

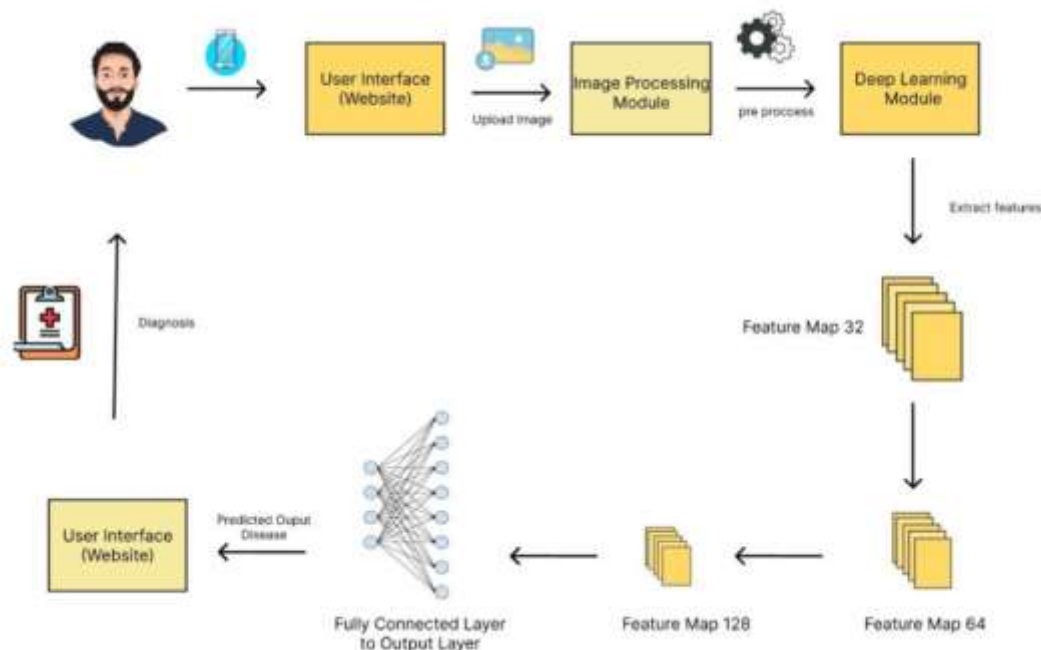
Early Diagnosis of Diseases Using Image Processing of Human Nails

Introduction:

In the realm of healthcare, the observation of human nails' colour and shape can often forewarn the presence of various diseases. Be it a small white blemish or a subtle pinkish tinge, even the slightest irregularity in the nail's appearance might signify an underlying ailment. The condition of one's nails can reveal much about the well-being of their liver, lungs, and heart. Doctors closely examine patients' nails to aid in the identification of potential ailments. Typically, a healthy individual displays pink nails that are smooth and consistently coloured. Any deviation affecting the growth and appearance of fingernails or toenails could indicate an anomaly. The condition of a person's nails can be indicative of their overall health status. The necessity for systems to analyse nails for disease prediction arises from the inherent subjectivity of the human eye concerning colours, limited resolution, and the potential oversight of subtle colour changes in a few nail pixels that may lead to erroneous conclusions. Conversely, computers can identify even minor colour variations on nails.

To tackle the aforementioned challenge, we are developing a model designed for the prevention and early detection of Nail Disease. Essentially, nail disease diagnosis hinges on various attributes such as colour, shape, and texture. Here, individuals can capture images of their nails, which are then forwarded to the trained model. The model scrutinizes the images and determines whether the person is afflicted with a nail disease and identifies its type.

Technical Architecture:



Prerequisites:

To successfully execute this project, the following software, concepts, and packages are essential:

Anaconda Navigator: A free and open-source distribution of the Python and R programming languages, used for data science and machine learning applications. It is compatible with Windows, Linux, and macOS. Anaconda includes tools such as JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, RStudio, and Visual Studio Code. For this project, Google Colab and Spyder will be utilized.

Necessary Python Packages:

NumPy: An open-source numerical Python library that supports multidimensional arrays and matrix data structures, facilitating various mathematical operations.

Scikit-learn: A Python-based, free machine learning library that incorporates algorithms such as support vector machines, random forests, and k-nearest neighbours. It is compatible with Python numerical and scientific libraries like NumPy and SciPy.

Flask: A web framework employed for constructing web applications.

Python Packages Installation: Open the Anaconda prompt as an administrator and execute the following commands:

```
"pip install numpy"
```

```
"pip install pandas"
```

```
"pip install scikit-learn"
```

```
"pip install tensorflow==2.3.2"
```

```
"pip install keras==2.3.1"
```

```
"pip install Flask"
```

Deep Learning Concepts:

VGG16: Utilized as a transfer learning method, VGG16 constitutes a pre-trained model trained on 1000 classes of images, serving as a fundamental for image analysis.

Flask: A widely-used Python web framework, functioning as a third-party Python library for the development of web applications.

Project Flow:

User Interaction: The user engages with the UI (User Interface) to select the image.

Image Analysis: The chosen image is analysed by the integrated model within the Flask application.

VGG16 Model Analysis: The VGG16 Model scrutinizes the image, and the predictions are displayed on the Flask UI.

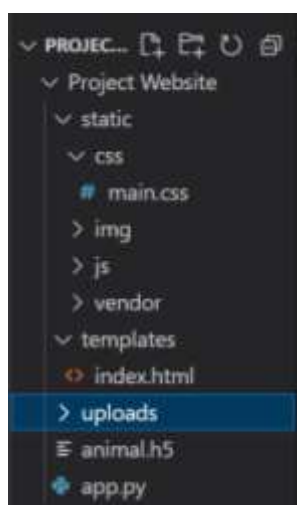
To accomplish this, the activities and tasks outlined below need to be completed:

1. Data Collection
2. Importing Model Building Libraries
3. Loading the Model
4. Addition of Flatten Layers
5. Addition of Output Layer
6. Creation of a Model Object
7. Configuration of the Learning Process
8. Importation of the Image Data Generator Library
9. Configuration of the Image Data Generator Class
10. Application of Image Data Generator functionality to Trainset and Test set
11. Model Training
12. Model Saving
13. Model Testing
14. HTML File Creation
15. Python Code Building
16. Application Execution
17. Final Output Display

Project Structure:

Organize the project in the following structure within a folder named "Project Website":

- Flask Application Files: Within the "Project Website" folder, the following files are essential for the Flask application:
- A folder named "templates" that houses the following HTML pages: index.html
- A Python script named "app.py" responsible for server-side scripting.
- Model File: Store the saved model, "Vgg-16-nail-disease.h5," within the "Project Website" folder.
- Within the "Project Website" folder, the following additional subfolders are required:
- A folder named "static" which includes the following subfolders:
 1. A "css" folder containing the "main.css" file.
 2. An "img" folder.
 3. A "js" folder.
 4. A "vendor" folder.



Task – 1 : Data Collection

The following dataset has been used.

Dataset:

<https://drive.google.com/drive/folders/1AXTYsbiarS1TCAgfj0mancTSrJYYMWMs?usp=sharing>

The dataset has already been divided into train and test folders. Therefore, we need not further divide it.

Task – 2 : Importing Model Building Libraries

Importing the necessary libraries.

```
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from glob import glob
import numpy as np
import matplotlib.pyplot as plt
```

Task – 3 : Loading the Model

```
vgg = VGG16(input_shape=imagesize + [3], weights='imagenet', include_top=False)
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [=====] - 0s 0us/step

Task – 4 : Addition of Flatten Layers

For VGG16 model, we need to keep the Hidden layer training as false, because it has trained weights

```
for layer in vgg.layers:
    layer.trainable = False
```

```
x = Flatten()(vgg.output)
```

Task – 5 : Addition of Output Layer

Our dataset has 17 classes, so the output layer needs to be changed as per the dataset

```
prediction = Dense(17, activation = 'softmax')(x)
```

17 indicates no of classes, SoftMax is the activation function we use for categorical output Adding fully connected layer

Task – 6 : Creation of a Model Object

```
model = Model(inputs=vgg.input,outputs=prediction)
```

We have created inputs and outputs in the previous steps and we are creating a model and fitting to the vgg16 model, so that it will take inputs as per the given and displays the given no of classes.

```
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 255, 255, 3)]	0
block1_conv1 (Conv2D)	(None, 255, 255, 64)	1792
block1_conv2 (Conv2D)	(None, 255, 255, 64)	36928
block1_pool (MaxPooling2D)	(None, 127, 127, 64)	0
block2_conv1 (Conv2D)	(None, 127, 127, 128)	73856
block2_conv2 (Conv2D)	(None, 127, 127, 128)	147584
block2_pool (MaxPooling2D)	(None, 63, 63, 128)	0
block3_conv1 (Conv2D)	(None, 63, 63, 256)	295168
block3_conv2 (Conv2D)	(None, 63, 63, 256)	590080
block3_conv3 (Conv2D)	(None, 63, 63, 256)	590080
block3_pool (MaxPooling2D)	(None, 31, 31, 256)	0
block4_conv1 (Conv2D)	(None, 31, 31, 512)	1180160
block4_conv2 (Conv2D)	(None, 31, 31, 512)	2359808
block4_conv3 (Conv2D)	(None, 31, 31, 512)	2359808
block4_pool (MaxPooling2D)	(None, 15, 15, 512)	0
block5_conv1 (Conv2D)	(None, 15, 15, 512)	2359808
block5_conv2 (Conv2D)	(None, 15, 15, 512)	2359808
block5_conv3 (Conv2D)	(None, 15, 15, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Summary of the model:

flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 17)	426513

```

Total params: 15141201 (57.76 MB)
Trainable params: 426513 (1.63 MB)
Non-trainable params: 14714688 (56.13 MB)

```

Task – 7 : Configuration of the Learning Process

1. The compilation marks the concluding stage in the model creation process. Once compiled, the training phase can commence. The loss function is employed to identify errors or discrepancies within the learning process. Keras mandates the specification of a loss function during the model compilation phase.
2. Optimization represents a crucial procedure that fine-tunes the input weights by assessing the predictions against the defined loss function. In this case, the Adam optimizer is employed for optimization purposes.
3. Metrics serve as tools for evaluating the overall performance of your model. They share similarities with the loss function; however, they are not actively involved in the training process.

```

model.compile(
    loss='categorical_crossentropy',
    optimizer = 'adam',
    metrics = ['precision'],run_eagerly=True
)

```

Task – 8 : Importation of the Image Data Generator Library

Within this phase, we will focus on enhancing the image data to mitigate undesirable distortions and accentuate critical image features essential for subsequent processing. This involves the implementation of various geometric transformations such as rotation, scaling, translation, and more.

Image data augmentation is a valuable technique employed to effectively increase the scale of a training dataset by generating modified versions of the existing images within the dataset. The Keras deep learning neural network library offers the functionality to train models using image data augmentation through the utilization of the ImageDataGenerator class.

Let us proceed by importing the ImageDataGenerator class from the TensorFlow Keras library.

```

from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

Task – 9 : Configuration of the Image Data Generator Class

We instantiate the ImageDataGenerator class and configure the specific data augmentation techniques. These techniques primarily include:

1. Image shifts utilizing the width_shift_range and height_shift_range arguments.
2. Image flips facilitated by the horizontal_flip and vertical_flip arguments.
3. Image rotations enabled through the rotation_range argument.
4. Image brightness adjustments managed by the brightness_range argument.
5. Image zoom functionality controlled by the zoom_range argument.

By constructing an instance of the ImageDataGenerator class, we can effectively apply these augmentation techniques to both the training and test datasets.

```
train_datagen = ImageDataGenerator(rescale =1./255,  
                                   shear_range =0.2,  
                                   zoom_range=0.2,  
                                   horizontal_flip =True)  
test_datagen=ImageDataGenerator(rescale =1./255 )
```

Task – 10 : Application of Image Data Generator functionality to Trainset and Test set

Let us proceed by implementing the ImageDataGenerator functionality to both the Training set and Test set using the code provided below. This will be accomplished by employing the "flow_from_directory" function.

The function returns batches of images from the specified subdirectories.

Arguments:

- Directory: Refers to the directory housing the dataset. If the labels are "inferred," the directory should consist of subdirectories, each containing images corresponding to a specific class. Otherwise, the directory structure will be disregarded.
- batch_size: Denotes the size of the data batches, set to 10.
- target_size: Specifies the dimensions for resizing images post-reading from the disk.
- class_mode:
 - 'int': Indicates that the labels are encoded as integers (e.g., suitable for sparse_categorical_crossentropy loss).
 - 'categorical': Implies that the labels are encoded as a categorical vector (e.g., appropriate for categorical_crossentropy loss).
 - 'binary': Signifies that the labels (limited to 2) are encoded as float32 scalars with values 0 or 1 (e.g., used for binary_crossentropy).
 - None: Suggests the absence of labels.

Loading our data and performing Data Augmentation

```

training_set = train_datagen.flow_from_directory(train_path,
                                                target_size = (255,255),
                                                batch_size = 64,
                                                class_mode = 'categorical')
test_set = test_datagen.flow_from_directory(test_path,
                                             target_size = (255,255),
                                             batch_size = 64,
                                             class_mode = 'categorical')

```

We notice that 655 images belong to 17 classes for training and 183 images belonging to 17 classes for testing purposes.

List of classes we have:

```

{'Darier_s disease': 0,
 'Muehrck-e_s lines': 1,
 'alopecia areata': 2,
 'beau_s lines': 3,
 'bluish nail': 4,
 'clubbing': 5,
 'eczema': 6,
 'half and half nailes (Lindsay_s nails)': 7,
 'koilonychia': 8,
 'leukonychia': 9,
 'onycholycis': 10,
 'pale nail': 11,
 'red lunula': 12,
 'splinter hemmorrhage': 13,
 'terry_s nail': 14,
 'white nail': 15,
 'yellow nails': 16}

```

Task – 11 : Model Training

Now, we proceed to train our model utilizing the designated image dataset. The model undergoes training for a total of 11 epochs, with the current state of the model being saved after each epoch if the encountered loss is the least recorded up to that point. Notably, the training loss exhibits a consistent decrease across almost every epoch throughout the 11-epoch training cycle, indicating potential room for further model refinement.

The "fit_generator" function is employed to facilitate the training of the deep learning neural network.

Arguments:

- **steps_per_epoch:** This parameter determines the total number of steps taken from the generator once an epoch is completed and the subsequent epoch begins. The value of

steps_per_epoch can be calculated by dividing the total number of samples in the dataset by the batch size.

- Epochs: An integer denoting the desired number of epochs for training the model.
- Validation_data: This argument can assume one of the following forms:
 - An inputs and targets list.
 - A generator.
 - An inputs, targets, and sample_weights list, facilitating the evaluation of the loss and metrics for any model once each epoch concludes.
- Validation_steps: This argument is utilized only when the validation_data is a generator. It dictates the total number of steps taken from the generator before it is halted at the conclusion of each epoch. The value of validation_steps can be determined by dividing the total number of validation data points in the dataset by the validation batch size.

```
import sys
#fit the model
r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=11,
    steps_per_epoch=len(training_set)//3,
    validation_steps=len(test_set)//3
)
```

Accuracy after 11 epochs:

```
<ipython-input-33-8343a613715a>:3: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Pl
r = model.fit_generator(
Epoch 1/11
3/3 [=====] - 13s 1s/step - loss: 0.4664 - accuracy: 0.8951 - val_loss: 0.3508 - val_accuracy: 0.9219
Epoch 2/11
3/3 [=====] - 6s 2s/step - loss: 0.4990 - accuracy: 0.8906 - val_loss: 0.3634 - val_accuracy: 0.9531
Epoch 3/11
3/3 [=====] - 6s 2s/step - loss: 0.3643 - accuracy: 0.9635 - val_loss: 0.3288 - val_accuracy: 0.9375
Epoch 4/11
3/3 [=====] - 7s 3s/step - loss: 0.4542 - accuracy: 0.8802 - val_loss: 0.3934 - val_accuracy: 0.9375
Epoch 5/11
3/3 [=====] - 5s 2s/step - loss: 0.4245 - accuracy: 0.9219 - val_loss: 0.3191 - val_accuracy: 0.9688
Epoch 6/11
3/3 [=====] - 7s 2s/step - loss: 0.4078 - accuracy: 0.9323 - val_loss: 0.3519 - val_accuracy: 0.9688
Epoch 7/11
3/3 [=====] - 6s 2s/step - loss: 0.4663 - accuracy: 0.9323 - val_loss: 0.3629 - val_accuracy: 0.9375
Epoch 8/11
3/3 [=====] - 6s 2s/step - loss: 0.3462 - accuracy: 0.9688 - val_loss: 0.3258 - val_accuracy: 0.9688
Epoch 9/11
3/3 [=====] - 5s 2s/step - loss: 0.4886 - accuracy: 0.9091 - val_loss: 0.3702 - val_accuracy: 0.9375
Epoch 10/11
3/3 [=====] - 4s 2s/step - loss: 0.4202 - accuracy: 0.9371 - val_loss: 0.3700 - val_accuracy: 0.9375
Epoch 11/11
3/3 [=====] - 6s 2s/step - loss: 0.3769 - accuracy: 0.9323 - val_loss: 0.3139 - val_accuracy: 0.9531
```

Task – 12 : Model Saving

```
model.save('vgg-16-nail-disease.h5')
```

The model is saved with .h5 extension.

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model using `load_model`

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
import numpy as np
```

```
model = load_model('vgg-16-nail-disease.h5')
```

Task – 13 : Model Testing

Taking an image as input and checking the results

```
img_data = image.load_img('/content/th.jpeg', target_size=(255,255))
img_data
```



Resizing the image

```
image_resized = tf.reshape(img_data, (-1, 255, 255, 3, 1))
image_resized = tf.squeeze(image_resized)
image_resized
```

```
x = image.img_to_array(image_resized)
x = np.expand_dims(x,axis = 0)
x
```

Predicting:

```

pred = np.argmax(model.predict(x),axis=1)
op = ['Darier_s disease', 'Muehrck-e_s lines', 'aloperia areata', 'beau_s lines',
      'bluish nail', 'clubbing', 'eczema', 'half and half nailes (Lindsay_s nails)',
      'koilonychia', 'leukonychia', 'onycholycis', 'pale nail', 'red lunula',
      'splinter hemmorrhage', 'terry_s nail', 'white nail', 'yellow nails']
result = str(op[pred[0]])
result

1/1 [=====] - 0s 20ms/step
'white nail'

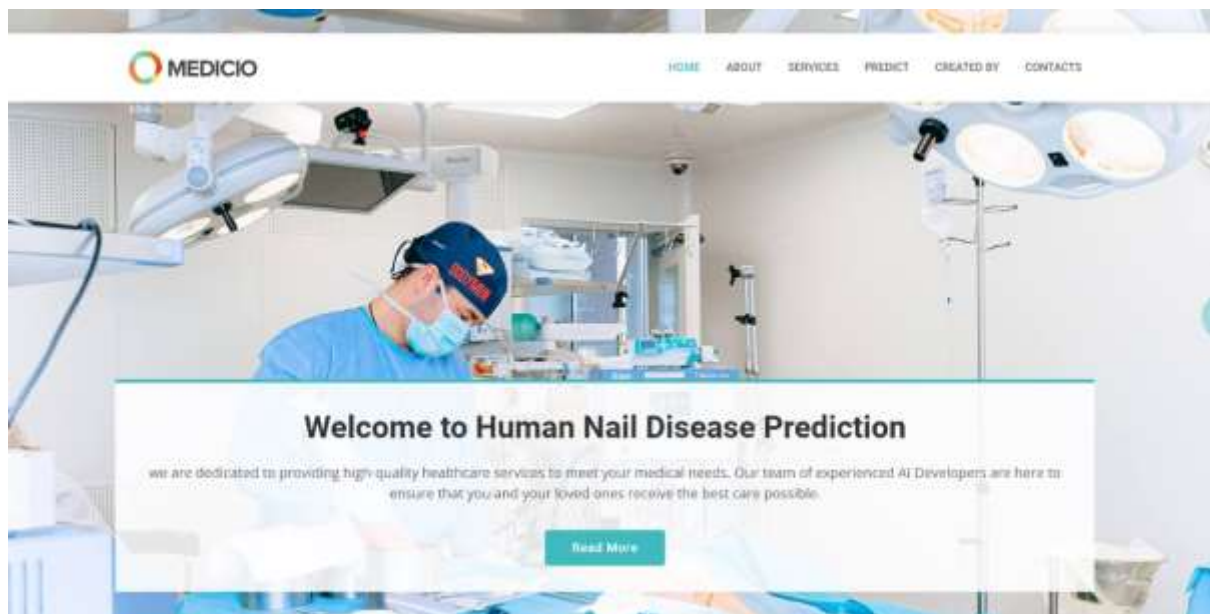
```

Task – 14 : HTML File Creation

Having completed the model training, our next step involves constructing a Flask application that will operate within the confines of our local browser, offering a user interface for interaction.


Within the Flask application, the input parameters are extracted from the HTML page. These parameters are subsequently utilized by the model to predict the estimated cost for the incurred damage, with the results promptly displayed on the HTML page, thereby informing the user. Upon user interaction with the UI and selection of the "Image" button, a subsequent page is presented, enabling the user to choose the desired image and acquire the corresponding prediction output.

We created the index.html file using html




Our Address
VIT-AP, Amaravati


Email Us
saranan.21bce5479@vitapstudent.ac.in
satgurunish.21bce5479@vitapstudent.ac.in
anusha.21bce5479@vitapstudent.ac.in
vishu.21bce5479@vitapstudent.ac.in


Call Us
+91 7066540121
+91 9950754433
+91 7888102861
+91 9381068505

CONTACT

Whether you have a question about our services, need assistance with scheduling an appointment, we're here for you. Please reach out to us (through OnSite (or) Email (or) Phone, and our friendly staff will be happy to assist you.



SERVICES

We offer a comprehensive healthcare services that are tailored to meet your unique health needs. Our services are designed with a patient-centric approach, combining medical expertise with cutting-edge technology to ensure you receive the best care.

Easy Disease Recognition

We can identify potential health issues at their earliest stages, giving you the advantage of early intervention and improved health.

Faster Prediction

We provide "Faster Prediction" services for quicker and more accurate medical diagnoses. Our advanced diagnostic tools help to ensure that you receive timely information about your health.

At Home Service

We understand that convenience is essential when it comes to healthcare. Our "At Home Service" is designed to bring medical care to your doorstep.

ABOUT US

Our mission is to redefine healthcare by providing compassionate, expert care while embracing cutting-edge technology. We believe that health is not just the absence of disease; it's about the quality of life, and we're here to enhance it.



Journey of MedicioAI

MedicioAI has been a trusted healthcare partner for countless individuals and families. We have continually evolved to meet the changing needs of our patients and the advancements in medical science.

Our Values

- Excellence:** We are dedicated to excellence in medical care, constantly improving and innovating to deliver the best outcomes.
- Compassion:** We understand that healthcare is not just about treating illnesses but also about providing support and understanding.
- Integrity:** Our commitment to ethical practices ensures that you can trust us with your health.

We believe that technology is a powerful tool in healthcare. Our focus on advanced technology, including the groundbreaking "Human Tumor Disease Prediction" system, allows us to provide accurate and early diagnoses.



Task – 15 : Python Code Building

Step 1: Import libraries

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, render_template, request
import os
import numpy as np
```

Step 2 : Load our model to create flask application

```
app = Flask(__name__)
model = load_model(r"nail.h5", compile = False)
@app.route('/')

```

Step 3 : Redirect to index page

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

Step 4 : Showcasing prediction on UI

In this section, we define a function that requests the selected file from the HTML page via the post method. The received image file is then saved to the "uploads" folder within the same directory, utilizing the OS library. Subsequently, we employ the "load image" class from the Keras library to retrieve the saved image from the specified path. Various image

processing techniques are applied to the retrieved image, which is then forwarded to the model for class prediction.

The outcome is a numerical value representing a specific class (e.g., 0, 1, 2, etc.), which resides at the 0th index of the variable "preds." This numerical value is assigned to the declared index variable. The corresponding class name is then derived and assigned to the "predict" variable, which is subsequently rendered on the HTML page for user reference.

```
@app.route('/nailresult', methods=["GET", "POST"])
def nres():
    if request.method == "POST":
        f = request.files['image']
        basepath = os.path.dirname(__file__) #getting the current path i.e where app.py is present
        #print("current path", basepath)
        filepath = os.path.join(basepath, 'uploads', f.filename) #from anywhere in the system we can give image but we want the
        #print("upload folder is", filepath)
        f.save(filepath)

        img = image.load_img(filepath, target_size=(224, 224))
        x = image.img_to_array(img) #img to array
        x = np.expand_dims(x, axis=0) #used for adding one more dimension
        #print(x)
        img_data = preprocess_input(x)
        prediction = np.argmax(model.predict(img_data))

        index = ['Darier_s disease', 'Muehrck-e_s lines', 'alopecia areata', 'beau_s lines', 'bluish nail',
                 'clubbing', 'eczema', 'half and half nails (Lindsay_s nails)', 'koilonychia', 'leukonychia',
                 'onycholysis', 'pale nail', 'red lunula', 'splinter hemorrhage', 'terry_s nail', 'white nail', 'yellow nails']
        nresult = str(index[prediction])
        nresult
```

Task – 16 : Application Execution

Finally, we will run the application.

```
""" Running our application """
if __name__ == "__main__":
    app.run(debug=False, port=8080)
```

Follow the steps outlined below to execute your Flask application:

1. Open the Anaconda prompt from the Start menu.
2. Navigate to the directory where your "app.py" file is located.
3. Type the command "python app.py" in the Anaconda prompt.
4. The local host where your application is running, typically indicated as <http://127.0.0.1:8080/>, will be displayed.
5. Copy the aforementioned local host URL and paste it into your preferred web browser. This action will direct you to the web page interface.
6. Proceed to input the necessary values, and subsequently click the "Predict" button to initiate the prediction process.
7. The resulting prediction will be displayed on the web page for your observation and analysis.

Task – 17 : Final Output Display

Prediction 1 :

PREDICTION

Our predictive models can detect early signs of diseases and health conditions from your health data to identify patterns and potential risks. Our technology-driven predictions are highly accurate, reducing false positives and negatives.

Upload Image Here To Identify the Disease

Choose...



Result: The classified Disease is : Yellow Nails

Prediction 2 :

PREDICTION

Our predictive models can detect early signs of diseases and health conditions from your health data to identify patterns and potential risks. Our technology-driven predictions are highly accurate, reducing false positives and negatives.

Upload Image Here To Identify the Disease

Choose...



Result: The classified Disease is : Beau_s Lines

Thus, the project is ended.

THANK YOU