



Skin Health Monitoring: Detection and Early Intervention for skin diseases

Supervisor:

Ms.Thamali Dassanayake

This is our
team

Co - Supervisor :

Ms.Karthiga Rajendran

External Supervisor:

Dr. Anushan Kailainathan

STUDENT NAME	REGISTRATION NUMBER
IT21010194	LAKSHANA.K
IT21150098	RUSHANTH.B

CONTENT



- BACKGROUND
- INTRODUCTION
- RESEARCH PROBLEM
- RESEARCH QUESTION
- RESEARCH GAP
- OBJECTIVES
- SYSTEM DIAGRAM

BACKGROUND

Skin tone diversity enriches human culture but complicates diagnosis and management of skin issues.

Visual inspection and subjective assessments dominate current diagnostic approaches, leading to misdiagnosis and inappropriate treatment.

Access to professional dermatological care is limited, necessitating accessible home remedies.

INTRODUCTION

Skin issues affect millions worldwide, yet accurate diagnosis based on skin tone remains challenging.

Current diagnostic methods lack precision and accessibility, hindering effective treatment.

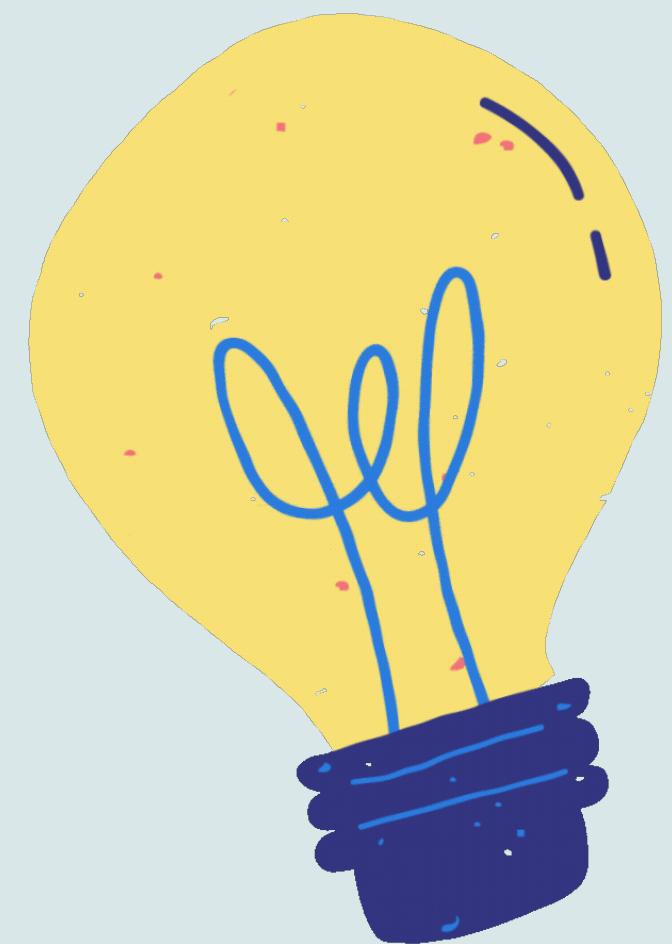
Clear guidance on affordable and practical home remedies is also lacking.

RESEARCH PROBLEM

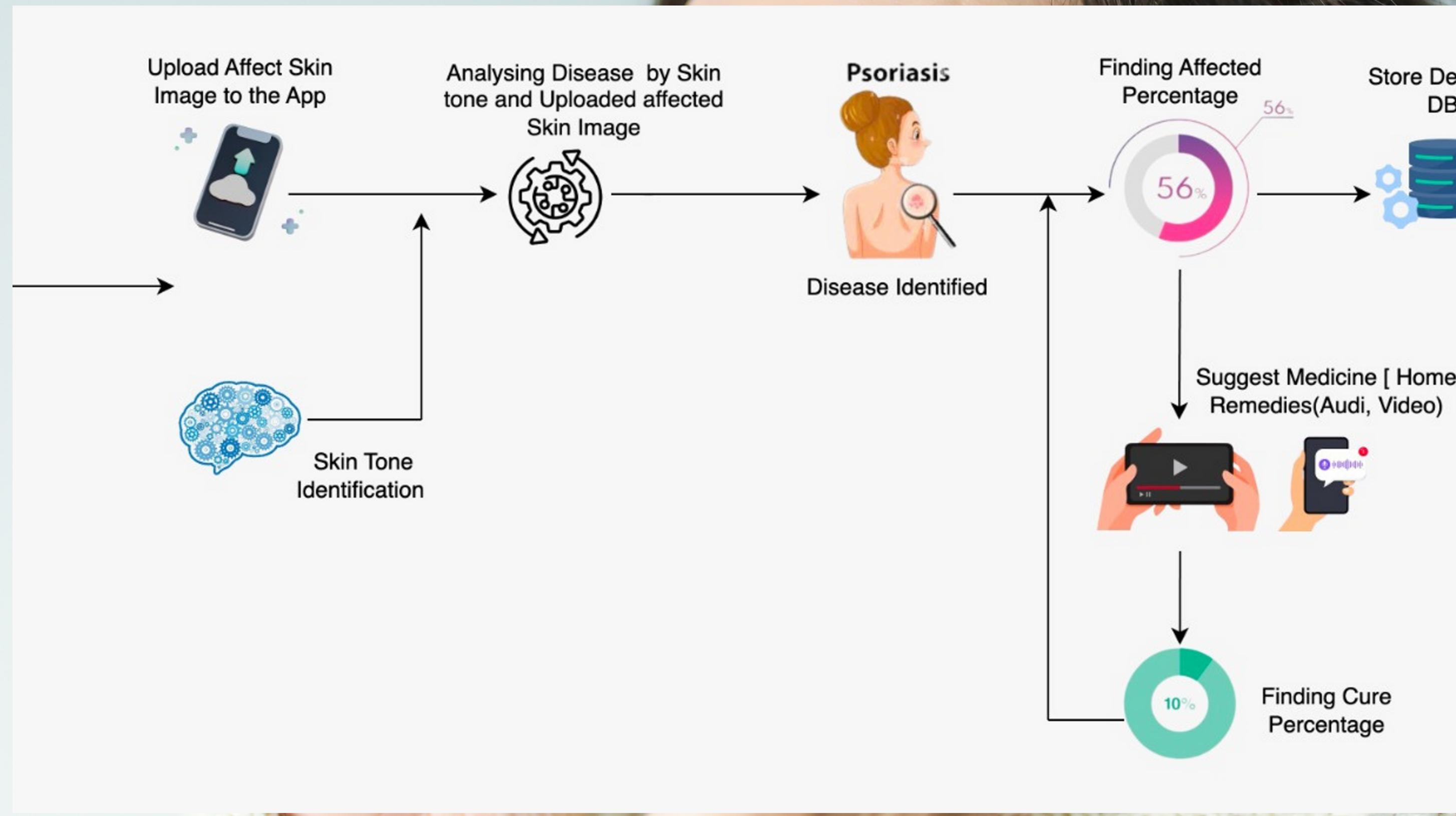
- To help with skin problems, we need better ways to figure out what type of skin issue someone has based on their skin tone and color.
- Right now, it's hard to find easy and accurate ways to do this, making it tough to get the right treatment.
- People also don't have clear advice on how to take care of their skin issues, especially when they want affordable and easy solutions at home.

OBJECTIVES

Identifies the disease based on the skin tone of the person and it gives the stage of the disease and provides a solution suitable for the disease



SYSTEM DIAGRAM



RESEARCH GAP

While there is a growing need for accessible and accurate methods to diagnose skin issues based on skin tone, existing approaches are often limited in their effectiveness and accessibility. Additionally, there is a lack of clear guidance on affordable and easy-to-use home remedies for managing various skin problems.

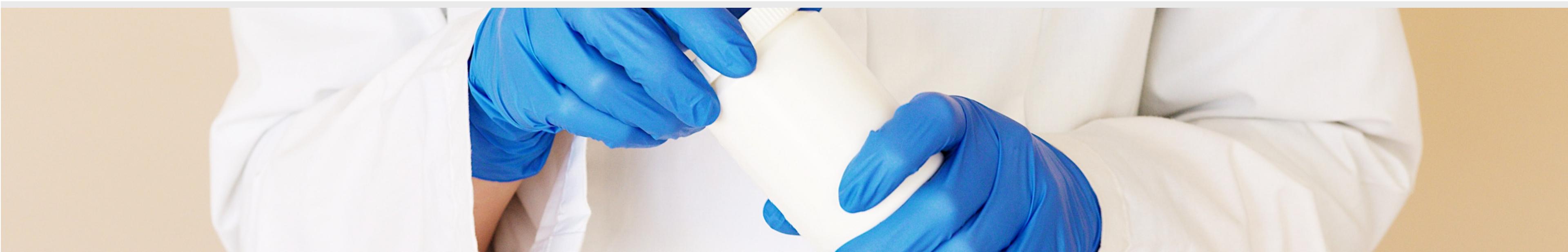


RESEARCH QUESTION

What methods can be developed to accurately identify different types of skin issues based on an individual's skin tone and color, considering the limitations in current diagnostic approaches?



Predicting the skin disease based on the skin tone using the image uploaded & Web Scraping



INTRODUCTION



Identifying skin diseases based on skin tone can be intricate due to the broad spectrum of pigmentation among individuals. While melanin levels influence skin tone, using it as the sole criterion for diagnosis may lead to inaccuracies. Therefore, a holistic approach integrating various factors such as symptoms, medical history, and diagnostic tests is imperative for precise identification and tailored treatment of skin conditions across diverse skin tones.

Research Problems



- Limited Awareness and Self-Treatment Practices
- Subjectivity in Traditional Diagnostic Approaches
- Inadequate Predictive Capabilities
- Insufficient Public Health Insights





Research Gap

- Holistic Predictive Models
- User-Friendly Integration
- In-depth Evaluation and Validation
- Public Health Impact Assessment

OBJECTIVES

Main Objective

Identifying skin disease of different skin tones based on the image uploaded.

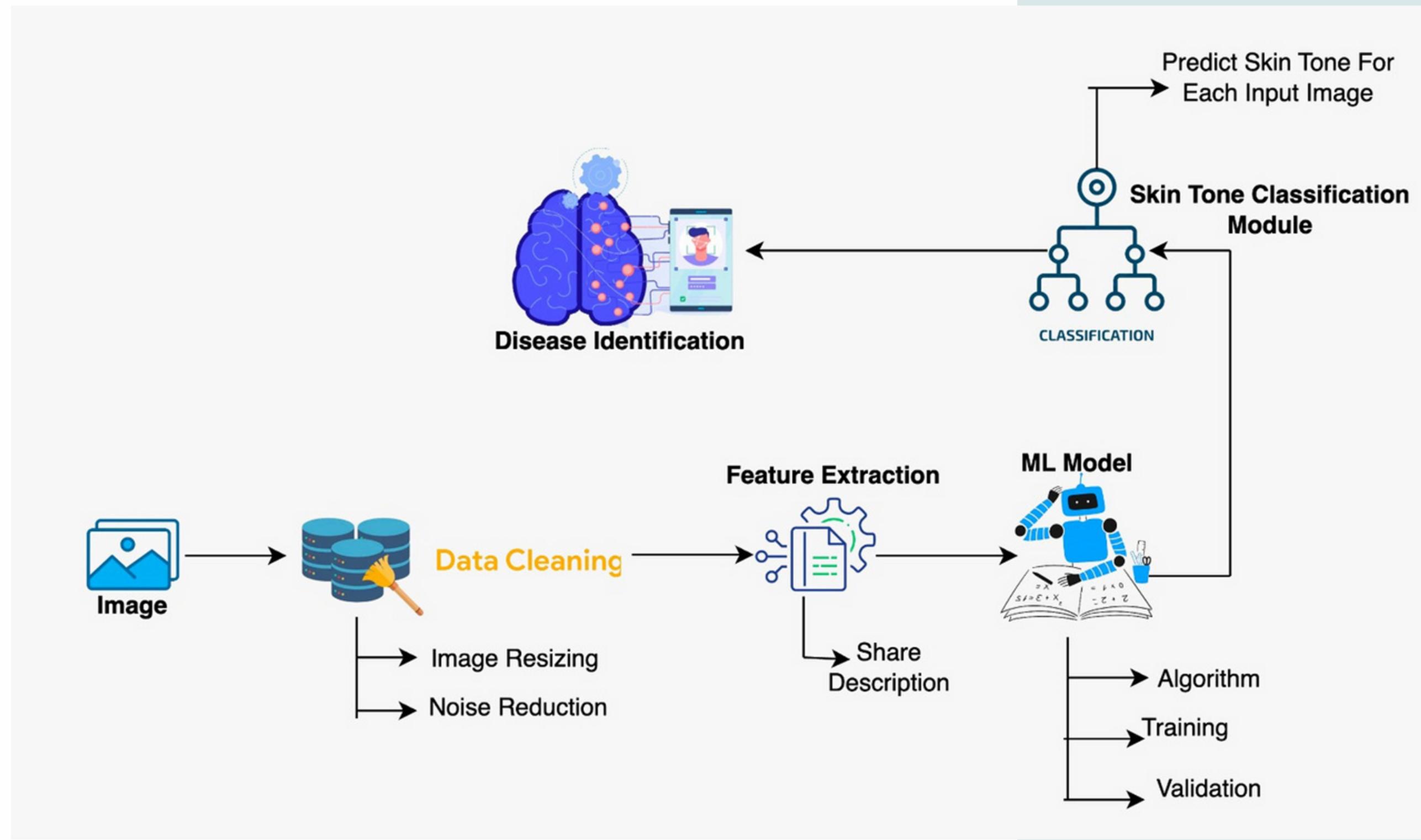
Sub Objectives

- Predict User's Skin Tone
- Image-Based Skin Disease Identification
- Duration Prediction for Identified Skin Diseases
- Web Scraping for Public Awareness

METHODOLOGY

- INDIVIDUAL SYSTEM ARCHITECTURE
- TOOLS AND TECHNOLOGIES
- REQUIREMENTS
- CURRENT PROGRESS
- NEXT EXPECTED PROGRESS

INDIVIDUAL SYSTEM ARCHITECTURE





TOOLS & TECHNOLOGY

Language : Python

Frontend : Flutter

Libraries : Tensorflow, CNN, Seaborn, Matplotlib, Keras

REQUIREMENTS

- Functional Requirements : Extract images from scanned images
- Non Functional Requirements : Accuracy, Speed, Scalability, Reliability
- System Requirements : Any IOS or Android devices with 2GB of RAM, Camera, and internet connectivity.
- User Requirements : Simple interface, Easy to operate

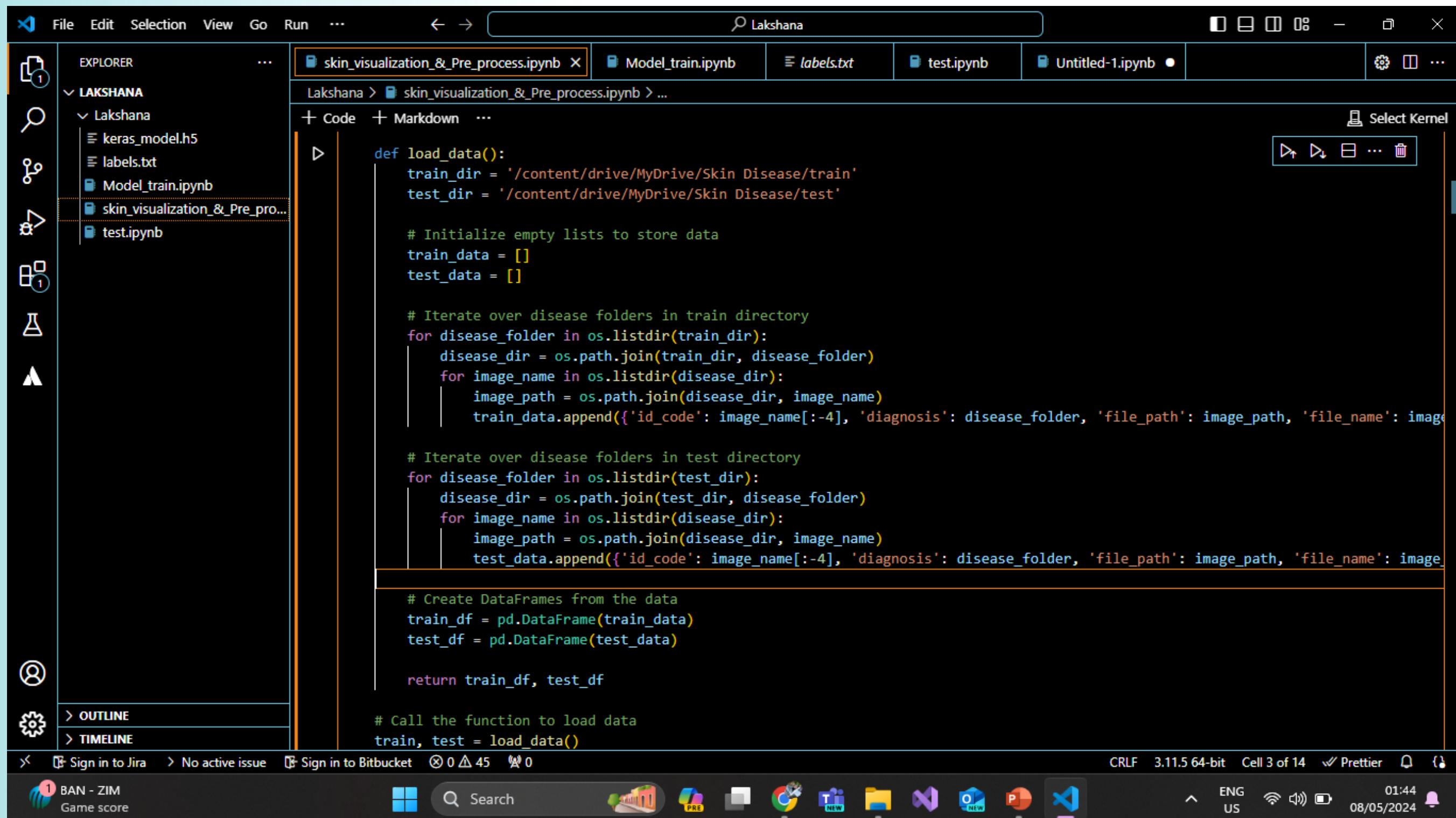
CURRENT PROGRESS

- Acquired a dataset to ensure the model produces precise predictions
- Utilizing the image processing algorithms, I've constructed a prediction model
- API Deployment
- Implemented mobile application and integrated frontend and backend

NEXT EXPECTED PROGRESS

- Testing the application
- Bug fixing
- Enhancing the UI of the application

DATASET PREVIEW



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** Lakshana
- Toolbar:** Back, Forward, Select Kernel, etc.
- Explorer:** Shows a folder structure under LAKSHANA named Lakshana, containing keras_model.h5, labels.txt, Model_train.ipynb, skin_visualization_&_Pre_process.ipynb, and test.ipynb. The skin_visualization_&_Pre_process.ipynb file is selected.
- Code Cell:** Contains Python code for dataset loading:

```
def load_data():
    train_dir = '/content/drive/MyDrive/Skin Disease/train'
    test_dir = '/content/drive/MyDrive/Skin Disease/test'

    # Initialize empty lists to store data
    train_data = []
    test_data = []

    # Iterate over disease folders in train directory
    for disease_folder in os.listdir(train_dir):
        disease_dir = os.path.join(train_dir, disease_folder)
        for image_name in os.listdir(disease_dir):
            image_path = os.path.join(disease_dir, image_name)
            train_data.append({'id_code': image_name[:-4], 'diagnosis': disease_folder, 'file_path': image_path, 'file_name': image_name})

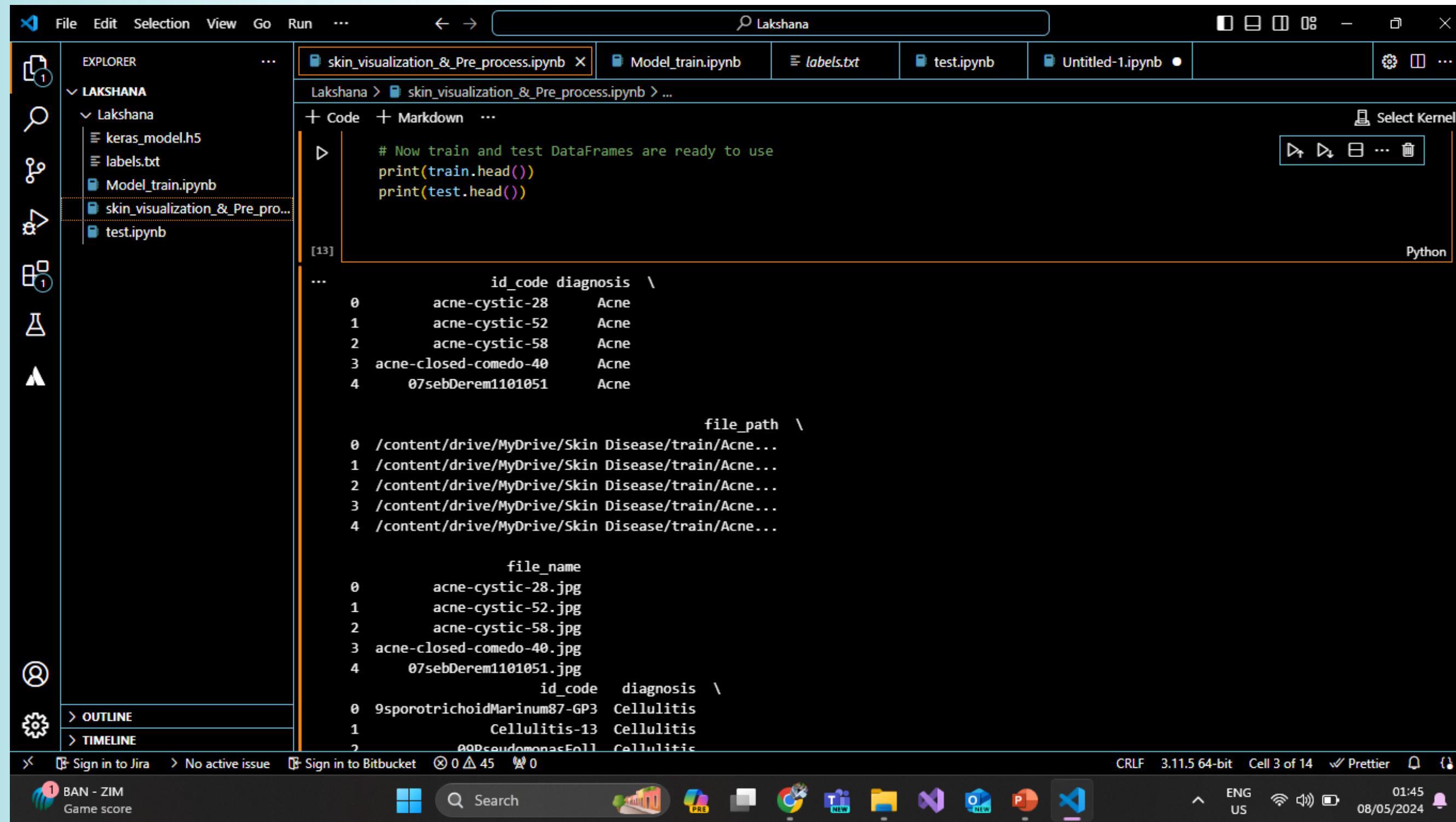
    # Iterate over disease folders in test directory
    for disease_folder in os.listdir(test_dir):
        disease_dir = os.path.join(test_dir, disease_folder)
        for image_name in os.listdir(disease_dir):
            image_path = os.path.join(disease_dir, image_name)
            test_data.append({'id_code': image_name[:-4], 'diagnosis': disease_folder, 'file_path': image_path, 'file_name': image_name})

    # Create DataFrames from the data
    train_df = pd.DataFrame(train_data)
    test_df = pd.DataFrame(test_data)

    return train_df, test_df

# Call the function to load data
train, test = load_data()
```
- Bottom Status Bar:** CRLF, 3.11.5 64-bit, Cell 3 of 14, Prettier, etc.
- Taskbar:** BAN - ZIM Game score, Search, File Explorer, Google Chrome, Microsoft Teams, Microsoft Word, Microsoft Powerpoint, Microsoft Visual Studio Code, etc.
- System Tray:** ENG US, 01:44, 08/05/2024, etc.

DATASET PREVIEW



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** Lakshana
- Toolbar:** Select Kernel, Python
- Left Sidebar (EXPLORER):** Shows a file tree under LAKSHANA/Lakshana, including keras_model.h5, labels.txt, Model_train.ipynb, skin_visualization_&_Pre_process.ipynb, and test.ipynb.
- Code Cell:** Contains Python code for previewing datasets.

```
# Now train and test DataFrames are ready to use
print(train.head())
print(test.head())

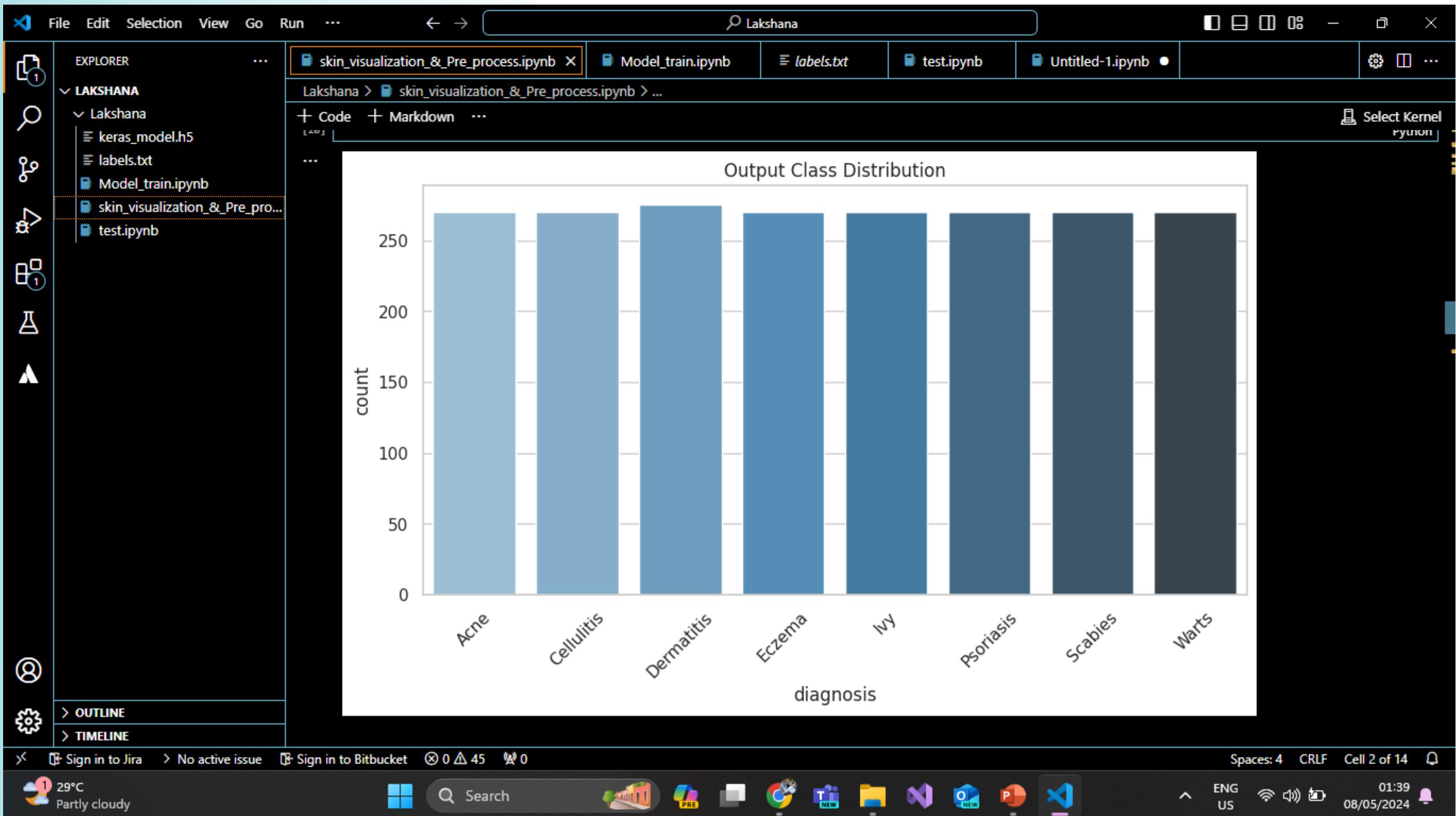
...
id_code diagnosis \
0    acne-cystic-28    Acne
1    acne-cystic-52    Acne
2    acne-cystic-58    Acne
3  acne-closed-comedo-40    Acne
4    07sebDerem1101051    Acne

file_path \
0 /content/drive/MyDrive/Skin Disease/train/Acne...
1 /content/drive/MyDrive/Skin Disease/train/Acne...
2 /content/drive/MyDrive/Skin Disease/train/Acne...
3 /content/drive/MyDrive/Skin Disease/train/Acne...
4 /content/drive/MyDrive/Skin Disease/train/Acne...

file_name
0    acne-cystic-28.jpg
1    acne-cystic-52.jpg
2    acne-cystic-58.jpg
3  acne-closed-comedo-40.jpg
4    07sebDerem1101051.jpg

id_code diagnosis \
0  9sporotrichoidMarinum87-GP3  Cellulitis
1          Cellulitis-13  Cellulitis
2  aQspseudomonasFall  Cellulitis
```
- Bottom Status Bar:** CRLF, 3.11.5 64-bit, Cell 3 of 14, Prettier, 01:45, ENG US, 08/05/2024

COMPLETION OF CODE

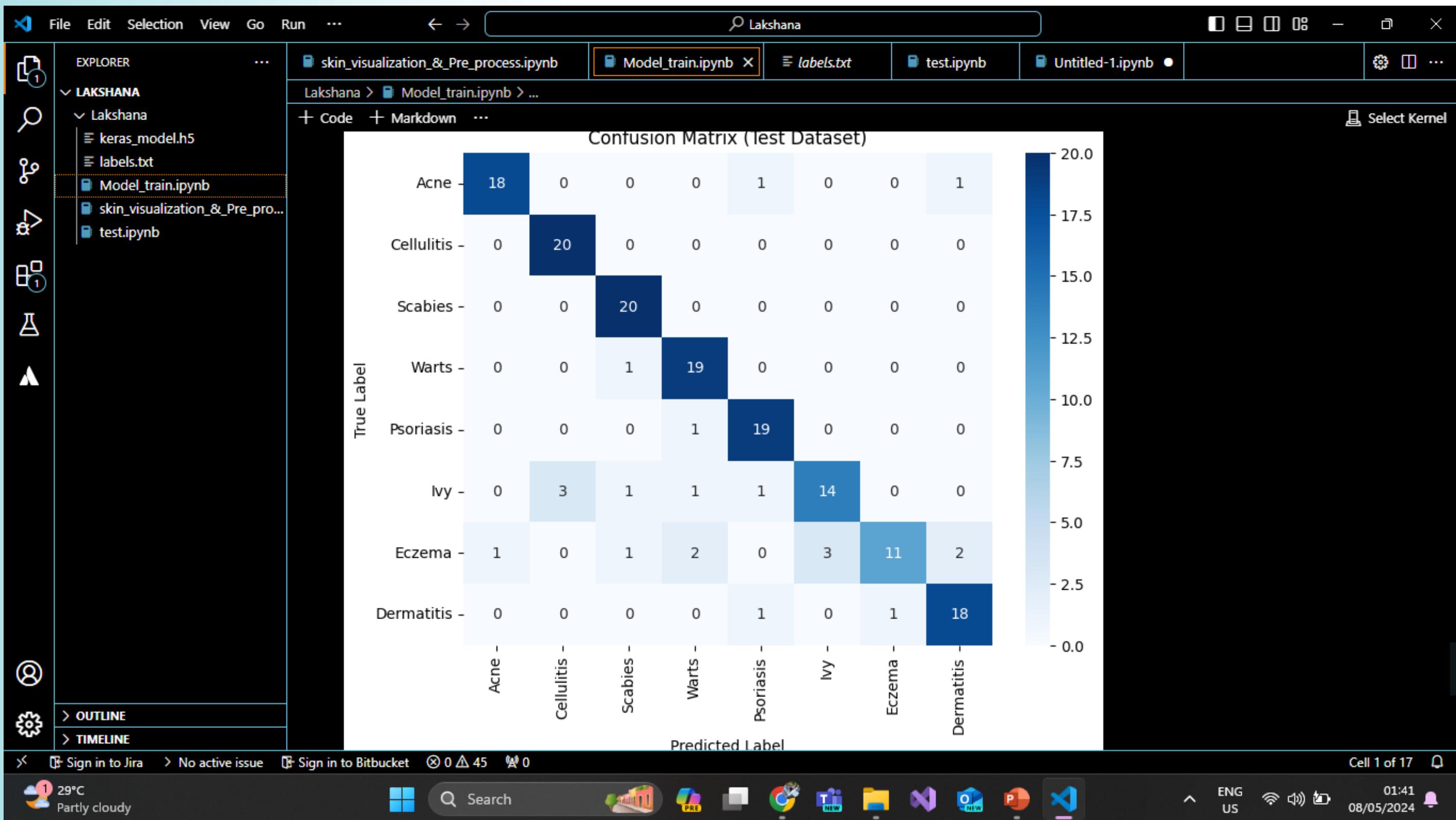


COMPLETION OF CODE

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** Lakshana
- Toolbar:** Back, Forward, Refresh, Stop, Minimize, Maximize, Close, Select Kernel
- Explorer:** Shows a tree view of the project structure under 'LAKSHANA' and 'Lakshana'. Files listed include keras_model.h5, labels.txt, Model_train.ipynb, skin_visualization_&_Pre_process.ipynb, test.ipynb.
- Code Cell:** Contains Python code for training a CNN model. It includes comments for loss computation, accuracy calculation, and printing statistics every 1000 batches. The output shows training accuracy for 10 epochs from 1 to 10.
- Output Cell:** Displays the printed output of the code, showing training accuracy for each epoch: Epoch 1, Training Accuracy: 90.12%; Epoch 2, Training Accuracy: 92.47%; Epoch 3, Training Accuracy: 93.07%; Epoch 4, Training Accuracy: 95.36%; Epoch 5, Training Accuracy: 96.26%; Epoch 6, Training Accuracy: 97.11%; Epoch 7, Training Accuracy: 96.71%; Epoch 8, Training Accuracy: 97.06%; Epoch 9, Training Accuracy: 96.26%; Epoch 10, Training Accuracy: 94.71%.
- Bottom Bar:** Sign in to Jira, Sign in to Bitbucket, 0 △ 45 ⌂ 0, Cell 1 of 17, Weather icon (29°C, Partly cloudy), Search bar, Taskbar icons (OneDrive, Microsoft Edge, Microsoft Teams, File Explorer, Visual Studio, Power BI, Microsoft Word), ENG US, 01:41, 08/05/2024.

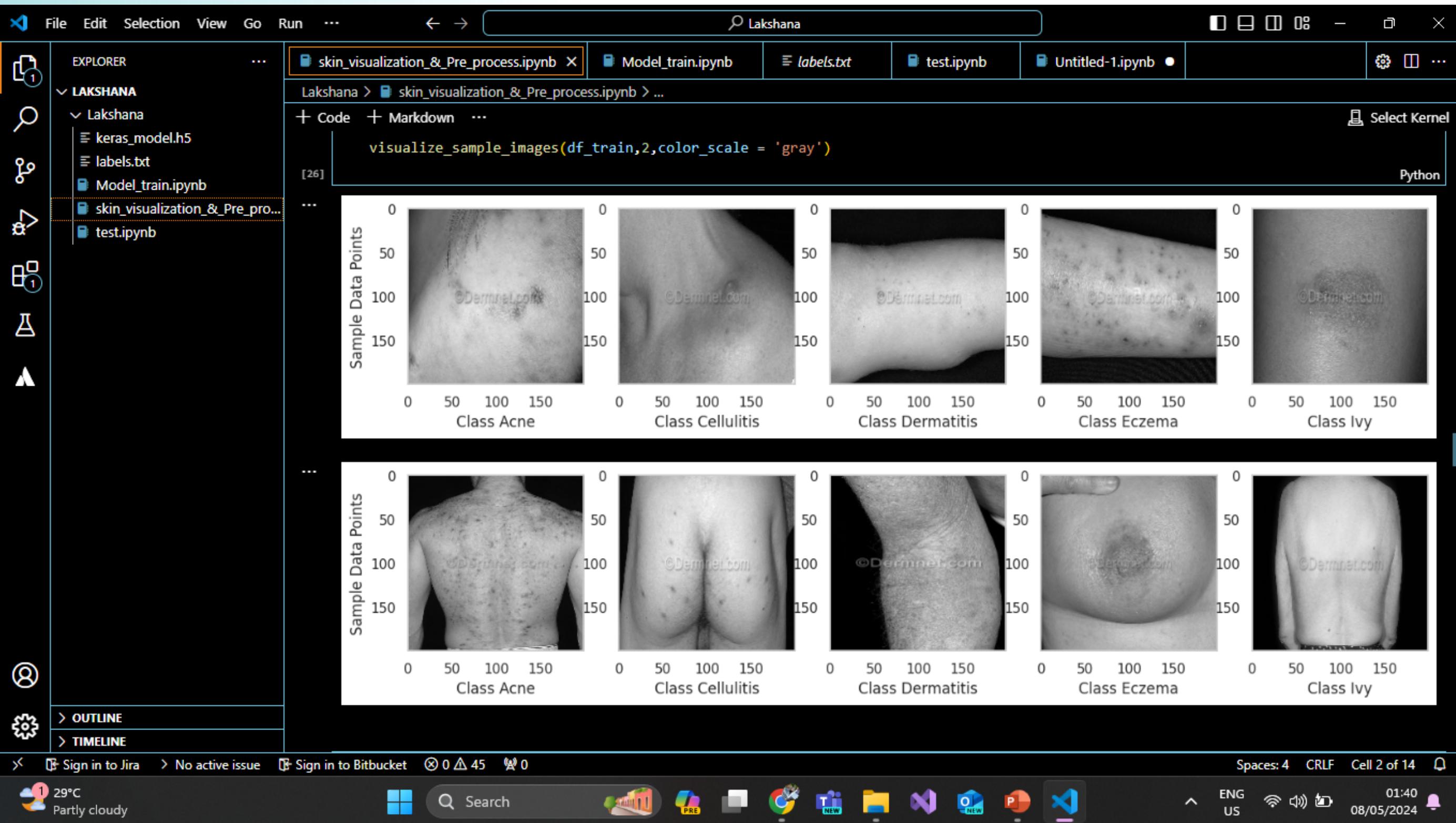
COMPLETION OF CODE



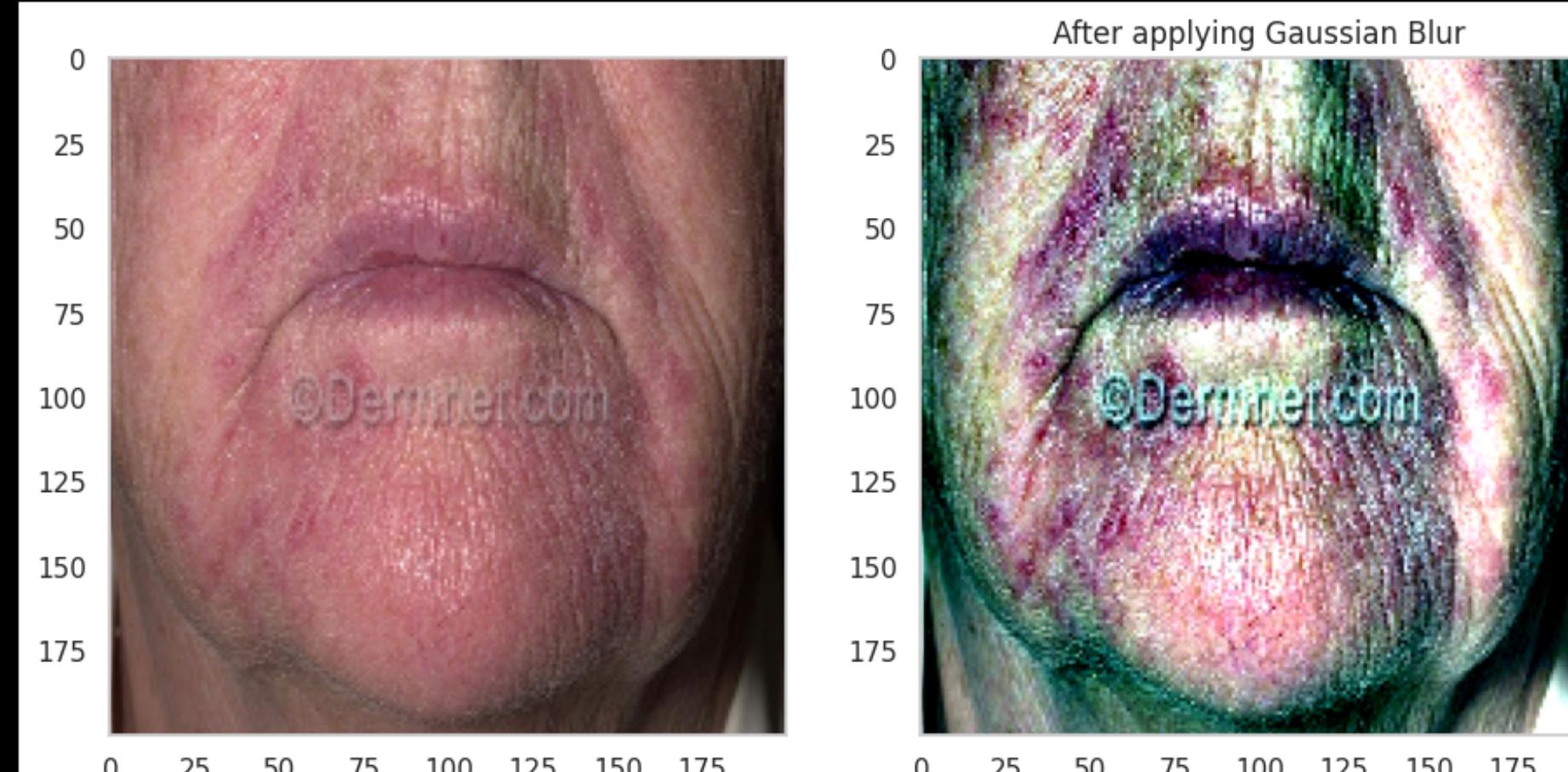
EXAMPLE OF PREDICTED IMAGES



EXAMPLE OF PREDICTED IMAGES



EXAMPLE OF PREDICTED IMAGES



Lakshana > skin_visualization_&_Pre_process.ipynb > ...

```
+ Code + Markdown ...  
axarr[0].imshow(img)  
axarr[1].imshow(img_t)  
plt.title('After applying Gaussian Blur')  
plt.show()
```

Python

File Edit Selection View Go Run ... ← → 🔍 Lakshana

EXPLORER LAKSHANA Lakshana keras_model.h5 labels.txt Model_train.ipynb skin_visualization_&_Pre_pro... test.ipynb [27]

Select Kernel

OUTLINE TIMELINE

Sign in to Jira No active issue Sign in to Bitbucket 0 △ 45 ⌂ 0 Spaces: 4 CRLF Cell 2 of 14 ENG US 01:40 08/05/2024

29°C Partly cloudy Search

Integration with Flask API



The screenshot shows a Google Colab notebook titled "SkinToneFlask.ipynb". The code in the notebook is as follows:

```
from flask import Flask, request, jsonify
from keras.models import load_model
from PIL import Image, ImageOps
import numpy as np
import io
from pyngrok import ngrok

app = Flask(__name__)

# Load the model
model = load_model('/content/drive/MyDrive/models/skin_tone/skin tone model.h5', compile=False)

# Load the labels
with open('/content/drive/MyDrive/models/skin_tone/skin tone labels.txt', 'r') as file:
    class_names = file.readlines()

# Define a function to preprocess image data
def preprocess_image(image_data):
    # Load image
    image = Image.open(io.BytesIO(image_data)).convert("RGB")
    # Resize and crop the image
    size = (224, 224)
    image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)
    # Convert image to a numpy array
    image_array = np.asarray(image)
    # Normalize the image
    normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1
    # Create the array with the right shape
    data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
```

Integration with NGROK

The screenshot shows a Google Colab notebook titled "SkinToneFlask.ipynb". The code cell contains Python code for a Flask application. It uses the `ngrok` library to expose a local Flask server on port 5000 via a public URL. The output of the cell shows the generated ngrok tunnel and the Flask app's log messages.

```
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

return jsonify({
    'class': class_name,
    'confidence_score': float(confidence_score) # Convert to float for JSON serialization
})

if __name__ == '__main__':
    # Set ngrok auth token and create tunnel
    ngrok.set_auth_token("2fvc0xfwZfJqbwQK9SVQ8GYt8c_4zxSzzURNS2ZHuyXYbsVf")
    public_url = ngrok.connect(5000)
    print(f" * ngrok tunnel \"NgrokTunnel: \"https://9448-34-82-113-186.ngrok-free.app\" -> \"http://localhost:5000\"\" -> \"http://127.0.0.1:5000\"")

    # Run Flask app
    app.run()

* ngrok tunnel "NgrokTunnel: "https://9448-34-82-113-186.ngrok-free.app" -> "http://localhost:5000"" -> "http://127.0.0.1:5000"
* Serving Flask app '__main__'
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
1/1 [=====] - 1s 1s/step
INFO:werkzeug:127.0.0.1 - - [09/Sep/2024 05:59:03] "POST /predict HTTP/1.1" 200 -
1/1 [=====] - 0s 39ms/step
INFO:werkzeug:127.0.0.1 - - [09/Sep/2024 05:59:26] "POST /predict HTTP/1.1" 200 -
```

Testing the API

The screenshot shows a dual-pane interface. On the left is a Jupyter Notebook cell containing Python code for making a POST request to an API endpoint. On the right is a browser window showing the API's response.

Jupyter Notebook Cell Content:

```
from google.colab import files
import requests

# upload files
uploaded = files.upload()

# ngrok url
url = "https://9448-34-82-113-186.ngrok-free.app/predict"

# img path
img_pth = list(uploaded.keys())[0]

# open the img file in binary mode & send it in POST request
with open(img_pth, 'rb') as img_file:
    files = {'image': img_file}
    response = requests.post(url, files=files)

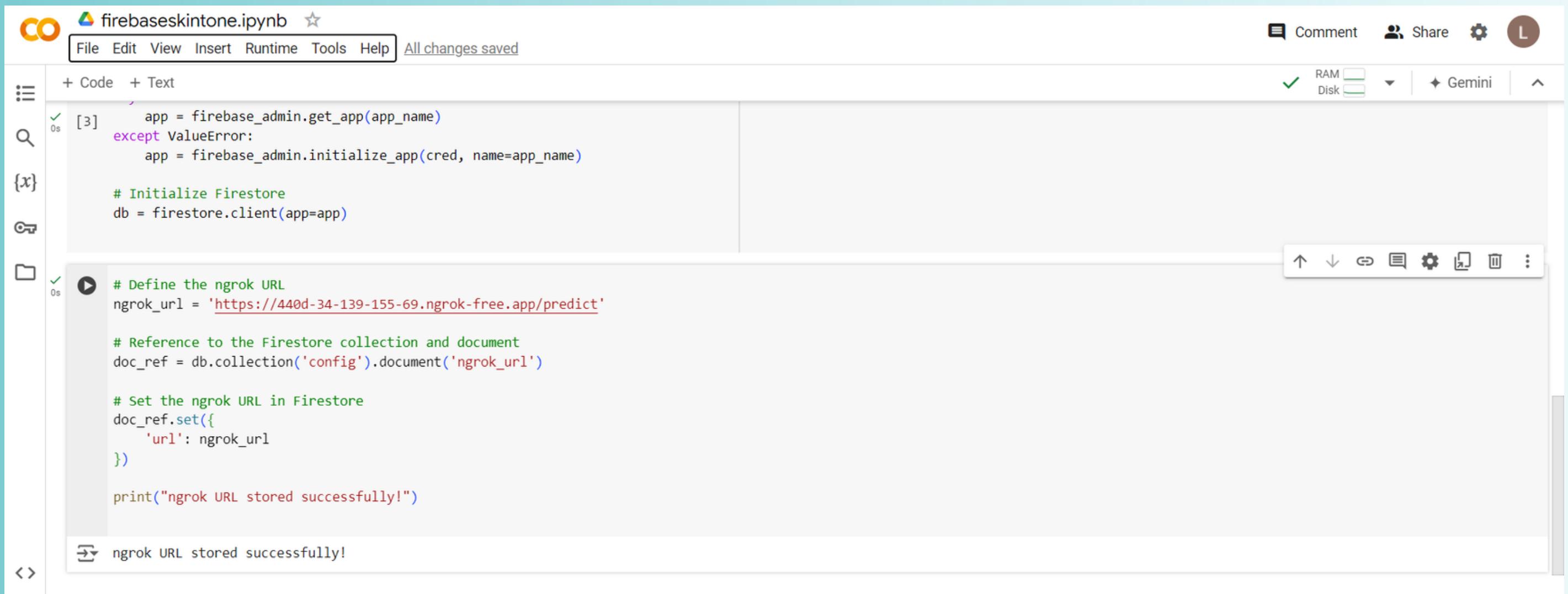
print(response.json())
```

Browser Response:

200 OK 1.77 s 57 B Just Now

```
1 {  
2   "class": "0 dark",  
3   "confidence_score": 0.9999951124191284  
4 }
```

Storing the API URL in Firebase



The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** firebaseskintone.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved
- Code Cell:** Contains Python code for interacting with Firebase Admin SDK and Firestore.
- Output Cell:** Shows the execution result: "ngrok URL stored successfully!"
- Left Sidebar:** Includes icons for Code, Text, Search, Variables, Keys, and Files.
- Top Right:** Comment, Share, Settings, and a user profile icon.
- Bottom Right:** RAM and Disk status indicators.

```
[3] app = firebase_admin.get_app(app_name)
except ValueError:
    app = firebase_admin.initialize_app(cred, name=app_name)

# Initialize Firestore
db = firestore.client(app=app)

# Define the ngrok URL
ngrok_url = 'https://440d-34-139-155-69.ngrok-free.app/predict'

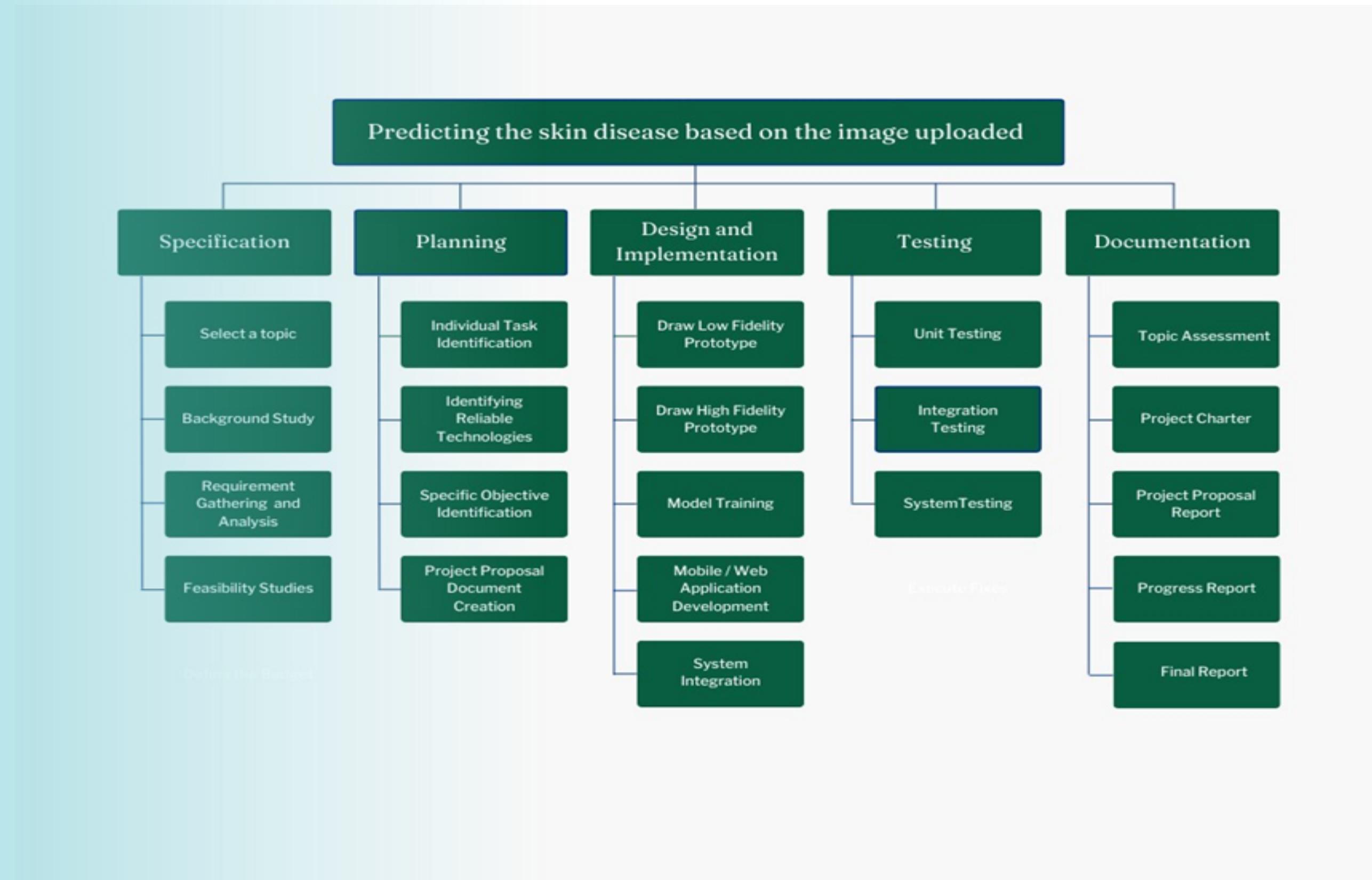
# Reference to the Firestore collection and document
doc_ref = db.collection('config').document('ngrok_url')

# Set the ngrok URL in Firestore
doc_ref.set({
    'url': ngrok_url
})

print("ngrok URL stored successfully!")

→ ngrok URL stored successfully!
```

WORK BREAKDOWN STRUCTURE



Determine the stage of the disease



1. INTRODUCTION

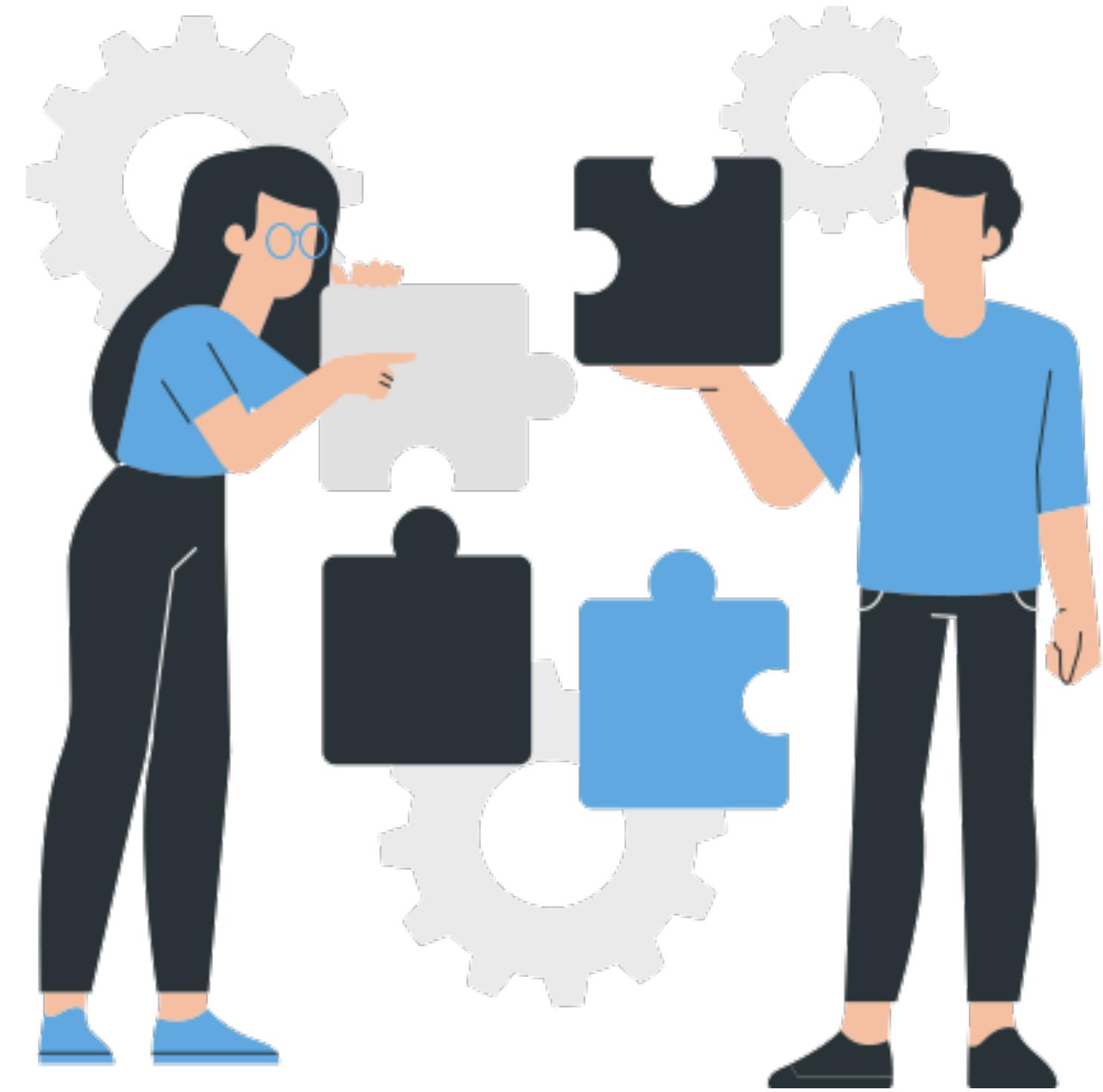
My contribution is to develop a model to do Calculation of effective percentage and cure percentage

1. Analyze and clean dataset.
2. Develop a model using the most accurate algorithms.
3. Evaluate the model through testing and outcome prediction.
4. Validate the model's effectiveness.



Research Problems

**Identify the skin diseases stages
initial & Chronic**



Research Gap

Identify the skin disease stages

- 1.initial**
- 2.Chronic**

OBJECTIVES

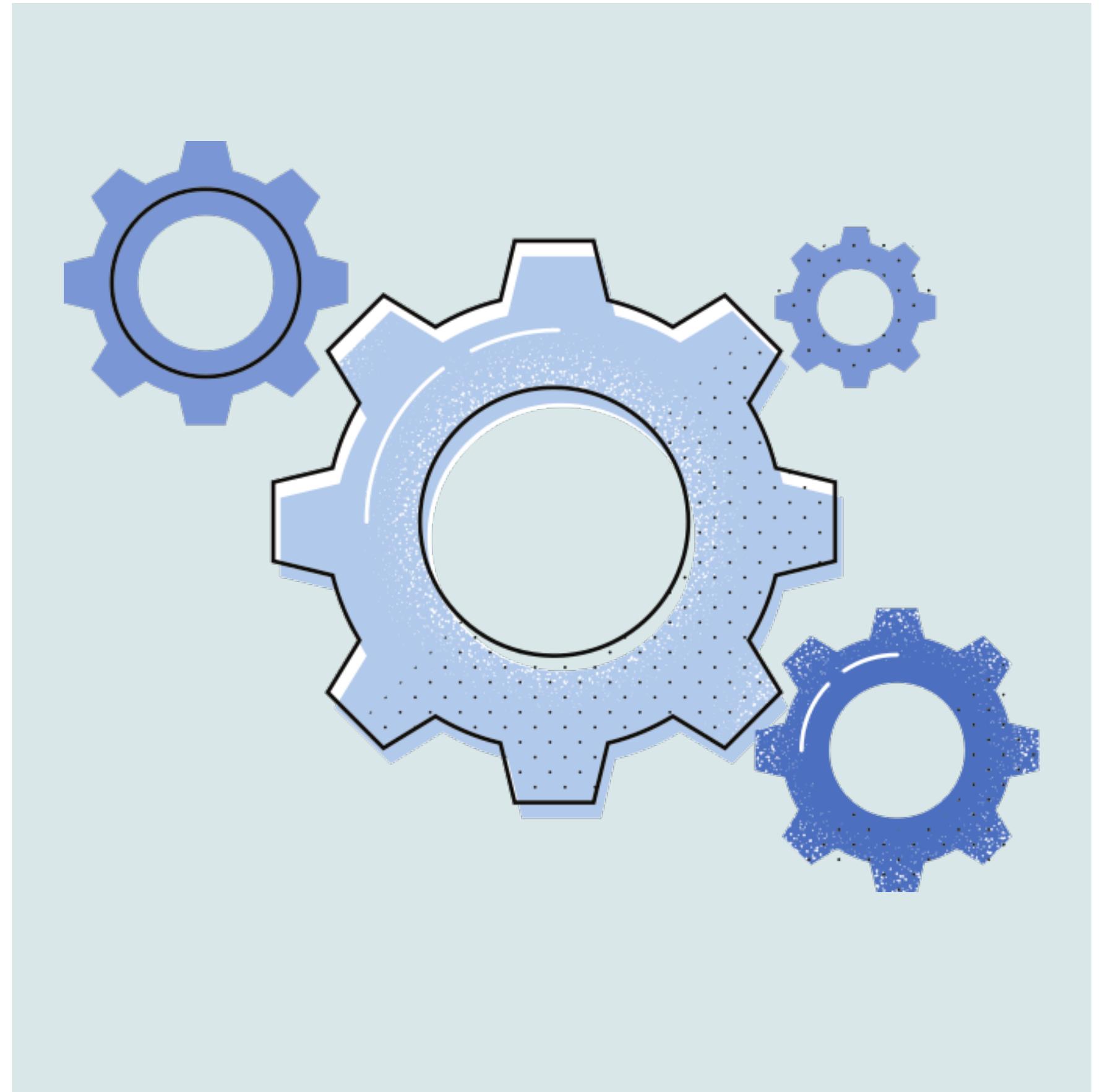
Main Objectives

Disease Identification

Stage Classification

Sub Objectives

Add validation



METHODOLOGY

Data Collection

Image Preprocessing

Machine Learning Model Training

Feature Extraction

Model Integration

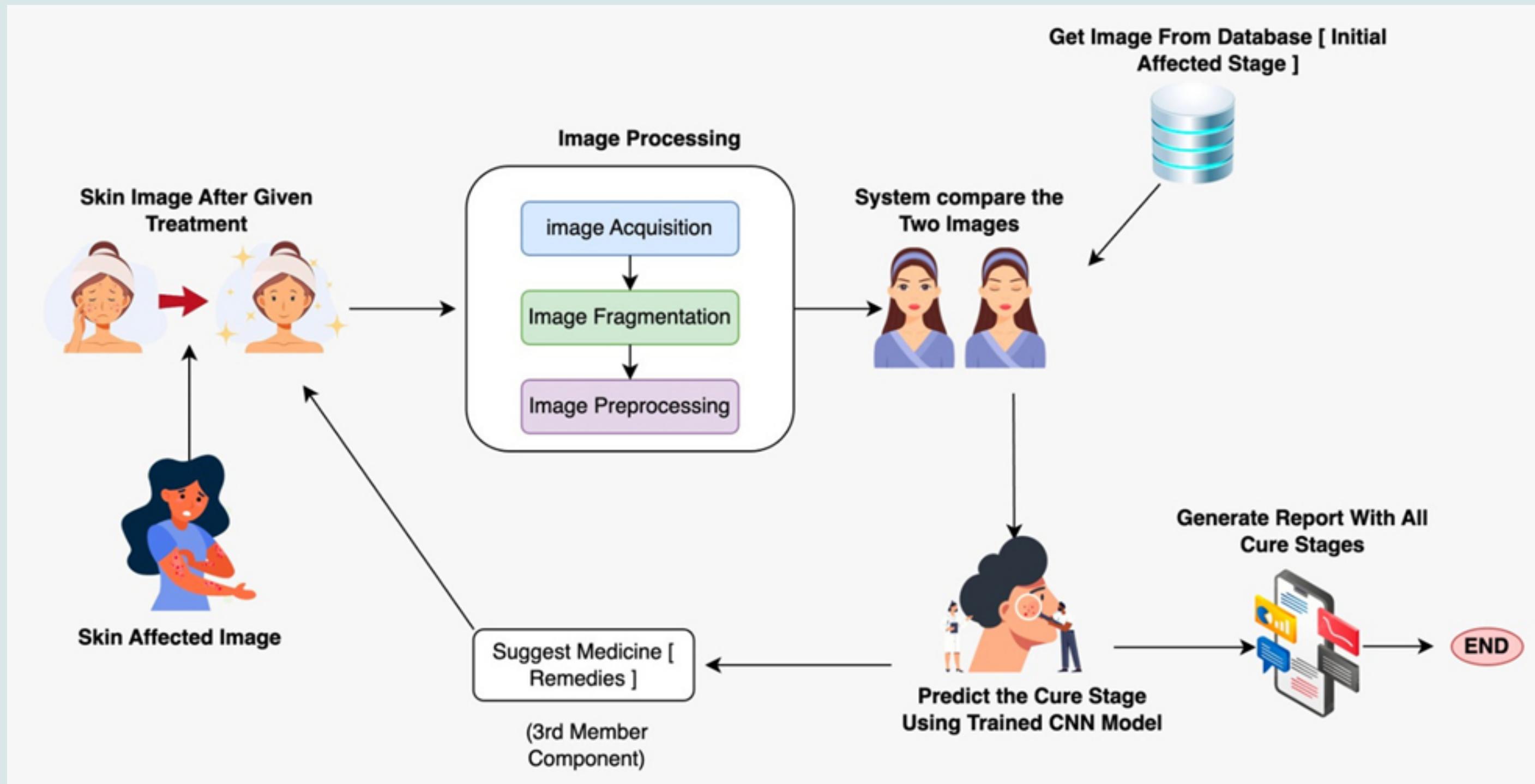
User Interface Design

Deployment and Maintenance

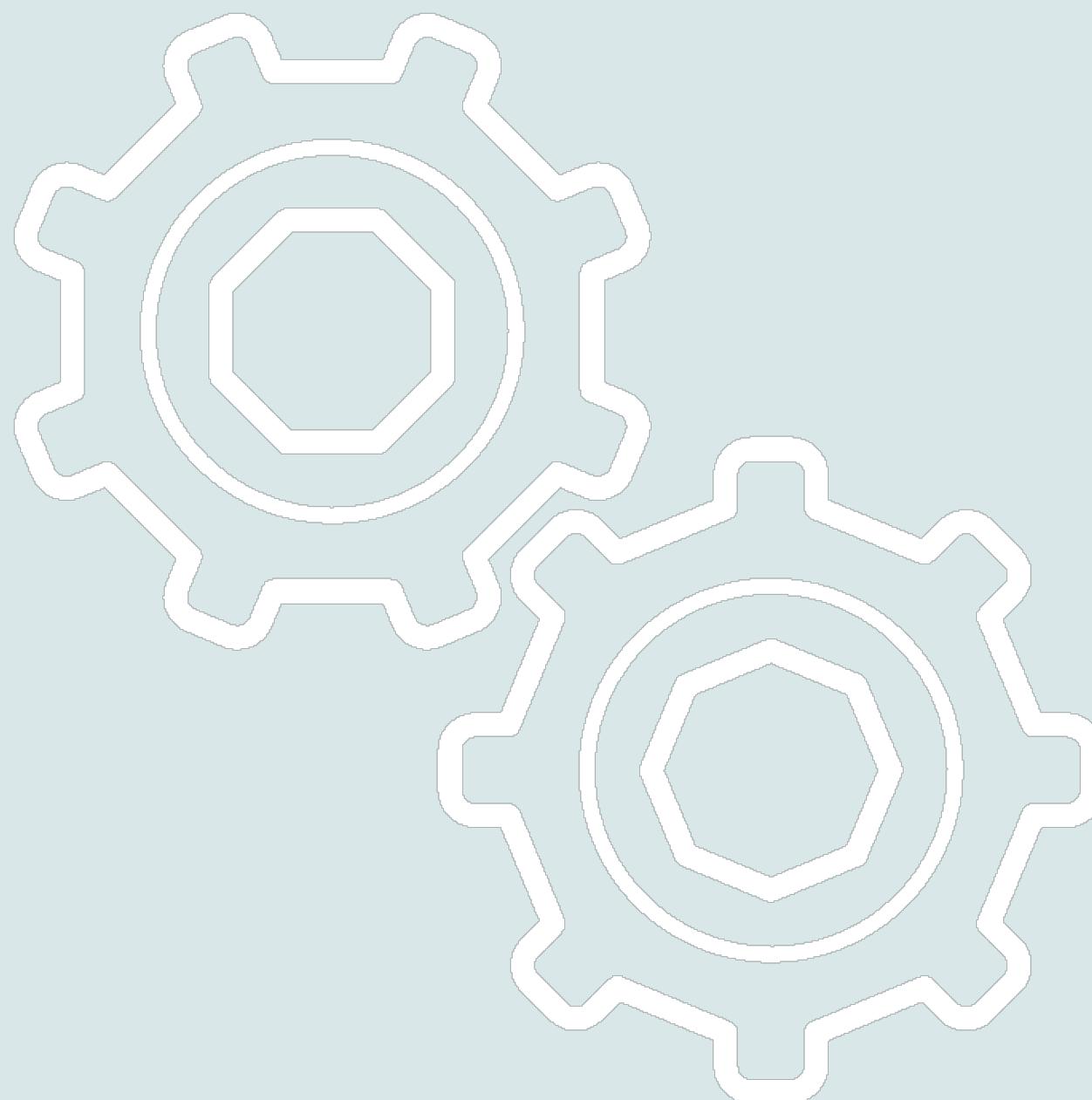
Validation and Testing



INDIVIDUAL SYSTEM ARCHITECTURE



TOOLS & TECHNOLOGY

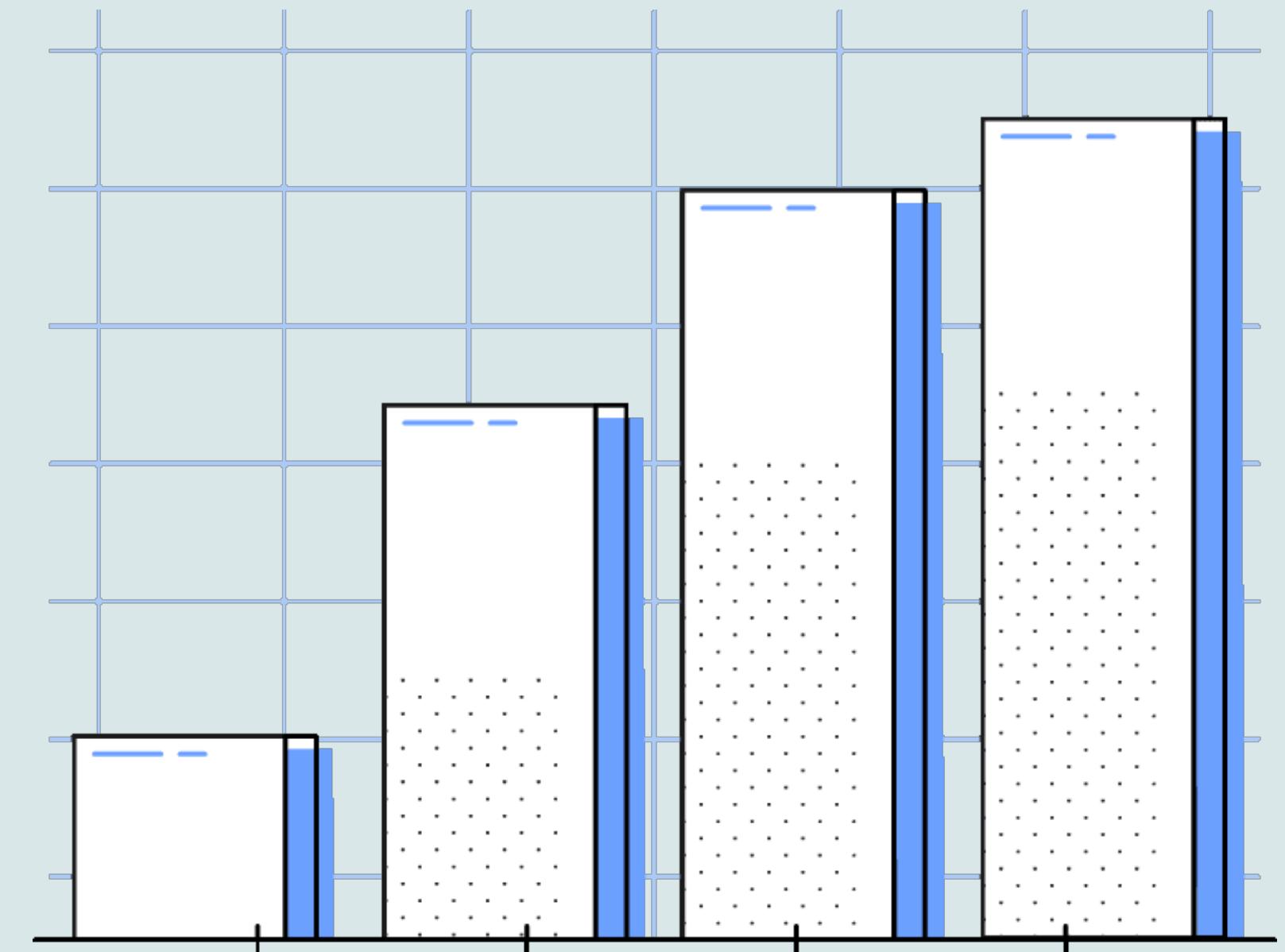


- 1.Tensorflow
- 2.Python
- 3.Numpy
- 4.Google drive
- 5.Git
- 6.Image augmentation
- 7.Flutter

CURRENT PROGRESS

1. data collection(process)

2. model training
(Dermatitis Chronic & Tenia Chronic &
Binary Validation)



NEXT EXPECTED PROGRESS

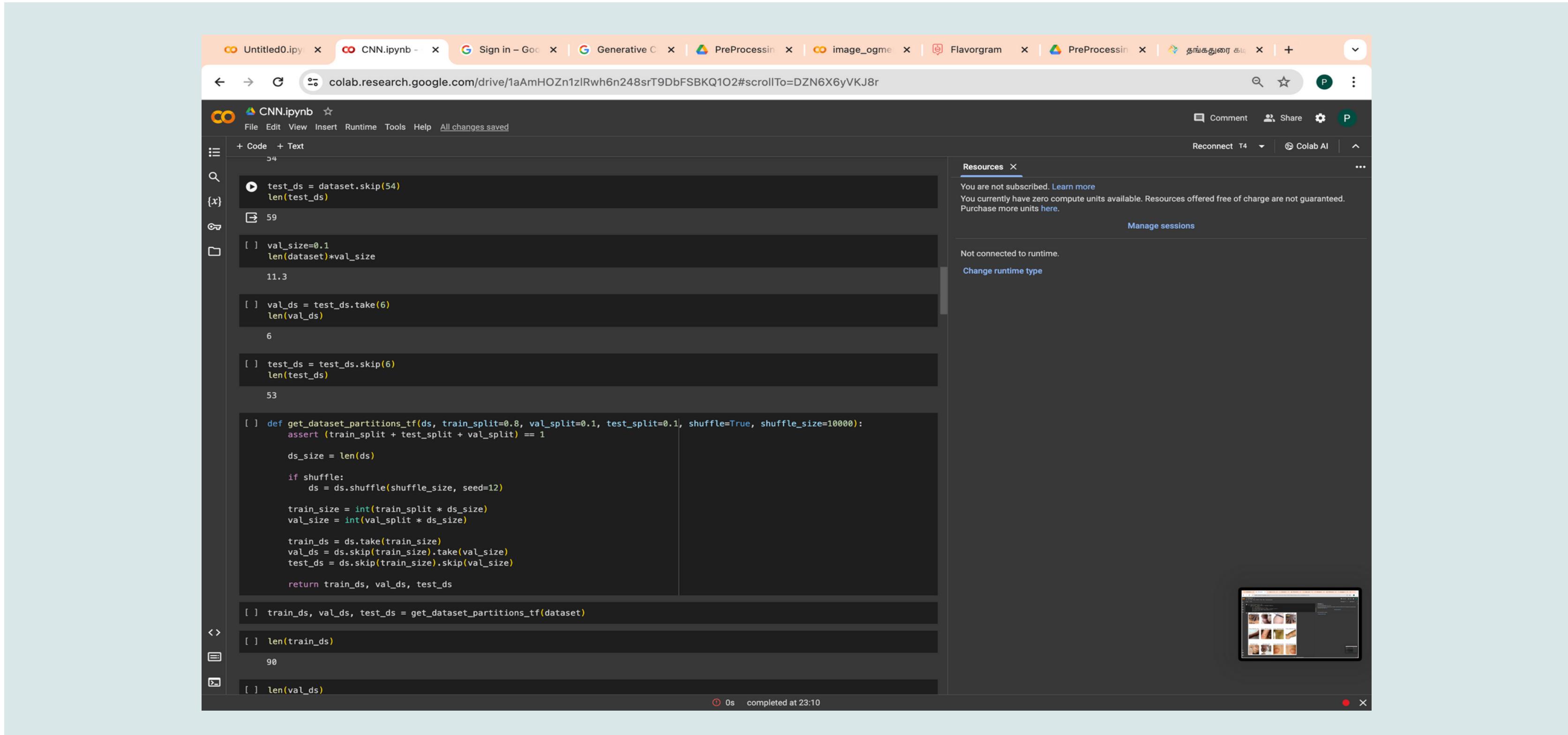
Deployment

UI design

binary validation to increase accuracy



Model Training Code



The screenshot shows a Google Colab notebook titled "CNN.ipynb". The code in the notebook is as follows:

```
test_ds = dataset.skip(54)
len(test_ds)

59

[ ] val_size=0.1
len(dataset)*val_size
11.3

[ ] val_ds = test_ds.take(6)
len(val_ds)
6

[ ] test_ds = test_ds.skip(6)
len(test_ds)
53

[ ] def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
    return train_ds, val_ds, test_ds

[ ] train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

[ ] len(train_ds)
90

[ ] len(val_ds)
```

The Resources panel indicates that the user is not subscribed and has zero compute units available. It also shows that the runtime is not connected.

Model Training Code

The screenshot shows a Google Colab notebook titled "CNN.ipynb". The code cell contains the following Python code:

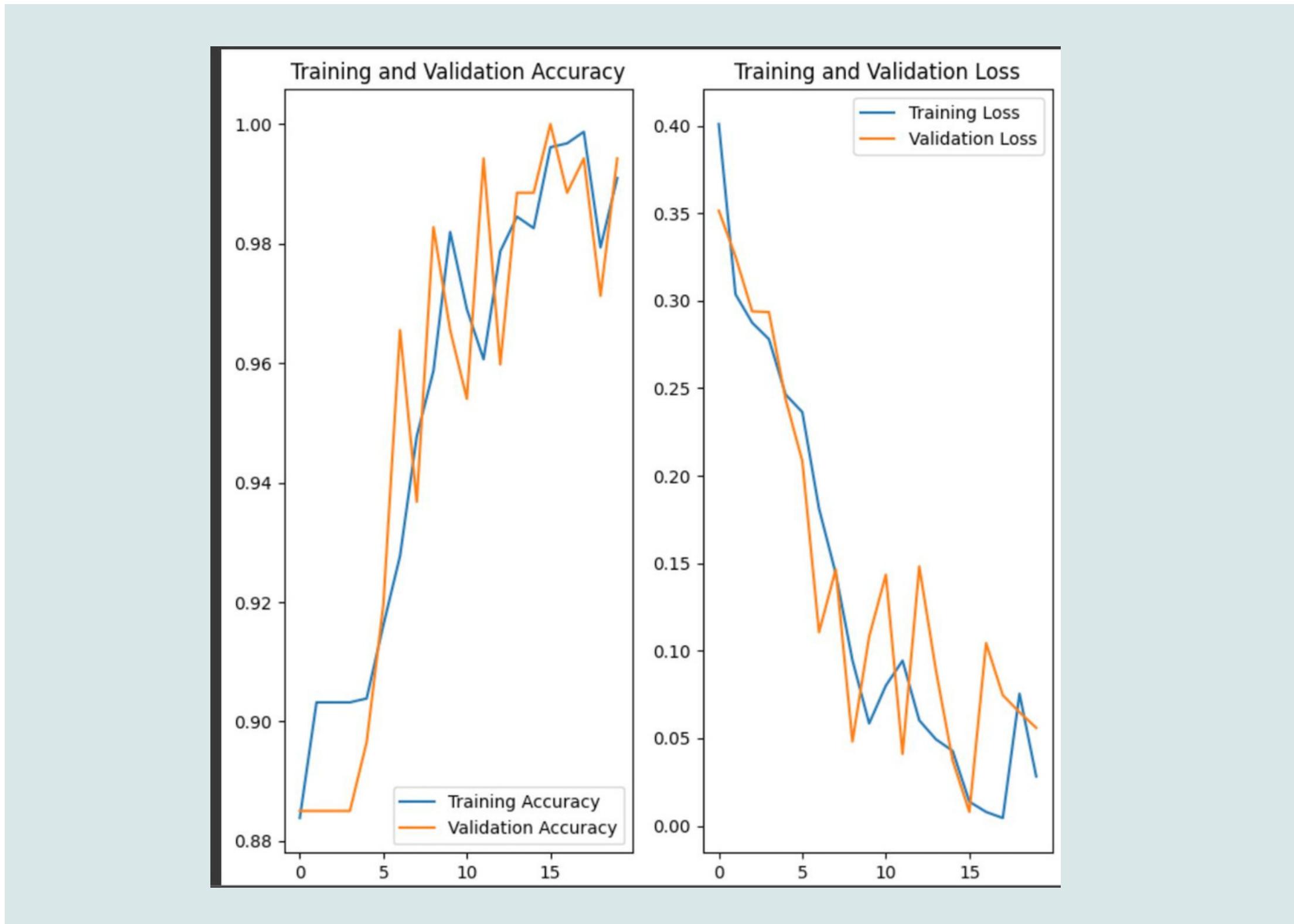
```
[ ] model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
```

The output cell displays the model's summary:

```
model.summary()
Model: "sequential_1"
+-----+
Layer (type)      Output Shape     Param #
+-----+
sequential (Sequential)  (32, 256, 256, 3)      0
conv2d (Conv2D)      (32, 254, 254, 32)     896
max_pooling2d (MaxPooling2D) (32, 127, 127, 32)  0
conv2d_1 (Conv2D)      (32, 125, 125, 64)    18496
max_pooling2d_1 (MaxPooling2D) (32, 62, 62, 64)  0
conv2d_2 (Conv2D)      (32, 60, 60, 64)    36928
max_pooling2d_2 (MaxPooling2D) (32, 30, 30, 64)  0
conv2d_3 (Conv2D)      (32, 28, 28, 64)    36928
max_pooling2d_3 (MaxPooling2D) (32, 14, 14, 64)  0
conv2d_4 (Conv2D)      (32, 12, 12, 64)    36928
max_pooling2d_4 (MaxPooling2D) (32, 6, 6, 64)   0
conv2d_5 (Conv2D)      (32, 4, 4, 64)    36928
max_pooling2d_5 (MaxPooling2D) (32, 2, 2, 64)   0
flatten (Flatten)      (32, 256)        0
dense (Dense)          (32, 64)        16448
dense_1 (Dense)        (32, 3)         195
+-----+
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
```

The Resources panel indicates that no compute units are available.

COMPLETION OF CODE



EXAMPLE OF PREDICTED IMAGES

The screenshot shows the Postman application interface. On the left, the sidebar displays collections like 'vit-sports' and various API references. The main workspace shows a POST request to 'http://localhost:9000/RT/skin-lesion'. The 'Body' tab is selected, showing a 'form-data' structure with a key 'image' pointing to a file named 'p-52-img-1.jpeg'. Below the request, the response section shows a status of '200 OK' with a JSON payload:

```
1 {  
2   "confidence": 0.9947029948234558,  
3   "result": "Tinea infection last"  
4 }
```

Image Augmentation

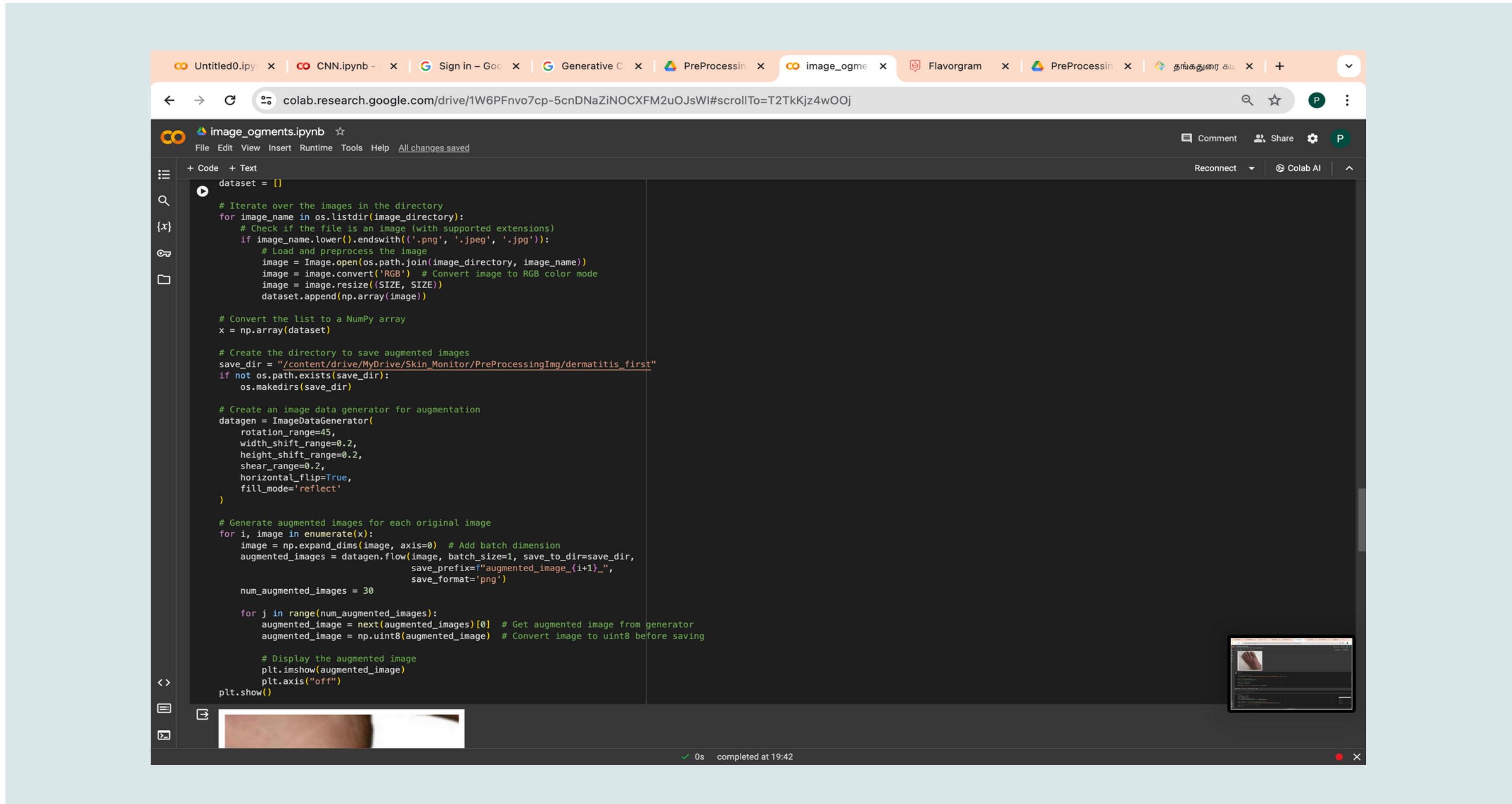
The screenshot shows a Google Colab notebook titled "CNN.ipynb". The code cell contains the following Python code:

```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```

Below the code, there are three rows of four images each, showing various skin conditions. The first row contains four images labeled "dermatitis_last". The second row contains four images labeled "Tinea_infection_first". The third row contains four images labeled "dermatitis_last". The images show different types of skin infections and dermatitis.

The Colab interface includes a "Resources" sidebar with a message about compute units and a "Not connected to runtime" message. The bottom of the screen shows the status bar with "0s completed at 23:10".

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". The code implements image augmentation on a dataset of skin images. It iterates over a directory of images, loads them, converts them to RGB, and resizes them. Then, it creates a data generator with specific augmentation parameters (rotation, width shift, height shift, shear, horizontal flip, fill mode) and generates 30 augmented versions of each original image. Finally, it displays one of the augmented images using plt.imshow().

```
dataset = []

# Iterate over the images in the directory
for image_name in os.listdir(image_directory):
    # Check if the file is an image (with supported extensions)
    if image_name.lower().endswith('.png', '.jpeg', '.jpg'):
        # Load and preprocess the image
        image = Image.open(os.path.join(image_directory, image_name))
        image = image.convert('RGB') # Convert image to RGB color mode
        image = image.resize((SIZE, SIZE))
        dataset.append(np.array(image))

# Convert the list to a NumPy array
x = np.array(dataset)

# Create the directory to save augmented images
save_dir = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/dermatitis_first"
if not os.path.exists(save_dir):
    os.makedirs(save_dir)

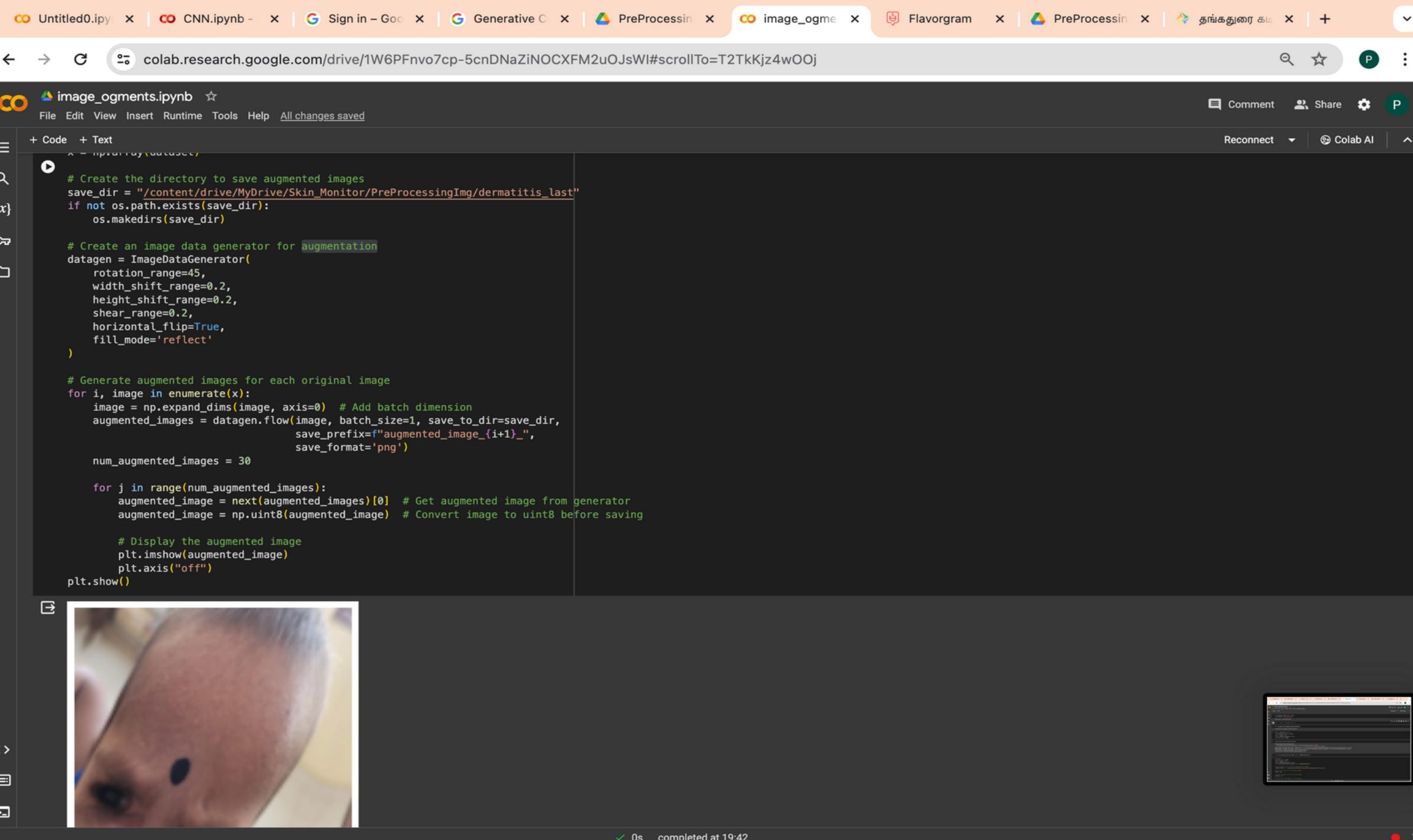
# Create an image data generator for augmentation
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode='reflect'
)

# Generate augmented images for each original image
for i, image in enumerate(x):
    image = np.expand_dims(image, axis=0) # Add batch dimension
    augmented_images = datagen.flow(image, batch_size=1, save_to_dir=save_dir,
                                    save_prefix=f"augmented_image_{i+1}_",
                                    save_format='png')
    num_augmented_images = 30

    for j in range(num_augmented_images):
        augmented_image = next(augmented_images)[0] # Get augmented image from generator
        augmented_image = np.uint8(augmented_image) # Convert image to uint8 before saving

        # Display the augmented image
        plt.imshow(augmented_image)
        plt.axis("off")
        plt.show()
```

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". The code in the cell generates augmented images from a dataset. It creates a directory to save augmented images, initializes an image data generator with specific parameters (rotation, width shift, height shift, shear, horizontal flip, fill mode), and then iterates through the original images, applying the generator to each to produce 30 augmented versions. The generated augmented image is displayed at the bottom left.

```
# Create the directory to save augmented images
save_dir = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/dermatitis_last"
if not os.path.exists(save_dir):
    os.makedirs(save_dir)

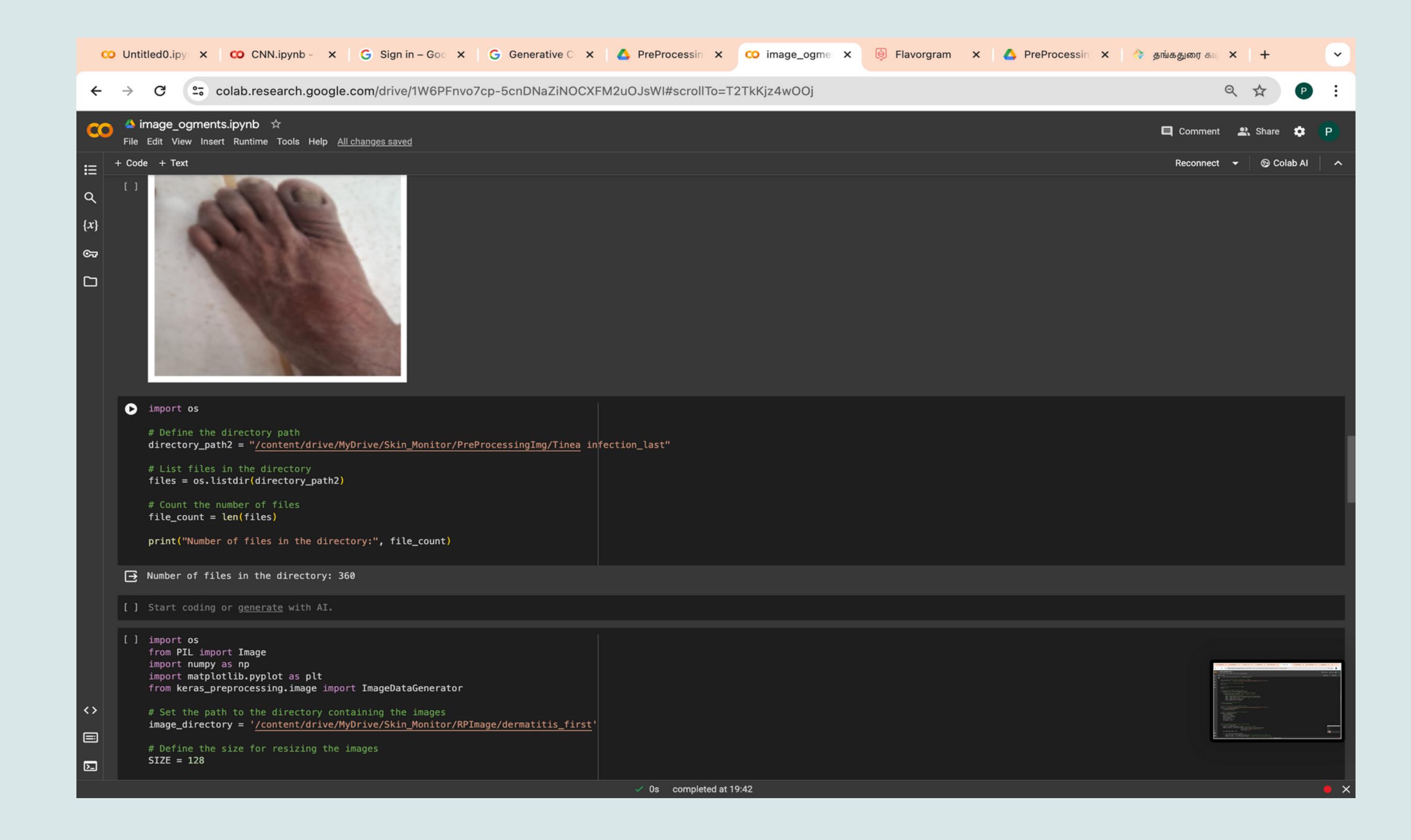
# Create an image data generator for augmentation
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True,
    fill_mode='reflect'
)

# Generate augmented images for each original image
for i, image in enumerate(x):
    image = np.expand_dims(image, axis=0) # Add batch dimension
    augmented_images = datagen.flow(image, batch_size=1, save_to_dir=save_dir,
                                    save_prefix=f"augmented_image_{i+1}_",
                                    save_format='png')
    num_augmented_images = 30

    for j in range(num_augmented_images):
        augmented_image = next(augmented_images)[0] # Get augmented image from generator
        augmented_image = np.uint8(augmented_image) # Convert image to uint8 before saving

        # Display the augmented image
        plt.imshow(augmented_image)
        plt.axis("off")
        plt.show()
```

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". At the top, there are several tabs: Untitled0.ipynb, CNN.ipynb, Sign in - Google, Generative, PreProcessin, image_ogments (which is the active tab), Flavogram, PreProcessin, and தங்கதுறை கடமை. The URL in the address bar is colab.research.google.com/drive/1W6PFnvo7cp-5cnDNaZiNOCXFMyuOJsWI#scrollTo=T2TkKjz4wOOj.

The notebook interface includes a toolbar with File, Edit, View, Insert, Runtime, Tools, Help, and a "Comment" button. On the left, there's a sidebar with icons for Code, Text, Search, and a file tree.

The main area displays a photograph of a person's foot with a skin condition. Below the image, a code cell contains Python code for listing files in a directory:

```
import os  
  
# Define the directory path  
directory_path2 = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/Tinea_infection_last"  
  
# List files in the directory  
files = os.listdir(directory_path2)  
  
# Count the number of files  
file_count = len(files)  
  
print("Number of files in the directory:", file_count)
```

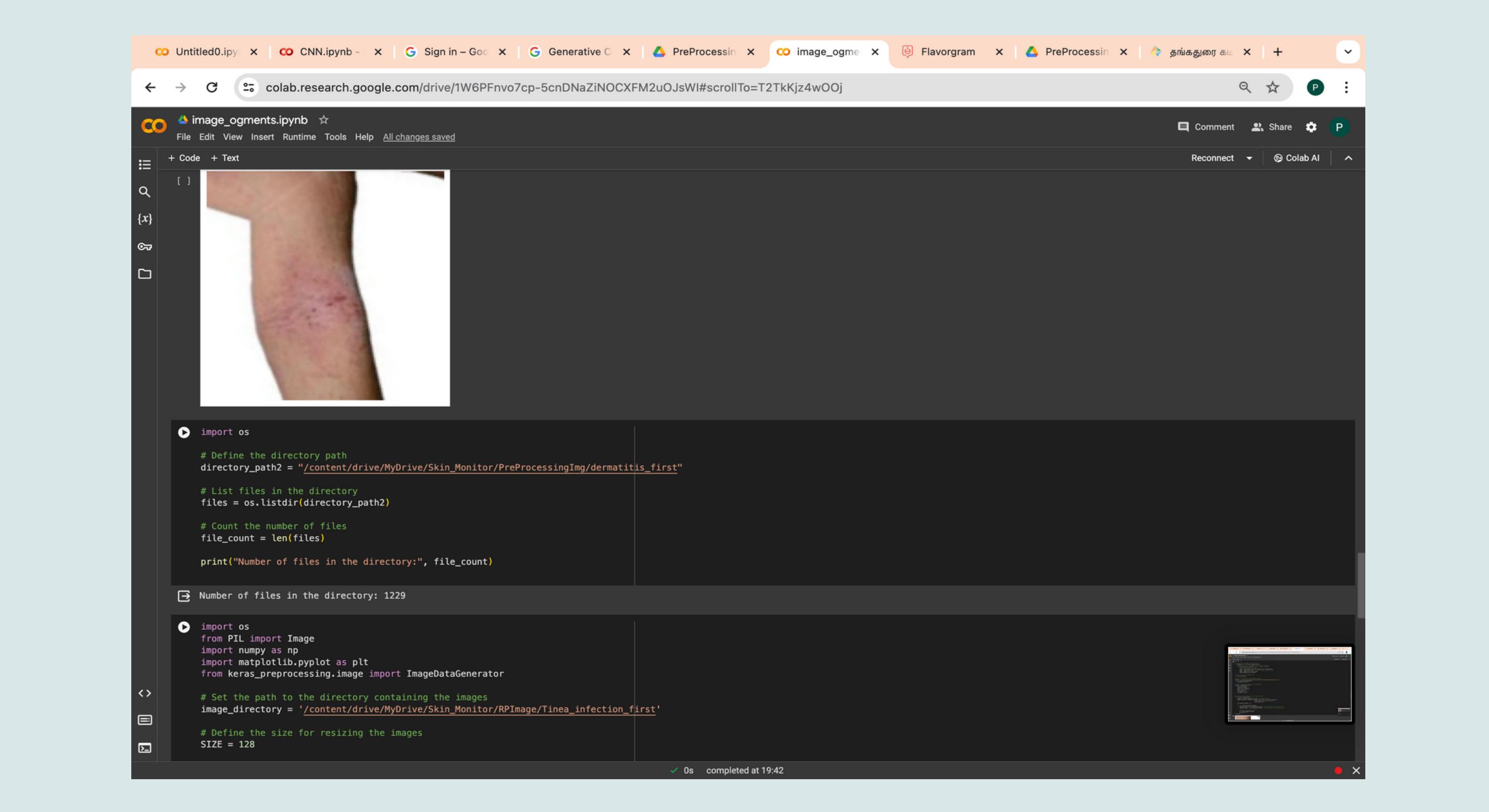
The output of this code is: Number of files in the directory: 360.

Below this, another code cell starts with "Start coding or generate with AI." and contains:

```
import os  
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
from keras.preprocessing.image import ImageDataGenerator  
  
# Set the path to the directory containing the images  
image_directory = '/content/drive/MyDrive/Skin_Monitor/RPIImage/dermatitis_first'  
  
# Define the size for resizing the images  
SIZE = 128
```

The status bar at the bottom indicates "0s completed at 19:42".

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". The notebook interface includes a toolbar at the top with various tabs like "Untitled0.ipynb", "CNN.ipynb", "Sign in - Google", "Generative", "PreProcessing", "image_ogments" (which is the active tab), "Flavorgram", "PreProcessing", and "தங்களுக்காக". Below the toolbar is a search bar and a user profile icon.

The main workspace displays a thumbnail of a skin image showing a rash or dermatitis. To the left of the image is a sidebar with icons for "Code" (selected) and "Text".

The code editor contains two code cells:

```
import os  
  
# Define the directory path  
directory_path2 = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/dermatitis_first"  
  
# List files in the directory  
files = os.listdir(directory_path2)  
  
# Count the number of files  
file_count = len(files)  
  
print("Number of files in the directory:", file_count)
```

Output from the first cell:

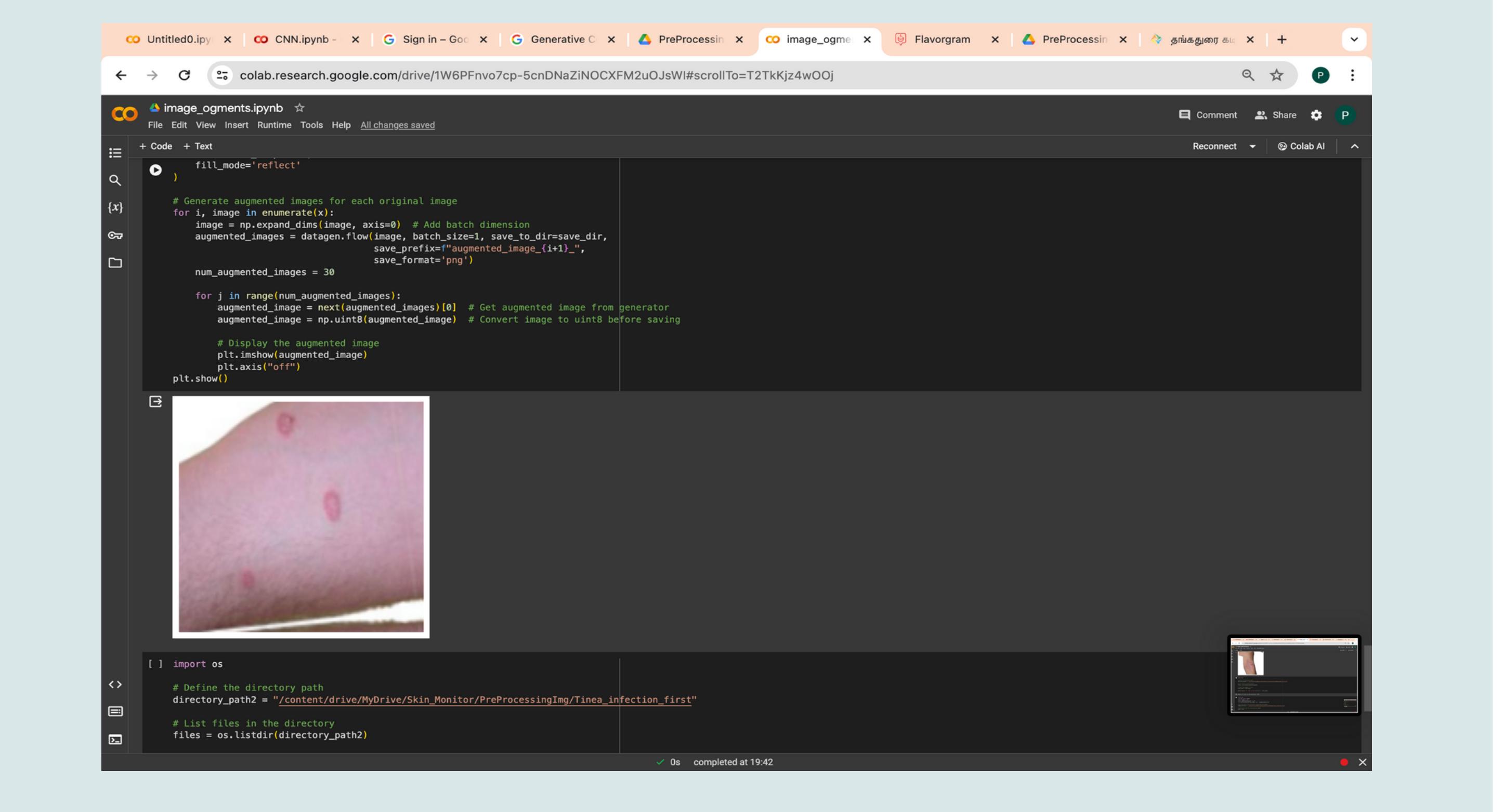
```
Number of files in the directory: 1229
```



```
import os  
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
from keras.preprocessing.image import ImageDataGenerator  
  
# Set the path to the directory containing the images  
image_directory = '/content/drive/MyDrive/Skin_Monitor/RPImage/Tinea_infection_first'  
  
# Define the size for resizing the images  
SIZE = 128
```

Output from the second cell shows a small preview window of the image data.

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". The code cell contains Python code for generating augmented images from a dataset. The output of the code is a visual representation of a skin lesion with three red circular marks, indicating the original image and its two augmented versions. Below this, another code cell imports os and lists files in a directory named "Tinea_infection_first". A small thumbnail of the same skin lesion image is visible in the background of the code cell.

```
fill_mode='reflect'

)
# Generate augmented images for each original image
for i, image in enumerate(x):
    image = np.expand_dims(image, axis=0) # Add batch dimension
    augmented_images = datagen.flow(image, batch_size=1, save_to_dir=save_dir,
                                    save_prefix=f"augmented_image_{i+1}_",
                                    save_format='png')
    num_augmented_images = 30

    for j in range(num_augmented_images):
        augmented_image = next(augmented_images)[0] # Get augmented image from generator
        augmented_image = np.uint8(augmented_image) # Convert image to uint8 before saving

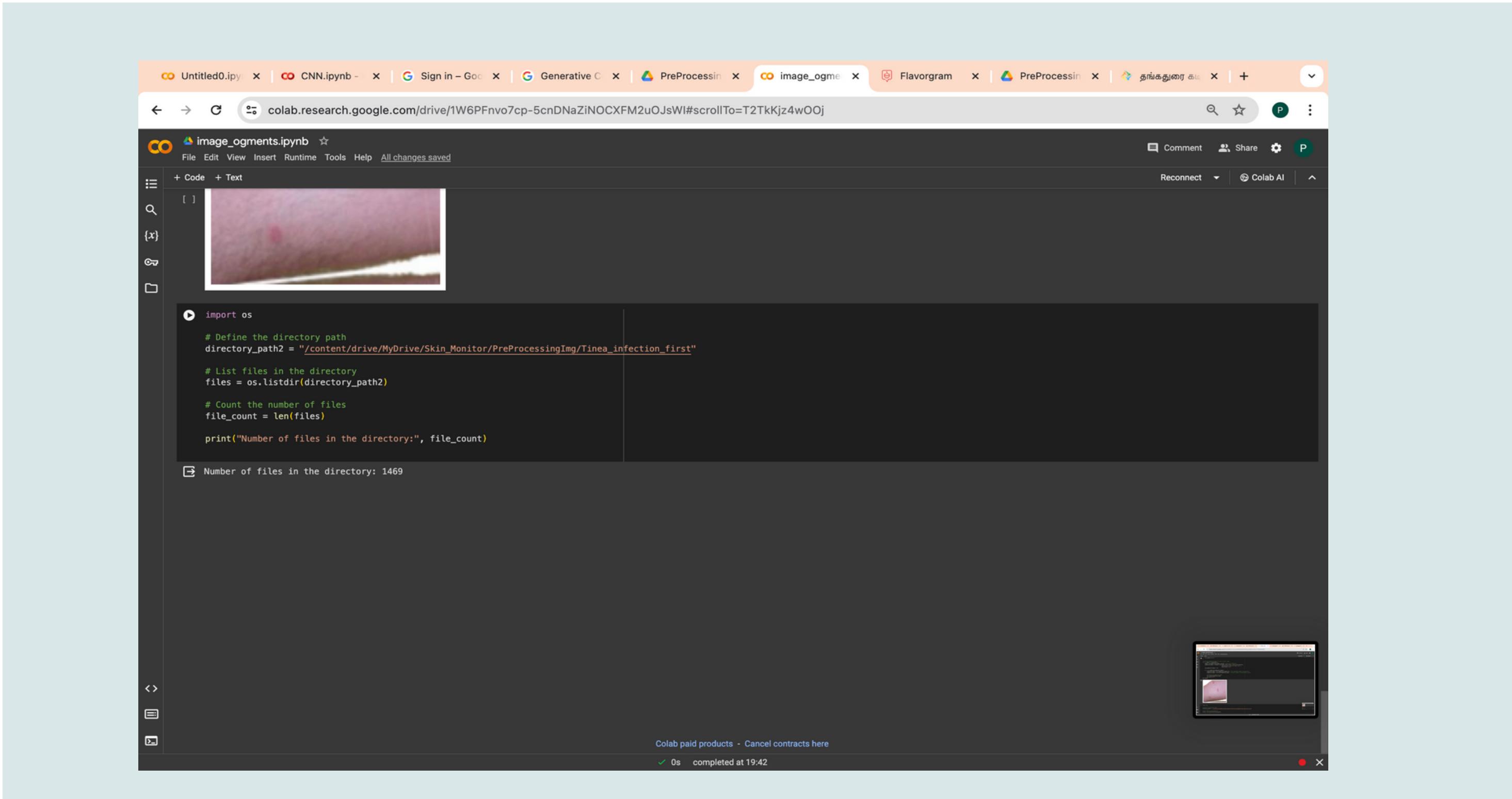
        # Display the augmented image
        plt.imshow(augmented_image)
        plt.axis("off")
    plt.show()

[ ] import os

<>
[ ] # Define the directory path
directory_path2 = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/Tinea_infection_first"

[ ] # List files in the directory
files = os.listdir(directory_path2)
```

Image Augmentation



The screenshot shows a Google Colab notebook titled "image_ogments.ipynb". The notebook interface includes a toolbar at the top with various tabs like "Untitled0.ipynb", "CNN.ipynb", "Sign in - Google", "Generative", "PreProcessing", "image_ogments" (which is the active tab), "Flavorgram", "PreProcessing", and "தங்களுரை கடமை". Below the toolbar is a search bar and a sidebar with icons for code, text, search, and files.

The main area displays a code cell with the following Python script:

```
import os  
  
# Define the directory path  
directory_path2 = "/content/drive/MyDrive/Skin_Monitor/PreProcessingImg/Tinea_infection_first"  
  
# List files in the directory  
files = os.listdir(directory_path2)  
  
# Count the number of files  
file_count = len(files)  
  
print("Number of files in the directory:", file_count)
```

The output of the code cell shows:

```
Number of files in the directory: 1469
```

Below the code cell, there is a thumbnail preview of a skin image showing a red, circular lesion.

Image Augmentation

The screenshot shows a Google Colab notebook titled "CNN.ipynb". The code in the notebook performs the following steps:

- Mounts Google Drive to the "/content/drive" directory.
- Changes the working directory to "/content/drive/MyDrive/Skin_Monitor".
- Imports TensorFlow, Keras, and Matplotlib libraries.
- Defines constants: BATCH_SIZE = 32, IMAGE_SIZE = 256, CHANNELS=3, and EPOCHS=20.
- Creates a dataset using `tf.keras.preprocessing.image_dataset_from_directory` with parameters: "PreProcessingImg", seed=123, shuffle=True, image_size=(IMAGE_SIZE, IMAGE_SIZE), batch_size=BATCH_SIZE.
- Prints a message indicating 3595 files belonging to 3 classes: 'Tinea infection_last', 'Tinea_infection_first', and 'dermatitis_last'.
- Prints the shape of the first batch of images and labels: (32, 256, 256, 3).

The Resources panel indicates that the user is not subscribed and has zero compute units available. It also shows that the session is not connected to a runtime.

Binary Validation

The image displays two side-by-side Jupyter Notebook interfaces, both titled "RP_train_local_machine".

Left Notebook:

- File Explorer:** Shows files including `image_ogments.ipynb`, `CNN (1).ipynb`, `trainModel.ipynb`, `newtest.ipynb`, and `modelfile.h5`.
- Code Editor:** The code is for a CNN model to classify dermatitis. It includes imports for TensorFlow, numpy, pandas, and matplotlib. It defines paths for training and validation data, creates data generators, and builds a Sequential model with Conv2D, MaxPooling2D, and Dense layers.
- Output:** Shows the command `pip install tensorflow numpy pandas matplotlib` being run in the terminal.

Right Notebook:

- File Explorer:** Similar file structure to the left notebook.
- Code Editor:** The code adds metrics to the model compilation and includes a section for saving the trained model to `modelfile.h5`.
- Output:** Shows the execution of the training loop. The output includes progress bars for training and validation steps, loss values, and accuracy metrics. The final output shows the model has been saved to `modelfile.h5`.

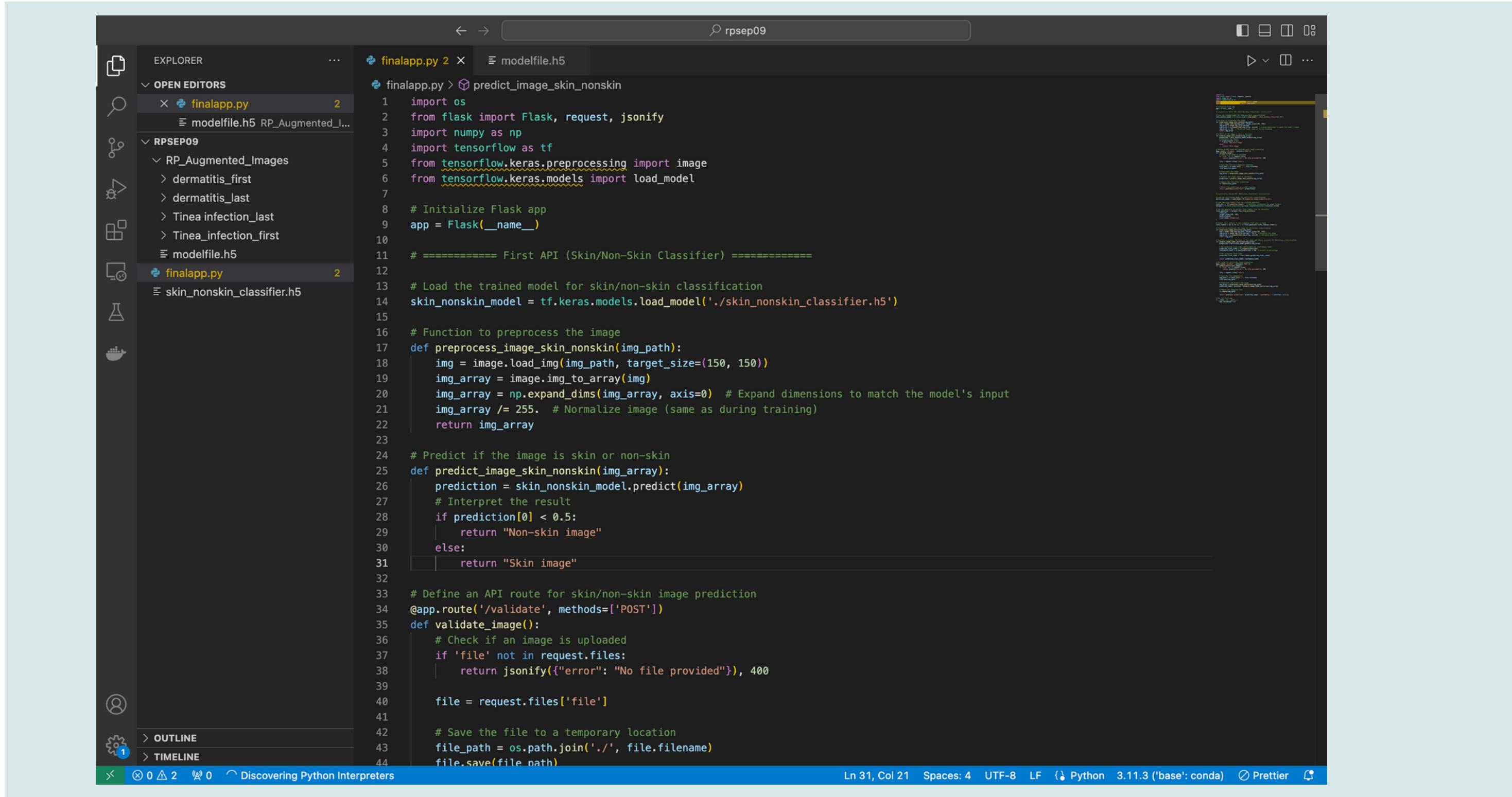
The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER** pane on the left:
 - OPEN EDITORS**: `image_dermoglyphs.ipynb`, `CNN (1).ipynb`, `trainModel.ipynb`, `newtest.ipynb`
 - RP_TRAIN_LOCAL_MACHINE**:
 - `image_dermoglyphs.ipynb`
 - `CNN (1).ipynb`
 - `trainModel.ipynb`
 - `newtest.ipynb`
 - `modelfile.h5`
 - `RP_Augmented_Images.ipynb`
 - `dermoglyphs`:
 - `dermatitis_first`
 - `dermatitis_last`
 - `Tinea_infection_first`
 - `Tinea_infection_last`
 - `RP_Augmented_Images`:
 - `dermatitis_first`
 - `dermatitis_last`
 - `Tinea_infection_first`
 - `Tinea_infection_last`
 - `modelfile.h5`
 - `CNN (1).ipynb`
 - `image_dermoglyphs.ipynb`
 - `newtest.ipynb`
 - `trainModel.ipynb`- CELLS** pane at the bottom:
 - Cell 1: `# Function to predict label for a new image`
 - Cell 2: `def predict_image_label(img_path, model, train_generator):`
 - Cell 3: `# Load the image`
 - Cell 4: `img = image.load_img(img_path, target_size=(150, 150))`
 - Cell 5: `# Preprocess the image`
 - Cell 6: `img_array = np.array(img) / 255.0`
 - Cell 7: `img_array = np.expand_dims(img_array, axis=0)`
 - Cell 8: `# Make a prediction`
 - Cell 9: `prediction = model.predict(img_array)`
 - Cell 10: `# Get the predicted class`
 - Cell 11: `class_indices = train_generator.class_indices`
 - Cell 12: `class_indices = {v: k for k, v in class_indices.items()}) # Invert the dictionary`
 - Cell 13: `predicted_class = class_indices[np.argmax(prediction)]`
 - Cell 14: `return predicted_class`
 - Cell 15: `# Load the model for prediction`
 - Cell 16: `loaded_model = load_model('modelfile.h5')`
- OUTPUT** pane on the right:
 - Cell 1: Output of `trainModel.ipynb` showing training metrics for Epoch 13/25.
 - Cell 2: Output of `trainModel.ipynb` showing validation metrics for Epoch 13/25.
 - Cell 3: Output of `newtest.ipynb` showing the loaded model's prediction for a new image.

Binary Validation

The screenshot shows the Postman application interface. On the left, there's a sidebar with a file tree containing 'my-spec.yaml'. The main area shows a POST request to 'http://127.0.0.1:5000/validate'. The 'Body' tab is selected, showing a 'Multipart' section with one item named 'file'. The response on the right is a 200 OK status with a response time of 629 ms and a size of 33 B, timestamped '2 Minutes Ago'. The response body is a JSON object: { "prediction": "Skin image" }. The bottom status bar indicates the collection is 'Online'.

Integration with Flask API



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- Title Bar:** The title bar displays the workspace name "rpsep09".
- Explorer View:** The left sidebar shows the file structure under "OPEN EDITORS". It includes "finalapp.py" (2 instances), "modelfile.h5", and a folder "RP_Augmented_Images" containing subfolders "dermatitis_first", "dermatitis_last", "Tinea infection_last", "Tinea_infection_first", and "modelfile.h5".
- Code Editor:** The main editor area contains the "finalapp.py" code for a Flask API. The code imports necessary libraries (os, flask, numpy, tensorflow, tensorflow.keras.preprocessing, tensorflow.keras.models), initializes a Flask app, loads a trained model ("skin_nonskin_classifier.h5"), defines a function to preprocess images, defines a function to predict if an image is skin or non-skin, and defines an API route for validation.
- Terminal:** A terminal window is visible on the right side of the interface, showing command-line output.
- Status Bar:** The bottom status bar shows the current file is "finalapp.py", line count is 31, column count is 21, spaces used are 4, encoding is UTF-8, LF, Python 3.11.3 ('base': conda) is the interpreter, and Prettier is the formatter.

Integration with NGROK

```
balendranrushanth — ngrok http 8080 — 117x22
ngrok
(Ctrl+C to quit)

Share what you're building with ngrok https://ngrok.com/share-your-ngrok-story

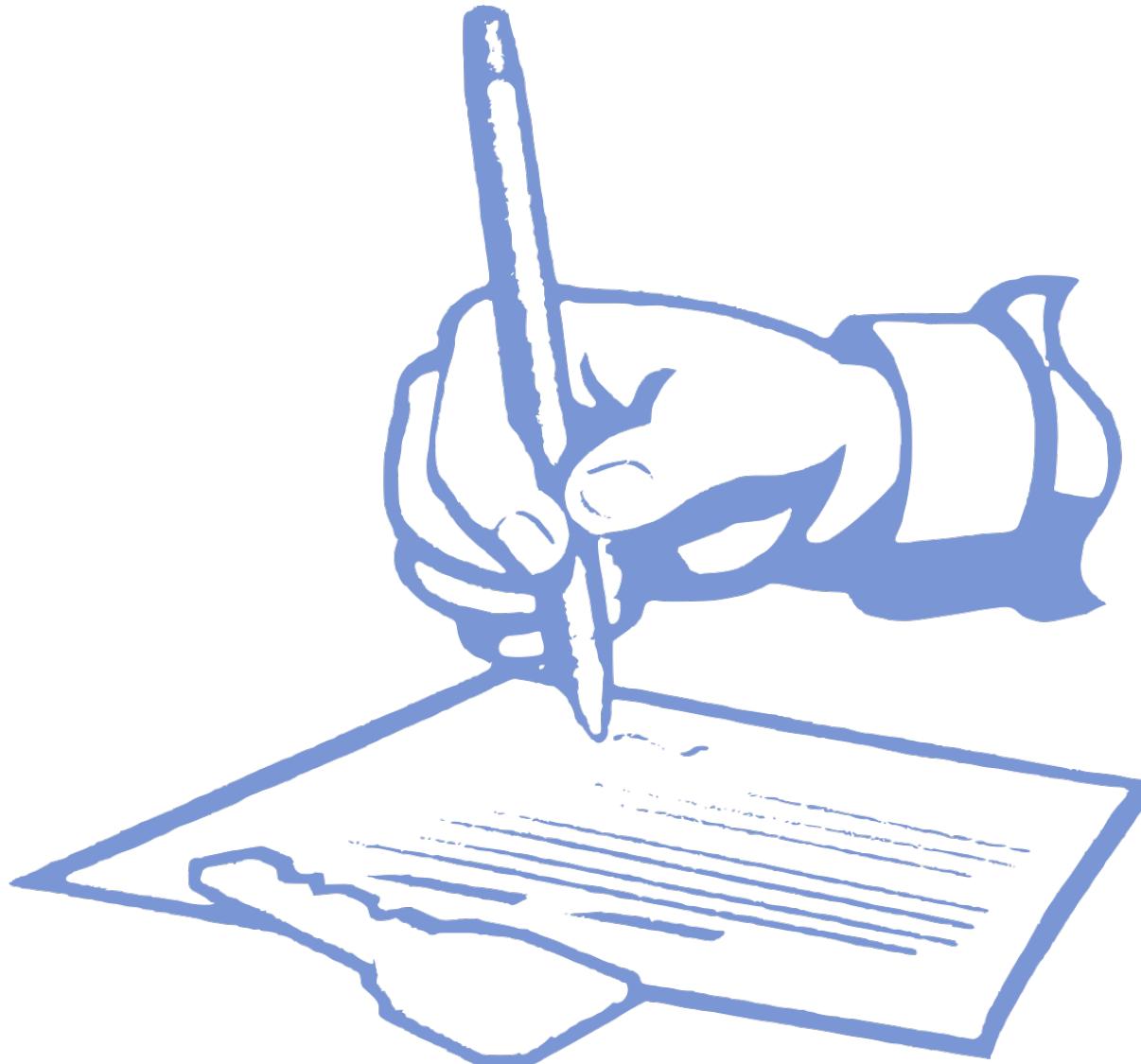
Session Status          online
Account                 rushanth9999@gmail.com (Plan: Free)
Update                  update available (version 3.16.0, Ctrl-U to update)
Version                3.15.1
Region                 Asia Pacific (ap)
Latency                224ms
Web Interface          http://127.0.0.1:4040
Forwarding             https://2e28-2a09-bac1-4320-00-38-5e.ngrok-free.app -> http://localhost:8080

Connections            ttl     opn      rt1      rt5      p50      p90
                      46       0      0.00      0.00     0.39     1.32

HTTP Requests
-----
23:45:53.448 +531POST /predict           200 OK
23:45:48.946 +481POST /validate         200 OK
23:44:12.669 +121POST /validate         200 OK
```

Dataset Collection

Patatient conformation form



Title of Project: Health Skin Monitoring: Detection and Early Intervention for Skin Issues
Principal Investigator: Dr.Anushan

Purpose of the Study:
The purpose of this research project is to study and analyze various skin diseases to enhance our understanding of dermatological conditions and develop improved treatments.

Procedure:
As part of this study, photographs of the affected area(s) of your skin will be taken using a digital camera or other imaging device. These photographs will be used solely for research purposes and may be included in research publications or presentations. Your identity will be kept confidential, and the photographs will not be used for any other purpose without your explicit consent.

Potential Risks:
There are minimal risks associated with participating in this study. The photographs may reveal personal information about your health condition, which will be kept confidential to the best of our ability.

Benefits:
By participating in this study, you will contribute valuable information that may help improve the diagnosis and treatment of skin diseases, benefitting both current and future patients.

Voluntary Participation:
Participation in this research project is entirely voluntary. You have the right to refuse or withdraw from the study at any time, without any penalty or loss of benefits to which you are otherwise entitled.

Confidentiality:
All information collected during this study, including photographs and personal data, will be kept strictly confidential. Your identity will be protected, and only authorized personnel involved in the research project will have access to the data.

Contact Information:
If you have any questions or concerns about the study, you can contact the Principal Investigator at [insert contact information].

I have read and understood the information provided above, and I agree to participate in this research project.

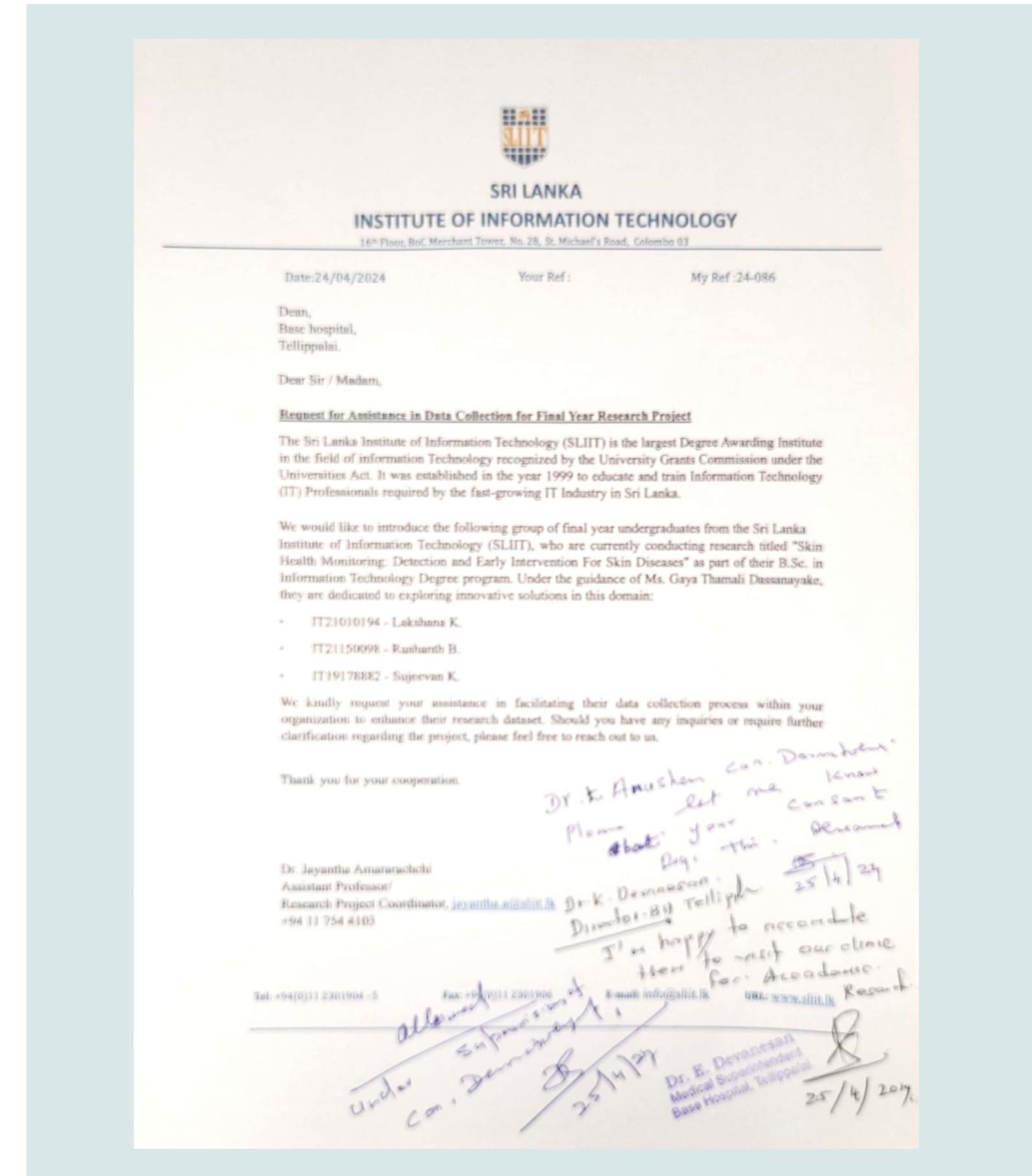
Patient Name (Printed): _____

Patient Signature: _____

Date: _____

Dataset Collection

Deal approval



Dataset Collection

Deal approval




**SRI LANKA
INSTITUTE OF INFORMATION TECHNOLOGY**
16th Floor, BoC Merchant Tower, No. 28, St. Michael's Road, Colombo 03

Date:25/03/2024 Your Ref : - My Ref:24-086

Dean,
Base hospital,
Manthikai,
Point Pedro.

Dear Sir / Madam,

Request for Assistance in Data Collection for Final Year Research Project

The Sri Lanka Institute of Information Technology (SLIIT) is the largest Degree Awarding Institute in the field of information Technology recognized by the University Grants Commission under the Universities Act. It was established in the year 1999 to educate and train Information Technology (IT) Professionals required by the fast-growing IT Industry in Sri Lanka.

We would like to introduce the following group of final year undergraduates from the Sri Lanka Institute of Information Technology (SLIIT), who are currently conducting research titled "Skin Health Monitoring: Detection and Early Intervention For Skin Diseases" as part of their B.Sc. in Information Technology Degree program. Under the guidance of Ms. Gaya Thamali Dassanayake, they are dedicated to exploring innovative solutions in this domain:

- IT21010194 - Lakshana K.
- IT21150098 - Rushanth B.
- IT19178882 - Sujeevan K.

We kindly request your assistance in facilitating their data collection process within your organization to enhance their research dataset. Should you have any inquiries or require further clarification regarding the project, please feel free to reach out to us.

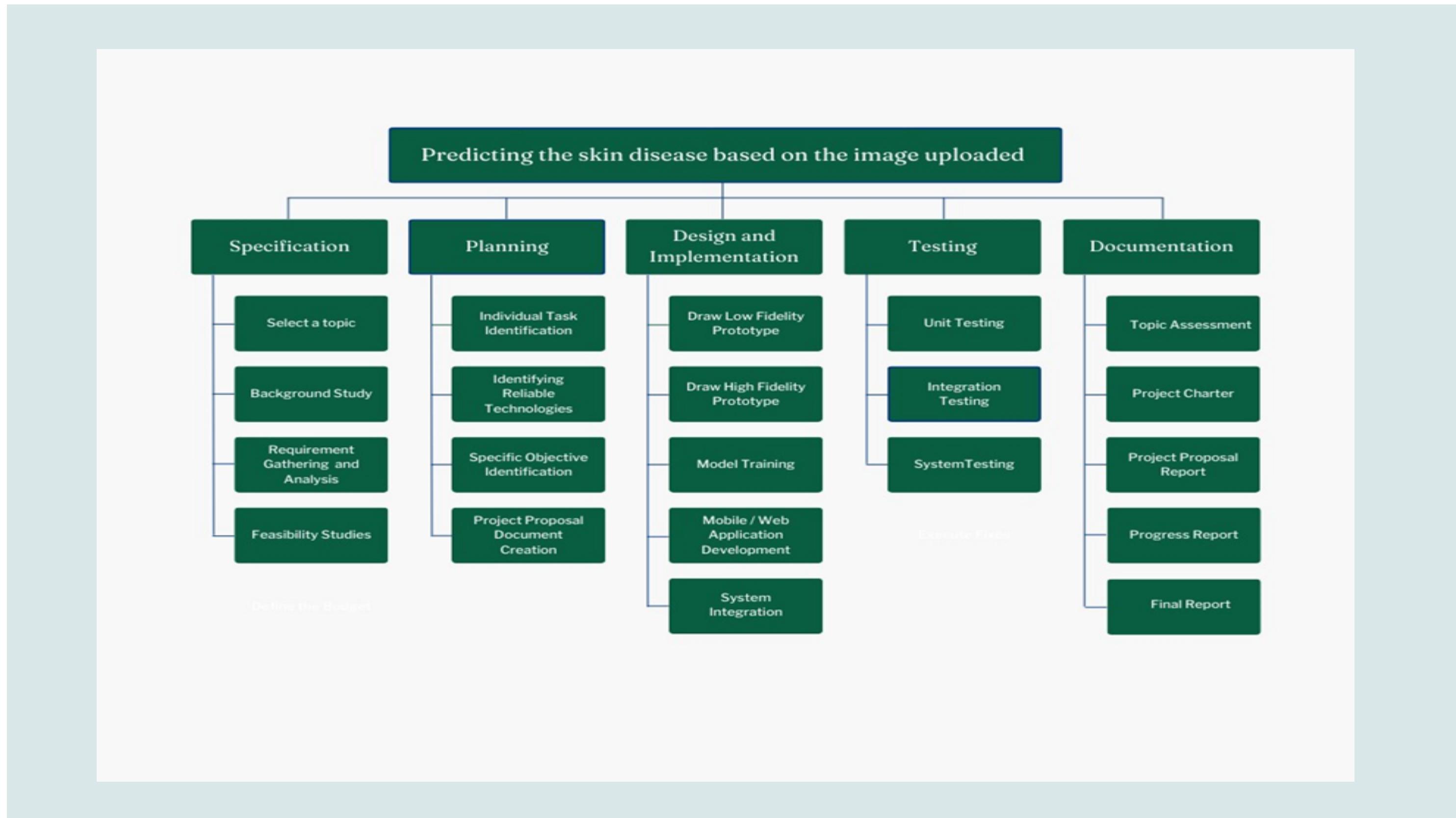
Thank you for your cooperation.

*Approved .
for Dr. Anushan (Cons. Dermato-
logist)
for Dr. Jayantha Amarachchi (Patient-
Research Project-
Shanka. (Mrs) Mayalin.
Medical Superintendent
Base Hospital
Point Pedro*

Dr. Jayantha Amarachchi
Assistant Professor/
Research Project Coordinator,
jayantha.a@sliit.lk
+94 11 754 4103

Tel: +94(0)11 2301904 - 5 Fax: +94(0)11 2301906 E-mail: info@sliit.lk URL: www.sliit.lk

WORK BREAKDOWN STRUCTURE

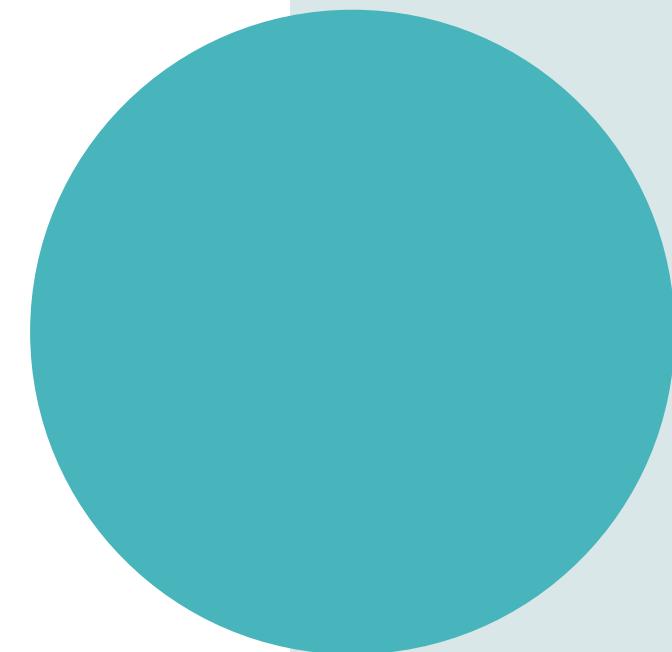


COMMERCIALIZATION

CHOOSE YOUR BEST PLAN

UPGRADE TO PREMIUM FOR THE ULTIMATE SKINCARE EXPERIENCE – UNLIMITED, AD-FREE, AND EXCLUSIVELY YOURS.

Free free <ul style="list-style-type: none">• Basic skincare analysis and recommendations.• Limited usage.• Standard support.• Ad-supported. Try for 30 days	Premium PER MONTH \$9.99 <ul style="list-style-type: none">• Full access to advanced features.• Unlimited usage.• Priority support.• Ad-free experience.• Exclusive content. Try for 30 days	Business PER MONTH \$39.99 <ul style="list-style-type: none">• Team access.• Customization for business needs.• Advanced analytics.• White-labeling option.• Dedicated support.• Integration options. Try for 30 days
--	---	---



Thank You