AI – Based localization and classification of skin disease with erythema

1. **INTRODUCTION**

1.1 Project Overview

This project focused on localising the region of infection and also generating its scientific name given an image as input using Convolutional Neural Networks (CNN) particularly the YOLO(You Only Look Once) model. The Deep Learning (DL) algorithm either tells us that if there is a presence of skin disease or not and if so, it's label. The various steps involved in the project are as follows:

- Creating the Dataset
- Data Pre-Processing
- Model Building
- UI Design Using Flask
- Prediction

1.2 Purpose

The process of identification of skin diseases is very much crucial for quick and robust recovery. And the process of manual identification can be inaccurate and such cases might lead to extensive and deepened spread of the said disease thereby making the treatment and recovery difficult. Thus we intend to make use of a computer to aid the manual diagnosis to solve the above problems.

2. **LITERATURE SURVEY**

2.1 Existing problem

1. Computer-aided diagnosis for CT colonography.

CT colonography, or virtual colonoscopy, is a promising alternative screening tool for colon cancer. Computer-aided diagnosis (CAD) for CT colonography has the potential to increase radiologists' diagnostic performance in the detection of polyps and to reduce variability of the diagnostic accuracy among readers. Technical developments have advanced CAD for CT colonography substantially during the last several years. This paper describes the key techniques used for CAD for detection of polyps and masses in CT colonography, the current detection performance, and challenges and the future of CAD.

The ultimate goal of CAD is to improve the performance of radiologists in the detection of polyps and masses, thus establishing sensitivity and specificity is only the first step in the evaluation of the benefit of CAD.CAD must be shown to improve the performance of radiologists.

CAD algorithms depend on a shape analysis that assumes that polyps appear to have a cap-like shape, i.e., they appear as polypoid lesions. But some like, sessile polyps that do not protrude sufficiently into the lumen, whose shape is significantly deviates from polypoid, or those that lose a portion due to the partial volume effect, may be missed by CAD.

2. Application of region-based segmentation and neural network edge detection to skin lesions.

This work investigates the application of two approaches to the skin lesion segmentation problem; iterative segmentation (IS) and neural network edge detection (NNED). The aim is to quantitatively analyze the error in locating the border due to the application of an automated segmentation method. The automatic skin segmentation (ASS) method presented by Xu et al. [12] is also used here to verify the other two proposed methods. These approaches are compared for synthetic lesions at different image signal to noise ratios (SNRs).

The use of synthetic lesions is advantageous in initial analysis and verification, as by knowing the true position of the lesion border the different methods can be quantitatively and more accurately compared.

In practice, this case may not always exist especially when considering a wide range of lesion scenes with widely different properties e.g. effect of noise and image details such as skin texture and hair. Moreover, the subsequent initial thresholding only finds an approximate lesion boundary, which does not always represent true region boundaries even if this region is refined using edge information in the image.

3.ERYTHEMA DETECTION IN DIGITAL SKIN IMAGES.

This work present a 3-layer segmentation scheme for automatic erythema detection. First, a skin region is detected with a histogrambased Bayesian classifier. Next, the extracted skin image is represented in terms of melanin and hemoglobin components based on Independent Component Analysis (ICA). At last, a trained Support Vector Machine (SVM) is applied to identify erythema areas using feature attributes from hemoglobin and melanin component images.

Before psoriasis lesion segmentation from skin is performed with a mixture of two Gaussians. One problem is that initial parameters of Gaussians from training samples often result in slow convergence and unstable recognition accuracy.With features extracted by ICA, SVM classification is applied to classify abnormal redness from normal skin so there will be fast convergence and stable recognition.

SVMs are not suitable for large datasets.The complexity of the algorithm's training is highly dependent on the size of the dataset.

4. Skin Disease Analysis and Tracking based on Image Segmentation.

The proposed method is based on two steps - the first step is a preprocessing one which consists in image segmentation to detect the edge of the infected skin region. In the second one, another proposed method is applied to measure the wound 'size' and control the illness evolution. A comparative study was realized to select the most suitable segmentation technique referred to a proposed criterion based on 'edge accuracy' EAC. The new criterion was compared with the 'surface accuracy' based on ROC space.

These methods have encountered many problem like transparency tracing grid preprinted in 1mm 2 areas so take more counting time and are not suited to routine practice as well it's so hard to keep the transparency grid or the ruler against the ulcer and it can be discomfort to patients afraid of the risk of contamination or cross-infection. But in this work a computerized method is used to measure the skin disease (i.e.)the measurement of the surface of the infected skin using the counting the digital pixels.

FCM method has some disadvantages such as its need for a large amount of time to converge and it is more sensitive to the noise and outliers in the data. In ROC,Confidence scores used to build ROC curves may be difficult to assign. False-positive and false-negative diagnoses have different misclassification costs.

5. Independent component analysis of skin color image.

The spatial distributions of melanin and hemoglobin in skin are separated by independent component analysis of skin color image. The independent component analysis(ICA) is a technique that extracts the original signals from mixtures of many independent sources without a prior information on the sources and the process of the mixture.

It reduces the dimensions to avoid the problem of overfitting. It decomposes the mixed signal into its independent sources signals.

Traditional Indepedent Component Analysis Algorithm leads to local minimum solution, and the suitable source signals are not isolated .But it is overcomed by artificial immune system.

2.2 References

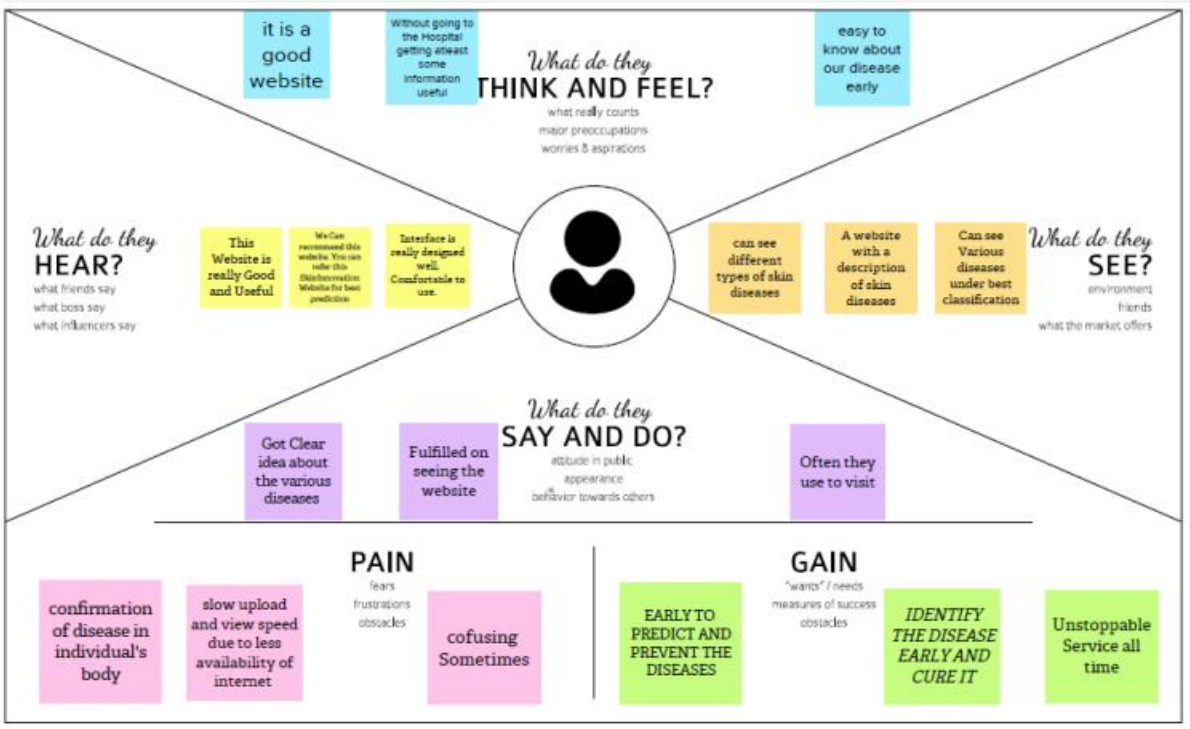
- Doi, K. Computer-aided diagnosis in medical imaging: Historical review, current status and future potential. Comput. Med. Imaging Graph.(2007).
- Trabelsi, O., Tlig, L., Sayadi, M. & Fnaiech, F., Skin disease analysis and tracking based on image segmentation. 2013 International Conference on Electrical Engineering and Sofware Applications, Hammamet.(2013).
- Rajab, M. I., Woolfson, M. S. & Morgan, S. P. Application of region-based segmentation and neural network edge detection to skin lesions. Comput. Med. Imaging Graph.(2004).
- Keke, S., Peng, Z. & Guohui, L., Study on skin color image segmentation used by fuzzy-c-means arithmetic. In 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery, Yantai.(2010).

2.3 Problem Statement Definition

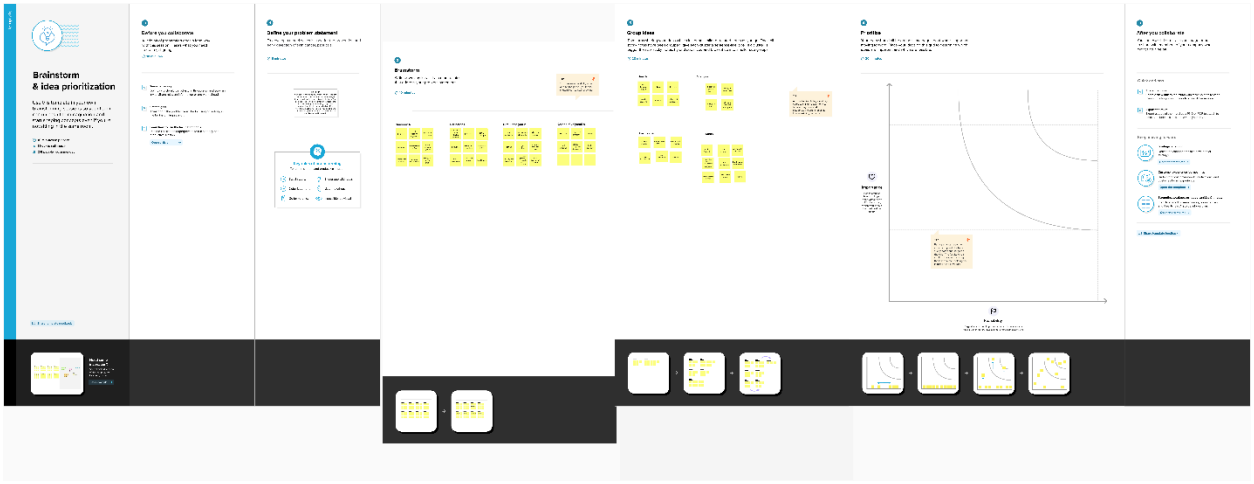
Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Suffering from symptoms of Skin diseases	Predict if I have skin diseases or not and if yes, the type skin disease	I am not financially stable to take expensive scans and professional advice	Prediction of skin diseases is a tedious process	Secure from severe cause and effect that can occur if not treated early
PS-2	Dermatologist	To detect skin disease with erythema	To analyze the kind of Skin disease	To predict the type of Skin disease	To prevent from severe cause and effect that can occur if not treated early

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



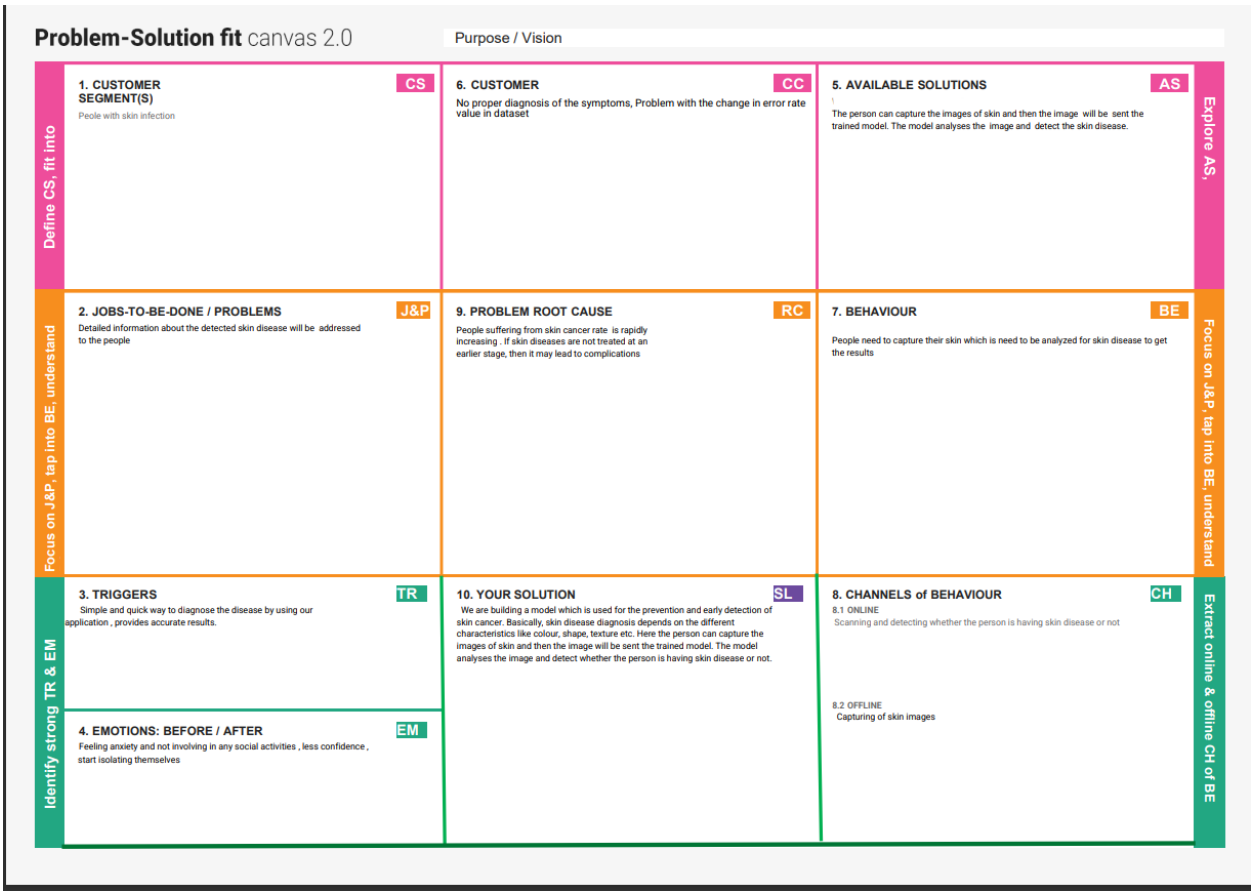
3.2 Ideation & Brainstorming



3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• Erythema is the skin disease associated with redness or skin rash. It is caused due to some allergic reaction or infection. Sometimes they are chronic, infectious and may develop into skin cancer. • The diagnosis and treatment of skin diseases like erythema takes longer time and causes financial and physical cost to the patient and also it leads to wrong prediction.
2.	Idea / Solution description	• We are building a model that is used for the prevention and early detection of Erythema. • An image processing-based approach is used to diagnose the erythema. The approach works on the inputs of a color image. Then resize the image to extract features using pre-trained convolutional neural network.
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 detailed description of the disease.
4.	Social Impact / Customer Satisfaction	• It is easy to access as everyone is using mobile phones now-a-days. • It is a time and money-saving process.
		• It is highly secured and the data is not delivered to third persons.
5.	Scalability of the Solution	• It can be used by dermatologists (skin specialist doctors) when they find difficulty in diagnosing the skin disease and may require expensive laboratory tests to correctly identify the type and stage of the skin disease.

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

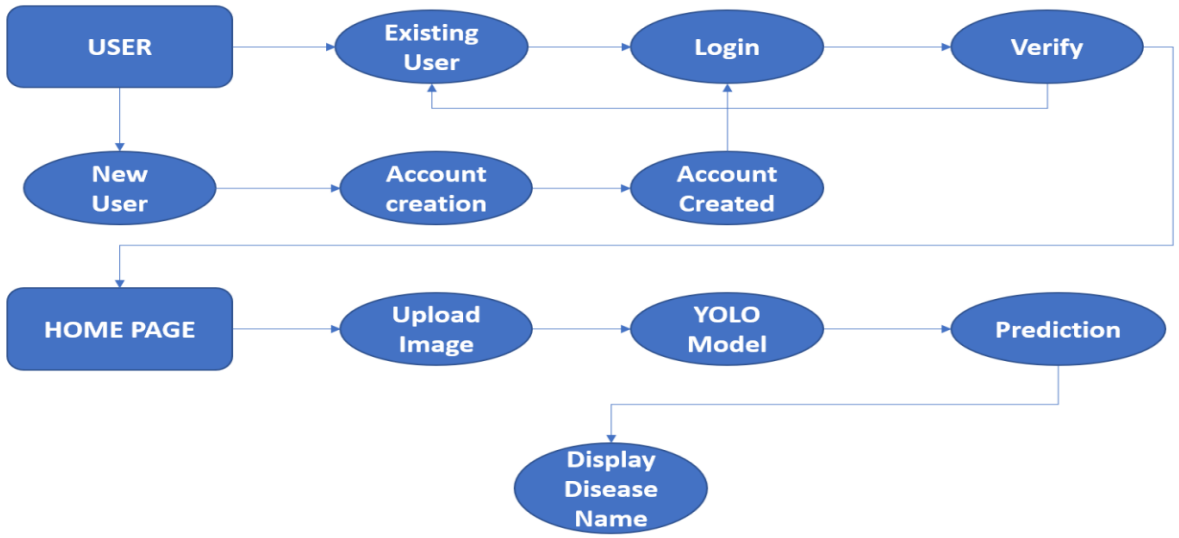
R No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Email.
FR-2	Patient Image Capturing Process	Provide Access to Upload Image Through Gallery.
FR-3	Disease Prediction	Prediction of disease in the uploaded Image.

4.2 Non-Functional requirements

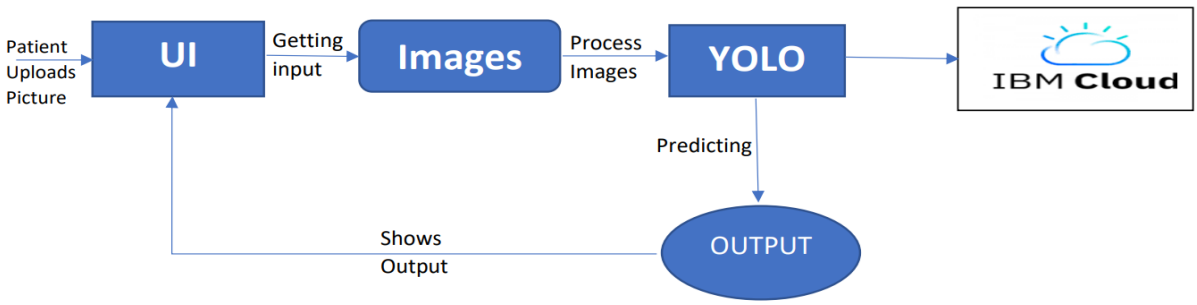
R No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Our application will be easy to use.
NFR-2	**Security**	Data privacy and security will be provided.
NFR-3	**Reliability**	Our application will perform consistently in the specified timings without failure.
NFR-4	**Performance**	Website's load time should be less for the users.
NFR-5	**Availability**	How likely the system is accessible to a user at a point of time,
NFR-6	**Scalability**	The ability to handle increasing and decreasing workloads.

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



6. PROJECT PLANNING & SCHEDULING

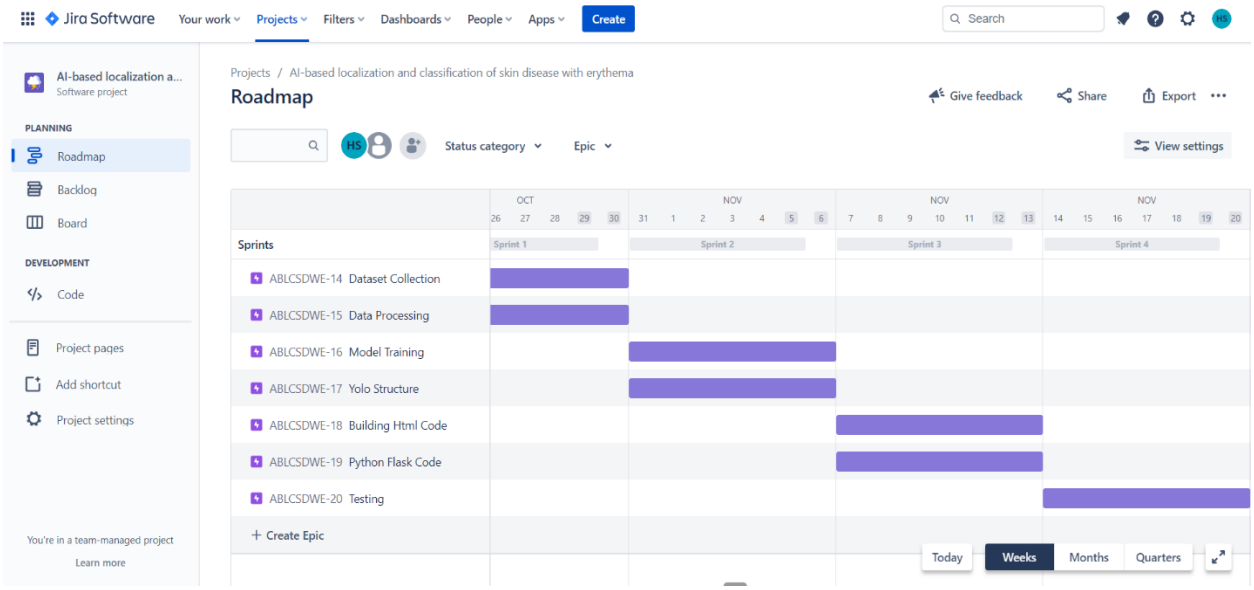
6.1 Sprint Planning & Estimation

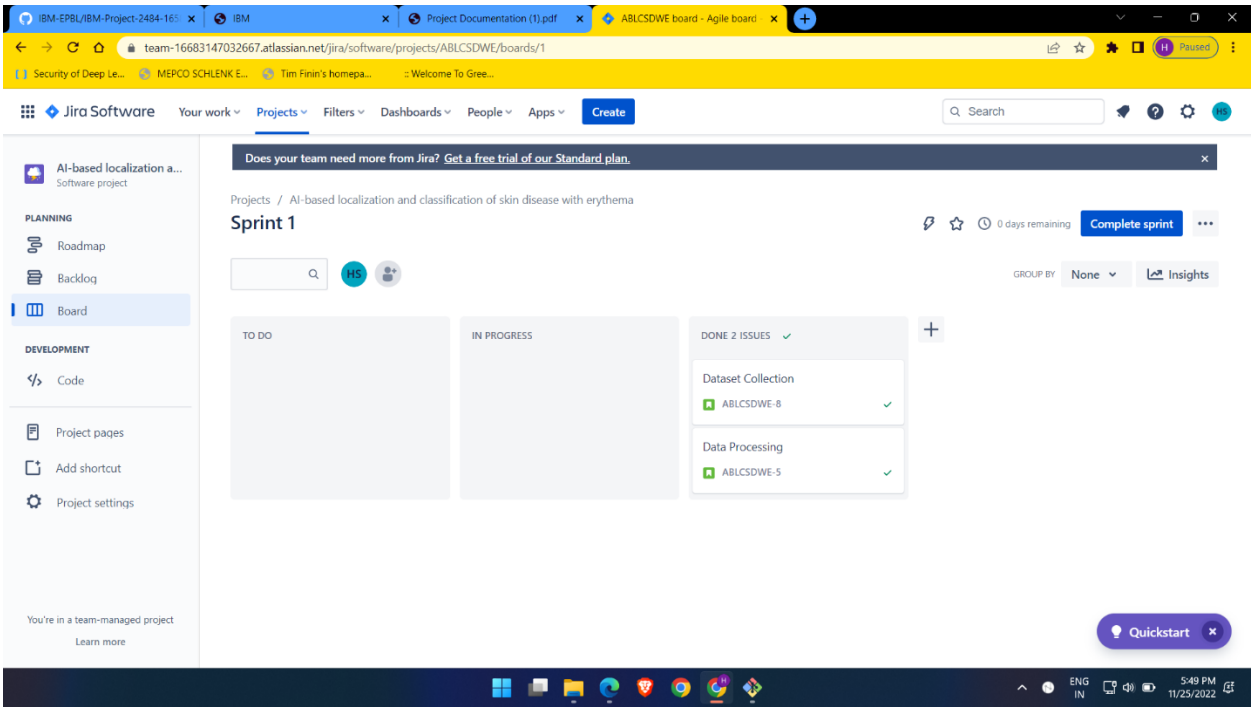
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	5 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

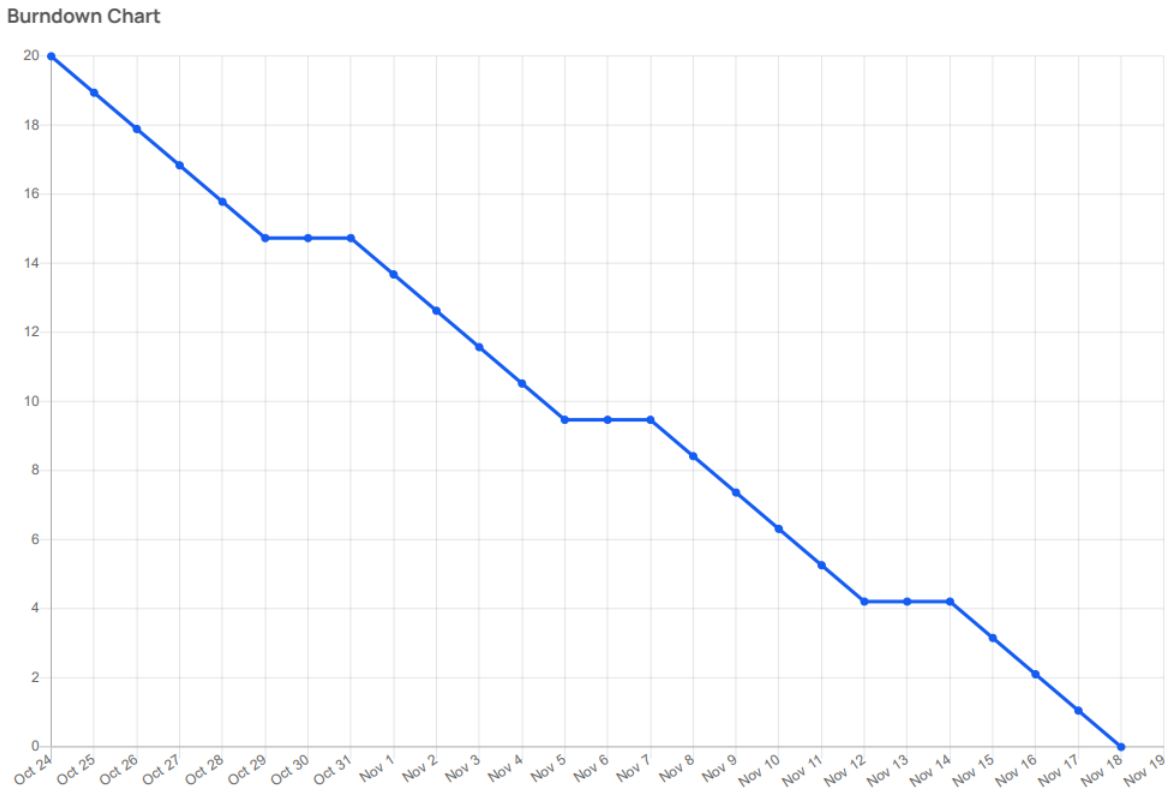
6.2 Sprint Delivery Schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	Collect dataset and preprocess the images	2	High	Hariharan S, Girivasan R
Sprint-2	Model Building	USN-2	Split the dataset and train the model	1	High	Nirai Pandiyan P, Chinna Muniyandi A
Sprint-3	Build the Website	USN-3	Build html pages and the backend using flask	2	Low	Hariharan S, Nirai Pandiyan P
Sprint-4	Final touch	USN-4	Correction, Restructuring and final documentation	2	Medium	Girivasan R, Chinna Muniyandi A

6.3 JIRA




IBM-EPBL/IBM-Project-2484-161
IBM
Project Documentation (1).pdf
ABLCSDWE board - Agile board
team-16683147032667.atlassian.net/jira/software/projects/ABLCSDWE/boards/1
Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
AI-based localization a...
Software project
PLANNING
Roadmap
Backlog
Board
DEVELOPMENT
Code
Project pages
Add shortcut
Project settings
You're in a team-managed project
Learn more
Does your team need more from Jira? Get a free trial of our Standard plan.
Projects / AI-based localization and classification of skin disease with erythema
Sprint 1
0 days remaining
Complete sprint
GROUP BY
None
Insights
TO DO
IN PROGRESS
DONE 2 ISSUES
Dataset Collection
ABLCSDWE-8
Data Processing
ABLCSDWE-5
Quickstart
ENG
IN
5:49 PM
11/25/2022


Burndown Chart
20
18
16
14
12
10
8
6
4
2
0
Oct 24
Oct 25
Oct 26
Oct 27
Oct 28
Oct 29
Oct 30
Oct 31
Nov 1
Nov 2
Nov 3
Nov 4
Nov 5
Nov 6
Nov 7
Nov 8
Nov 9
Nov 10
Nov 11
Nov 12
Nov 13
Nov 14
Nov 15
Nov 16
Nov 17
Nov 18
Nov 19

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Python Flask Code:

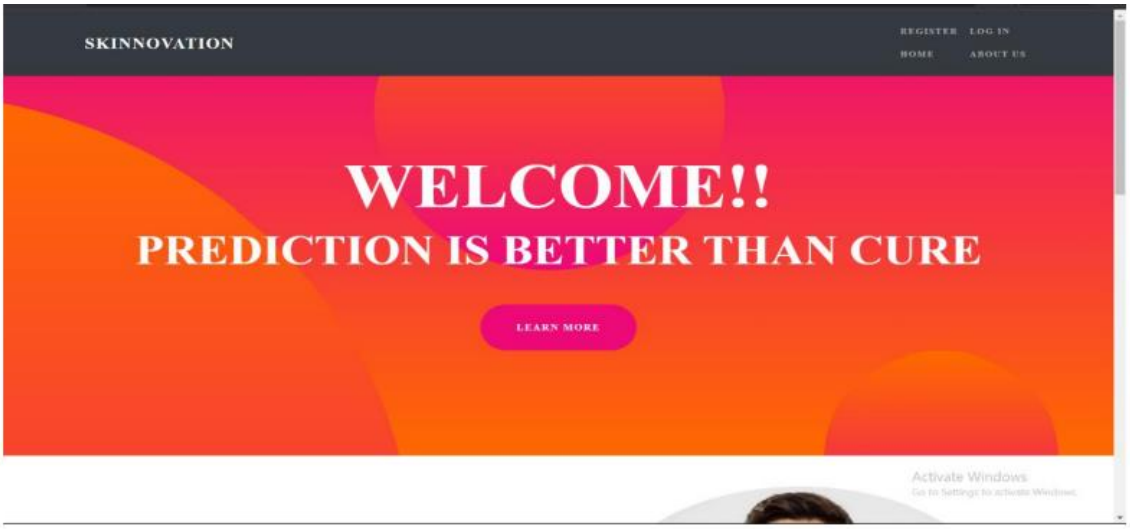
```
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
from flask import Flask, request,flash, render_template, redirect,url_for,session
from cloudant.client import Cloudant
from twilio.rest import Client
model = load_model(r"erythema_detection.h5")
app = Flask(__name__)
app.secret_key="abc"
app.config['UPLOAD_FOLDER'] = "User_Images"
# Authenticate using an IAM API key
client = Cloudant.iam('7d22d94d-826f-45f8-b4b3-978d8295f90e-bluemix',
                      '5om7V-HPoW2u-ZysZHXSWHeeud2-zmUEsueMa-lR0b1k', connect=True)
# Create a database using an initialized client
my_database = client.create_database('my_db')
if my_database.exists():
   print("Database '{0}' successfully created.".format('my_db'))
# default home page or route

user = ""

@app.route('/')
def index():
   return render_template('homepage.html', pred="Login", vis ="visible")


# registration page
@app.route('/register',methods=["GET","POST"])
def register():
   if request.method == "POST":
      name =  request.form.get("name")
      mail = request.form.get("email")
      pswd = request.form.get("password")
      print(name)
      print(mail)
      print(pswd)
      data = {
        'name': name,
        'mail': mail,
```

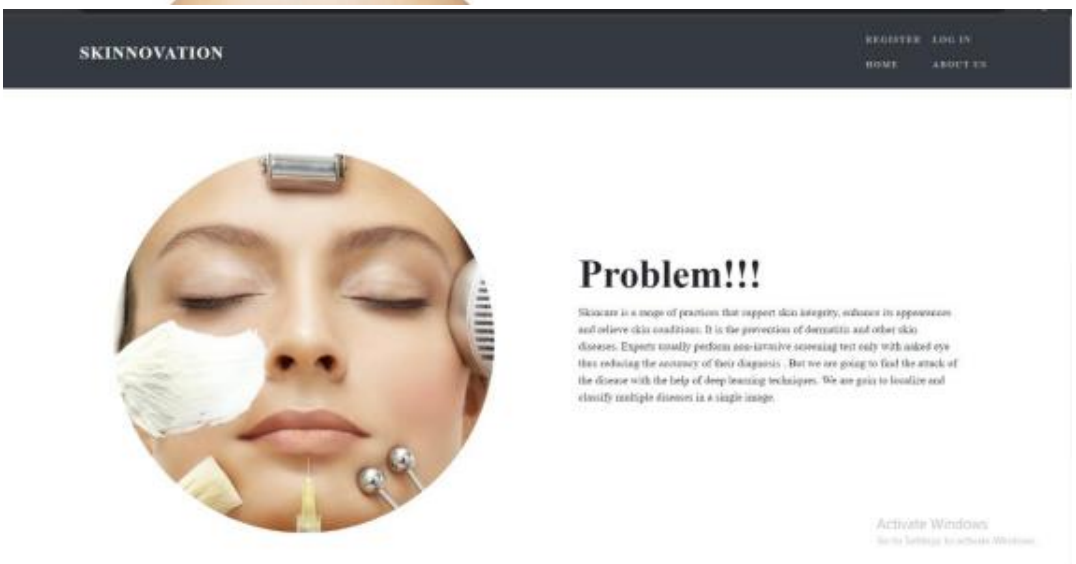
```
        'psw': pswd
    }
    print(data)
    query = {'mail': {'$eq': data['mail']}}
    docs = my_database.get_query_result(query)
    print(docs)
    print(len(docs.all()))
    if (len(docs.all()) == 0):
        url = my_database.create_document(data)
        return render_template("register.html", 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 ")
  else:
    return render_template('register.html')


@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == "GET":
        user = request.args.get('email')
        passw = request.args.get('password')
        print(user, passw)
        query = {'mail': {'$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')
        else:
            if ((user == docs[0][0]['mail'] and passw == docs[0][0]['psw'])):
                session['user'] = user
                flash("Logged in as " + str(user))
                return render_template('prediction.html', pred="Logged in as "+str(user), vis ="hidden", vis2="visible")
            else:
                return render_template('login.html', pred="The password is wrong.")
    else:
        return render_template('login.html')


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

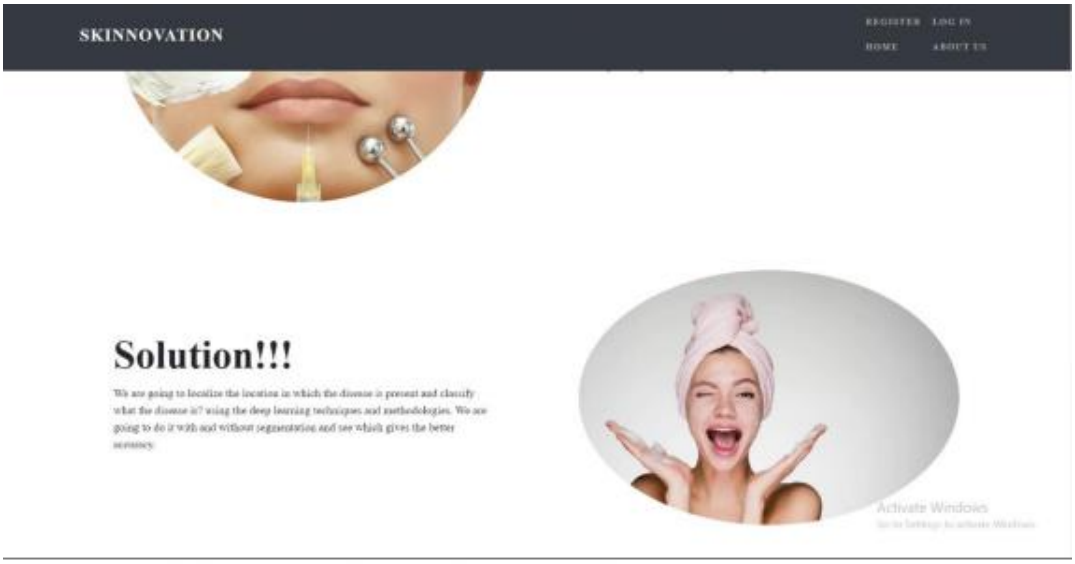
```
        session.pop('user',None)
        return render_template('logout.html')

@app.route("/predict",methods=["GET", "POST"])
def predict():
        if request.method == "POST":
            f = request.files['file']
            # getting the current path 1.e where app.py is present
            basepath = os.path.dirname(__file__)
            #print ( " current path " , basepath )
            # from anywhere in the system we can give image but we want that
            filepath = os.path.join(str(basepath), 'User_Images', str(f.filename))
            #print ( " upload folder is " , filepath )
            f.save(filepath)
            img = image.load_img(filepath, target_size=(299, 299))
            x = image.img_to_array(img)  # ing to array
            x = np.expand_dims(x, axis=0)  # used for adding one more dimension
            #print ( x )
            img_data = preprocess_input(x)
            prediction = np.argmax(model.predict(img_data), axis=1)
            index = [' No erythema ','Erythema chronicum migrans','Erythema infectiosum','Erythema
marginatum','Palmar erythema']
            result = str(index[prediction[0]])
            print(result)
            return render_template('prediction.html', prediction=result, fname = filepath)
        else:
            return render_template("prediction.html")

@app.route('/index')
def home():
    login=False
    if 'user' in session:
        login=True
    return render_template("homepage.html", pred="Login", vis ="visible",login=login)

if __name__ == "__main__":
    app.debug = False
    app.run()
```

7.2 Html pages

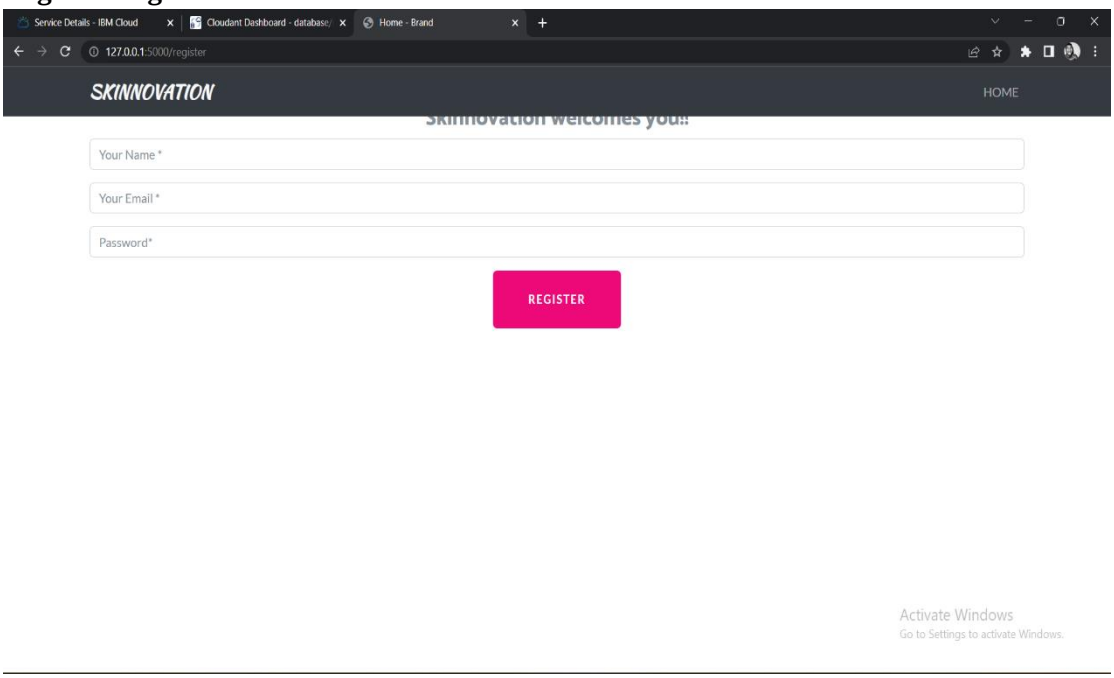
HomePage:



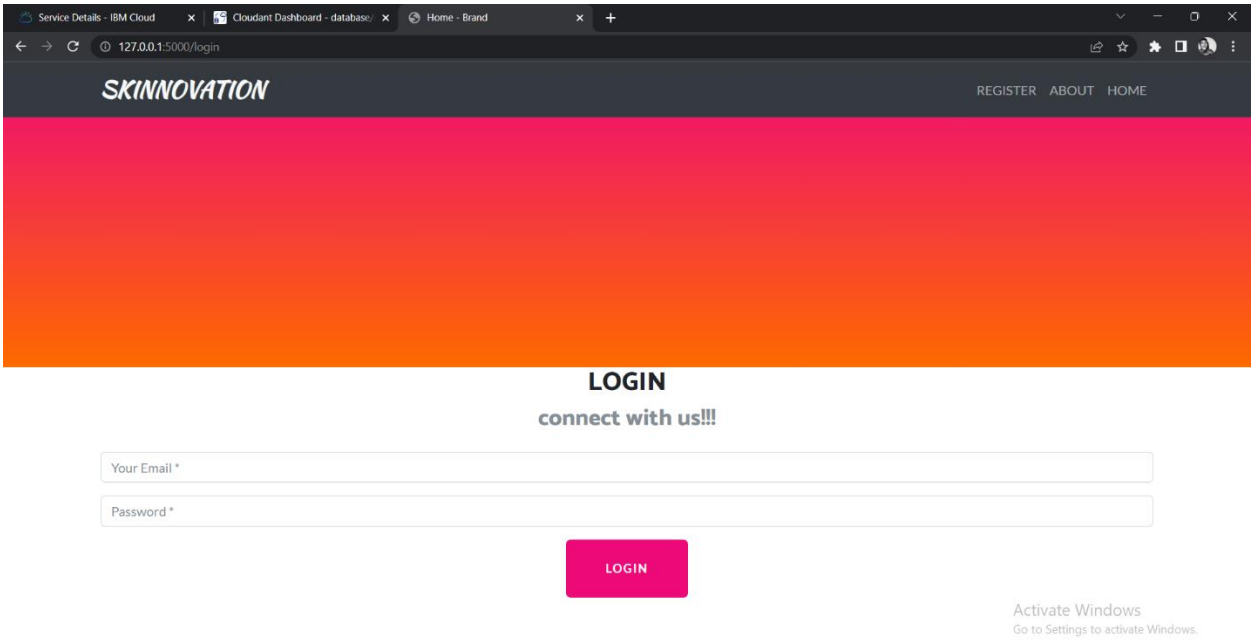




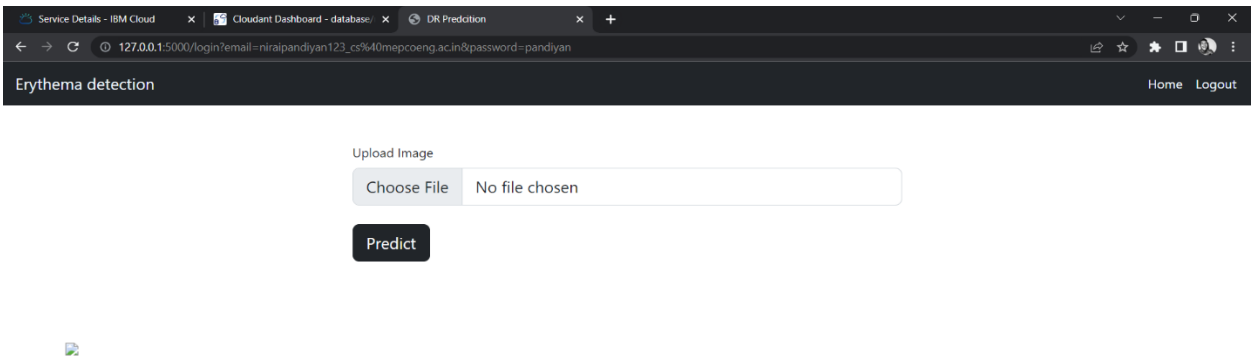
Register Page:



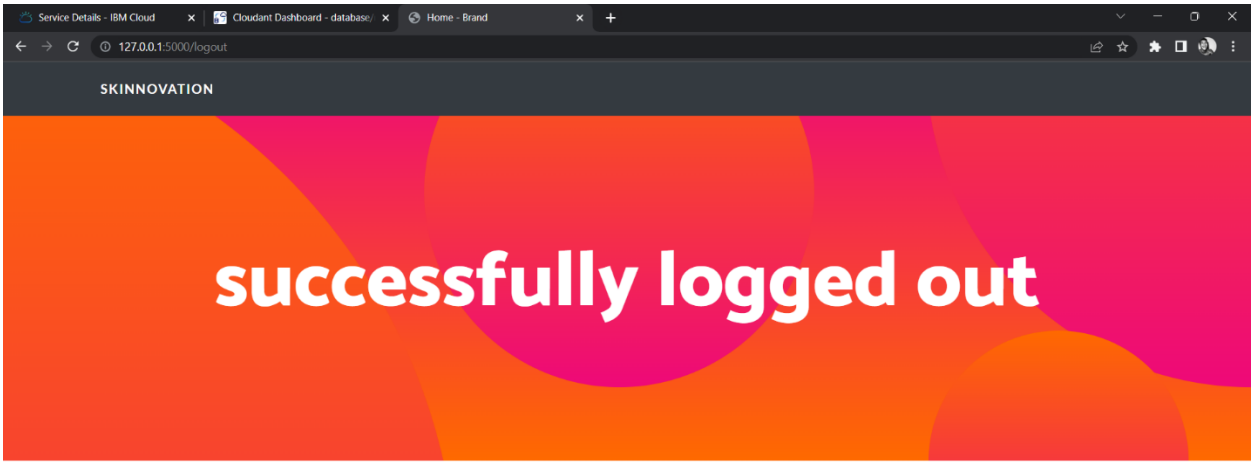
Login Page:



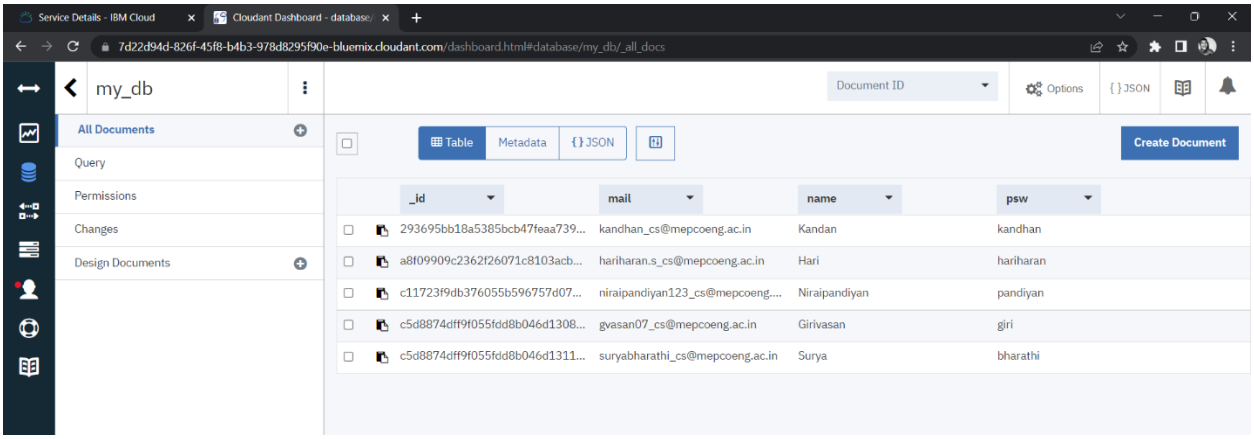
Predict Page:



LogoutPage:



7.3 Database Schema:



8. TESTING

8.1 Test Cases

Test Case No.	Action	Expected Output	Actual Output	Result
1	Registration	Storing Name, email, and password in IBM Cloud	Stores name,email, and password in Database	Pass
2	Login	Login with registered credentials	Login with registered credentials	Pass
3	Disease Prediction	Prediction of disease	It should predict disease	Pass

8.2 User Acceptance Testing

Section	Total cases	Not tested	Fail	Pass
Registration	5	0	0	5
Login	5	0	0	5
Disease detection	10	0	0	10
Final report output	2	0	0	2

9. RESULTS

9.1 Performance Metrics

S.No.	Parameter	Values
1.	Accuracy	Training Accuracy – 89% Validation Accuracy – 95%
2.	Confidence Score (Yolo)	Class Detected – 93% Confidence Score – 90%

10. ADVANTAGES

Very little human intervention and doesn't require much technical knowledge both in aspects of medical and computer science

Continued improvement of prediction by including new user inputs to the training set

Can identify any number of types of diseases provided enough number of samples in each of that type

11. DISADVANTAGES

No surety can be given for the results of prediction

Model training phase of the model can take hours of time depending on the dataset

Periodic updation of the training set is mandatory for keeping up with any sorts of evolution

12. CONCLUSION

From a proper analysis of positive points and constraints on the component, it can be safely concluded that the product is a highly efficient. This application is working properly and meeting all of Doctors' requirements. This component can be easily plugged with many other systems.

13. FUTURE SCOPE

Model can be trained alongside other models that were trained using biological factors such as vitals of the patient, blood reports for accurate prediction and not just being dependent upon the physical appearance.

14. APPENDIX

Model Code:

```
import os
import sys


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 = os.path.join(get_parent_dir(1), "2_Training", "src")
utils_path = os.path.join(get_parent_dir(1), "Utils")

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), "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

if __name__ == "__main__":
    # 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

if file_types:
    input_paths = GetFileList(FLAGS.input_path, endings=file_types)
else:
    input_paths = GetFileList(FLAGS.input_path)

# 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()
```

App.py:

```
import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
from flask import Flask, request,flash, render_template, redirect,url_for,session
from cloudant.client import Cloudant
from twilio.rest import Client
model = load_model(r"erythema_detection.h5")
app = Flask(__name__)
app.secret_key="abc"
app.config['UPLOAD_FOLDER'] = "User_Images"
# Authenticate using an IAM API key
client = Cloudant.iam('7d22d94d-826f-45f8-b4b3-978d8295f90e-bluemix',
                      '5om7V-HPoW2u-ZysZHXSWHeeud2-zmUEsueMa-lR0b1k', connect=True)
# Create a database using an initialized client
my_database = client.create_database('my_db')
```

```
if my_database.exists():
    print("Database '{0}' successfully created.".format('my_db'))
# default home page or route

user = ""

@app.route('/')
def index():
    return render_template('homepage.html', pred="Login", vis ="visible")



# registration page
@app.route('/register',methods=["GET","POST"])
def register():
    if request.method == "POST":
        name =  request.form.get("name")
        mail = request.form.get("email")
        pswd = request.form.get("password")
        print(name)
        print(mail)
        print(pswd)
        data = {
          'name': name,
          'mail': mail,
          'psw': pswd
        }
        print(data)
        query = {'mail': {'$eq': data['mail']}}
        docs = my_database.get_query_result(query)
        print(docs)
        print(len(docs.all()))
        if (len(docs.all()) == 0):
            url = my_database.create_document(data)
```

```
            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 ")
    else:
        return render_template('register.html')


@app.route('/login', methods=['GET','POST'])
def login():
        if request.method == "GET":
            user = request.args.get('email')
            passw = request.args.get('password')
            print(user, passw)
            query = {'mail': {'$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')
            else:
                if ((user == docs[0][0]['mail'] and passw == docs[0][0]['psw'])):
                    session['user'] = user
                    flash("Logged in as " + str(user))
                    return render_template('prediction.html', pred="Logged in as "+str(user), vis ="hidden", vis2="visible")
                else:
                    return render_template('login.html', pred="The password is wrong.")
        else:
            return render_template('login.html')


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

```
def logout():
    session.pop('user',None)
    return render_template('logout.html')

@app.route("/predict",methods=["GET", "POST"])
def predict():
    if request.method == "POST":
        f = request.files['file']
        # getting the current path 1.e where app.py is present
        basepath = os.path.dirname(__file__)
        #print ( " current path " , basepath )
        # from anywhere in the system we can give image but we want that
        filepath = os.path.join(str(basepath), 'User_Images', str(f.filename))
        #print ( " upload folder is " , filepath )
        f.save(filepath)
        img = image.load_img(filepath, target_size=(299, 299))
        x = image.img_to_array(img)  # ing to array
        x = np.expand_dims(x, axis=0)  # used for adding one more dimension
        #print ( x )
        img_data = preprocess_input(x)
        prediction = np.argmax(model.predict(img_data), axis=1)
        index = [' No erythema ','Erythema chronicum migrans','Erythema infectiosum','Erythema marginatum','Palmar erythema']
        result = str(index[prediction[0]])
        print(result)
        return render_template('prediction.html', prediction=result, fname = filepath)
    else:
        return render_template("prediction.html")

@app.route('/index')
def home():
  login=False
  if 'user' in session:
    login=True
```

```
    return render_template("homepage.html", pred="Login", vis ="visible",login=login)


if __name__ == "__main__":
    app.debug = False
    app.run()
```

GitHub:

Github link

Demo Video Link:

Video Link