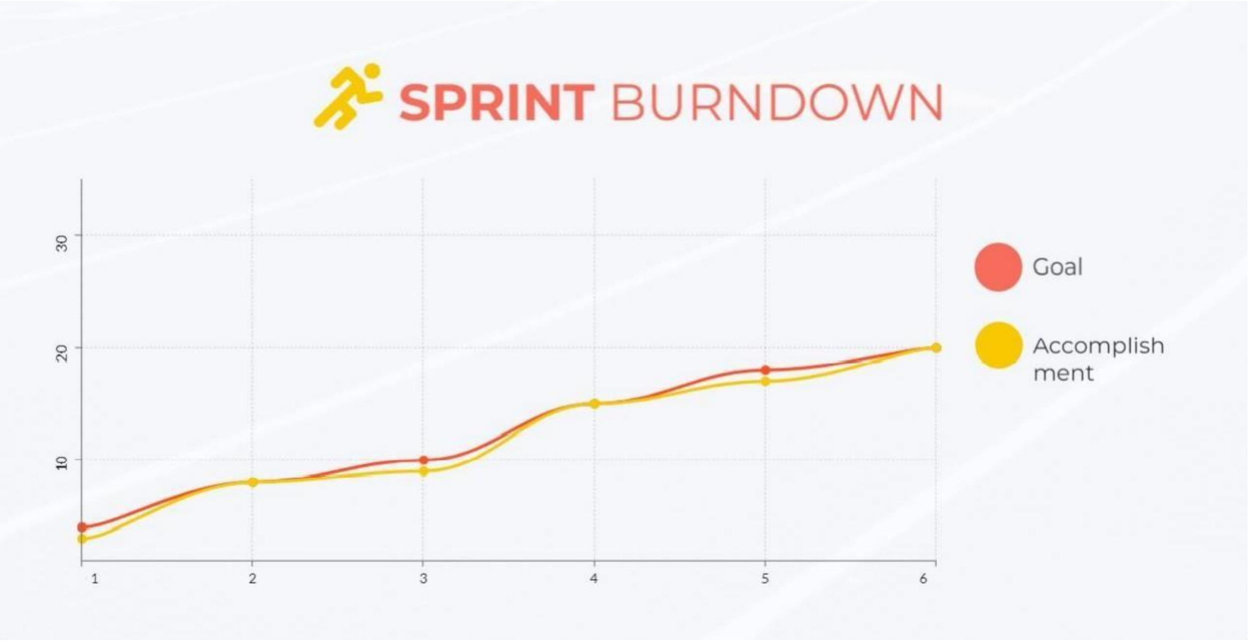
Velocity:

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)

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

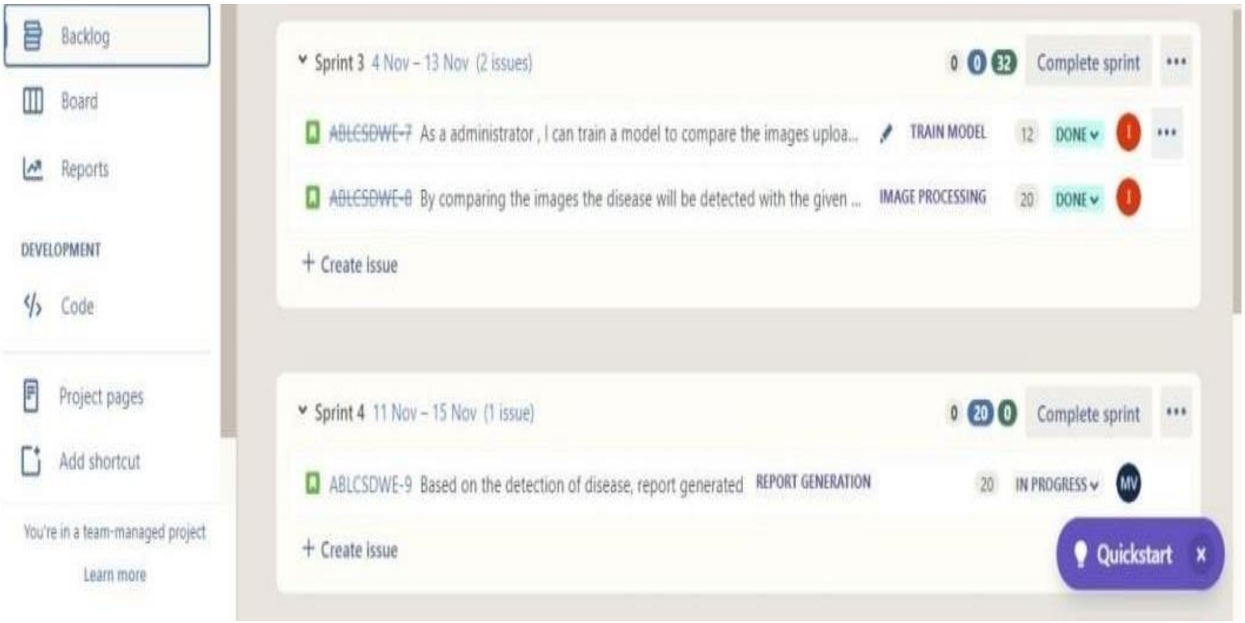
Burndown Chart:

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:



Sprint 3 4 Nov – 13 Nov (2 issues) 0 0 32

~~ABLCSDWE-7~~ As a administrator , I can train a model to compare the images uploa... TRAIN MODEL 12 DONE

~~ABLCSDWE-8~~ By comparing the images the disease will be detected with the given ... IMAGE PROCESSING 20 DONE

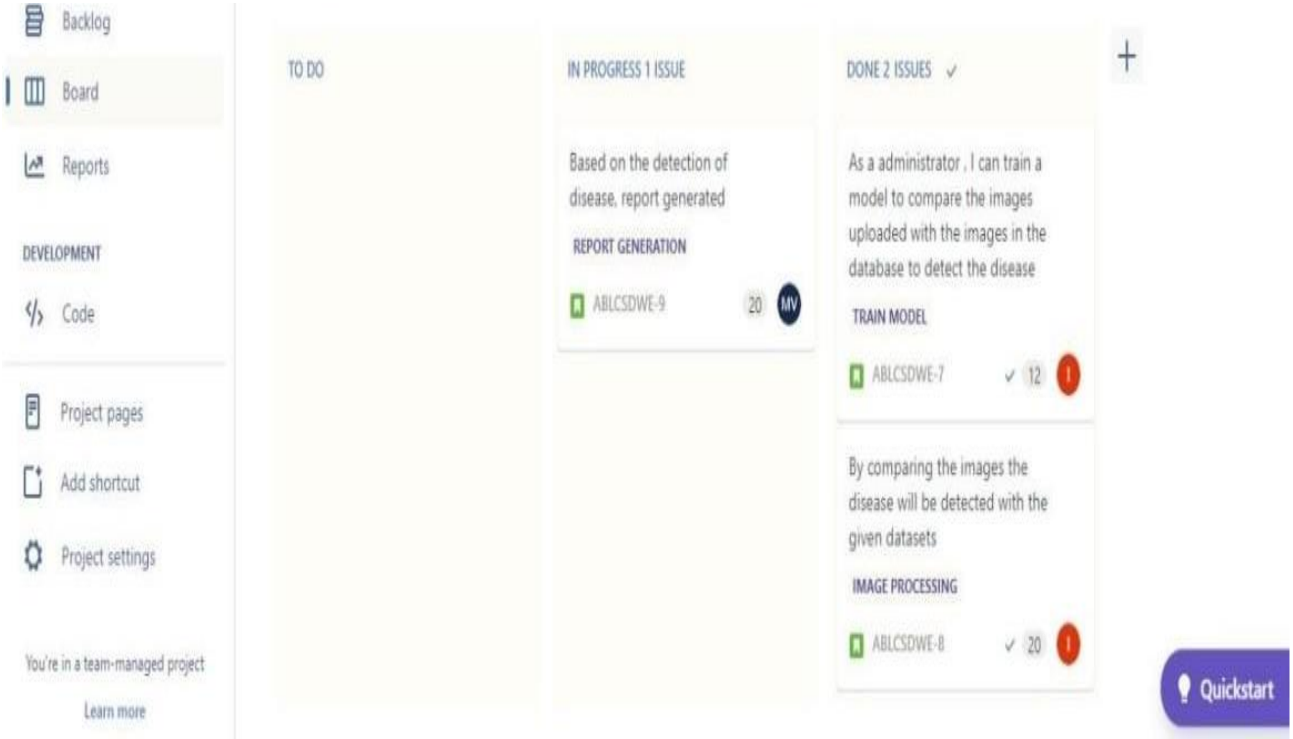
+ Create issue

Sprint 4 11 Nov – 15 Nov (1 issue) 0 20 0

ABLCSDWE-9 Based on the detection of disease, report generated REPORT GENERATION 20 IN PROGRESS MV

+ Create issue

Board:



TO DO

IN PROGRESS 1 ISSUE

Based on the detection of disease, report generated

REPORT GENERATION

ABLCSDWE-9 20 MV

DONE 2 ISSUES ✓

As a administrator , I can train a model to compare the images uploaded with the images in the database to detect the disease

TRAIN MODEL

ABLCSDWE-7 ✓ 12

By comparing the images the disease will be detected with the given datasets

IMAGE PROCESSING

ABLCSDWE-8 ✓ 20

7.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 is
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 Da
ta_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
```

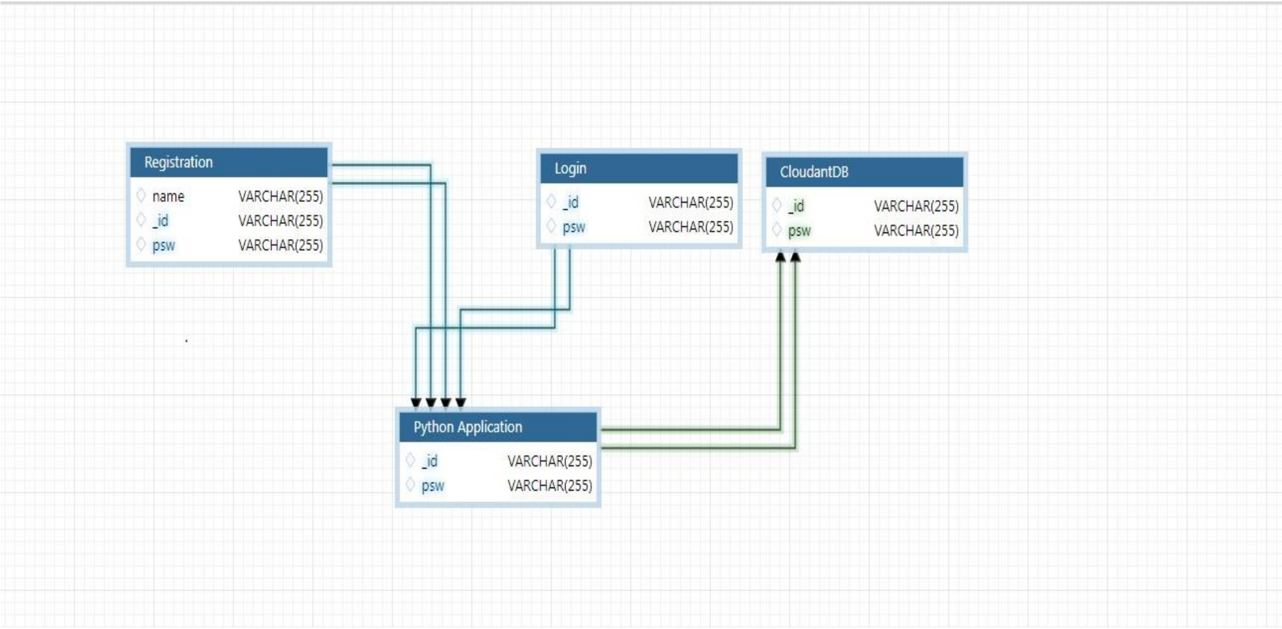
```
        performance and less
        accuracy. Default is False.",
    )
    parser.add_argument( "--random_seed", type=float, default=None, help="Random seed
        value to make script deterministic. Default is 'None', i.e.
non-deterministic.",
    )
    parser.add_argument( "--epochs", type=float, default=51, help="Number of epochs for
        training last layers and number of epochs for fine-
tuning 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:



8.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

9.RESULTS

9.1 Performance Metrics

S.No.	Parameter	Values
1.	Model Summary	To evaluate object detection 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% Validation Accuracy – 95%
3.	Confidence Score (Only Yolo Projects)	Class Detected – 93% Confidence Score – 90%

10.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.

11.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.

12.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, readingradiology 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, andgenerally 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 patientengagement in their own care and streamline patient access to care.

13.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 timeit 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 dicrectory 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-
4298-88e8-3f887f8bbd2d-
bluemix','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 link:

https://github.com/IBM-EPBL/IBM-Project-12394-1659450057

Project Demo Link:
https://drive.google.com/file/d/1v2n_geg25zzLYdD8zfzbr3V1YeSXXQw8/view?usp=share_link