AI BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMIA

Team ID:PNT2022TMID12466

KARPAGAM COLLEGE OF ENGINEERING

ROLLNO : 717819P206

TABLE OF CONTENTS

INTRODUCTION

1.1 Project Overview

Skin is the largest and most sensitive part of the human body which protects our inner vital parts and organs from the outside environment, hence avoiding contact with bacteria and viruses. Skin also helps in body temperature regulation. The skin consists of cells, pigmentation, blood vessels, and other components. It is comprised of 3 main layers, namely, the epidermis, the dermis, and the hypodermis.

Epidermis, being the outermost skin layer, forms a waterproof and protective sheath around the body's surface. The dermis, found beneath the epidermis, comprises of connective tissues and protects the body from stress and strain. A basement membrane tightly joins the dermis with the epidermis. The hypodermis, also called subcutaneous tissue, is not actually a part of the skin and lies below the dermis. It attaches the skin to the underlying bone and muscle and also supplies blood vessels and nerves to it.

1.2 Purpose

Classification of a disease is difficult due to the strong similarities between common skin disease symptoms. Therefore, it would be beneficial to exploit the strengths of CAD using artificial intelligence techniques, in order to improve the accuracy of dermatology diagnosis.The segmentation and classification of skin diseases has been gaining attention in the field of artificial intelligence because of its promising results.

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. 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.

LITERATURE REVIEW

2.1 EXISTING PROBLEMS:

If skin diseases are not treated at an earlier stage, then it may lead to complications in the body including spreading of the infection from one individual to the other. The skin diseases can be prevented by investigating the infected region at an early stage. The characteristic of the skin images is diversified so that it is a challenging job to devise an efficient and robust algorithm for automatic detection of skin disease and its severity. Skin tone and skin colour play an important role in skin disease detection. Colour and coarseness of skin are visually different. Automatic processing of such images for skin analysis requires quantitative discriminator to differentiate the diseases.

To overcome the above problem we are building a model which is used for the prevention and early detection of skin cancer, psoriasis. 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.

2.2 References

1. Yoshida, H. & Dachman, A. H. Computer-aided diagnosis for CT colonography. Semin. Ultrasound CT MRI 25, 419–431. https://doi.org/10.1053/j.sult.2004.07.002 (2004).
2. Trabelsi, O., Tlig, L., Sayadi, M. & Fnaiech, F., Skin disease analysis and tracking based on image segmentation. 2013 International Conference on Electrical Engineering and Software Applications, Hammamet, 1–7. https://doi.org/10.1109/ICEESA.2013.6578486 (2013).
3. 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. 28, 61–68. https://doi.org/10.1016/S0895-6111(03)00054-5 (2004).
4. 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, 612–615. https://doi.org/10.1109/FSKD.2010.5569451 (2010).

5. Hongmao, S. Quantitative Structure-Activity Relationships: Promise, Validations, and Pitfalls in A Practical Guide to Rational Drug Design 163–192 (Woodhead Publishing, Sawston, 2016). https://doi.org/10.1016/B978-0-08-100098-4.00005-3.
6. Lu, J., Manton, J. H., Kazmierczak E. & Sinclair, R., Erythema detection in digital skin images. In 2010 IEEE International Conference on Image Processing, Hong Kong, 2545–2548. https://doi.org/10.1109/ICIP.2010.5653524 (2010).

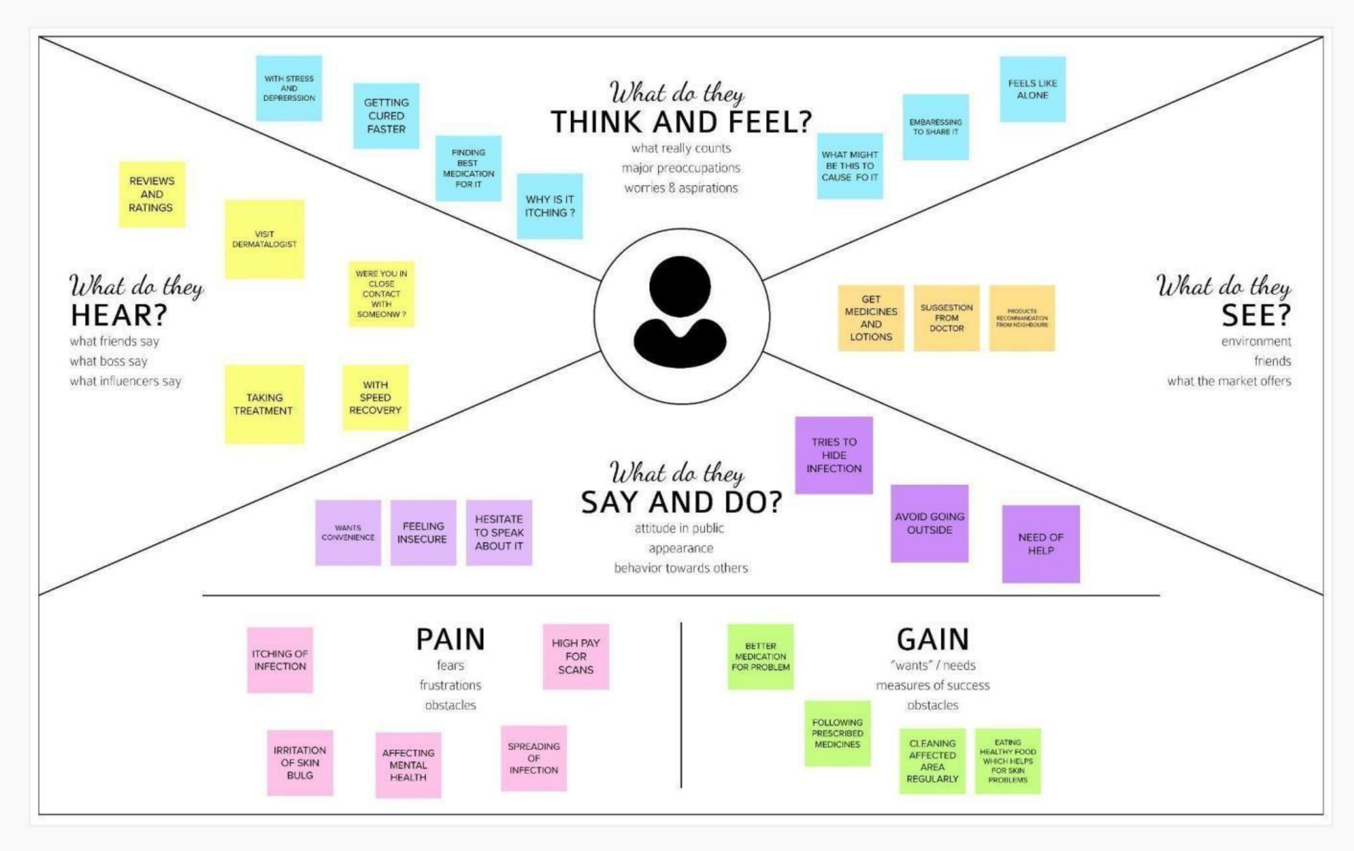
2.3 Problem Statement Definition

The characteristic of the skin images is diversified so that it is a challenging job to devise an efficient and robust algorithm for automatic detection of skin disease and its severity.We are building a model which is used for the prevention and early detection of skin cancer, psoriasis. Basically, skin disease diagnosis depends on the different characteristics like color, shape, texture etc…

- The person can capture the images of skin and then the image will be sent to the trained model. The model analyzes the image and detects whether the person is having skin disease or not.

IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS:



3.2 Ideation & Brainstorming

Brainstorming is a great way to generate a lot of ideas that you would not be able to generate by just sitting down with a pen and paper. The intention of brainstorming is to leverage the collective thinking of the group, by engaging with each other, listening, and building on other ideas. Conducting a brainstorm also creates a distinct segment of time when you intentionally turn up the generative part of your brain and turn down the evaluative part. You can use brainstorming throughout any design or work process, of course, to generate ideas for design solutions, but also any time you are trying to generate ideas, such as planning where to do empathy work, or thinking about product and services related to your project

3.3 Proposed Solution

Given an image of the skin, we decompose the image to normalize and extract high-level features. Using a 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 model.

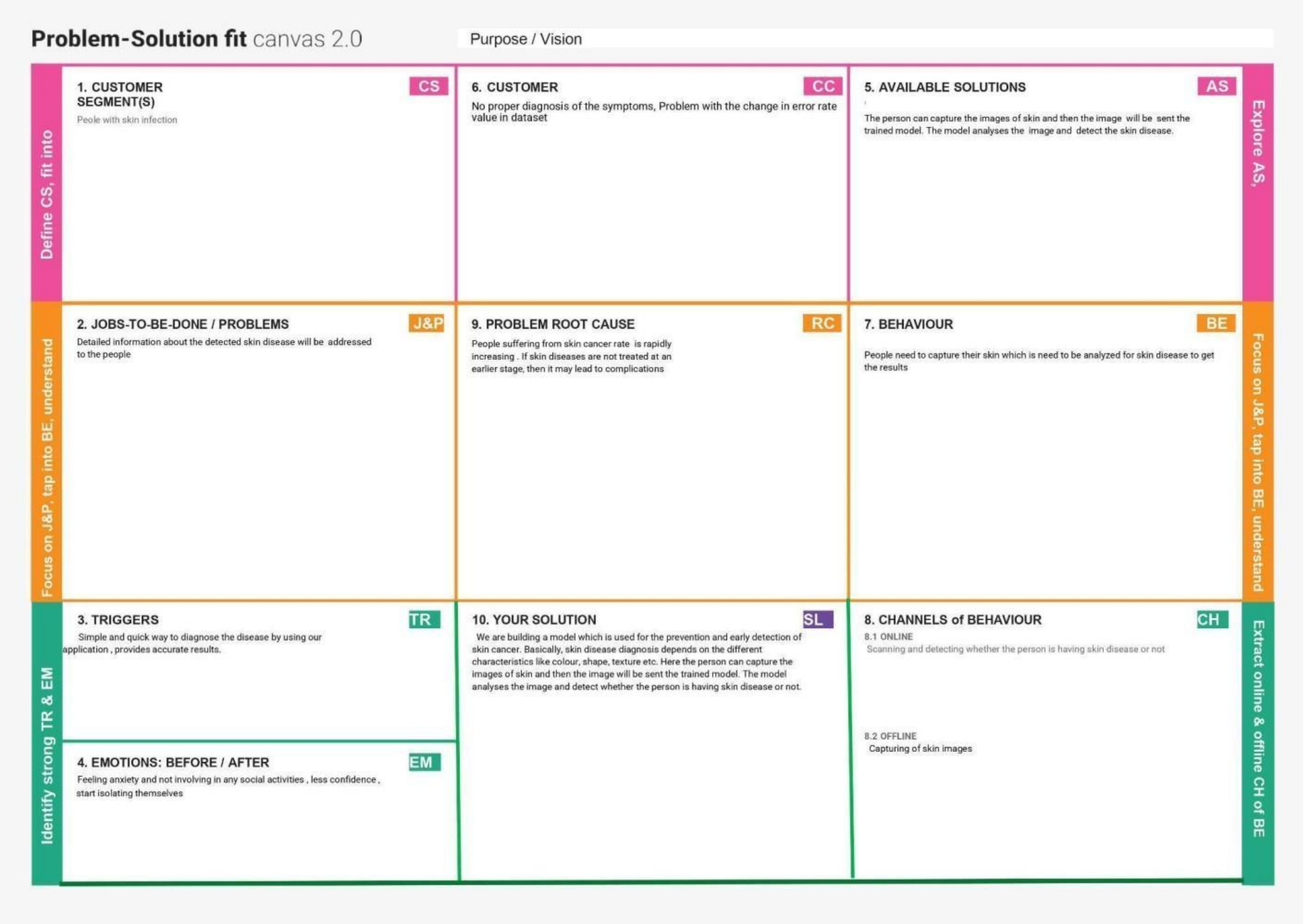
Furthermore, classification of a disease is difficult due to the strong similarities between common skin disease symptoms. Therefore, it would be beneficial to use CAD with AI techniques to improve the accuracy of dermatology

Two of the more prominent approaches for skin disease segmentation and classification are clustering algorithms and support vector machines (SVMs).

The methods described above lack the ability to localize and classify multiple diseases within one image. However, we have developed a method to address this problem.

In the past, skin disease models have been applied to either segmentation or classification. In this project, we sequentially combine both models by using the output of a segmentation model as input to a classification model.

3.4 Problem Solution fit



Problem-Solution fit canvas 2.0

Purpose / Vision

Define CS, fit into

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.

Explore AS,

Focus on J&P, tap into BE, understand

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

Focus on J&P, tap into BE, understand

Identify strong TR & EM

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

Extract online & offline CH of BE

REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Mobile Number Registration through Google Account
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Patient Image Capturing Process	Providing Access to Capture Image Through Camera Provide Access to Upload Image Through Gallery.
FR-4	Patient Medicine Reminder	Remind the patients to take their Medicine/ointments At the right time through the remaining alarm.
FR-5	Suggestion Box	Patients can take suggestions from the doctors through chats.
FR-6	Flareup Cycles	Patients can know their medicine level from doctors through messages.

4.2 NON-FUNCTIONAL REQUIREMENTS:

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Our Mobile phone application designed to improve the quality of patient‑held photos, and was developed to generate and hold their own skin images to help guide their skin care.
NFR-2	**Security**	Data privacy and security practices may vary based on users and their age
NFR-3	**Reliability**	Easy to use app to get personalized answers to your skin conditions questions
NFR-4	**Performance**	Good treatments are available for a variety of skin conditions including rash, itchy skin, skin fungus etc.
NFR-5	**Availability**	Our app helps you to screen your skin symptoms and prepare for your practitioner visit.

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.

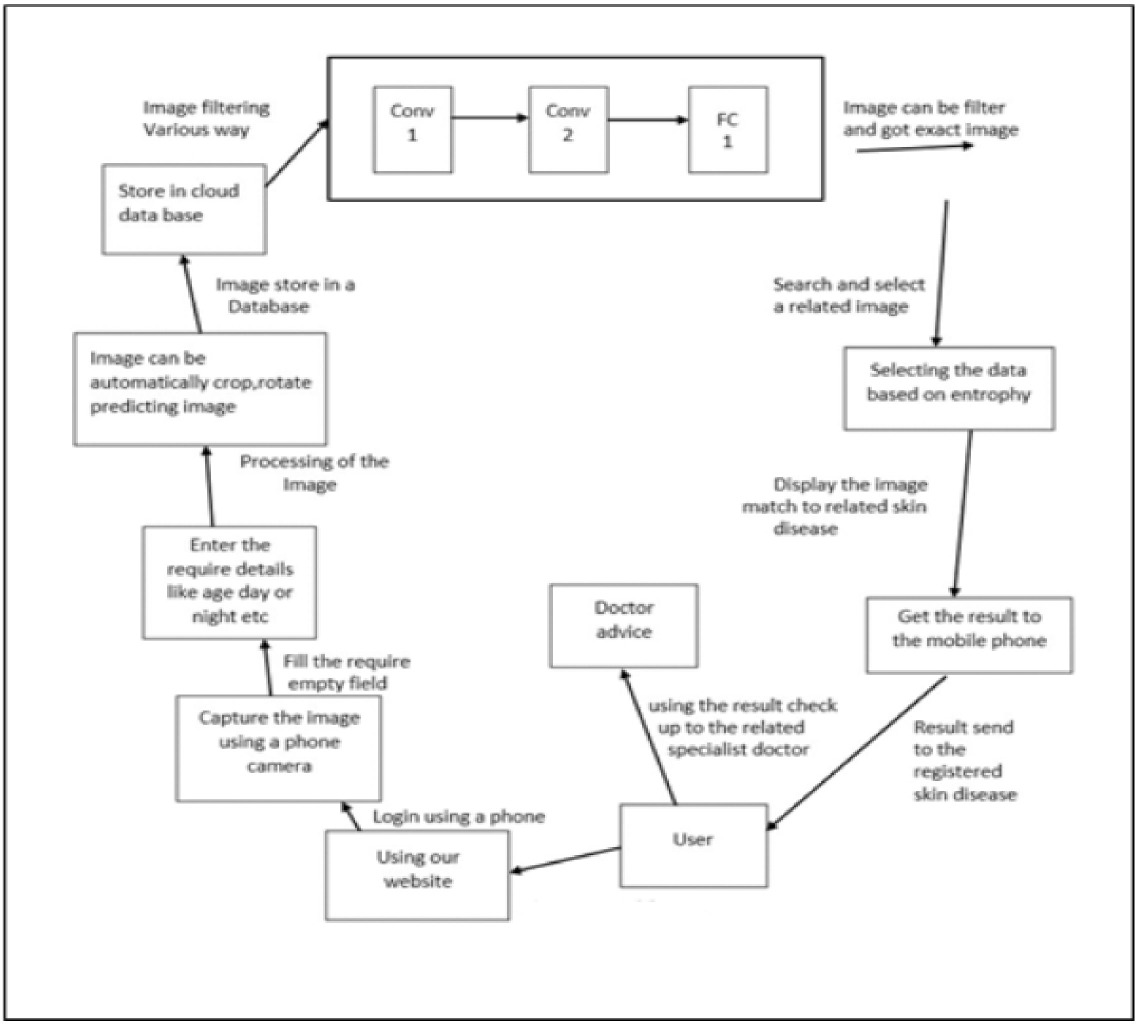


Image filtering Various way
Conv 1
Conv 2
FC 1
Image can be filter and got exact image
Store in cloud data base
Image store in a Database
Search and select a related image
Image can be automatically crop,rotate predicting image
Selecting the data based on entropy
Processing of the Image
Display the image match to related skin disease
Enter the require details like age day or night etc
Doctor advice
Get the result to the mobile phone
Fill the require empty field
using the result check up to the related specialist doctor
Capture the image using a phone camera
Result send to the registered skin disease
Login using a phone
User
Using our website

5.2 Solution & Technical Architecture

Solution architecture as well as technical 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:

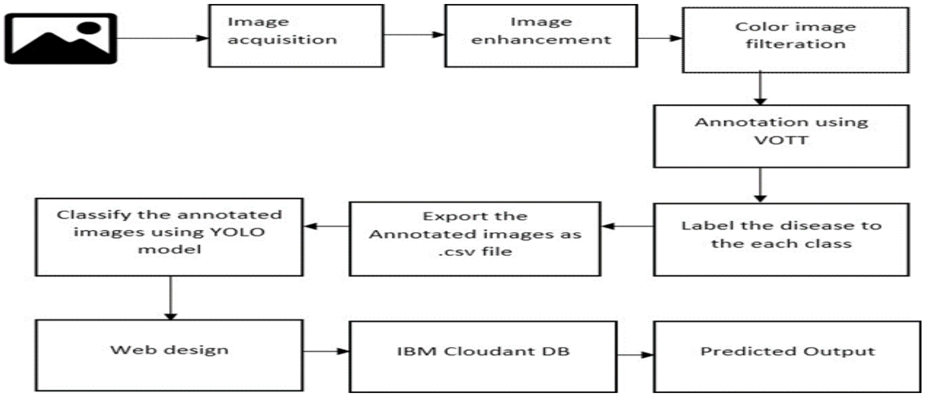
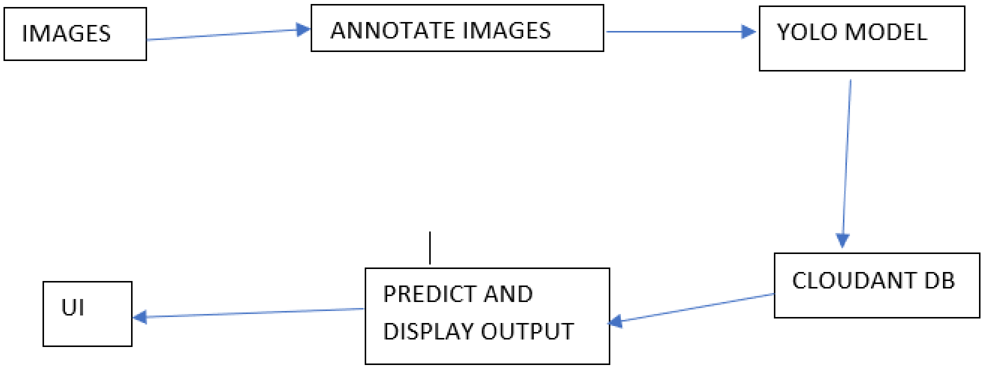


Image acquisition
Image enhancement
Color image filteration
Annotation using VOTT
Label the disease to the each class
Export the Annotated images as .csv file
Classify the annotated images using YOLO model
Web design
IBM Cloudant DB
Predicted Output

Technical Architecture:



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	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
		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
		USN-3	As a user, I can register for the application through Gmail	I can register & access the dashboard with Gmail	Medium	Sprint-1
	Login	USN-4	As a user , I can log into the application by entering email & password	I can use a login id and password	High	Sprint-1
	Dashboard	USN-5	As a user, I can see the configuration in a dashboard and use them	I can use all features in dashboard	Medium	Sprint-2
Customer (Web user)	Register	USN-1	As a user, I can register for the application entering my email,password and confirming my password	I can access my account/dashboard	High	Sprint-1
	Login	USN-2	As a user , I can log into the site by entering email & password	I can use a login id and password	High	Sprint-1
Customer Care Executive	Suggest a doctor	USN-2	Depend upon the skin disease the doctor can be suggested	Suggest a specialist doctor	Medium	Sprint-2
Administrato r	Maintain	USN-1	A data given by a users are maintained by a administrator	Data is kept in safe	High	Sprint-1

PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story Number	Story Points	Priority	Team Members
Sprint 1	Registration	USN 1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Anjana
Sprint 1	Confirmation	USN 2	As a user, I can register for the application by entering my email, password, and confirming my password.	1	High	Jeevika
Sprint 1	Login	USN 3	As a user, I can login for the application throughGmail	2	Medium	Premja
Sprint 1	Login	USN 4	As a user, I can log into the application by entering email& password	2	High	Premja
Sprint 1	Dashboard	USN 5	As a user, I can log into the application by entering email& password	2	High	Riyaz
Sprint 1	Data input	USN 7	As a user, I can log into the application by entering email& password	2	High	Riyaz
Sprint 1	Train Model	USN 8	As a user, I can log into the application by entering email& password	1	Medium	Anjana

Sprint 1	Image processing	USN 9	As a user, I can log into the application by entering email& password	2	High	Riyaz
Sprint 1	Report generation	USN 10	Based on the detection of disease, report generated	2	High	Jeevika

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**

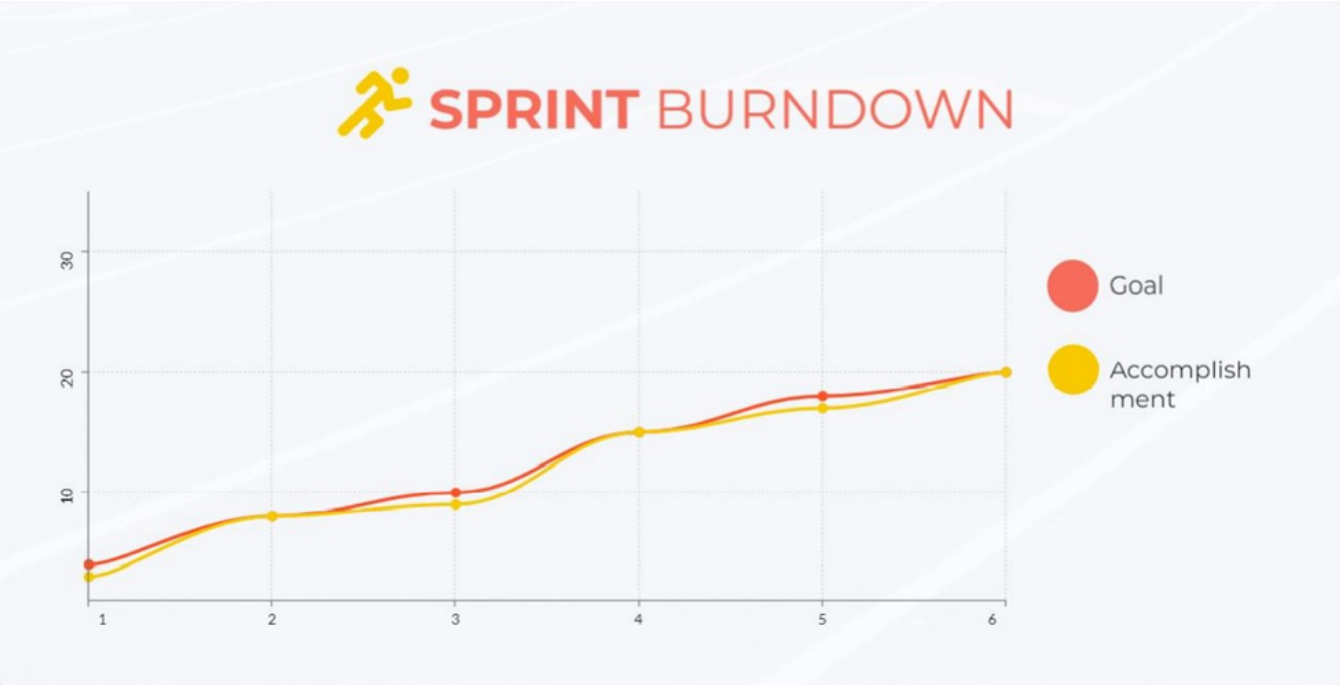
Velocity:

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)

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

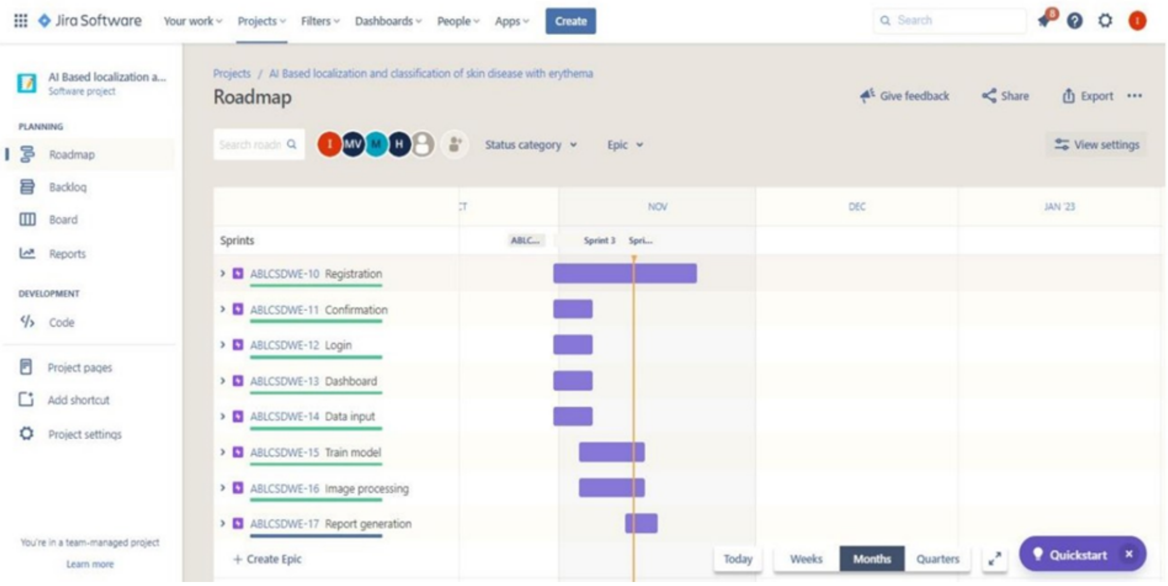
Burndown Chart:

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:



Projects / AI Based localization and classification of skin disease with erythema

Roadmap

ABLCSDWE-10 Registration

ABLCSDWE-11 Confirmation

ABLCSDWE-12 Login

ABLCSDWE-13 Dashboard

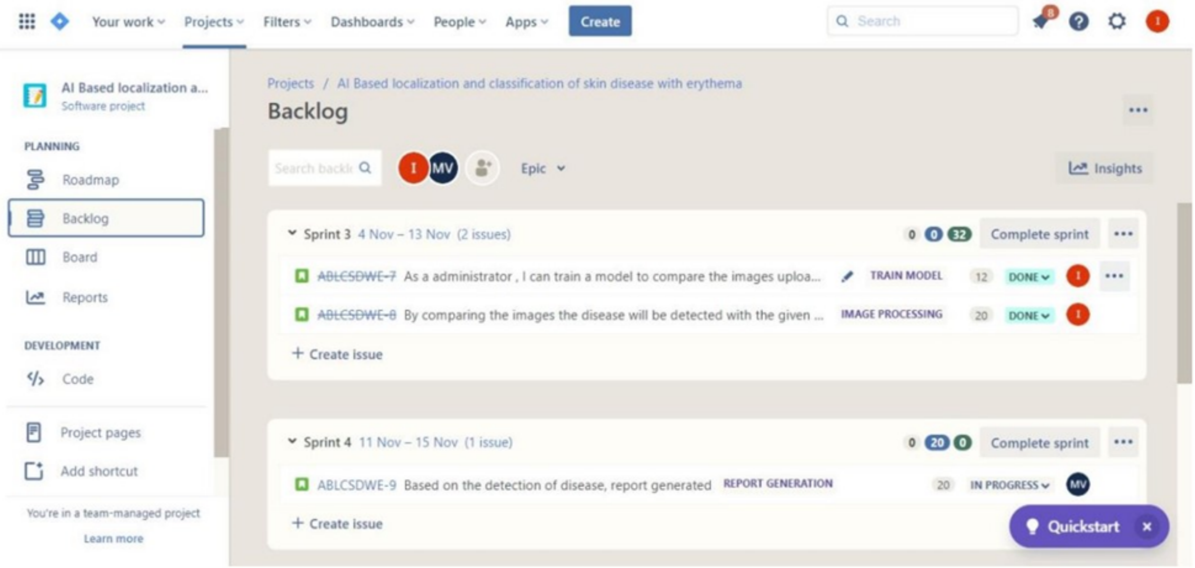
ABLCSDWE-14 Data input

ABLCSDWE-15 Train model

ABLCSDWE-16 Image processing

ABLCSDWE-17 Report generation

Backlog:



Projects / AI Based localization and classification of skin disease with erythema

Backlog

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

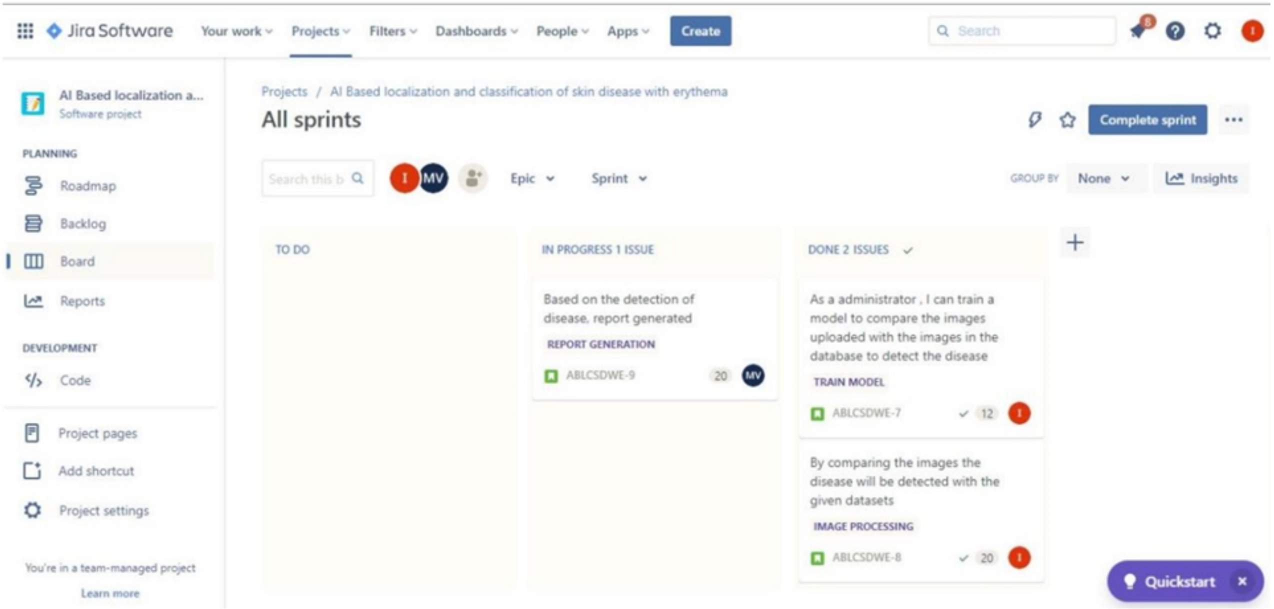
~~ABLCSDWE-7~~ As a administrator , I can train a model to compare the images uploa... TRAIN MODEL 12 DONE

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

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

Board:



Based on the detection of disease, report generated

REPORT GENERATION

ABLCSDWE-9 20 MV

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 I

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

IMAGE PROCESSING

ABLCSDWE-8 ✓ 20 I

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 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}
```

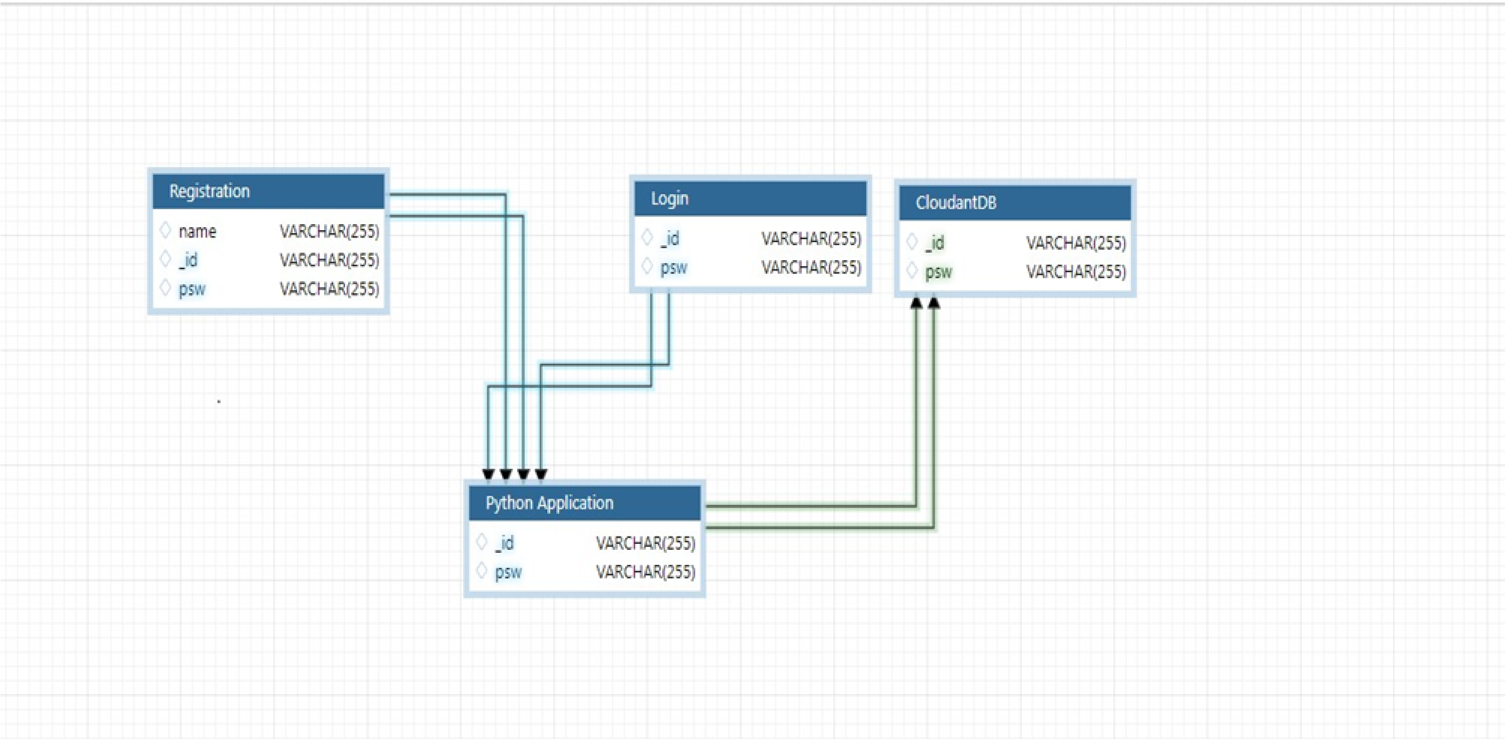
```
        )  # recompile to apply the change
      print("Unfreeze all layers.")

         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:


Registration
name VARCHAR(255)
_id VARCHAR(255)
psw VARCHAR(255)
Login
_id VARCHAR(255)
psw VARCHAR(255)
CloudantDB
_id VARCHAR(255)
psw VARCHAR(255)
Python Application
_id VARCHAR(255)
psw VARCHAR(255)

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

User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

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.

Performance metrics are a part of every machine learning pipeline. They tell you if you're making progress, and put a number on it. All machine learning models, whether it's linear regression, or a SOTA technique like BERT, need a metric to judge performance.

- ✔ R-CNN and YOLO is used for model creation. The mean average precision compares the detected with new box and finally returns a returns a score.

- ✔ Accuracy which we obtained was Training Accuracy – 86% and Validation Accuracy – 94%
- ✔ The confidence score which we got are Class Detected – 93% and Confidence Score – 90%

ADVANTAGES & DISADVANTAGES

Advantages:

- ✔ Artificial Intelligence helps to solve complex problems that require difficult calculations and can be done without any error.
- ✔ Perform Repetitive Jobs and faster decision taking.
- ✔ AI can streamline workflows.
- ✔ AI systems eliminate the risk of human error, producing a more accurate result.

Disadvantages:

- ✔ High production cost
- ✔ Lacking Out of Box Thinking and unemployment.
- ✔ AI is making humans lazy with its applications automating the majority of the work
- ✔ AI cannot be accessed and utilized akin to human intelligence, but it can store infinite data.

CONCLUSION

AI model helps one to model images with sufficient accuracy .
Yolo model helped us to mark down the areas where the disease is located with the help of segmentation. The AI-based methods reduces manual stress and tension by increasing the speed of diagnosis with minimal error rate.These models are easy and flexible to use which helps one to detect disease by one self.

FUTURE SCOPE

Research involving AI is making encouraging progress in the diagnosis of skin diseases. Despite the various claims of deep learning algorithms surpassing clinicians' performance in the diagnosis of skin disease, there are far more challenges faced by these algorithms to become a complete diagnostic system. Because such experiments are performed in controlled settings, algorithms are never tested in the real-life diagnosis of patients. The real-world diagnosis process requires taking into account a patient's ethnicity, skin, hair and eye color, occupation, illness, medicines, existing sun damage, the number of nevi, and lifestyle habits (such as sun exposure, smoking, and alcohol intake), clinical history, the respond to previous treatments, and other information from the patient's medical records.

However, current deep learning models predominantly rely on only patients' imaging data. Moreover, such systems often risk a misdiagnosis whenever they are applied to skin lesions or conditions that are not present in the training dataset.

Computer vision and dermatologist societies need to work together to improve current AI solutions and enhance the diagnostic accuracy of methods used for the diagnosis of skin diseases. AI has the potential to deliver a paradigm shift in the diagnosis of skin diseases, and thus a cost-effective, remotely accessible, and accurate healthcare solution.

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):
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 = C:\Users\amurali\OneDrive - Informatica\Desktop\yolo_structure\2_Training\src’
print(src_path)
utils_path = r' C:\Users\amurali\OneDrive - Informatica\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
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
# 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)
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)
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)
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),
}
)
out_df = pd.DataFrame(
columns=[ "image", "image_path", "xmin",
"ymin",
"xmax",
"ymax",
"label", "confidence", "x_size",
"y_size",
]
)
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 = ""
 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)
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
)
)
return render_template('prediction.html')
app.run(debug=True)
```

GitHub

Github:

https://github.com/IBM-EPBL/IBM-Project-21954-1

659799030