

PROJECT REPORT

AI BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMA

Submitted by,

PNT2022TMID29594

HARIPRAKASH S - 513119104008

HARI PRASAD V - 513119104009

SATHISH KUMAR E - 513119104032

SIVA KA - 513119104035

Table of Content

Chapter No.	Title	Page No.
1	INTRODUCTION	
1.1	Project OverView	5
1.2	Purpose	5
2	LITERATURE SURVEY	
2.1	Existing Problem	6
2.2	References	6
2.3	Problem Statement Definition	6
3	IDEATION & PROPOSED SOLUTION	
3.1	Empathy Map Canvas	7
3.2	Ideation & BrainStorming	7
3.3	Proposed Solution	8
3.4	Problem Solution Fit	10
4	REQUIREMENT ANALYSIS	
4.1	Functional Requirement	11
4.2	Non-Function Ruquirement	11
5	PROJECT DESIGN	
5.1	Data Flow Diagrams	12
5.2	Solution & Technical Architecture	13
5.3	User Stories	14

6	PROJECT PLANNING & SCHEDULING	
6.1	Spring Planning & Estimation	15
6.2	Sprint Delicery Schedule	16
6.3	Reports From JIRA	18
7	CODING & SOLUTIONING	
7.1	Feature 1	20
7.2	Feature 2	25
7.3	Database Schema	36
8	TESTING	
8.1	Test Cases	38
8.2	User Acceptence Testing	38
9	RESULTS	
9.1	Performance Metrics	40
10	ADVANTAGES & DISADVANTAGES	41
11	CONCLUSION	42
12	FUTURE SCOPE	42
13	APPENDIX	
13.1	Source Code	43
13.2	GitHub & Project Demo Link	59

List of Figures

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

CHAPTER 1

INTRODUCTION

1.1 Project Overview

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

1.2 Purpose

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

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Problem:

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

2.2 Reference

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

https://link.springer.com/chapter/10.1007/978-981-19-0863-7_10

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

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

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

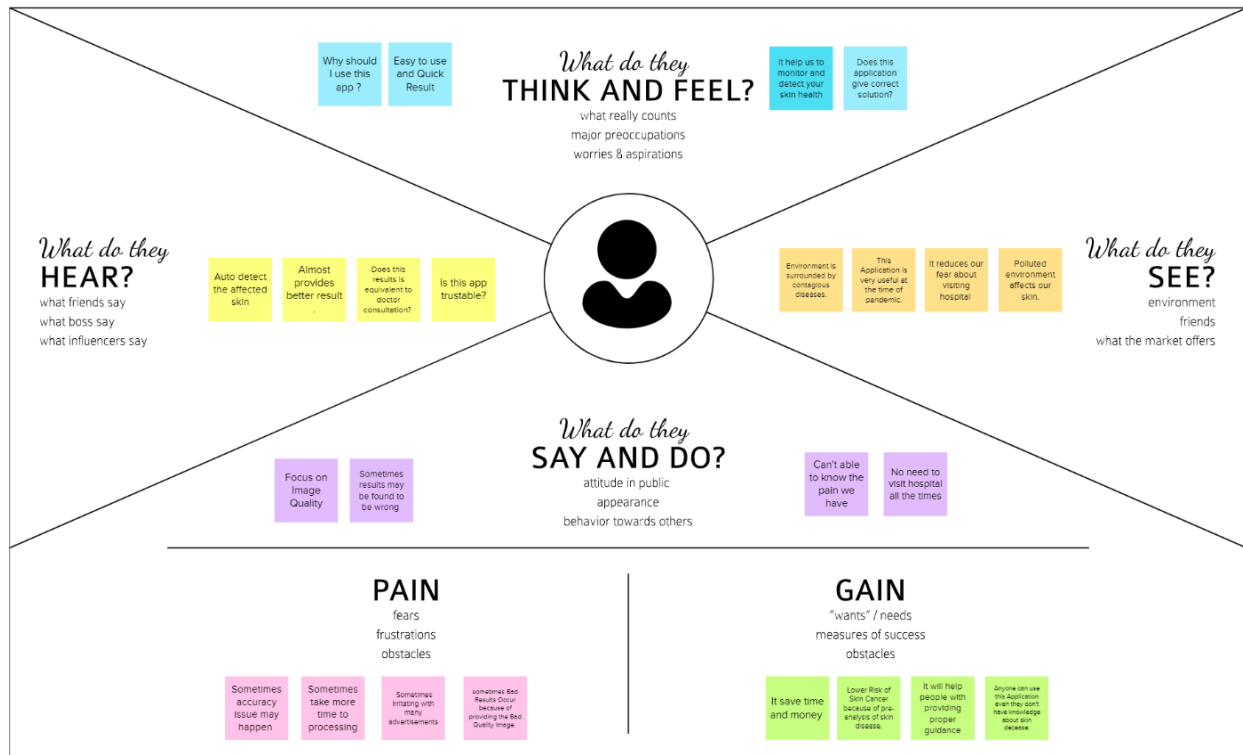
2.3 Problem Statement Definition

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

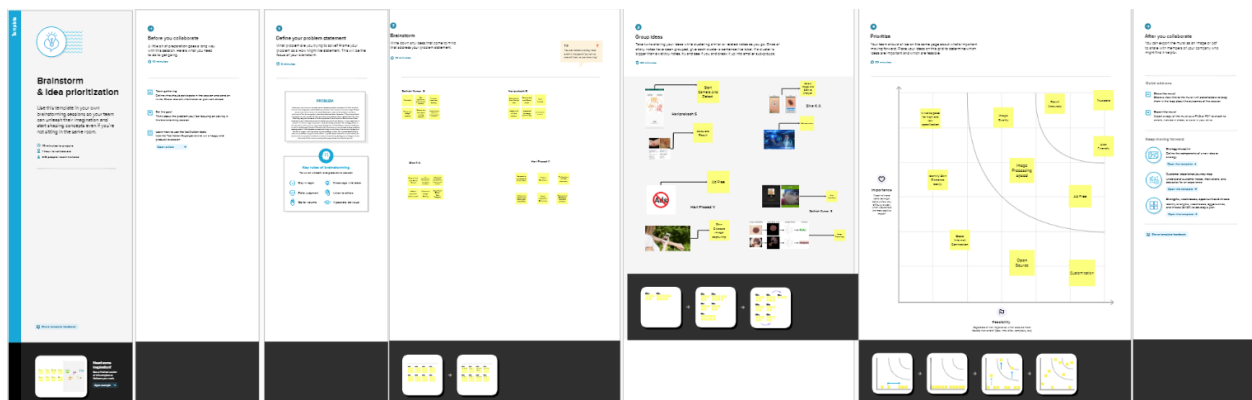
CHAPTER 3

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)	<ul style="list-style-type: none">• AI-Based Localization and Classification of Skin Disease with Erythema.• 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	<ul style="list-style-type: none">• To overcome the above problem, 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 colour image. Then resize the image to extract features using pre-trained convolutional neural network. After that it classifies feature using Multiclass SVM. Finally, the results are shown to the user, including the type of disease, spread, and severity.
3.	Novelty / Uniqueness	<ul style="list-style-type: none">• 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 and treatment suggestions.• It has high level of accuracy
4.	Social Impact / Customer Satisfaction	<ul style="list-style-type: none">• It is easy to access as everyone is using mobile phones now-a-days.• It is a time and money-saving process.

		<ul style="list-style-type: none"> It is highly secured and the data is not delivered to third persons.
5.	Business Model (Revenue Model)	<ul style="list-style-type: none"> Based on the report, a dermatologist is assigned who will cater specific needs and build a unique regime. It recommends skin care products which will be delivered to doorsteps
6.	Scalability of the Solution	<ul style="list-style-type: none"> 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. 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:

Define CS, fit into CC	<div>1. CUSTOMER SEGMENT(S)<div>CS</div></div> <div>Patients are customers here</div>	<div>6. CUSTOMER CONSTRAINTS<div>CC</div></div> <div><div><div>Need of Experts consultation</div><div>Budget Problem</div></div></div>	<div>5. AVAILABLE SOLUTIONS<div>AS</div></div> <div>After being affected, a patient visits a doctor for a consultation. It requires more travel time and has a budget constraint. It is not suitable for immediate consultation, but it includes an expert prescription that ensures the correct solution and care.</div>	Explore AS, differentiate
	<div>2. JOBS-TO-BE-DONE / PROBLEMS<div>J&P</div></div> <div>When afflicted with a disease, patients should take picture of the affected area and upload it.</div>	<div>9. PROBLEM ROOT CAUSE<div>RC</div></div> <div>Common causes of skin diseases include: Bacteria trapped in your pores or hair follicles. Conditions that affect your thyroid, kidneys or immune system. Contact with environmental triggers, such as allergens or another person's skin.</div>	<div>7. BEHAVIOUR<div>BE</div></div> <div>The problem of skin disease is directly connected to the patient. When he or she feels irritation or redness in skin, they can address the issues</div>	
Focus on J&P, tap into BE, understand RC	<div>3. TRIGGERS<div>TR</div></div> <div>Patients can know about this from their neighbors, colleagues, and friends.</div>	<div>10. YOUR SOLUTION<div>SL</div></div> <div>To overcome the 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 color, shape, texture etc. Here the person can capture the images of skin and then the image will be sent to the trained model. The model analyses the image and detect whether the person is having skin disease or not</div>	<div>8. CHANNELS of BEHAVIOUR<div>CH</div></div> <div>8.1 ONLINE Users upload pictures of the affected area.</div> <div>8.2 OFFLINE After the classification of disease, they can consult a doctor and cure the disease.</div>	Focus on J&P, tap into BE, understand RC
	<div>4. EMOTIONS: BEFORE / AFTER<div>EM</div></div> <div>Before: The patient may experience discomfort, Insecurity and mental fear after being infected. After: They can know about the affected disease.</div>			
Identify strong TR & EM				

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Functional requirements:

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

4.2 Non-Functional requirements:

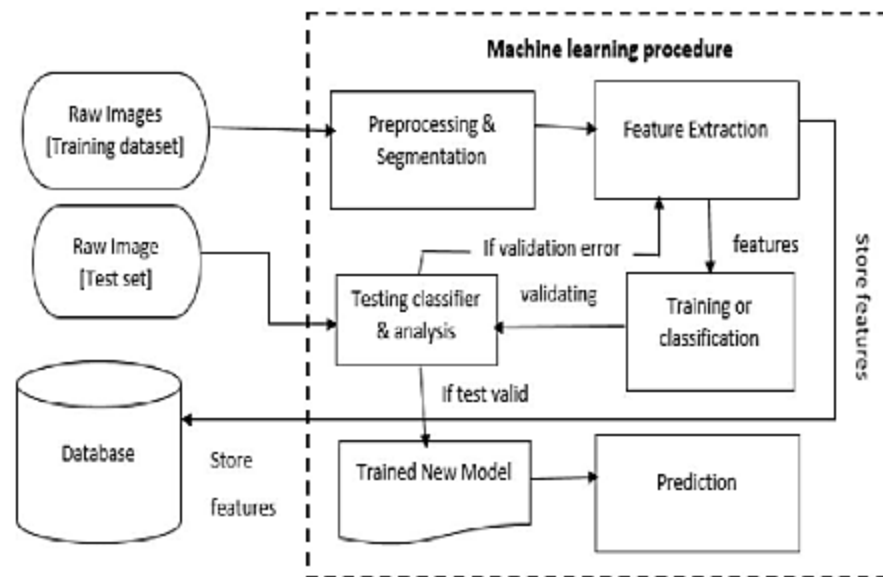
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Used to classify skin disease with erythema
NFR-2	Security	It offers greater security and prevents unauthorized individuals from accessing user's data.
NFR-3	Reliability	Even with more users, there will be a good Performance without failure.
NFR-4	Performance	With greater accuracy, the performance is high.
NFR-5	Availability	With a good system, all authorized users can access it.
NFR-6	Scalability	Performance will be good even with the higher user traffic.

CHAPTER 5

PROJECT DESIGN

5.1 Data Flow Diagram:

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

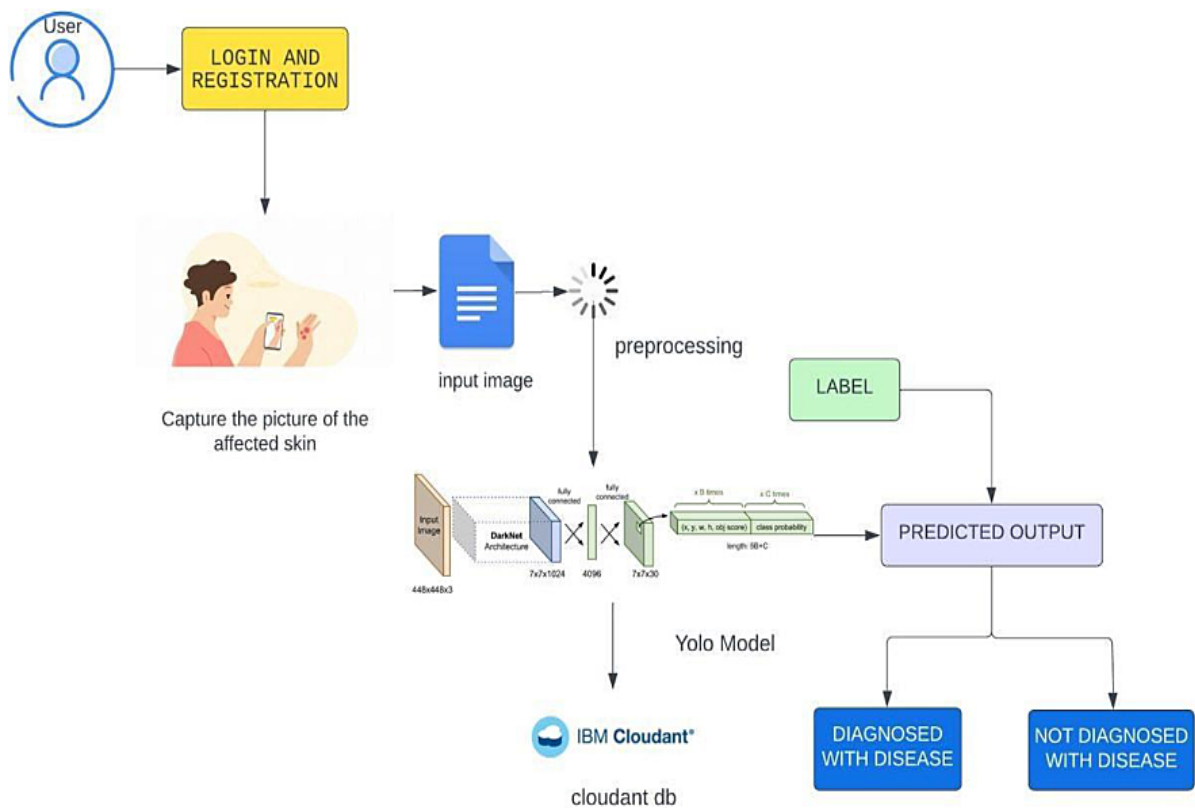


5.2 Solution & Technical Architecture:

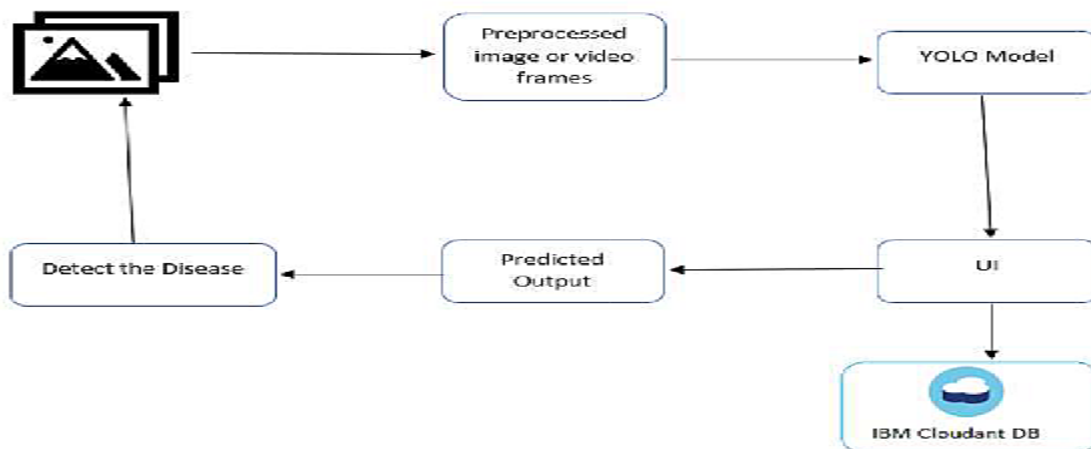
Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Solution Architecture Diagram:



Technical Architecture:



5.3 User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
	Confirmation	USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
	Login	USN-3	As a user, I can login for the application through Gmail	I can access my account / dashboard	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-5	As a user, I can see the my profile, medical history, upload image , getting report services provided by the application	I can get into one of the services and use it	Medium	Sprint-2
	Data input	USN-6	As a user, I can upload the images of the affected skin area	I can submit it to the application	High	Sprint-2
Administrator	Train model	USN-7	As a administrator , I can train a model to compare the images uploaded with the images in the database to detect the disease	I can test the model whether it meets the criteria	High	Sprint-3
Trained model	Image processing	USN-8	By comparing the images the disease will be detected with the given datasets	All the necessary operation performed and information extracted	High	Sprint-3
	Report generation	USN-9	Based on the detection of disease, report generated	The results will be shown on the screen to the patients	High	Sprint-4

CHAPTER 6

PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-1	Confirmation	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-1	Login	USN-3	As a user, I can login for the application through Gmail	2	Medium	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	2	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-2	Dashboard	USN-5	As a user, I can see the my profile, medical history, upload image , getting report services provided by the application	1	Medium	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A

Sprint-2	Data input	USN-6	As a user, I can upload the images of the affected skin area	1	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-3	Train model	USN-7	As a administrator , I can train a model to compare the images uploaded with the images in the database to detect the disease	2	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-3	Image processing	USN-8	By comparing the images the disease will be detected with the given datasets	2	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A
Sprint-4	Report generation	USN-9	Based on the detection of disease, report generated	2	High	Hari Prasad V Hari Prakash S Sathishkumar E Siva K A

6.2 Sprint Delivery Schedule:

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	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	14 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

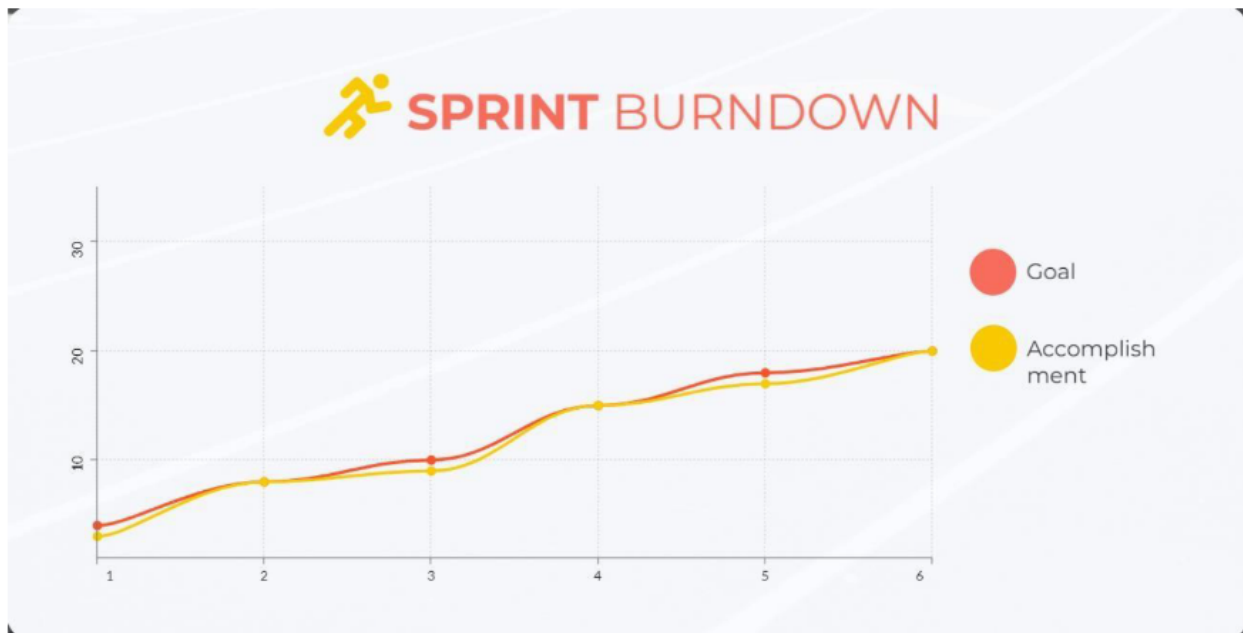
Velocity:

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

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

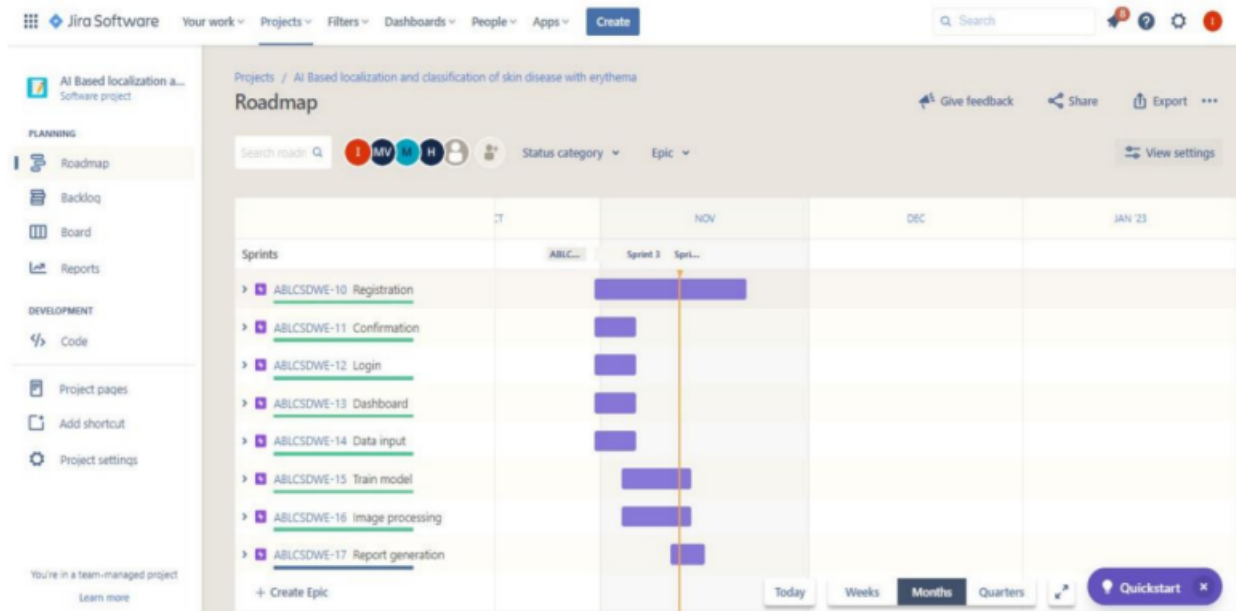
Burndown Chart:

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

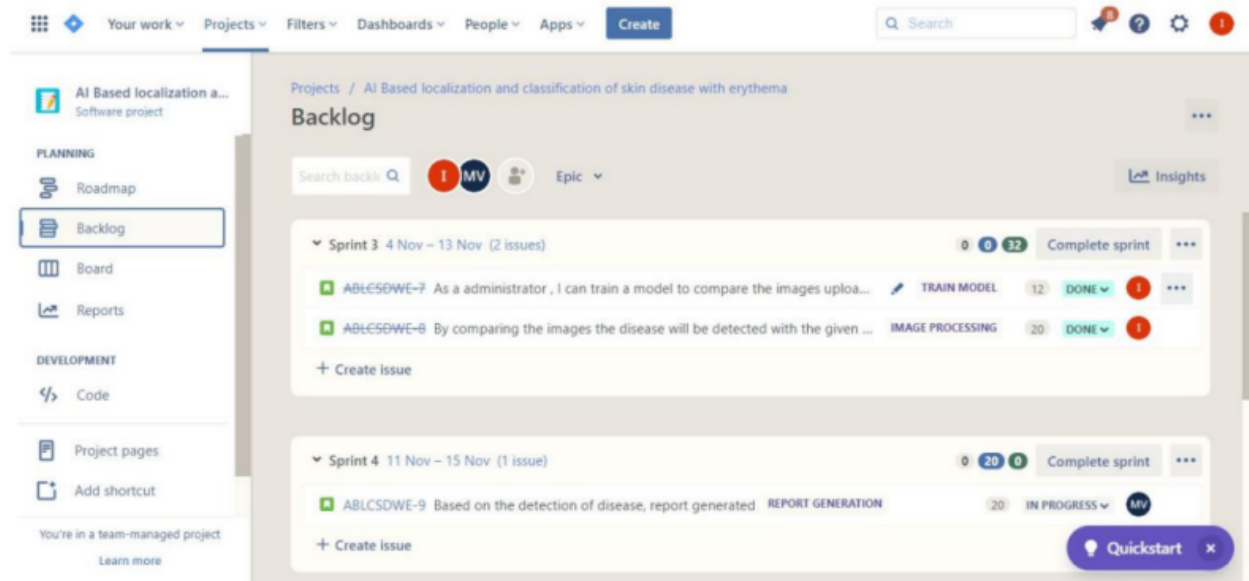


6.4 Reports from JIRA:

Road Map:



Backlog:



Board:

The screenshot shows the Jira Software interface for a project named "AI Based localization and classification of skin disease with erythema". The main view is the "All sprints" board, which is organized into three columns: "TO DO", "IN PROGRESS 1 ISSUE", and "DONE 2 ISSUES".

TO DO: This column is currently empty.

IN PROGRESS 1 ISSUE: This column contains one issue titled "Based on the detection of disease, report generated" with the sub-header "REPORT GENERATION". The issue is assigned to "ABLCSDWE-9" and has a time estimate of 20 minutes. It is marked with a "MV" (Move) icon.

DONE 2 ISSUES: This column contains two issues:

- TRAIN MODEL:** Issue "ABLCSDWE-7" with a time estimate of 12 minutes. It is marked with a checkmark and a red circle with an exclamation mark.
- IMAGE PROCESSING:** Issue "ABLCSDWE-8" with a time estimate of 20 minutes. It is marked with a checkmark and a red circle with an exclamation mark.

The interface includes a sidebar on the left with navigation options: "Roadmap", "Backlog", "Board" (selected), "Reports", "Code", "Project pages", "Add shortcut", and "Project settings". The top navigation bar shows "Your work", "Projects", "Filters", "Dashboards", "People", "Apps", and a "Create" button. A search bar and notification icons are also present in the top right.

CHAPTER 7

CODING & SOLUTIONING

7.1 Feature 1

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

Code:

```
import os
import uuid
import flask
from urllib import *
from PIL import Image
from numpy import number
from keras.models import load_model
from flask import Flask , render_template , request , send_file
from keras.preprocessing.image import load_img , img_to_array
from keras.applications.mobilenet import preprocess_input, decode_predictions
from keras.models import model_from_json
from keras.models import load_model
from urllib.request import urlopen
import requests

app = Flask(__name__)
j_file = open(r'C:\Users\sivak\Documents\AISKin Project\Hari\New folder
(2)\skin-disease-classification-main\envs\skin-disease-classification-
main\model.json', 'r')
```

```

loaded_json_model = j_file.read()
j_file.close()
model = model_from_json(loaded_json_model)
model.load_weights('model.h5')

```

```

ALLOWED_EXT = set(['jpg' , 'jpeg' , 'png' , 'jif'])
def allowed_file(filename):
    return '.' in filename and \
        filename.rsplit('.', 1)[1] in ALLOWED_EXT

```

```

classes = [
    'Erythema Multiforme',
    'Erythema Nodosum',
    'Benign Keratosis',
    'Erythema Psoriasis',
    'Melanoma',
    'Erythema Marginatum',
    'Erythema Toxicum'

```

```

]

```

```

def predict(filename , model):
    img = load_img(filename , target_size = (224, 224))
    img = img_to_array(img)
    img = img.reshape(1,224,224,3)

    img = img.astype('float32')
    img = img/255.0
    result = model.predict(img)

```

```

dict_result = {}
for i in range(7):
    dict_result[result[0][i]] = classes[i]

res = result[0]
res.sort()
res = res[::-1]
prob = res[:3]

prob_result = []
class_result = []
for i in range(3):
    prob_result.append((prob[i]*100).round(2))
    class_result.append(dict_result[prob[i]])

return class_result , prob_result

```

```

@app.route('/')
def home():
    return render_template("index.html")

```

```

@app.route('/about/')
def about():
    return render_template("about.html")

```

```

@app.route('/contact/')
def contact():
    return render_template("contact.html")

```

```
# more changes to be made in contact
```

```
# @app.route('/about1/')
```

```
# def about():
```

```
#     return "this is about page"
```

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

```
def login():
```

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

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

```
def success():
```

```
    error = "
```

```
    target_img = os.path.join(os.getcwd(),'static/images')
```

```
    if request.method == 'POST':
```

```
        if(request.form):
```

```
            link = request.form.get('link')
```

```
            try :
```

```
                resource =urlopen(link)
```

```
                unique_filename = str(uuid.uuid4())
```

```
                filename = unique_filename+".jpg"
```

```
                img_path = os.path.join(target_img , filename)
```

```
                output = open(img_path , "wb")
```

```
                output.write(resource.read())
```

```
                output.close()
```

```
                img = filename
```

```
                class_result , prob_result = predict(img_path , model)
```

```
                predictions = {
```

```
                    "class1":class_result[0],
```

```
                    "class2":class_result[1],
```

```
                    "class3":class_result[2],
```

```

        "prob1": prob_result[0],
        "prob2": prob_result[1],
        "prob3": prob_result[2],
    }
except Exception as e :
    print(str(e))
    error = 'This image from this site is not accesible or inappropriate input'
    if(len(error) == 0):
        return render_template('success.html' , img = img , predictions =
predictions)
    else:
        return render_template('index.html' , error = error)
elif (request.files):
    file = request.files['file']
    if file and allowed_file(file.filename):
        file.save(os.path.join(target_img , file.filename))
        img_path = os.path.join(target_img , file.filename)
        img = file.filename
        class_result , prob_result = predict(img_path , model)
        predictions = {
            "class1":class_result[0],
            "class2":class_result[1],
            "class3":class_result[2],
            "prob1": prob_result[0],
            "prob2": prob_result[1],
            "prob3": prob_result[2],
        }
    else:
        error = "Please upload images of jpg , jpeg and png extension only"
        if(len(error) == 0):
            return render_template('success.html' , img = img , predictions =
predictions)

```



```

        else:
            return render_template('index.html' , error = error)
    else:
        return render_template('index.html')

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

```

7.2 Feature 2

Training Yolo To prepare for the training process, convert the YOLOv3 model to the Keras format. The YOLOv3 Detector can then be trained by Train_YOLO.py file.

Code:

```

import os
import sys
import argparse
import warnings

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

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

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

```

```

sys.path.append(utils_path)

import numpy as np
import keras.backend as K
from keras.layers import Input, Lambda
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import (
    TensorBoard,
    ModelCheckpoint,
    ReduceLROnPlateau,
    EarlyStopping,
)
from keras_yolo3.yolo3.model import (
    preprocess_true_boxes,
    yolo_body,
    tiny_yolo_body,
    yolo_loss,
)
from keras_yolo3.yolo3.utils import get_random_data
from PIL import Image
from time import time
import tensorflow.compat.v1 as tf
import pickle

from Train_Utils import (
    get_classes,
    get_anchors,
    create_model,
    create_tiny_model,
    data_generator,
    data_generator_wrapper,

```

```

    ChangeToOtherMachine,
)

keras_path = os.path.join(src_path, "keras_yolo3")
Data_Folder = os.path.join(get_parent_dir(1), "Data")
Image_Folder = os.path.join(Data_Folder, "Source_Images", "Training_Images")
VoTT_Folder = os.path.join(Image_Folder, "vott-csv-export")
YOLO_filename = os.path.join(VoTT_Folder, "data_train.txt")

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

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

FLAGS = None

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

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

```

```

parser.add_argument(
    "--classes_file",
    type=str,
    default=YOLO_classname,
    help="Path to YOLO classnames. Default is " + YOLO_classname,
)
parser.add_argument(
    "--log_dir",
    type=str,
    default=log_dir,
    help="Folder to save training logs and trained weights to. Default is "
    + log_dir,
)
parser.add_argument(
    "--anchors_path",
    type=str,
    default=anchors_path,
    help="Path to YOLO anchors. Default is " + anchors_path,
)
parser.add_argument(
    "--weights_path",
    type=str,
    default=weights_path,
    help="Path to pre-trained YOLO weights. Default is " + weights_path,
)
parser.add_argument(
    "--val_split",
    type=float,
    default=0.1,
    help="Percentage of training set to be used for validation. Default is 10%.",
)
parser.add_argument(

```

```

    "--is_tiny",
    default=False,
    action="store_true",
    help="Use the tiny Yolo version for better performance and less accuracy.
Default is False.",
)
parser.add_argument(
    "--random_seed",
    type=float,
    default=None,
    help="Random seed value to make script deterministic. Default is 'None', i.e.
non-deterministic.",
)
parser.add_argument(
    "--epochs",
    type=float,
    default=51,
    help="Number of epochs for training last layers and number of epochs for
fine-tuning layers. Default is 51.",
)
parser.add_argument(
    "--warnings",
    default=False,
    action="store_true",
    help="Display warning messages. Default is False.",
)
)
FLAGS = parser.parse_args()
if not FLAGS.warnings:
    tf.logging.set_verbosity(tf.logging.ERROR)
    os.environ['TF_CPP_MIN_LOG_LEVEL']='3'
    warnings.filterwarnings("ignore")
np.random.seed(FLAGS.random_seed)

```

```

log_dir = FLAGS.log_dir
class_names = get_classes(FLAGS.classes_file)
num_classes = len(class_names)
anchors = get_anchors(FLAGS.anchors_path)
weights_path = FLAGS.weights_path
input_shape = (416, 416) # multiple of 32, height, width
epoch1, epoch2 = FLAGS.epochs, FLAGS.epochs
is_tiny_version = len(anchors) == 6 # default setting
if FLAGS.is_tiny:
    model = create_tiny_model(
        input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path
    )
else:
    model = create_model(
        input_shape, anchors, num_classes, freeze_body=2,
weights_path=weights_path
    ) # make sure you know what you freeze
log_dir_time = os.path.join(log_dir, "{}".format(int(time())))
logging = TensorBoard(log_dir=log_dir_time)
checkpoint = ModelCheckpoint(
    os.path.join(log_dir, "checkpoint.h5"),
    monitor="val_loss",
    save_weights_only=True,
    save_best_only=True,
    period=5,
)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.1, patience=3,
verbose=1)
early_stopping = EarlyStopping(
    monitor="val_loss", min_delta=0, patience=10, verbose=1
)

```

```

val_split = FLAGS.val_split
with open(FLAGS.annotation_file) as f:
    lines = f.readlines()
lines = ChangeToOtherMachine(lines, remote_machine="")
np.random.shuffle(lines)
num_val = int(len(lines) * val_split)
num_train = len(lines) - num_val
model = tf.nn.rnn_cell.LSTMCellWrapper(
    tf.nn.rnn_cell.LSTMCell(
        optimizer=Adam(lr=1e-3),
        loss={
            # use custom yolo_loss Lambda layer.
            "yolo_loss": lambda y_true, y_pred: y_true * 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"])

```

```

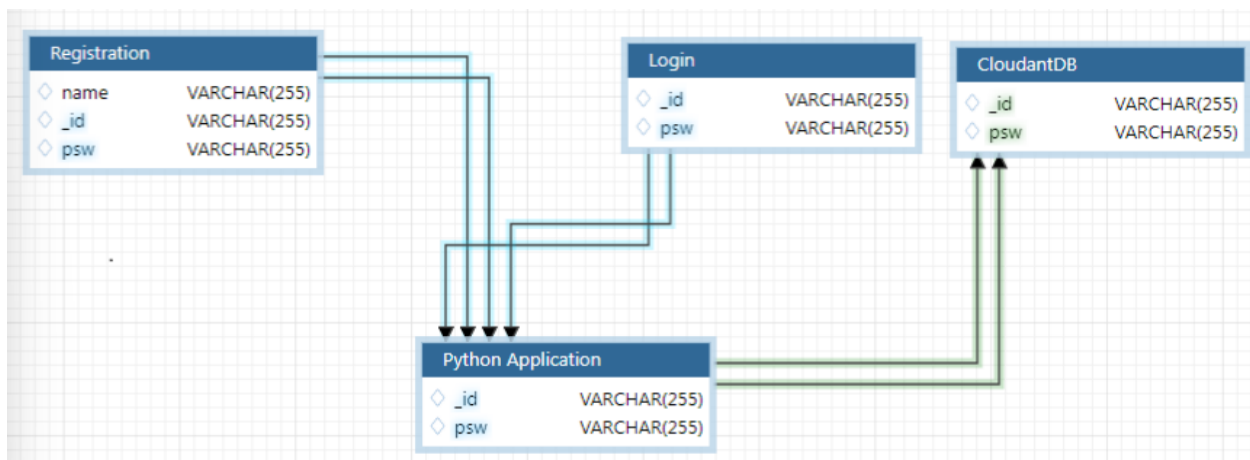
file = open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w")
with open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w") as f:
    for item in step2_val_loss:
        f.write("%s\n" % item)
file.close()

```

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:



Chapter 8

TESTING

8.1 TEST CASES:

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute
Sign_up_Page 00	Functional	Homepage	Verify if the user is able to enter details to sign-up	1.Enter the skin disease prediction url 2.Click on signup option 3.Verify Singup popup displayed or not
Login_Page _TD-1	UI	Homepage	Verify the UI elements in Login/Signup popup	1.Enter the skin disease prediction url 2.Click on signup option 3.Verify login/Singup popup with below UI elements: a.email text box b.password text box c.Login button d.New user? Create account link e.Last password? Recovery password link
Login_Page _TD-2	Functional	Homepage	Verify if the user is able to log in to the website with Valid credentials	1.Enter and click on go 2.Click on log in button 3.Enter Valid username/email in the text box 4.Enter valid password in the password text box

				5.Click on login button
Upload_data_page00	Functional	Upload data page	Verify if the user is able to upload the image of the problem	1.Enter the url and click on go 2.Enter the valid credentials and click on login 3. Click on upload button,drag on drop the image 4.Click on upload file button
Display_results_page_00	Functional	Display results page	Verify if the disease predicted from the image uploaded	1.Enter the URL and click on go 2.Enter the credentials and click on login 3.Click on upload image and upload the problem image 4.Click on upload and display results

Test Data	Expected Result	Actual Result	Status	Comments	Bug ID
	Signup popup should display	Working as expected	Pass	Steps are clear to follow	
	Application should show below UI elements: a.email text box b.password text box c.Login button with	Working as expected	Fail	Steps are not clear to follow	BUG-512126

	orange color d.New customer? Create account link e.Last password? Recovery password link				
Username: chalam@gmail password: Testing123	Users should be able to login and navigate to the file upload page.	Working as expected	Pass	Everything is defined well	BUG-312432
https://rb.gy/zt5ikl	User should be able to upload problem image	Working as expected	Pass	Working perfectly	BUG-435674
	The predicted results should be displayed upon the uploaded image	Working as expected	Pass		BUG-645376

8.2 ACCEPTANCE TESTING

Purpose of this Document :

The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] project at the time of the release to User Acceptance Testing (UAT).

Defect Analysis :

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved.

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	9	5	1	4	19
Duplicate	2	1	2	2	7
External	1	1	1	2	5
Fixed	10	1	3	19	33
Not Reproduced	1	1	2	1	5
Skipped	2	0	3	1	6
Won't fix	1	4	1	1	7
Totals	26	13	12	30	82

Test Case Analysis:

This report shows the number of test cases that have passed, failed, and untested.

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	52	0	0	52
Security	3	0	0	3
Outsource Shipping	2	0	0	2
Exception Reporting	8	0	0	88
Final Report Output	5	0	0	5
Version Control	1	0	0	1


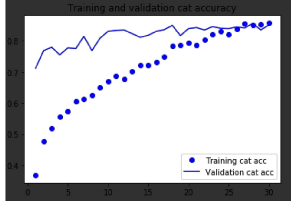
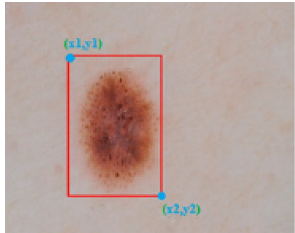
Chapter 9

RESULTS

9.1 PERFORMANCE METRICS:

Model Performance Testing:

Project team shall fill the following information in the model performance testing template.

S.No.	Parameter	Values	Screenshot
1	Model Summary	-	
2	Accuracy	Training Accuracy - 85% Validation Accuracy - 87%	
3	Confidence Score(Only Yolo Projects)	Confidence Score - 0.85 85% Z-value - 1.44	

ADVANTAGES AND DISADVANTAGES

Advantages:

- This system usually takes seconds to minutes to diagnose when confronted with a picture of a skin lesion. The inputs, algorithms, and outputs can be accessible to anyone with access to the web. This takes a brief timeframe than the stand by and go time given to a dermatologist arrangement.
- The calculations are regularly versatile and that they can gain from adding new pictures over the long haul.
- This system is prepared to proceed as confirmed dermatologists, and their precision will in any case improve in future.
- It can predict lesions so quickly with lesser cost than given to a dermatologist

Disadvantages:

- In general, the advantage of AI is that it can help doctors perform tedious repetitive tasks. For example, if sufficient blood is scanned, an AI-powered microscope can detect low-density infections in micrographs of standard, field-prepared thick blood films, which is considered to be time-consuming, difficult, and tedious owing to the low density and small parasite size and abundance of similar non-parasite objects .
- Medicine is an area that is not yet fully understood. Information is not completely transparent. The characteristics of dermatology determine that the majority of the data cannot be obtained. At the same time, the AI technology route is immature, the identification accuracy of which must be improved owing to the uncertainty of manual diagnosis. There is no strict correspondence between the symptoms and results of a disease and no clear boundary between the different diseases.

CONCLUSION:

We have shown that even without a large dataset and high-quality images, it is possible to achieve sufficient accuracy rates. In addition, we have shown that current state-of-the-art CNN models can outperform models created by previous research, through proper data preprocessing, self-supervised learning, transfer learning, and special CNN architecture techniques. The ensemble method and feature selection applied to dermatology datasets yields a better performance as compared to individual classifier algorithms. The ensemble method provides a more accurate and effective skin disease prediction. Furthermore, with accurate segmentation, we gain knowledge of the location of the disease, which is useful in the preprocessing of data used in classification, as it allows the CNN model to focus on the area of interest. Lastly, unlike previous studies, our method provides a solution to classify multiple diseases within a single image. With higher quality and a larger quantity of data, it will be viable to use state-of-the-art models to enable the use of CAD in the field of dermatology.

FUTURE SCOPE:

In future, this model may bind with various websites which can provide real-time data for skin disease prediction. Also, we may add large historical data on skin disease which can help to improve the accuracy of the machine learning model. We can build an android app as a user interface for interacting with the user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates, and train it on clusters of data rather than the whole dataset.

APPENDIX

SOURCE CODE:

```
from numpy.random import seed
seed(101)

from tensorflow.python.framework.random_seed import set_random_seed
set_random_seed(101)

import pandas as pd
import numpy as np

import tensorflow
from tensorflow.python.keras.layers import Dense, Dropout
from keras.optimizers import adam_v2
from tensorflow.python.keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint

import os

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline

os.listdir('./input')

base_dir = 'base_dir'
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)

val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)

nv = os.path.join(train_dir, 'nv')
os.mkdir(nv)
```

```

mel = os.path.join(train_dir, 'mel')
os.mkdir(mel)
bkl = os.path.join(train_dir, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(train_dir, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(train_dir, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(train_dir, 'vasc')
os.mkdir(vasc)
df = os.path.join(train_dir, 'df')
os.mkdir(df)

nv = os.path.join(val_dir, 'nv')
os.mkdir(nv)
mel = os.path.join(val_dir, 'mel')
os.mkdir(mel)
bkl = os.path.join(val_dir, 'bkl')
os.mkdir(bkl)
bcc = os.path.join(val_dir, 'bcc')
os.mkdir(bcc)
akiec = os.path.join(val_dir, 'akiec')
os.mkdir(akiec)
vasc = os.path.join(val_dir, 'vasc')
os.mkdir(vasc)
df = os.path.join(val_dir, 'df')
os.mkdir(df)

df_data = pd.read_csv('HAM10000_metadata.csv')

df_data.head()

df = df_data.groupby('lesion_id').count()

df = df[df['image_id'] == 1]

df.reset_index(inplace=True)

df.head()

def identify_duplicates(x):

    unique_list = list(df['lesion_id'])

```

```

    if x in unique_list:
        return 'no_duplicates'
    else:
        return 'has_duplicates'

df_data['duplicates'] = df_data['lesion_id']
df_data['duplicates'] = df_data['duplicates'].apply(identify_duplicates)

df_data.head()

df_data['duplicates'].value_counts()

df = df_data[df_data['duplicates'] == 'no_duplicates']

df.shape

y = df['dx']

_, df_val = train_test_split(df, test_size=0.17, random_state=101, stratify=y)

df_val.shape

df_val['dx'].value_counts()

def identify_val_rows(x):
    val_list = list(df_val['image_id'])

    if str(x) in val_list:
        return 'val'
    else:
        return 'train'

df_data['train_or_val'] = df_data['image_id']
df_data['train_or_val'] = df_data['train_or_val'].apply(identify_val_rows)

df_train = df_data[df_data['train_or_val'] == 'train']

print(len(df_train))
print(len(df_val))

df_train['dx'].value_counts()

df_val['dx'].value_counts()

```

```
df_data.set_index('image_id', inplace=True)
```

In [15]:

```
folder_1 = os.listdir('../input/ham10000_images_part_1')
```

```
folder_2 = os.listdir('../input/ham10000_images_part_2')
```

```
train_list = list(df_train['image_id'])
```

```
val_list = list(df_val['image_id'])
```

```
for image in train_list:
```

```
    fname = image + '.jpg'
```

```
    label = df_data.loc[image, 'dx']
```

```
    if fname in folder_1:
```

```
        src = os.path.join('../input/ham10000_images_part_1', fname)
```

```
        dst = os.path.join(train_dir, label, fname)
```

```
        shutil.copyfile(src, dst)
```

```
    if fname in folder_2:
```

```
        src = os.path.join('../input/ham10000_images_part_2', fname)
```

```
        dst = os.path.join(train_dir, label, fname)
```

```
        shutil.copyfile(src, dst)
```

```
for image in val_list:
```

```
    fname = image + '.jpg'
```

```
    label = df_data.loc[image, 'dx']
```

```
    if fname in folder_1:
```

```
        src = os.path.join('../input/ham10000_images_part_1', fname)
```

```
        dst = os.path.join(val_dir, label, fname)
```

```
        shutil.copyfile(src, dst)
```

```
    if fname in folder_2:
```

```
        src = os.path.join('../input/ham10000_images_part_2', fname)
```

```
        dst = os.path.join(val_dir, label, fname)
```

```
        shutil.copyfile(src, dst)
```

```
print(len(os.listdir('base_dir/train_dir/nv')))
```

```
print(len(os.listdir('base_dir/train_dir/mel')))
```

```

print(len(os.listdir('base_dir/train_dir/bkl')))
print(len(os.listdir('base_dir/train_dir/bcc')))
print(len(os.listdir('base_dir/train_dir/akiec')))
print(len(os.listdir('base_dir/train_dir/vasc')))
print(len(os.listdir('base_dir/train_dir/df')))

print(len(os.listdir('base_dir/val_dir/nv')))
print(len(os.listdir('base_dir/val_dir/mel')))
print(len(os.listdir('base_dir/val_dir/bkl')))
print(len(os.listdir('base_dir/val_dir/bcc')))
print(len(os.listdir('base_dir/val_dir/akiec')))
print(len(os.listdir('base_dir/val_dir/vasc')))
print(len(os.listdir('base_dir/val_dir/df')))

class_list = ['mel', 'bkl', 'bcc', 'akiec', 'vasc', 'df']

for item in class_list:

    aug_dir = 'aug_dir'
    os.mkdir(aug_dir)
    img_dir = os.path.join(aug_dir, 'img_dir')
    os.mkdir(img_dir)

    img_class = item

    img_list = os.listdir('base_dir/train_dir/' + img_class)

    for fname in img_list:
        src = os.path.join('base_dir/train_dir/' + img_class, fname)
        dst = os.path.join(img_dir, fname)
        shutil.copyfile(src, dst)

    path = aug_dir
    save_path = 'base_dir/train_dir/' + img_class

    datagen = ImageDataGenerator(
        rotation_range=180,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        vertical_flip=True,

```

```

        #brightness_range=(0.9, 1.1),
        fill_mode='nearest')

batch_size = 50

aug_datagen = datagen.flow_from_directory(path,
                                          save_to_dir=save_path,
                                          save_format='jpg',
                                          target_size=(224, 224),
                                          batch_size=batch_size)

num_aug_images_wanted = 6000

num_files = len(os.listdir(img_dir))
num_batches = int(np.ceil((num_aug_images_wanted-num_files)/batch_size))

for i in range(0, num_batches):

    imgs, labels = next(aug_datagen)

    shutil.rmtree('aug_dir')

print(len(os.listdir('base_dir/train_dir/nv')))
print(len(os.listdir('base_dir/train_dir/mel')))
print(len(os.listdir('base_dir/train_dir/bkl')))
print(len(os.listdir('base_dir/train_dir/bcc')))
print(len(os.listdir('base_dir/train_dir/akiec')))
print(len(os.listdir('base_dir/train_dir/vasc')))
print(len(os.listdir('base_dir/train_dir/df')))

print(len(os.listdir('base_dir/val_dir/nv')))
print(len(os.listdir('base_dir/val_dir/mel')))
print(len(os.listdir('base_dir/val_dir/bkl')))
print(len(os.listdir('base_dir/val_dir/bcc')))
print(len(os.listdir('base_dir/val_dir/akiec')))
print(len(os.listdir('base_dir/val_dir/vasc')))
print(len(os.listdir('base_dir/val_dir/df')))

def plots(ims, figsize=(12,6), rows=5, interp=False, titles=None): # 12,6
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)

```

```

        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')

plots(imgs, titles=None)

train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'

num_train_samples = len(df_train)
num_val_samples = len(df_val)
train_batch_size = 10
val_batch_size = 10
image_size = 224

train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)

datagen = ImageDataGenerator(
    preprocessing_function= \
        tensorflow.keras.applications.mobilenet.preprocess_input)

train_batches = datagen.flow_from_directory(train_path,
                                             target_size=(image_size, image_size),
                                             batch_size=train_batch_size)

valid_batches = datagen.flow_from_directory(valid_path,
                                             target_size=(image_size, image_size),
                                             batch_size=val_batch_size)

test_batches = datagen.flow_from_directory(valid_path,
                                             target_size=(image_size, image_size),
                                             batch_size=1,

```

In [23]:


```
shuffle=False)
```

```
from keras.layers import Flatten, Dense, Dropout, BatchNormalization, Conv2D,
MaxPool2D
```

```
from keras.optimizers import Adam
```

```
from keras.callbacks import ReduceLROnPlateau
```

```
from keras.preprocessing.image import ImageDataGenerator
```

```
import matplotlib.pyplot as plt
```

```
from PIL import Image
```

```
import seaborn as sns
```

```
import numpy as np
```

```
import pandas as pd
```

```
import os
```

```
from tensorflow.keras.utils import to_categorical
```

```
from glob import glob
```

In [26]:

```
from keras.models import Sequential
```

```
import keras.models
```

```
input_shape = (224, 224, 3)
```

```
num_classes = 7
```

```
model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding =
'Same', input_shape=input_shape))
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding = 'Same',))
```

```
model.add(MaxPool2D(pool_size = (2, 2)))
```

```
model.add(Dropout(0.16))
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding = 'Same'))
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding = 'Same',))
```

```
model.add(MaxPool2D(pool_size = (2, 2)))
```

```
model.add(Dropout(0.20))
```

```
model.add(Conv2D(64, (3, 3), activation='relu', padding = 'same'))
```

```
model.add(Conv2D(64, (3, 3), activation='relu', padding = 'Same'))
```

```
model.add(MaxPool2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
```

```
model.add(Flatten())
```

```
model.add(Dense(256, activation='relu'))
```

```
model.add(Dense(128, activation='relu'))
```

```
model.add(Dropout(0.4))
```

```

model.add(Dense(num_classes, activation='softmax'))
model.summary()

optimizer = Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
In [29]:
model.compile(optimizer = optimizer , loss = "categorical_crossentropy",
metrics=["accuracy"])
In [30]:
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
patience=4,
verbose=1,
factor=0.5,
min_lr=0.00001)
In [31]:
epochs = 30
batch_size = 10
history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
validation_data=valid_batches,
validation_steps=val_steps,
epochs=epochs, verbose=1,
callbacks=[learning_rate_reduction])
from sklearn.metrics import classification_report, confusion_matrix
In [33]:
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)
In [34]:
val_loss, val_cat_acc = \
model.evaluate_generator(test_batches,
steps=len(df_val))

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)

model.save("model.h5")

model.metrics_names

import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']

```

```

val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()
plt.figure()

plt.show()

test_labels = test_batches.classes

test_labels

test_batches.class_indices

predictions = model.predict_generator(test_batches, steps=len(df_val),
verbose=1)

predictions.shape

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    print(cm)

```

In [38]:

```

plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)

fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()

```

In [43]:

```

test_labels.shape
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))

```

In [45]:

```

test_batches.class_indices

cm_plot_labels = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')

y_pred = np.argmax(predictions, axis=1)
y_true = test_batches.classes

```

In [48]:

```

from sklearn.metrics import classification_report

report = classification_report(y_true, y_pred, target_names=cm_plot_labels)

print(report)

from numpy.random import seed
seed(101)

from tensorflow import set_random_seed
set_random_seed(101)

import pandas as pd
import numpy as np

```

```

import tensorflow
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint

```

```

import os

```

```

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline

```

```

from tensorflow.keras.metrics import categorical_accuracy,
top_k_categorical_accuracy
import matplotlib.pyplot as plt

```

In [50]:

```

datagen = ImageDataGenerator(
    preprocessing_function= \
        tensorflow.keras.applications.mobilenet.preprocess_input)

train_batches = datagen.flow_from_directory(train_path,

target_size=(image_size, image_size),

                                         batch_size=train_batch_size)

valid_batches = datagen.flow_from_directory(valid_path,

target_size=(image_size, image_size),

                                         batch_size=val_batch_size)

test_batches = datagen.flow_from_directory(valid_path,

target_size=(image_size, image_size),

                                         batch_size=1,
                                         shuffle=False)

```

```

mobile = tensorflow.keras.applications.mobilenet.MobileNet()

mobile.summary()

type(mobile.layers)

len(mobile.layers)

x = mobile.layers[-6].output

x = Dropout(0.25)(x)
predictions = Dense(7, activation='softmax')(x)

model = Model(inputs=mobile.input, outputs=predictions)

model.summary()

for layer in model.layers[:-23]:
    layer.trainable = False

from tensorflow.keras.metrics import categorical_accuracy,
top_k_categorical_accuracy

def top_3_accuracy(y_true, y_pred):
    return top_k_categorical_accuracy(y_true, y_pred, k=3)

def top_2_accuracy(y_true, y_pred):
    return top_k_categorical_accuracy(y_true, y_pred, k=2)

model.compile(Adam(lr=0.01), loss='categorical_crossentropy',
              metrics=[categorical_accuracy, top_2_accuracy,
top_3_accuracy])

print(valid_batches.class_indices)
class_weights={
    0: 1.0, # akiec
    1: 1.0, # bcc
    2: 1.0, # bkl
    3: 1.0, # df
    4: 3.0, # mel # Try to make the model more sensitive to Melanoma.
    5: 1.0, # nv
    6: 1.0, # vasc
}

```

In [56]:

In [59]:

In [60]:

In [62]:

```

filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_top_3_accuracy',
verbose=1,
                                save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_top_3_accuracy', factor=0.5,
patience=2,
                                verbose=1, mode='max', min_lr=0.00001)

callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
                                class_weight=class_weights,
                                validation_data=valid_batches,
                                validation_steps=val_steps,
                                epochs=30, verbose=1,
                                callbacks=callbacks_list)

model.metrics_names

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate_generator(test_batches,
                        steps=len(df_val))

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
print('val_top_2_acc:', val_top_2_acc)
print('val_top_3_acc:', val_top_3_acc)

model.load_weights('model.h5')

val_loss, val_cat_acc, val_top_2_acc, val_top_3_acc = \
model.evaluate_generator(test_batches,
                        steps=len(df_val))

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
print('val_top_2_acc:', val_top_2_acc)
print('val_top_3_acc:', val_top_3_acc)

import matplotlib.pyplot as plt

acc = history.history['categorical_accuracy']

```

```

val_acc = history.history['val_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
train_top2_acc = history.history['top_2_accuracy']
val_top2_acc = history.history['val_top_2_accuracy']
train_top3_acc = history.history['top_3_accuracy']
val_top3_acc = history.history['val_top_3_accuracy']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, train_top2_acc, 'bo', label='Training top2
acc')
plt.plot(epochs, val_top2_acc, 'b', label='Validation top2 acc')
plt.title('Training and validation top2 accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, train_top3_acc, 'bo', label='Training top3
acc')
plt.plot(epochs, val_top3_acc, 'b', label='Validation top3 acc')
plt.title('Training and validation top3 accuracy')
plt.legend()

plt.show()

test_labels = test_batches.classes

test_labels

test_batches.class_indices

predictions = model.predict_generator(test_batches, steps=len(df_val),

```

In [68]:


```

verbose=1)

predictions.shape

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

test_labels.shape
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))

test_batches.class_indices

cm_plot_labels = ['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')

```

In [73]:

In [75]:

```
y_pred = np.argmax(predictions, axis=1)
```

```
y_true = test_batches.classes
```

In [78]:

```
from sklearn.metrics import classification_report
```

```
report = classification_report(y_true, y_pred, target_names=cm_plot_labels)
```

```
print(report)
```

```
model_json = model.to_json()
```

```
with open("model.json", "w") as j_file:  
    j_file.write(model_json)
```

In [80]:

```
model_json = model.to_json()
```

```
with open("model.json", "w") as json_file:  
    json_file.write(model_json)
```

```
# serialize weights to HDF5
```

```
model.save_weights("model1.h5")
```

```
print("Saved model to disk")
```

#app.py

import os

import uuid

import flask

import urllib

from PIL import Image

from numpy import number

from tensorflow.python.keras.models import load_model

**from flask import Flask , render_template , request ,
send_file,redirect,url_for,session**

from tensorflow.keras.preprocessing.image import load_img , img_to_array

```
from keras.applications.mobilenet import preprocess_input,  
decode_predictions
```

```
from keras.models import model_from_json
```

```
from keras.models import load_model
```

```
from mydb import connection as db
```

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

```
j_file = open('model.json', 'r')
```

```
loaded_json_model = j_file.read()
```

```
j_file.close()
```

```
model = model_from_json(loaded_json_model)
```

```
model.load_weights('model.h5')
```

```
ALLOWED_EXT = set(['jpg' , 'jpeg' , 'png' , 'jif'])
```

```
def allowed_file(filename):
```

```
    return '.' in filename and \
```

```
        filename.rsplit('.', 1)[1] in ALLOWED_EXT
```

```
classes = [
```

```
'Actine Keratoses',  
'Basal Cell Carcinoma',  
'Benign Keratosis',  
'Erythema Psoriasis',  
'Melanoma',  
'Melanocytic Nevi',  
'vascular Naevus'  
  
]
```

```
def predict(filename , model):  
  
    img = load_img(filename , target_size = (224, 224))  
  
    img = img_to_array(img)  
  
    img = img.reshape(1,224,224,3)  
  
  
    img = img.astype('float32')  
  
    img = img/255.0  
  
    result = model.predict(img)
```

```

dict_result = {}

for i in range(7):

    dict_result[result[0][i]] = classes[i]


res = result[0]

res.sort()

res = res[::-1]

prob = res[:3]


prob_result = []

class_result = []

for i in range(3):

    prob_result.append((prob[i]*100).round(2))

    class_result.append(dict_result[prob[i]])


return class_result , prob_result


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

def log():

    return render_template('login.html')

```

```

@app.route('/login', methods=['GET', 'POST'])

def login():

    msg = ''

    if request.method == 'POST':

        name = request.form["username"]

        password = request.form["password"]

        result_dict = db.login(name,password)

        msg = 'Invalid Username / Password'

        if result_dict != False:

            # session["name"] = request.form["username"]

            # msg = session["name"]

            # print(msg)

            return redirect('/index')

        return render_template('login.html', msg = msg)

    else:

        return render_template('login.html', msg = msg)

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

```

```
def register():  
  
    msg = ''  
  
    if request.method == 'POST':  
  
        name = request.form["username"]  
  
        email = request.form["email"]  
  
        password = request.form["password"]  
  
        print(name,email,password)  
  
        db.register(name,email,password)  
  
        return render_template('login.html', msg = msg)  
  
    else:  
  
        return render_template('register.html', msg = msg)
```

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

```
def home():  
  
    return render_template("index.html")
```

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

```
def about():
```

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

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

```
def contact():
```

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

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

```
# def login():
```

```
#     return render_template("login.html")
```

```
# more changes to be made in contact
```

```
# @app.route('/about1/')
```

```
# def about():
```

```
#     return "this is about page"
```



```

@app.route('/success' , methods = ['GET' , 'POST'])

def success():

    error = ''

    target_img = os.path.join(os.getcwd() , 'static/images')

    if request.method == 'POST':

        if(request.form):

            link = request.form.get('link')

            try :

                resource = urllib.request.urlopen(link)

                unique_filename = str(uuid.uuid4())

                filename = unique_filename+".jpg"

                img_path = os.path.join(target_img , filename)

                output = open(img_path , "wb")

                output.write(resource.read())

                output.close()

                img = filename

            class_result , prob_result = predict(img_path , model)

```

```

    predictions = {
        "class1":class_result[0],
        "class2":class_result[1],
        "class3":class_result[2],
        "prob1": prob_result[0],
        "prob2": prob_result[1],
        "prob3": prob_result[2],
    }

except Exception as e :

    print(str(e))

    error = 'This image from this site is not accesible or inappropriate
input'

    if(len(error) == 0):

        return render_template('success.html' , img = img , predictions =
predictions)

    else:

        return render_template('index.html' , error = error)

```

```
elif (request.files):  
  
    file = request.files['file']  
  
    if file and allowed_file(file.filename):  
  
        file.save(os.path.join(target_img , file.filename))  
  
        img_path = os.path.join(target_img , file.filename)  
  
        img = file.filename  
  
  
        class_result , prob_result = predict(img_path , model)  
  
  
        predictions = {  
  
            "class1":class_result[0],  
  
            "class2":class_result[1],  
  
            "class3":class_result[2],  
  
            "prob1": prob_result[0],  
  
            "prob2": prob_result[1],  
  
            "prob3": prob_result[2],  
  
        }
```

else:

error = "Please upload images of jpg , jpeg and png extension only"

if(len(error) == 0):

**return render_template('success.html' , img = img , predictions =
predictions)**

else:

return render_template('index.html' , error = error)

else:

return render_template('index.html')

if __name__ == "__main__":

app.run(debug = True)

GitHub & Project Demo Link

✓ <https://github.com/IBM-EPBL/IBM-Project-24103-1659937745>

✓ https://drive.google.com/file/d/1z4Z-Zym9YQY_YABPI9SDKpKU2VypKISg/view?usp=share_link