PROJECT REPORT

AI-BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMA

Team Leader	Jaisurya S
Team Members	Abishek V Bhuvaneshwari T Magesh A
Problem Statement	AI-Based Localization And Classification Of Skin Disease With Erythema

ABSTRACT

Skin is an extraordinary human structure. It frequently suffered from many known and unknown diseases. Therefore, the diagnosis of human skin diseases is the most uncertain and complicated branch of science. It has been observed that most of the cases remain unnoticed because of the lack of better medical infrastructure and facilities. This project is devoted to solving this challenge. Therefore, we are building an AI-based model that is used for the prevention and early detection of erythema. 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. Thus, to evaluate the performance of the proposed system several experiments are conducted on our dataset. This leads to detecting skin disease and providing the user with the disease name and treatment and a related prescription with high accuracy.

ACKNOWLEDGMENT

It gives us a great sense of pleasure to present the report of the Nalaiya Thiran Project undertaken during B. Tech. Fourth Year. This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received.

Our heartiest thanks to Dr. (Prof). **SUMATHI G,** Professor, Department of IT for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.

We owe special debt of gratitude to all faculty members, Department of IT, for their constant support and guidance throughout the course of our work. Their sincerity, thoroughness and perseverance have been a constant source of inspiration for us. They have showered us with all their extensively experienced ideas and insightful comments at virtually all stages of the project & have also taught us about the latest industry-oriented technologies.

We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.

Jaisurya S

Abishek V

Bhuvaneshwari T

Magesh A

LIST OF TABLES

Table 1- Literature survey…….

Table 2- Functional Requirements……

Table 3- Non-Functional Requirements……

Table 4- User Stories……

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1	**INTRODUCTION** 1.1 INTRODUCTION 1.2 MOTIVATION 1.3 IDEATION AND PROPOSED SOLUTION	
2	**LITERATURE SURVEY**	
3	**SOLUTION ARCHITECTURE** **And** **TECHNOLOGY ARCHITECTURE**	
4	**SYSTEM DESIGN** 4.1 REQUIREMENT ANALYSIS 4.2 DATA FLOW 4.3 USER STORIES	
5	**PROJECT PLANNING AND SCHEDULING** 5.1 SPRINT PLANNING AND ESTIMATION 5.2 SPRINT DELIVERY SCHEDULE	
6	**CODING AND SOLUTIONS**	
7	**RESULT**	
8	**CONCLUSION**	
	APPENDIX	

1. INTRODUCTION

1.1 INTRODUCTION

In dermatology, although skin disease is a common disease, one in which early detection and classification are crucial for the successful treatment and recovery of patients, dermatologists perform most noninvasive screening tests only with the naked eye. This may result in avoidable diagnostic inaccuracies as a result of human error, as the detection of the disease can be easily overlooked. Furthermore, the classification of 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. This paper shows that CAD may be a viable option in the field of dermatology using state-of-the-art deep learning models.

1.2 MOTIVATION

The diseases are not considered skin diseases, and skin tone is majorly suffered from the ultraviolet rays from the sun. However, dermatologists perform the majority of non-invasive screening tests simply with the naked eye, even though skin illness is a frequent disease for which early detection and classification are essential for patient success and recovery. The characteristic of the skin images is diversified so it is a challenging job to devise an efficient and robust algorithm for the automatic detection of skin disease and its severity.

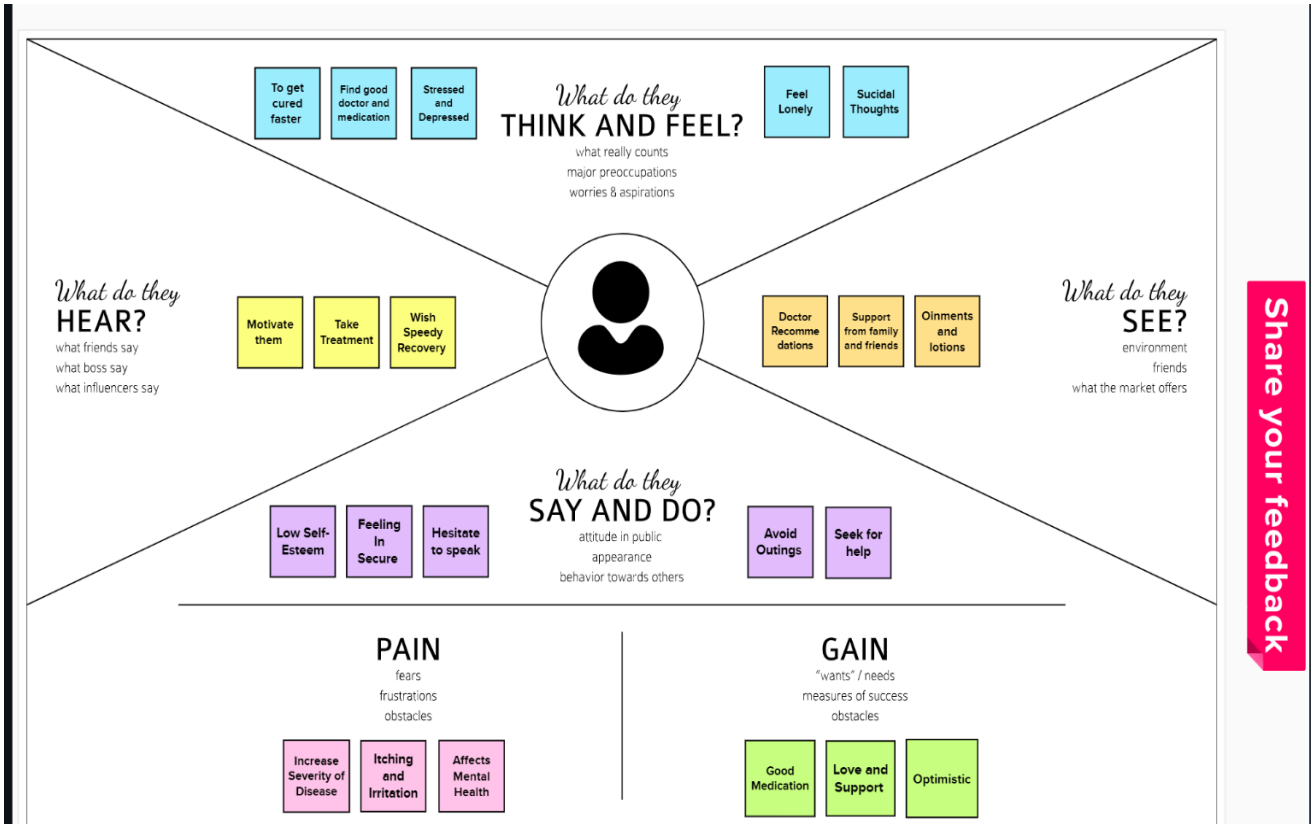
1.3 PROPOSED SOLUTION

Two-phase analysis model. The original image primarily enters a pre-processing stage, where normalization and decomposition occur. Afterward, the first step is segmentation, where clusters of abnormal skin are segmented and cropped. The second step is classification, where each cluster is classified into its corresponding class. The developed Model is Still under training.

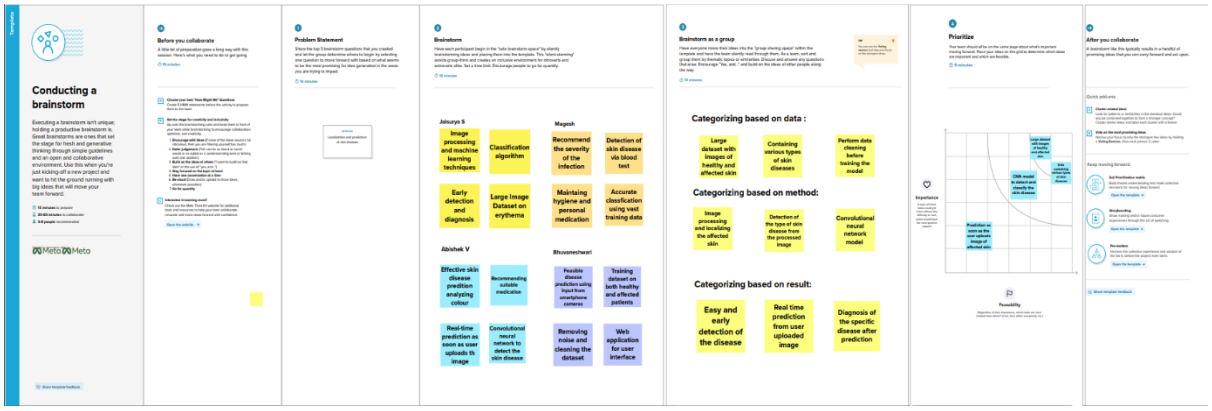
1.4 PROBLEM-SOLUTION FIT

Skin disease can appear in virtually any part of the body and there is a lack of data required to form an association between the probability of a skin disease based on the body part. A Solution model used for the prevention is early detection of skin cancer and psoriasis by image analyses to detect whether the person is having skin disease or not.

1.5 EMPATHY MAP



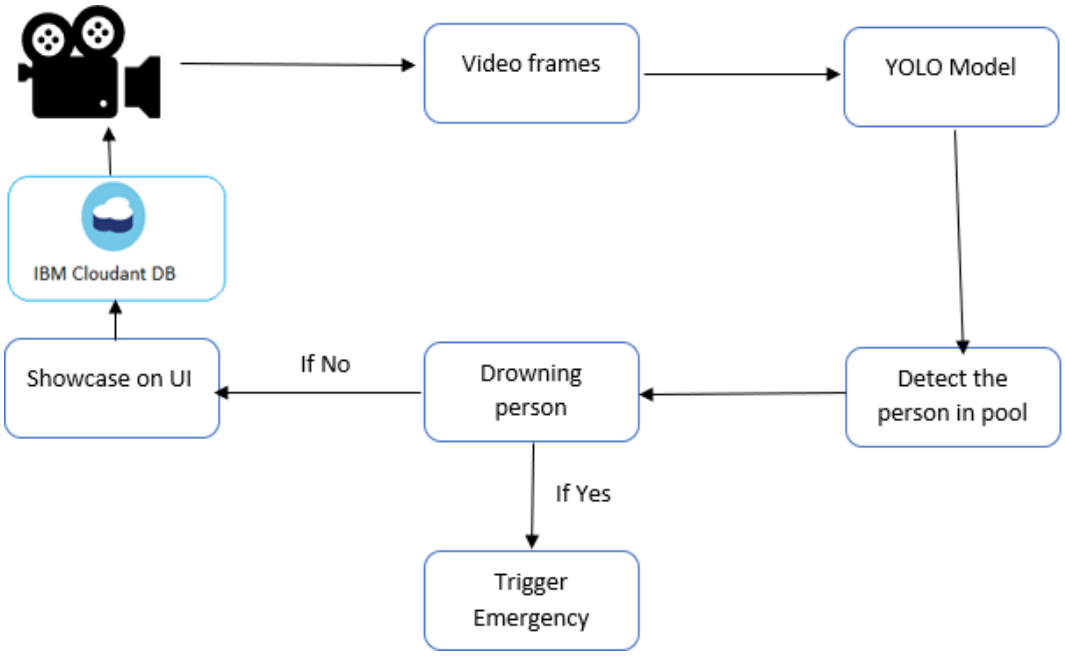
1.6 IDEATION AND BRAINSTORMING



2. LITERATURE SURVEY

S.NO	TITLE	AUTHOR	YEAR	FINDINGS
1	Intelligent System for Skin Disease Prediction using AI	Ahmed A. Elngar[1], Rishabh Kumar[2], Amber Hayat[2] and Prathamesh Churi[3]	2021	Dermatologist Disease Diagnosis using color-skin images has proposed a two-stage method to detect the disease based on color texture-based identification and by using classification to identify the name of the disease. The first stage has an accuracy of 95.99% and the second stage has a 94.016% accuracy.
2	A method of skin disease detection using Image Processing and machine learning	Nawal Soliman, & ALKolifi AlEnezi at el.	2015	Proposed an early detection method for image processing based on a Convolutional neural network to feature extraction and then using color to identify the features.
3	An image analysis System to detect skin diseases	Pravin S. Ambad & A. S. Shirsat et al.	2018	Proposed a system for the early identification of skin problems using statistical analysis and an ad boost classifier. Their research mainly focused on the early identification of skin cancer symptoms based on statistical analysis with correlation algorithms.
4	Skin disease recognition method based on image color and texture features	Li-sheng Wei, Quan Gan, and Tao ji at el.	2021	Proposed a model based on feature extraction of images using color texture and using segmentation and SVM on it to identify the disease.

3. SOLUTION AND TECHNOLOGY ARCHITECTURE



4. SYSTEM DESIGN

4.1 REQUIREMENT ANALYSIS

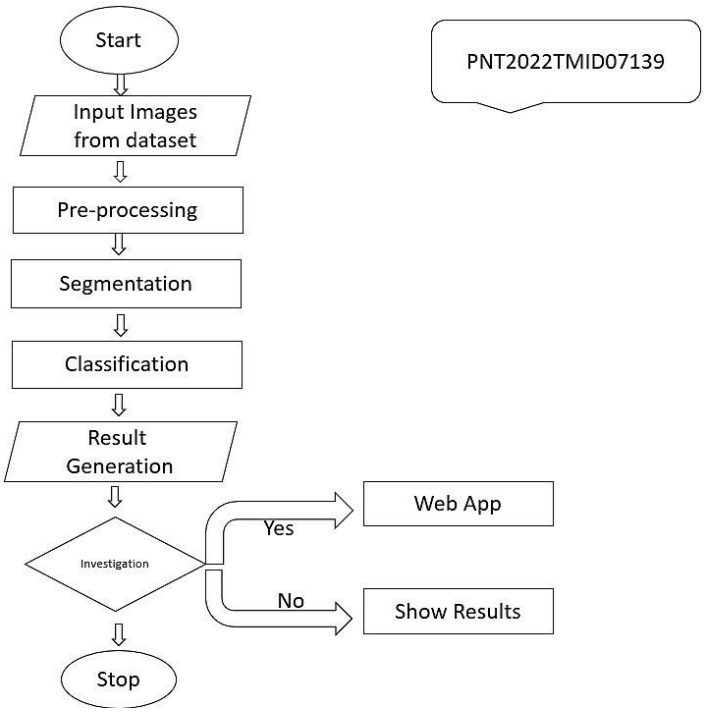
Functional requirements:

Image Acquisition, Pre-processing Steps such as Colour gradient generator on an image , Cropping and isolating region of interest and Thresholding and Clustering on image, Visual feature extraction, System Training YOLO Model for Skin disease classification with deep learning and CNN, Separate access of application for admin, Diagnosis of Skin disease and Data retrieval and Data manipulation.

Non-Functional requirements:

Software Quality Attributes, Prediction, Accuracy.

4.2 DATA FLOW DIAGRAM



4.3 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
Dat Collection	USN-2	The 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

5. PROJECT PLANNING AND SCHEDULING

5.1 SPRINT PLANNING AND 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	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A
Sprint-1	Data Collection	USN-2	Dataset should be collected from google or using a Chrome extension such as Fatkun Batch Downloader	3	High	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A
Sprint-1	Annotate Images	USN-3	Create A Project in VOTT (Microsoft's Visual Object Tagging Tool)	2	Medium	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A
Sprint-2	Training YOLO	USN-4	train our model using YOLO weights	2	Medium	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A

Sprint-2		USN-5	To Download and Convert PreTrained Weights	3	High	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A
Sprint-2		USN-6	To Train YOLOv3 Detector	3	High	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A
Sprint-3	Cloudant DB	USN-7	Register & Login to IBM Cloud	3	High	Jaisuriya S Abishek V
Sprint-3		USN-8	Create Service Instant and Credentials	2	Medium	Bhuvaneshwari T Magesh A
Sprint-3		USN-9	Launch DB and Create database	3	High	Jaisuriya S Bhuvaneshwari T
Sprint-3	Development Phase	USN-10	To build a web application	3	High	Abishek V Magesh A
Sprint-3		USN-11	Building HTML pages with python code	2	Medium	Jaisuriya S Magesh A
Sprint-3		USN-12	To run the application	3	High	Abishek V Bhuvaneshwari T
Sprint-4	Testing Phase	USN-13	As a user login to dashboard	2	Medium	Jaisuriya S Abishek V
Sprint-4		USN-14	As a user import the images with skin diseases to the software application	2	Medium	Jaisuriya S Bhuvaneshwari T

Sprint-4		USN-15	YOLO processes the image and give the necessary details	3	High	Jaisuriya S Abishek V Bhuvaneshwari T Magesh A

5.2 SPRINT DELIVERY SCHEDULE

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

6. CODING AND SOLUTIONS

6.1 Image Annotation:

The images in our training folder are manually labeled using Microsoft's Visual Object Tagging Tool (VoTT) . At least 100 images should be annotated for each category to get respectable results.

Code:

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


def get_parent_dir(n=1):
""" returns the n-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


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()
```

6.2 Training Yolo

```
"""
Retrain the YOLO model for your own dataset.
"""

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 yolo3.model import preprocess_true_boxes, yolo_body, tiny_yolo_body, yolo_loss
from yolo3.utils import get_random_data
```

```
def _main():
annotation_path = "data_train.txt"
log_dir = "logs/003/"
classes_path = "data_classes.txt"
# anchors_path = 'model_data/yolo-tiny_anchors.txt'
anchors_path = "model_data/yolo_anchors.txt"
class_names = get_classes(classes_path)
num_classes = len(class_names)
anchors = get_anchors(anchors_path)

input_shape = (416, 416)  # multiple of 32, hw
epoch1, epoch2 = 40, 40

is_tiny_version = len(anchors) == 6  # default setting
if is_tiny_version:
model = create_tiny_model(
input_shape,
anchors,
num_classes,
freeze_body=2,
weights_path="model_data/yolo-tiny.h5",
)
else:
model = create_model(
input_shape,
anchors,
num_classes,
freeze_body=2,
```

```
weights_path="model_data/yolo.h5",
)  # make sure you know what you freeze

logging = TensorBoard(log_dir=log_dir)
# checkpoint = ModelCheckpoint(log_dir + 'ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.h5',
#     monitor='val_loss', save_weights_only=True, save_best_only=True, period=3)
checkpoint = ModelCheckpoint(
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 = 0.1
with open(annotation_path) as f:
lines = f.readlines()
np.random.seed(10101)
np.random.shuffle(lines)
np.random.seed(None)
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 not bad 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
)
)
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(log_dir + "trained_weights_stage_1.h5")

# Unfreeze and continue training, to fine-tune.
```

```
# Train longer if the result is not good.
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 of the layers.")

batch_size = (
16  # 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
)
)
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(log_dir + "trained_weights_final.h5")

# Further training if needed.


def get_classes(classes_path):
"""loads the classes"""
with open(classes_path) as f:
class_names = f.readlines()
class_names = [c.strip() for c in class_names]
return class_names


def get_anchors(anchors_path):
"""loads the anchors from a file"""
with open(anchors_path) as f:
anchors = f.readline()
anchors = [float(x) for x in anchors.split(",")]
return np.array(anchors).reshape(-1, 2)


def create_model(
input_shape,
anchors,
num_classes,
load_pretrained=True,
freeze_body=2,
weights_path="model_data/yolo_weights.h5",
):
"""create the training model"""
```

```
K.clear_session()  # get a new session
image_input = Input(shape=(None, None, 3))
h, w = input_shape
num_anchors = len(anchors)

y_true = [
Input(
shape=(
h // {0: 32, 1: 16, 2: 8}[l],
w // {0: 32, 1: 16, 2: 8}[l],
num_anchors // 3,
num_classes + 5,
)
)
for l in range(3)
]

model_body = yolo_body(image_input, num_anchors // 3, num_classes)
print(
"Create YOLOv3 model with {} anchors and {} classes.".format(
num_anchors, num_classes
)
)

if load_pretrained:
model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
print("Load weights {}.".format(weights_path))
if freeze_body in [1, 2]:
# Freeze darknet53 body or freeze all but 3 output layers.
num = (185, len(model_body.layers) - 3)[freeze_body - 1]
```

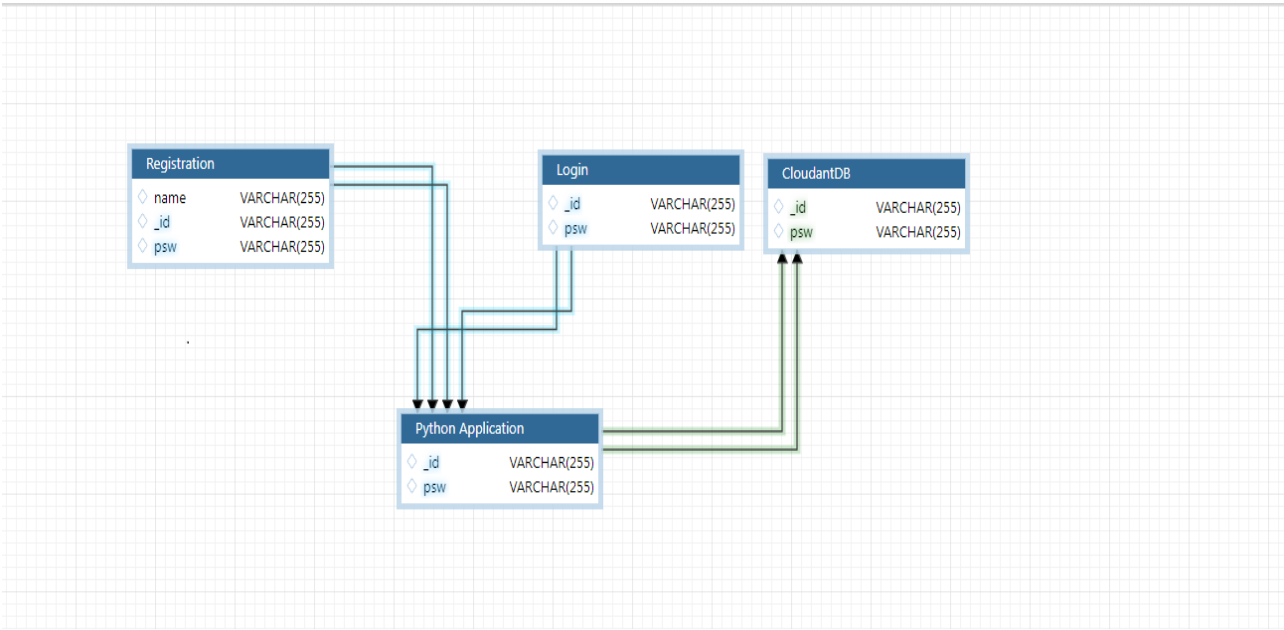
```
for i in range(num):
model_body.layers[i].trainable = False
print(
"Freeze the first {} layers of total {} layers.".format(
num, len(model_body.layers)
)
)

model_loss = Lambda(
yolo_loss,
output_shape=(1,),
name="yolo_loss",
arguments={
"anchors": anchors,
"num_classes": num_classes,
"ignore_thresh": 0.5,
},
)([*model_body.output, *y_true])
model = Model([model_body.input, *y_true], model_loss)

return model


def create_tiny_model(
input_shape,
anchors,
num_classes,
load_pretrained=True,
freeze_body=2,
weights_path="model_data/tiny_yolo_weights.h5",
```

```
):
"""create the training model, for Tiny YOLOv3"""
K.clear_session()  # get a new session
image_input = Input(shape=(None, None, 3))
h, w = input_shape
num_anchors = len(anchors)

y_true = [
Input(
shape=(
h // {0: 32, 1: 16}[l],
w // {0: 32, 1: 16}[l],
num_anchors // 2,
num_classes + 5,
)
)
for l in range(2)
]

model_body = tiny_yolo_body(image_input, num_anchors // 2, num_classes)
print(
"Create Tiny YOLOv3 model with {} anchors and {} classes.".format(
num_anchors, num_classes
)
)

if load_pretrained:
model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
print("Load weights {}.".format(weights_path))
if freeze_body in [1, 2]:
```

```
# Freeze the darknet body or freeze all but 2 output layers.
num = (20, len(model_body.layers) - 2)[freeze_body - 1]
for i in range(num):
model_body.layers[i].trainable = False
print(
"Freeze the first {} layers of total {} layers.".format(
num, len(model_body.layers)
)
)

model_loss = Lambda(
yolo_loss,
output_shape=(1,),
name="yolo_loss",
arguments={
"anchors": anchors,
"num_classes": num_classes,
"ignore_thresh": 0.7,
},
)([*model_body.output, *y_true])
model = Model([model_body.input, *y_true], model_loss)

return model


def data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes):
"""data generator for fit_generator"""
n = len(annotation_lines)
i = 0
while True:
```

```
image_data = []
box_data = []
for b in range(batch_size):
if i == 0:
np.random.shuffle(annotation_lines)
image, box = get_random_data(annotation_lines[i], input_shape, random=True)
image_data.append(image)
box_data.append(box)
i = (i + 1) % n
image_data = np.array(image_data)
box_data = np.array(box_data)
y_true = preprocess_true_boxes(box_data, input_shape, anchors, num_classes)
yield [image_data, *y_true], np.zeros(batch_size)


def data_generator_wrapper(
annotation_lines, batch_size, input_shape, anchors, num_classes
):
n = len(annotation_lines)
if n == 0 or batch_size <= 0:
return None
return data_generator(
annotation_lines, batch_size, input_shape, anchors, num_classes
)


if __name__ == "__main__":
_main()
```

6.3 Database Schema:



7. TESTING & RESULT

7.1 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%

7.2 APPLICATION TEST CASES

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. 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.

9. FUTURE SCOPE

This implementation of the Structural Co-Occurrence matrices for feature extraction in the skin diseases classification and the pre-processing techniques are handled by using the Median filter, this filter helps to remove the salt and pepper noise in the image processing; thus, it enhances the quality of the images,

and normally, the skin diseases are considered as the risk factor in all over the world. Our proposed approach provides 97% of the classification of the accuracy results while another existing model such as FFT + SCM gives 80%, SVM + SCM gives 83%, KNN + SCM gives 85%, and SCM + CNN gives 82%. Future work is dependent on the increased support vector machine's accuracy in classifying skin illnesses, and SCM is used to manage the feature extraction technique.

10. ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

Instant Response, improves prediction of Skin Disease, no referral needed, Saves Money and Time, and is Confidential Advice.

10.2 DISADVANTAGE

Network Connectivity and Accuracy.