**AI-BASED LOCALIZATION AND CLASSIFICATION OF SKIN DISEASE WITH ERYTHEMA**

**NALAYA THIRAN PROJECT**

**BASED LEARNING ON**

**PROFESSIONAL READLINESS**

**FOR INNOVATION,**

**EMPLOYEMENT AND**

**ENTERPRENEURSHIP**

**A PROJECT REPORT**

**Submitted**

**By**

**SUSI DEVI V**

**SUMAYA SAFRIN A N**

**NOUFIA KHADHEEJA M**

**SHALINI B**

**TEAM ID :** PNT2022TMID34065

**INDUSTRY MENTOR :** Sri Tulasi

**FACULTY MENTOR :** M.Mary Vespa

**COLLEGE NAME :** ARUNACHALA

COLLEGE OF ENGINEERING

FOR WOMEN

**PROJECT REPORT**

**1.INTRODUCTION**

1.1 Project Overview

1.2 Purpose

**2.LITERATURE SURVEY**

2.1 Existing problem

2.2 References

2.3 Problem Statement Definition

**3.IDEATION&PROPOSED SOLUTION**

3.1 Empathy Map Canvas

3.2 Proposed Solution

3.3 Problem solution fit

**4.REQUIREMENT ANALYSIS**

4.1 Functional requirements

4.2 Non-Functional requirements

**5.PROJECT DESIGN**

5.1 Data flow diagrams

5.2 Solution & Technical Architecture

5.3 User Stories

**6.PROJECT PLANNING &SCHEDULING**

6.1 Sprint planning and Estimation

6.2 Sprint planning Schedule

6.3 Reports from JIRA

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

7.1 Feature 1

7.2 Feature 2

7.3 Database Schema

**8.TESTING**

8.1 Test Cases

8.2 User Acceptance Testing

**9.RESULTS**

9.1Performance Metrics

**10.ADVANTAGES &DISADVANTAGES**

**11.CONCLUSION**

**12.FUTURE SCOPE**

**13.APPENDIX**

Source code

GitHub &Project Demo Link

**1.INTRODUCTION**

**1.1.Project Overview**

Now a day’s people are suffering from skin diseases, More than 125 million people suffering from Psoriasis also skin cancer rate is rapidly increasing over the last few decades especially Melanoma is most diversifying skin cancer. If skin diseases are not treated at an earlier stage, then it may lead to complications in the body including spreading of the infection from one individual to the other. The skin diseases can be prevented by investigating the infected region at an early stage. The characteristic of the skin images is diversified so that it is a challenging job to devise an efficient and robust algorithm for automatic detection of skin disease and its severity. Skin tone and skin colour play an important role in skin disease detection. Colour and coarseness of skin are visually different. Automatic processing of such images for skin analysis requires quantitative discriminator to differentiate the diseases.

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

**1.2.Purpose**

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 that it is a challenging job to devise an efficient and robust algorithm for automatic detection of skin disease and its severity. Automatic processing of such images for skin analysis requires quantitativediscriminator to differentiate the diseases.

**2. LITERATURE SURVEY**

**2.1.Existing problem**

A neglected public health problem Skin diseases are among the most common health problems in humans.Considering their significant impact on the individual, the family, the social life of patients, and their heavy economic burden, the public health importance of these diseases is underappreciated.

**2.2.References**

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

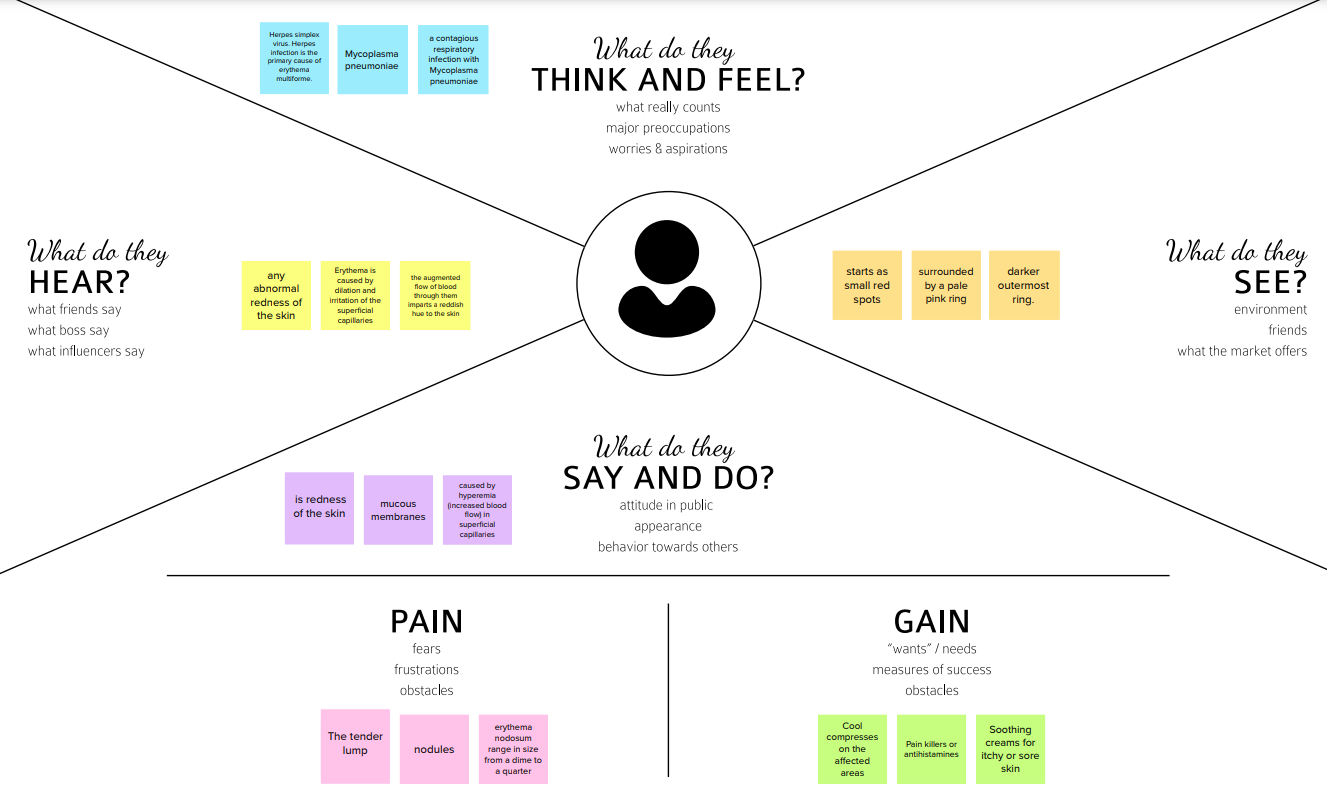
[2] S. Verma, M. A. Razzaque, U. Sangtongdee, C. Arpnikanondt, B. Tassaneetrithep, and A. Hossain, “Digital diagnosis of Hand, Foot, and mouth disease using hybrid deep neural networks,” IEEE Access, vol. 9, pp. 143481–143494, 2021.

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

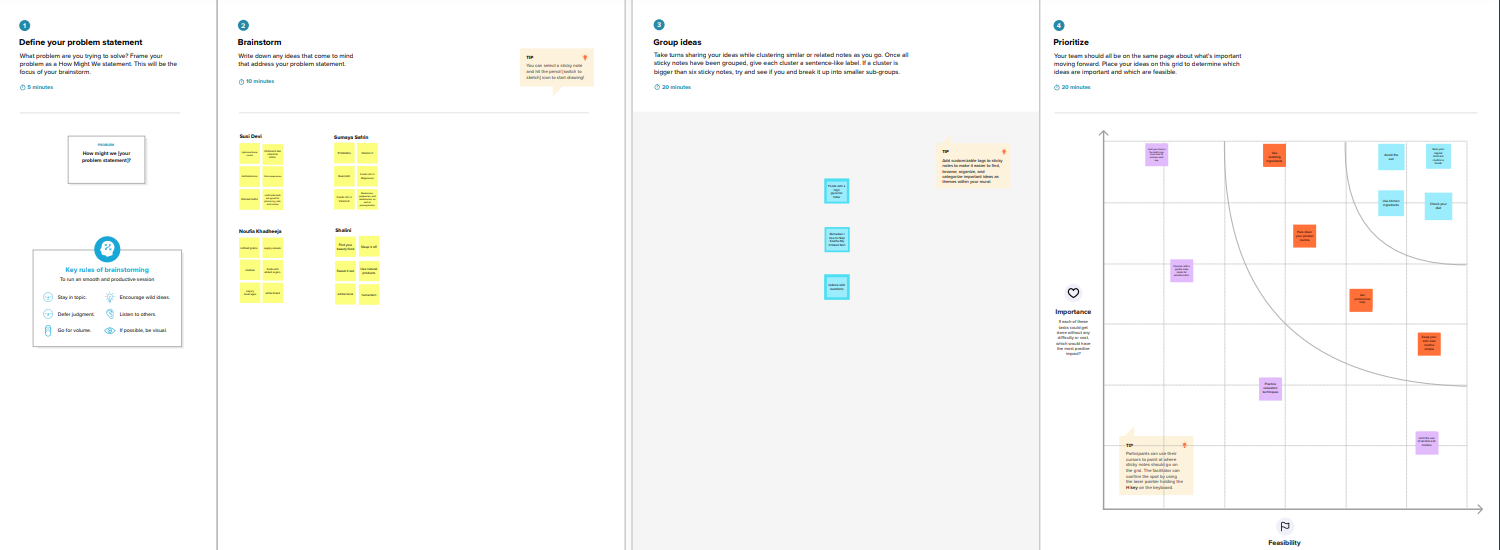
**2.3.Problem Statement Definition**

We're trying to find a solution to identify Skin Disease but Developed model is under training because given an image of skin, we can decompose, segment, and classify in a sequential manner which takes to Early detection of skin cancer, psoriasis.

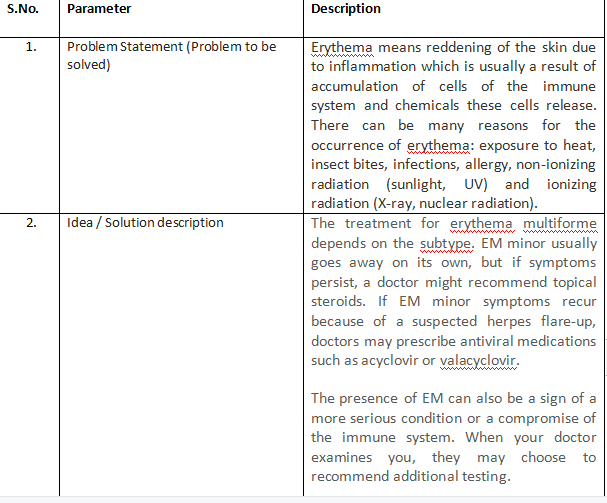
**3.IDEATION & PROPOSED SOLUTION**

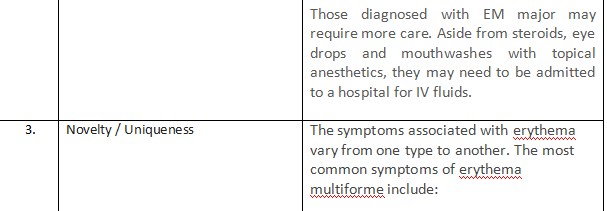
**3.1.Empathy Map Canvas**

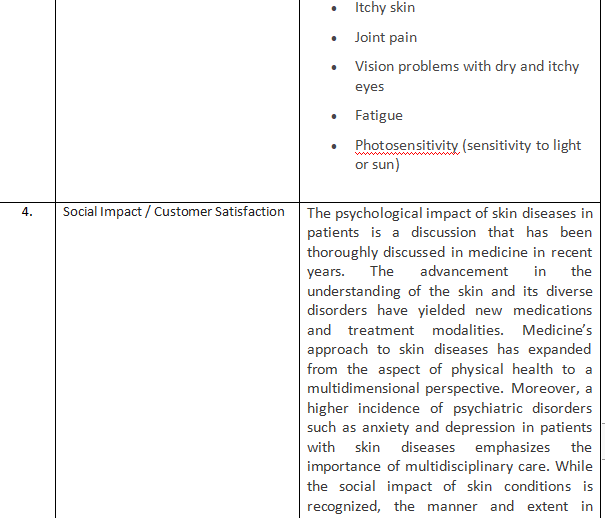
**3.2.Ideation & Brainstroming**

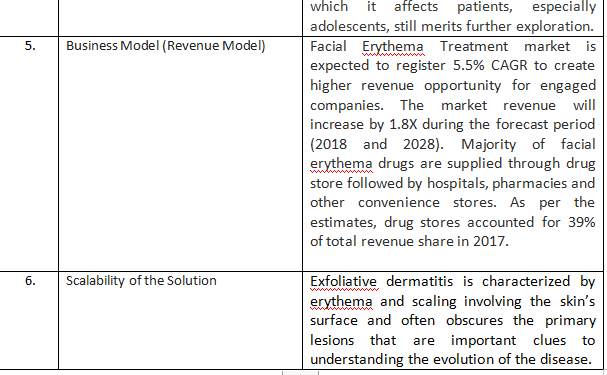


**3.3.Proposed Solution**

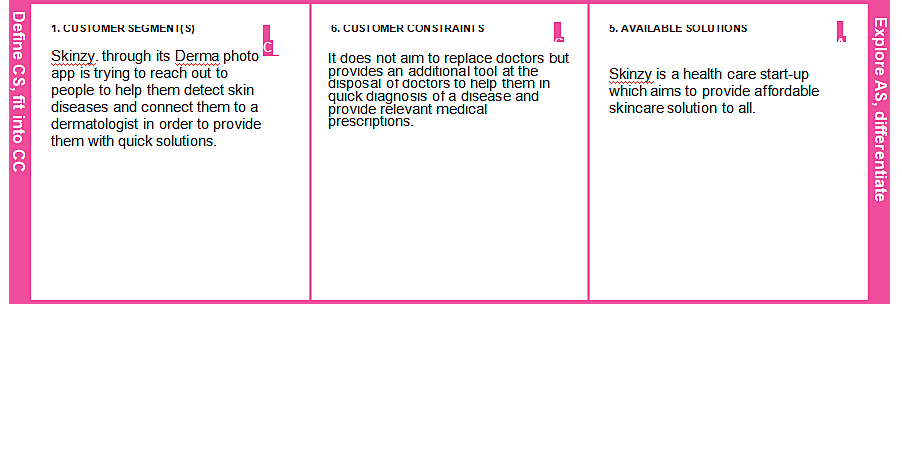


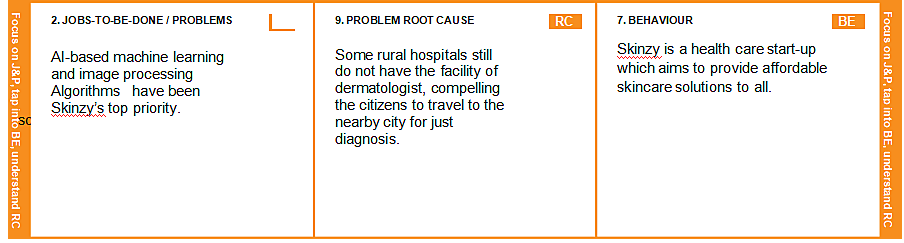


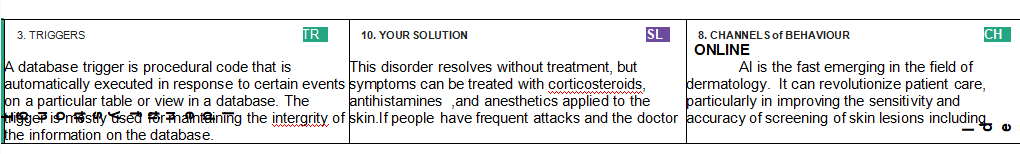


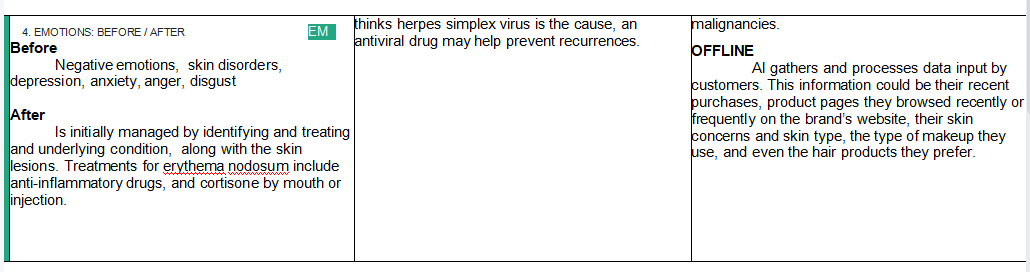


**3.4.Problem Solution fit**



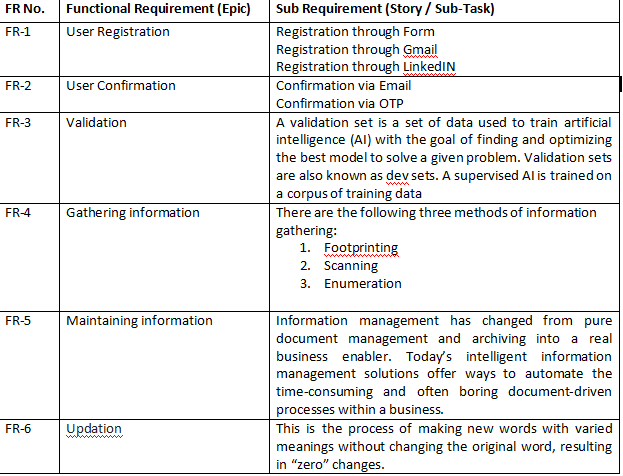




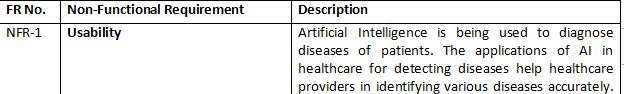


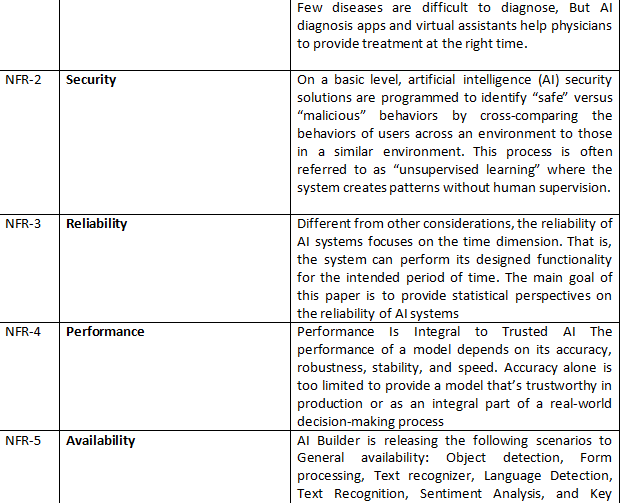
**4. REQUIREMENT ANALYSIS**

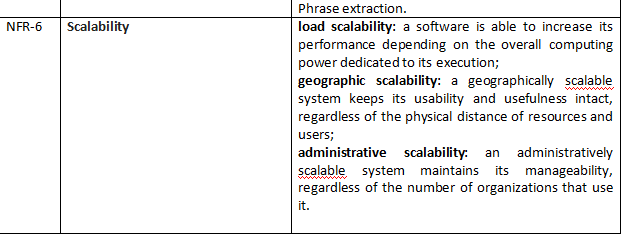
**4.1.Functional requirements**



**4.2.Non-Functional requirements**

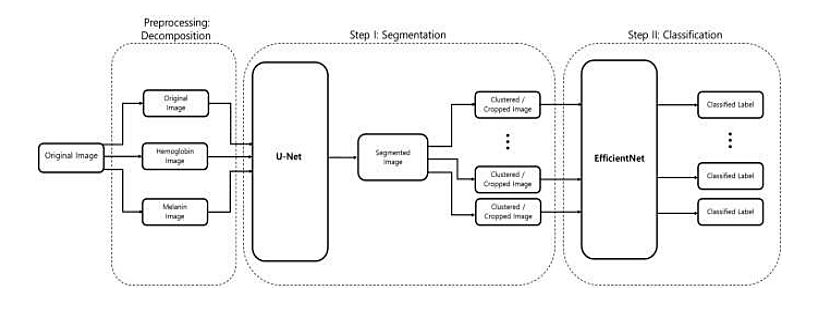




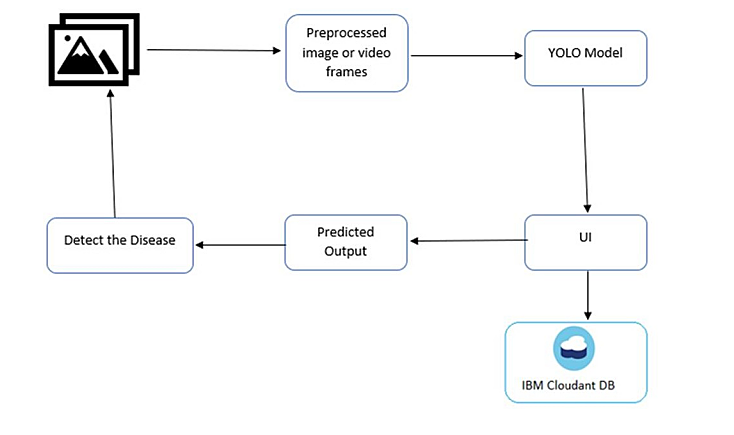


**5. PROJECT DESIGN**

**5.1.Data Flow Diagram**



**5.2.Solution and Technical Architecture**

**5.3 User Stories**

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

**6.PROJECT PLANNING & SCHEDULING**

**6.1.Sprint Planning and Estimation**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional Requirement (Epic** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| Sprint-1 | Pre-requisites | USN-1 | Install Python IDE, Python packages, Microsoft Visual Object Tagging Tool, Yolo Structure | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-1 | Create Dataset | USN-2 | Now we are going to collect the images of different skin disease from google | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja |
| Sprint-1 | Annotate Images | USN-3 | Create A Project in VOTT (Microsoft's Visual Object Tagging Tool) | 2 | Medium | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Spirnt-2 | Training YOLO | USN-4 | train our model using YOLO weights | 2 | Medium | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Spirnt-2 |  | USN-5 | To Download and Convert PreTrained Weights | 3 | High | Susi devi,Noufia khadheeja, Shalini |
| Spirnt-2 |  | USN-6 | To Train YOLOv3 Detector | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja |
| Sprint-3 | Cloudant DB | USN-7 | Register & Login to IBM Cloud | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-3 |  | USN-8 | Create Service Instant and Credentials | 2 | Medium | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-3 |  | USN-9 | Launch DB and Create database | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-3 | Development Phase | USN-10 | To build a web application | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-3 |  | USN-11 | Building HTML pages with python code | 2 | Medium | Sumaya Safrin,Noufia khadheeja |
| Sprint-3 |  | USN-12 | To run the application | 3 | High | Susi devi,Sumaya Safrin |
| Sprint-4 | Testing Phase | USN-13 | As a user login to dashboard | 2 | Medium | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-4 |  | USN-14 | As a user import the images with skin diseases to the software application | 2 | Medium | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |
| Sprint-4 |  | USN-15 | YOLO processes the image and give the necessary details | 3 | High | Susi devi,Sumaya Safrin,Noufia khadheeja, Shalini |

**6.2 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 Nov 2022 |

**7. CODING & SOLUTIONING**

import tensorflow as tf

import tensorflow\_hub as hub

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

from tensorflow.keras.utils import get\_file

from sklearn.metrics import roc\_curve, auc, confusion\_matrix

from imblearn.metrics import sensitivity\_score, specificity\_score

import os

import glob

import zipfile

import random

# to get consistent results after multiple runs

tf.random.set\_seed(7)

np.random.seed(7)

random.seed(7)

# 0 for benign, 1 for malignant

class\_names = ["benign", "malignant"]

**Preparing the Dataset**

def download\_and\_extract\_dataset():

# dataset from https://github.com/udacity/dermatologist-ai

# 5.3GB

train\_url = "https://s3-us-west-1.amazonaws.com/udacitydlnfd/datasets/skin-cancer/train.zip"

# 824.5MB

valid\_url = "https://s3-us-west-1.amazonaws.com/udacitydlnfd/datasets/skin-cancer/valid.zip"

# 5.1GB

test\_url = "https://s3-us-west-1.amazonaws.com/udacitydlnfd/datasets/skin-cancer/test.zip"

for i, download\_link in enumerate([valid\_url, train\_url, test\_url]):

temp\_file = f"temp{i}.zip"

data\_dir = get\_file(origin=download\_link,

fname=os.path.join(os.getcwd(), temp\_file))

print("Extracting", download\_link)

with zipfile.ZipFile(data\_dir, "r") as z:

z.extractall("data")

# remove the temp file

os.remove(temp\_file)

# comment the below line if you already downloaded the dataset

download\_and\_extract\_dataset()

# preparing data

# generate CSV metadata file to read img paths and labels from it

def generate\_csv(folder, label2int):

folder\_name = os.path.basename(folder)

labels = list(label2int)

# generate CSV file

df = pd.DataFrame(columns=["filepath", "label"])

i = 0

for label in labels:

print("Reading", os.path.join(folder, label, "\*"))

for filepath in glob.glob(os.path.join(folder, label, "\*")):

df.loc[i] = [filepath, label2int[label]]

i += 1

output\_file = f"{folder\_name}.csv"

print("Saving", output\_file)

df.to\_csv(output\_file)

# generate CSV files for all data portions, labeling nevus and seborrheic keratosis

# as 0 (benign), and melanoma as 1 (malignant)

# you should replace "data" path to your extracted dataset path

# don't replace if you used download\_and\_extract\_dataset() function

generate\_csv("data/train", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1}

generate\_csv("data/valid", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1})

generate\_csv("data/test", {"nevus": 0, "seborrheic\_keratosis": 0, "melanoma": 1})

# loading data

train\_metadata\_filename = "train.csv"

valid\_metadata\_filename = "valid.csv"

# load CSV files as DataFrames

df\_train = pd.read\_csv(train\_metadata\_filename)

df\_valid = pd.read\_csv(valid\_metadata\_filename)

n\_training\_samples = len(df\_train)

n\_validation\_samples = len(df\_valid)

print("Number of training samples:", n\_training\_samples)

print("Number of validation samples:", n\_validation\_samples)

train\_ds = tf.data.Dataset.from\_tensor\_slices((df\_train["filepath"], df\_train["label"]))

valid\_ds = tf.data.Dataset.from\_tensor\_slices((df\_valid["filepath"], df\_valid["label"]))

Number of training samples: 2000

Number of validation samples: 150

Copy

Let's load the images:

# preprocess data

def decode\_img(img):

# convert the compressed string to a 3D uint8 tensor

img = tf.image.decode\_jpeg(img, channels=3)

# Use `convert\_image\_dtype` to convert to floats in the [0,1] range.

img = tf.image.convert\_image\_dtype(img, tf.float32)

# resize the image to the desired size.

return tf.image.resize(img, [299, 299])

def process\_path(filepath, label):

# load the raw data from the file as a string

img = tf.io.read\_file(filepath)

img = decode\_img(img)

return img, label

valid\_ds = valid\_ds.map(process\_path)

train\_ds = train\_ds.map(process\_path)

# test\_ds = test\_ds

for image, label in train\_ds.take(1):

print("Image shape:", image.shape)

print("Label:", label.numpy())

Image shape: (299, 299, 3)

Label: 0

# training parameters

batch\_size = 64

optimizer = "rmsprop"

def prepare\_for\_training(ds, cache=True, batch\_size=64, shuffle\_buffer\_size=1000):

if cache:

if isinstance(cache, str):

ds = ds.cache(cache)

else:

ds = ds.cache()

# shuffle the dataset

ds = ds.shuffle(buffer\_size=shuffle\_buffer\_size)

# Repeat forever

ds = ds.repeat()

# split to batches

ds = ds.batch(batch\_size)

# `prefetch` lets the dataset fetch batches in the background while the model

# is training.

ds = ds.prefetch(buffer\_size=tf.data.experimental.AUTOTUNE)

return ds

valid\_ds = prepare\_for\_training(valid\_ds, batch\_size=batch\_size, cache="valid-cached-data")

train\_ds = prepare\_for\_training(train\_ds, batch\_size=batch\_size, cache="train-cached-data")

batch = next(iter(valid\_ds))

def show\_batch(batch):

plt.figure(figsize=(12,12))

for n in range(25):

ax = plt.subplot(5,5,n+1)

plt.imshow(batch[0][n])

plt.title(class\_names[batch[1][n].numpy()].title())

plt.axis('off')

show\_batch(batch)

**Output:**



# building the model

# InceptionV3 model & pre-trained weights

module\_url = "https://tfhub.dev/google/tf2- preview/inception\_v3/feature\_vector/4"

m = tf.keras.Sequential([ hub.KerasLayer(module\_url, output\_shape=[2048], trainable=False),

tf.keras.layers.Dense(1, activation="sigmoid") ])

m.build([None, 299, 299, 3])

m.compile(loss="binary\_crossentropy", optimizer=optimizer, metrics=["accuracy"])

m.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # =================================================================

keras\_layer (KerasLayer) multiple 21802784 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) multiple 2049

=================================================================

Total params: 21,804,833

Trainable params: 2,049

Non-trainable params: 21,802,784

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Training the Model**

We now have our dataset and the model, let's get them together:

model\_name = f"benign-vs-malignant\_{batch\_size}\_{optimizer}"

tensorboard = tf.keras.callbacks.TensorBoard(log\_dir=os.path.join("logs", model\_name))

# saves model checkpoint whenever we reach better weights

modelcheckpoint = tf.keras.callbacks.ModelCheckpoint(model\_name + "\_{val\_loss:.3f}.h5",

save\_best\_only=True, verbose=1)

history = m.fit(train\_ds, validation\_data=valid\_ds,

steps\_per\_epoch=n\_training\_samples // batch\_size,

validation\_steps=n\_validation\_samples // batch\_size,

verbose=1, epochs=100,

callbacks=[tensorboard, modelcheckpoint])

Here is a part of the output during training:

Train for 31 steps, validate for 2 steps

Epoch 1/100

30/31 [============================>.] - ETA: 9s - loss: 0.4609 - accuracy: 0.7760

Epoch 00001: val\_loss improved from inf to 0.49703, saving model to benign-vs

-malignant\_64\_rmsprop\_0.497.h5

31/31 [==============================] - 282s 9s/step - loss: 0.4646 - accuracy: 0.7722 - val\_loss: 0.4970

- val\_accuracy: 0.8125

<..SNIPED..>

Epoch 27/100

30/31 [============================>.] - ETA: 0s - loss: 0.2982 - accuracy: 0.8708

Epoch 00027: val\_loss improved from 0.40253 to 0.38991, saving model to benign-vs

-malignant\_64\_rmsprop\_0.390.h5

31/31 [==============================] - 21s 691ms/step - loss: 0.3025 - accuracy: 0.8684 - val\_loss:

0.3899 - val\_accuracy: 0.8359

<..SNIPED..>

Epoch 41/100

30/31 [============================>.] - ETA: 0s - loss: 0.2800 - accuracy: 0.8802

Epoch 00041: val\_loss did not improve from 0.38991

31/31 [==============================] - 21s 690ms/step - loss: 0.2829 - accuracy: 0.8790 - val\_loss:

0.3948 - val\_accuracy: 0.8281

Epoch 42/100 30/31 [============================>.] - ETA: 0s - loss: 0.2680 - accuracy: 0.8859

Epoch 00042: val\_loss did not improve from 0.38991

31/31 [==============================] - 21s 693ms/step - loss: 0.2722 - accuracy: 0.8831 - val\_loss:

0.4572 - val\_accuracy: 0.8047

**Model Evaluation**

First, let's load our test set, just like previously:

# evaluation

# load testing set

test\_metadata\_filename = "test.csv"

df\_test = pd.read\_csv(test\_metadata\_filename)

n\_testing\_samples = len(df\_test)

print("Number of testing samples:", n\_testing\_samples)

test\_ds = tf.data.Dataset.from\_tensor\_slices((df\_test["filepath"], df\_test["label"]))

def prepare\_for\_testing(ds, cache=True, shuffle\_buffer\_size=1000):

if cache:

if isinstance(cache, str):

ds = ds.cache(cache)

else:

ds = ds.cache()

ds = ds.shuffle(buffer\_size=shuffle\_buffer\_size)

return ds

test\_ds = test\_ds.map(process\_path)

test\_ds = prepare\_for\_testing(test\_ds, cache="test-cached-data")

The above code loads our test data and prepares it for testing:

Number of testing samples: 600

600 images of the shape (299, 299, 3) can fit our memory, let's convert our test set from tf.data into a NumPy array:

# convert testing set to numpy array to fit in memory (don't do that when testing

# set is too large)

y\_test = np.zeros((n\_testing\_samples,))

X\_test = np.zeros((n\_testing\_samples, 299, 299, 3))

for i, (img, label) in enumerate(test\_ds.take(n\_testing\_samples)):

# print(img.shape, label.shape)

X\_test[i] = img

y\_test[i] = label.numpy()

print("y\_test.shape:", y\_test.shape)

# load the weights with the least loss

m.load\_weights("benign-vs-malignant\_64\_rmsprop\_0.390.h5")

print("Evaluating the model...")

loss, accuracy = m.evaluate(X\_test, y\_test, verbose=0)

print("Loss:", loss, " Accuracy:", accuracy)

Copy

Output:

Evaluating the model...

Loss: 0.4476394319534302 Accuracy: 0.8

The below function does that:

def get\_predictions(threshold=None):

"""

Returns predictions for binary classification given `threshold` For instance, if threshold is 0.3, then it'll output 1

(malignant) for that sample if the probability of 1 is 30% or more (instead of 50%) """

y\_pred = m.predict(X\_test)

if not threshold:

threshold = 0.5

result = np.zeros((n\_testing\_samples,))

for i in range(n\_testing\_samples):

# test melanoma probability

if y\_pred[i][0] >= threshold:

result[i] = 1

# else, it's 0 (benign)

return result

threshold = 0.23

# get predictions with 23% threshold

# which means if the model is 23% sure or more that is malignant,

# it's assigned as malignant, otherwise it's benign

y\_pred = get\_predictions(threshold)

Now let's draw our confusion matrix and interpret it:

def plot\_confusion\_matrix(y\_test, y\_pred):

cmn = confusion\_matrix(y\_test, y\_pred)

# Normalise

cmn = cmn.astype('float') / cmn.sum(axis=1)[:, np.newaxis]

# print it

print(cmn)

fig, ax = plt.subplots(figsize=(10,10))

sns.heatmap(cmn, annot=True, fmt='.2f',

xticklabels=[f"pred\_{c}" for c in class\_names],

yticklabels=[f"true\_{c}" for c in class\_names],

cmap="Blues" )

plt.ylabel('Actual')

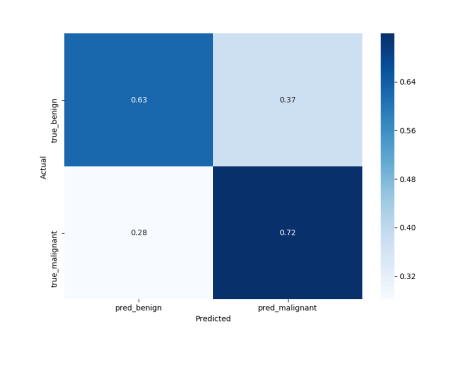
plt.xlabel('Predicted')

# plot the resulting confusion matrix

plt.show()

plot\_confusion\_matrix(y\_test, y\_pred)

**Output:**



def plot\_roc\_auc(y\_true, y\_pred):

""" This function plots the ROC curves and provides the scores. """

# prepare for figure

plt.figure()

fpr, tpr, \_ = roc\_curve(y\_true, y\_pred)

# obtain ROC AUC

roc\_auc = auc(fpr, tpr)

# print score

print(f"ROC AUC: {roc\_auc:.3f}")

# plot ROC curve

plt.plot(fpr, tpr, color="blue", lw=2, label='ROC curve (area = {f:.2f})'.format(d=1, f=roc\_auc))

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curves')

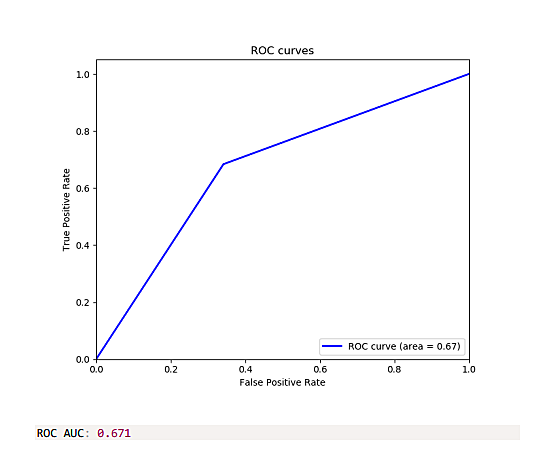
plt.legend(loc="lower right")

plt.show()

plot\_roc\_auc(y\_test, y\_pred)

Copy

**Output:**



**8.TESTING**

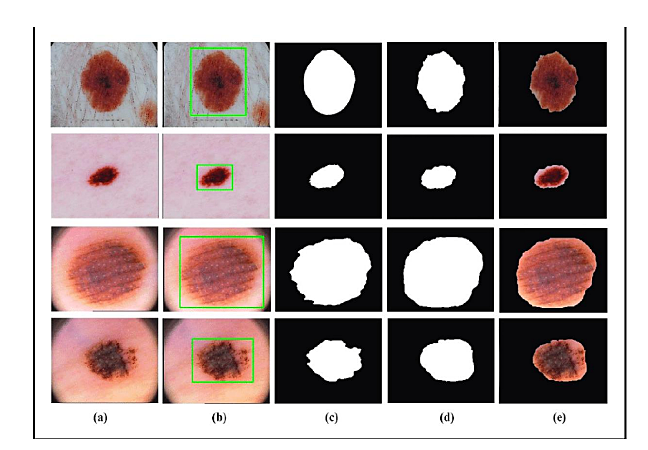
**8.1.Test Cases**

**No specific laboratory tests are indicated to make the diagnosis of erythema multiforme (EM)**, which should be arrived at clinically. The clinical picture can guide laboratory testing in severe cases. Cultures are indicated in severe cases and should be obtained from blood, sputum, and mucosal lesions.

**9. RESULTS**

**9.1.Performance Metrics**

The final results are based on the accuracy results in the form of the melanoma and the non-melanoma skin diseases classifications.



**10.ADVANTAGES & DISADVANTAGES**

**Advantages**

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

**Disadvantages**

Network Connectivity and Accuracy

**11.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 pre-processing, self-supervised learning, transfer learning, and special CNN architecture techniques. Furthermore, 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 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.

**12.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 18 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.

**13.APPENDIX**

**Githublink:[https://github.com/IBM-EPBL/IBM-Project-40490 1660630255](https://github.com/IBM-EPBL/IBM-Project-40490-1660630255)**