

Early Stage Disease Diagnosis System Using Human Nail Image Processing Using Deep Learning

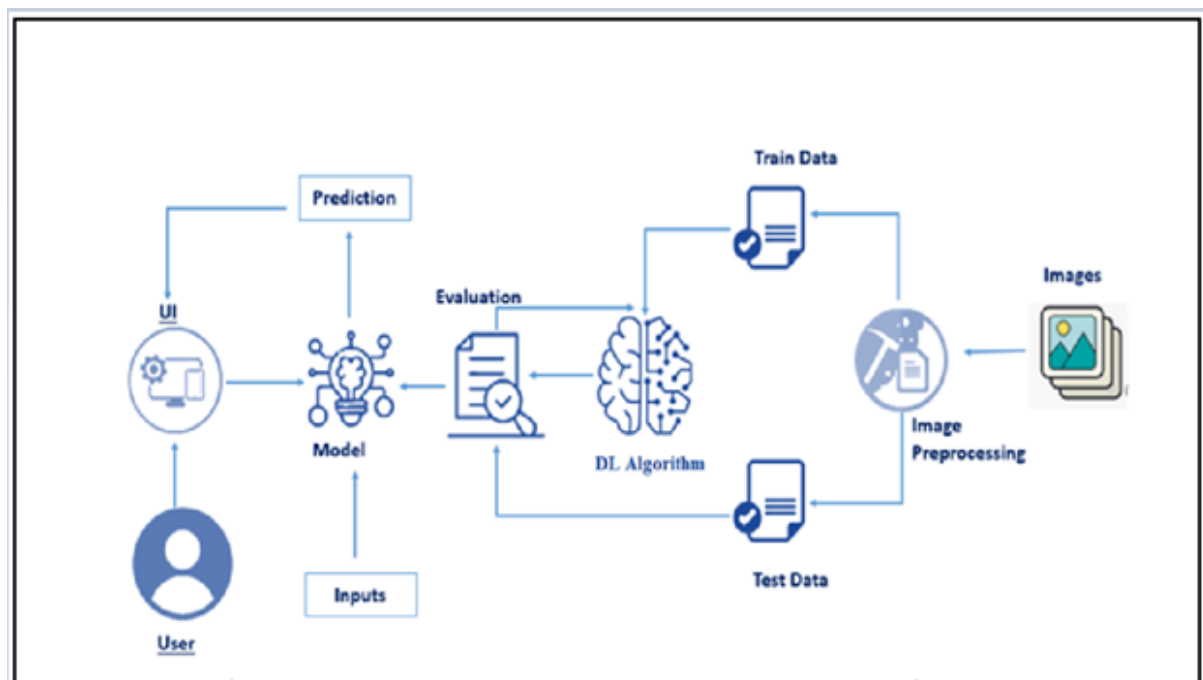
Develop an early stage disease diagnosis system using deep learning techniques to analyze human nail images. By analyzing nail characteristics and abnormalities, this project aims to detect and diagnose early signs of various diseases, aiding healthcare professionals and individuals in timely intervention and disease management.

Scenario 1 (Healthcare Providers): Implement the disease diagnosis system to assist healthcare professionals in early detection and diagnosis of nail-related diseases. Improve diagnostic accuracy, facilitate timely treatment, and enhance patient outcomes by detecting diseases at their early stages.

Scenario 2 (Individuals): Incorporate the nail image processing system into personal health monitoring applications for early detection of potential health issues. Empower individuals to monitor their nail health, detect abnormalities, and seek timely medical advice for preventive care and disease management.

Scenario 3 (Public Health Programs): Utilize the deep learning-based diagnosis system in public health initiatives for screening and monitoring nail health at the population level. Identify disease trends, implement preventive measures, and promote early intervention strategies to reduce the burden of nail-related diseases on public health.

Technical Architecture:



Prerequisites

To complete this project, you must require the following software, concepts, and packages

- Anaconda Navigator:
 - Refer to the link below to download Anaconda Navigator
- Python packages:
 - Open anaconda prompt as administrator
 - Type "pip install numpy" and click enter.
 - Type "pip install pandas" and click enter.
 - Type "pip install scikit-learn" and click enter.
 - Type "pip install matplotlib" and click enter.
 - Type "pip install scipy" and click enter.
 - Type "pip install seaborn" and click enter.
 - Type "pip install tensorflow" and click enter.
 - Type "pip install Flask" and click enter.

Prior Knowledge

You must have prior knowledge of the following topics to complete this project.

DL Concepts

- Neural Networks:: <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/>
- Deep Learning Frameworks:: <https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow>
- Transfer Learning: <https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a>
- VGG16: <https://www.geeksforgeeks.org/vgg-16-cnn-model/>
- Convolutional Neural Networks (CNNs): <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/> [s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning](https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning)
- Overfitting and Regularization: <https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/>
- Optimizers: <https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/>
- Flask Basics: https://www.youtube.com/watch?v=Ij4I_CvBnt0

Project Objectives

By the end of this project, you will:

- Know fundamental concepts and techniques used for Deep Learning.
- Gain a broad understanding of data.
- Know how to pre-process/clean the data using different data preprocessing techniques
- Know how to build a web application using the Flask framework.

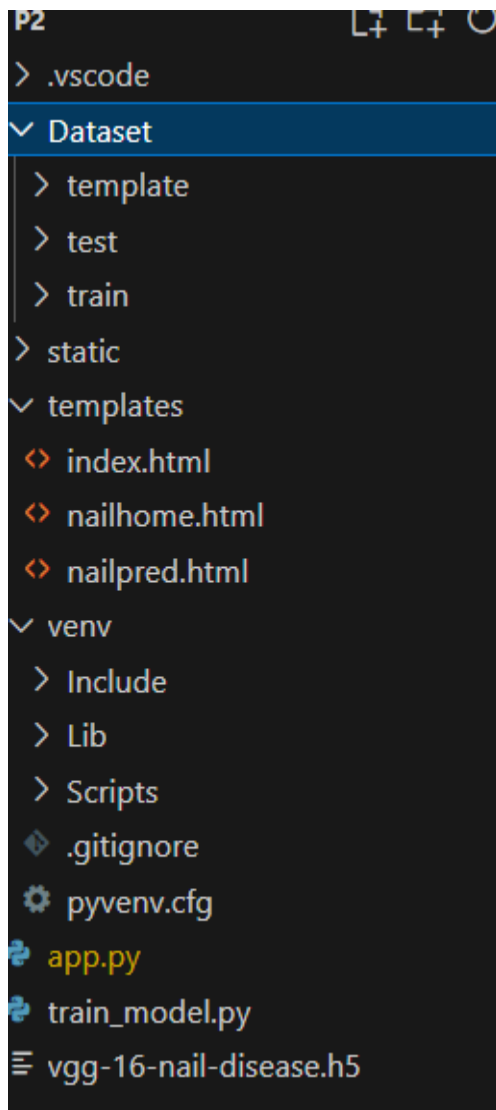
Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- VGG16 Model analyzes the image, then prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities listed below,

- Data Collection.
 - Create Train and Test Folders.
- Model Building
 - Importing the Model Building Libraries
 - Loading the model
 - Adding Flatten Layers
 - Adding Output Layer
 - Creating a Model object:
 - Configure the Learning Process
 - Import the ImageDataGenerator library
 - Configure ImageDataGenerator class
 - Apply ImageDataGenerator functionality to Trainset and Testset
- Training
 - Train the Model
 - Save the Model
- Testing
 - Test the model
- Application Building
 - Create an HTML file
 - Build Python Code
 - Run the application
 - Final Output

Project Structure



Create the Project folder which contains files as shown below

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- waste_classifier.h5 is our saved model. Further we will use this model for flask integration.
- Data Folder contains the Dataset used
- The Notebook file contains procedure for building the model.

1. Define Problem / Problem Understanding:

- Existing problem

Some of the existing solutions for solving this problem are: There are some existing systems which use color change of human palm for predicting disease. The system presented by Pandit et al. analyzes certain features in image and predicts probable disease using knowledge base of medical palmistry. Science of observing nails and palms to predict some diseases called Medical palmistry. Using example of the medical palmistry this model is discussed and presented for extraction of an interested portion of an image for further processing. This system presents work to get color of palm from the palm image and it works successfully on the different skin tones of human palm and increases accuracy of such observations of palm. The borders of palm are darker color than palm color in the scanned image of palm. Therefore exact boundary of palm may not found which can affect in the calculation of average color of palm. Image enhancement methods were used to overcome those problems. But this method is time consuming so to analyze nails it will take more time. So our system is using digital camera or mobile camera to get good quality image therefore it is easy to extract features of nails. There is another model, for nail color analysis which predicts diseases using digital image processing. The system presented by Hardik Pandit et. al observes the color of nails using the principles of medical science as basis and output of system is prediction of diseases, if found any. In healthcare domain doctors often observe human nails to as supporting information or symptoms in disease prediction. The same task is defined by the proposed model without any human intervention. The model gives more accurate results than human vision, because it overcomes the limitations of human eye like subjectivity and resolution power. As mentioned earlier different colors of nails indicate certain diseases. To implement this model, authors used computer based reference color values of ill nails to compare user's nail. That is, color values of nails of user's input images would be compared with these reference colors. If the case matches with any of reference color values, user would be victim of that disease. There will be 50 samples per color are taken for reference color values of ill nails. For example, for yellow nails, 50 different yellow nails color values are considered, and then their arithmetic mean is considered as a reference color value for yellow nails to RGB color components of reference color for that disease.

- Proposed solution:

There are different ways of disease diagnosis such as through various tests (blood test, urine test etc.) and symptoms available on various parts of body guides towards disease diagnosis. We are proposing a system which will take nail image as an input and output will be possible diseases prediction based on color changes. This is our first step towards a perfect system so we are not keen to accuracy of system. The main objective of this system design is to provide an application for use in healthcare domain this is

advantageous in terms of cost and time. The proposed system will take nail image as an input and will perform some processing on input image. Then finally it will predict probable diseases. This system can be used by people as well as by doctors in healthcare domain

2.Data Collection and Preparation:

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

2.1 Data Collection:

The data collection and preprocessing phase forms the foundation for accurate and reliable model training in the proposed early-stage disease diagnosis system using human nail images. The primary dataset used in this study was provided by **SmartBridge**, containing **17 classes** of nail diseases, including Alopecia Areata, Beau's Lines, Bluish Nails, Clubbing, Darier's Disease, Eczema, Lindsay's Nails, Koilonychia, Leukonychia, Muehrcke's Lines, Onycholysis, Pale Nails, Red Lunula, Splinter Hemorrhage, Terry's Nails, White Nails & Yellow Nails.

To address issues of class imbalance, similarity among certain disorders, and overfitting, a refined **Dataset2** with **12 consolidated classes** was created by merging visually similar diseases such as Alopecia Areata and Darier's Disease. Additionally, a **Healthy Nail** class was introduced to improve classification robustness.

Each image underwent several preprocessing steps to ensure consistency and enhance model generalization:

- **Resizing:** All images were resized to a uniform dimension of **224 × 224 × 3**, matching the input requirements of the VGG16 and DenseNet121 architectures.
- **Normalization:** Pixel values were scaled to a [0,1] range to stabilize training and improve convergence speed.
- **Data Augmentation:** Techniques such as random rotation, flipping, zooming, and brightness adjustments were applied to artificially expand the dataset and reduce overfitting.
- **Batch Normalization:** Implemented to standardize the input distribution across layers, enabling faster and more stable learning.
- **Whitening:** Applied to decorrelate pixel intensities, further improving feature extraction in convolutional layers.

These preprocessing techniques collectively enhanced image quality, improved learning efficiency, and ensured that the neural networks could generalize effectively across diverse nail image conditions.

2.2 Data Collection Plan and Raw Data Sources Identified

The datasets used for this research were obtained from the **SmartBridge AI Externship program**, supplemented by additional open-source medical image repositories and dermatological research archives, wherever necessary, to improve diversity and representativeness.

The data collection plan followed a structured approach:

1. **Dataset Identification:**
Identify existing datasets containing labeled nail images representing both healthy and diseased nail conditions.
2. **Source Verification:**
Validate the credibility and quality of datasets sourced from academic and clinical research repositories such as **ResearchGate**, **Kaggle**, and **DermNet**.
3. **Data Quality Assessment:**
Evaluate datasets based on image clarity, resolution, annotation accuracy, and class balance to ensure suitability for deep learning applications.
4. **Dataset Refinement:**
Curate and organize the final datasets (Dataset1 and Dataset2) based on visual similarity, disease relevance, and clinical significance.
5. **Ethical and Privacy Considerations:**
Ensure that all datasets used comply with open-access licensing and maintain patient anonymity.

Through this systematic data collection and preparation process, the study ensured a high-quality, diverse dataset suitable for training transfer learning models such as **VGG16**, **DenseNet121**, **ResNet50**, and hybrid architectures. This approach enhanced the robustness and diagnostic reliability of the proposed nail image disease classification system.

This dataset contains an additional class called 'HealthyNail', which also informs the patient that their nails are healthy too. We have made these classifications based on the visual characteristics of the diseases. Say, in most cases it is found that the visual characters of the nail diseases are similar for Alopecia Areata and Darier's Disease. Images Fig.1.a and Fig.1.b describes the same.















Fig1.a. Alopecia areata



Fig1.b. Darier's Disease

Si.No	Nail Disorder Name	Figure
1	Alopecia Areata	
2	Beau's Lines	
3	Bluish Nails	
4	Clubbing	
5	Darier's disease	
6	Half Moon Red Bands	
7	Healthy Nails	

8	Koilonychia	
9	Leukonychia	
10	Lindsay's Nail	
11	Muehrcke's Lines	
12	Onychogryphosis	
13	Onycholysis	
14	Nail Psoriasis	
15	Onychomycosis	
16	Splinter Hemorrhage	
17	Terry's Nails	
18	White Nails	
19	Yellow nails	

All preprocessing and training were conducted on **Google Colab**, leveraging GPU acceleration. The training and validation datasets were mounted from Google Drive using:

```
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

from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive', force_remount=True)

Google Drive was mounted in the Colab environment to access the image dataset stored in the drive. This allows the notebook to read images directly from cloud storage rather than local files, enabling easy sharing, backup, and scalability.

The VGG16 model, pre-trained on the large-scale ImageNet dataset, was imported without its top (fully connected) layers. This allowed us to reuse its convolutional base for feature extraction and customize the classification head for our nail disease dataset.

The **VGG16 model** was then fine-tuned on this dataset for **20 epochs**, resulting in a training accuracy of approximately **87.8%** and improved validation accuracy across epochs. The trained model was saved as:

```
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

This prepared model was later integrated into a Flask-based web application for disease prediction based on uploaded nail images.

2.3:Data Augmentation:

ImageDataGenerator was used to preprocess the images by normalizing pixel values and applying augmentation techniques. Augmentation created slightly altered versions of images (by rotating, zooming, and flipping), which effectively increased the dataset size.

Reason for Use:

These steps enhanced model generalization and robustness by reducing overfitting.

Normalization ensured faster gradient descent convergence, while augmentation helped the model learn invariant features under different orientations and lighting conditions.

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

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
# tell the model what cost and optimization method to use
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy'], run_eagerly=True
)
```

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

test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
training_set = train_datagen.flow_from_directory(trainPath,
                                                  target_size = (224, 224),
                                                  batch_size = 32,
                                                  class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(testPath,
                                             target_size = (224, 224),
                                             batch_size = 32,
                                             class_mode = 'categorical')
```

```
↔ Found 655 images belonging to 17 classes.
   Found 183 images belonging to 17 classes.
```

```
▶ training_set.class_indices  
  
{ '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 }
```

In this section, the dataset is divided into three parts — **training**, **validation**, and **testing** — to properly train and evaluate the model. The images are loaded from their respective directories using the `ImageDataGenerator` class, which is responsible for preprocessing and augmenting the data.

The **training data generator** applies several augmentation techniques such as rotation, width and height shifting, shearing, zooming, and horizontal flipping. These transformations artificially expand the training dataset, making the model more robust and less likely to overfit. Additionally, all pixel values are rescaled to the range `[0, 1]` for normalization.

For the **validation** and **test** datasets, only rescaling is performed to maintain consistency while evaluating model performance on unaltered images. Each image is resized to **224 × 224 pixels**, which is the standard input size for many pre-trained CNN architectures like **ResNet**, **VGG**, and **MobileNet**.

Finally, the class indices are printed to display the mapping between class names and their corresponding numeric labels. This step helps verify that the data has been correctly categorized before model training begins.

3.Split Data and Model Building

3.1:Train-Test-Split:

In this project, the dataset is divided into training and testing subsets to evaluate model performance effectively.

- The training data is stored in the path: `"/content/output_dataset/train"` and is used to train the model so it can learn patterns and features.
- The testing data is stored in the path: `"/content/output_dataset/test"` and is used to test the model's accuracy and generalization on unseen data.

This split ensures that the model doesn't just memorize the data but learns to make predictions effectively on new inputs.

```

Define Data Paths

This cell defines the paths to the training and test directories of the dataset.

Splitting

trainpath = "/content/output_dataset/train"
testpath="/content/output_dataset/test"

Initialize ImageDataGenerators (Alternative)

This cell initializes ImageDataGenerators for training and testing. The training generator includes data augmentation techniques like zooming and shearing, while the test generator only rescales the images.

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

```

3.2:Model Building:

Vgg16 Transfer-Learning Model:

The VGG16-based neural network is created using a pre-trained VGG16 architecture with frozen weights. The model is built sequentially, incorporating the VGG16 base, a flattening layer, dropout for regularization, and a dense layer with SoftMax activation for classification into three categories. The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss. During training, which spans 10 epochs, a generator is employed for the training data, and validation is conducted, incorporating call-backs such as Model Checkpoint and Early Stopping. The best-performing model is saved as `"waste_classifier.h5"` for potential future use. The model summary provides an overview of the architecture, showcasing the layers and parameters involved.

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

Downloading data from https://storage.googleapis.com/tensorflow/keras-applicati
58889256/58889256 1s 0us/step

for layer in vgg.layers:
    layer.trainable = False
    print(layer)

<InputLayer name=input_layer, built=True>
<Conv2D name=block1_conv1, built=True>
<Conv2D name=block1_conv2, built=True>
<MaxPooling2D name=block1_pool, built=True>
<Conv2D name=block2_conv1, built=True>
<Conv2D name=block2_conv2, built=True>
<MaxPooling2D name=block2_pool, built=True>
<Conv2D name=block3_conv1, built=True>
<Conv2D name=block3_conv2, built=True>
<Conv2D name=block3_conv3, built=True>
<MaxPooling2D name=block3_pool, built=True>
<Conv2D name=block4_conv1, built=True>
<Conv2D name=block4_conv2, built=True>
<Conv2D name=block4_conv3, built=True>
<MaxPooling2D name=block4_pool, built=True>
<Conv2D name=block5_conv1, built=True>
<Conv2D name=block5_conv2, built=True>
<Conv2D name=block5_conv3, built=True>
<MaxPooling2D name=block5_pool, built=True>
```

```
<Conv2D name=block4_conv1, built=True>
<Conv2D name=block4_conv2, built=True>
<Conv2D name=block4_conv3, built=True>
<MaxPooling2D name=block4_pool, built=True>
<Conv2D name=block5_conv1, built=True>
<Conv2D name=block5_conv2, built=True>
<Conv2D name=block5_conv3, built=True>
<MaxPooling2D name=block5_pool, built=True>
```

Start coding or generate with AI.

Add Custom Classification Layer

This cell adds a Flatten layer followed by a Dense layer with 3 units (for the 3 classes: Biodegradable, Recyclable, Trash) and a 'softmax' activation function to the output of the VGG16 model. This creates the new classification head for our model.

```
output=Dense(3,activation='softmax')(Flatten()(vgg.output)) #
```

Create the Final Model

This cell creates the final model by combining the pre-trained VGG16 base model with the newly added classification layers.

```
vgg16=Model(vgg.input,output)
```

A **Flatten** layer is then added to convert the extracted feature maps into a one-dimensional vector, followed by a **Dense** layer with **3 output neurons** and a **softmax activation function**.

This final layer is responsible for classifying images into three categories based on the dataset used.

```
# view the structure of the model
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808

Finally, the **VGG16 model** is redefined using the Model() function, combining the pre-trained base and the newly added output layers. The vgg16.summary() function provides a detailed overview of the model's structure, including the number of layers, parameters, and connections.

3.3 :Model Compilation and Training:

After defining the VGG16-based model, it is compiled and trained using appropriate optimization and regularization techniques to improve performance and prevent overfitting.

Finally, the model is trained using the **fit()** function for **20 epochs**, with both training and validation datasets provided. The training process records performance metrics such as loss and accuracy, which can later be visualized to assess model learning behavior.

```
import sys
# fit the model
r = model.fit(
    training_set,
    validation_data=test_set,
    epochs=20,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set))
```

`/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `Py``
`self.warn_if_super_not_called()`

Epoch	Time	Step	Accuracy	Loss	Val Accuracy	Val Loss
1/20	410s	19s/step	0.0995	3.3385	0.2404	2.9676
2/20	16s	753ms/step	0.2467	2.5060	0.4426	2.0553
3/20	20s	738ms/step	0.3868	2.1244	0.5301	1.6953
4/20	16s	737ms/step	0.4293	1.8591	0.5792	1.5021
5/20	16s	743ms/step	0.4996	1.7198	0.6230	1.3696
6/20	16s	738ms/step	0.5249	1.5215	0.6503	1.2481
7/20	21s	746ms/step	0.6580	1.2929	0.6011	1.5746
8/20	16s	739ms/step	0.6177	1.4035	0.6503	1.1764
9/20	16s	750ms/step	0.6298	1.2870	0.7432	1.0553
10/20	16s	747ms/step	0.6302	1.2332	0.7541	0.8446

4. Testing Model & Data Prediction:

4.1: Testing the model

After successfully training the **VGG16-based nail disease classification model**, its performance was evaluated using individual images from the **test dataset** to verify the accuracy of real-time predictions. This testing phase ensured that the model could correctly identify different nail diseases based on their visual patterns and color characteristics.

For each test image, the following steps were performed:

1. **Image Loading and Resizing:**

The image was loaded from the test dataset directory and resized to **224 × 224 × 3 pixels**, which matches the input size required by the VGG16 model.

2. **Image Conversion and Preprocessing:**

The loaded image was converted into a NumPy array and preprocessed using the `preprocess_input()` function to scale the pixel values and normalize them according to VGG16's trained format.

3. Prediction Using the Trained Model:

The preprocessed image was passed through the trained **VGG16 model**, which generated prediction probabilities across all **17 nail disease classes** such as Beau's Lines, Clubbing, Koilonychia, Leukonychia, Yellow Nails, etc.

4. Identifying the Predicted Class:

The class label with the **highest probability** was selected as the final prediction. This was achieved using the `np.argmax()` function, which identifies the index of the most likely class from the output array.

5. Displaying the Result:

The corresponding nail disease name was then displayed, verifying whether the model could accurately recognize the disease pattern from the given nail image.

This process confirmed that the trained model can correctly classify real-time nail images into their respective disease categories with high accuracy before deployment.

```
#load saved model file
model=load_model('vgg-16-nail-disease.keras')
```

```
#load one random image from local system
img=image.load_img('/content/drive/MyDrive/Dataset/test/clubbing/10.PNG',target
```

```
#convert image to array format
x = image.img_to_array(img)
```

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

```
x.shape
```

```
(224, 224, 3)
```

```
x = np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
print(img_data.shape)
```

```
(1, 224, 224, 3)
```

```
import matplotlib.pyplot as plt

x.shape

(224, 224, 3)

x = np.expand_dims(x,axis=0)
img_data=preprocess_input(x)
print(img_data.shape)

(1, 224, 224, 3)

model.predict(img_data)

1/1 ————— 1s 507ms/step
array([[0.0000000e+00, 4.9521736e-38, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 4.3147524e-15,
        0.0000000e+00, 0.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00, 1.0000000e+00, 0.0000000e+00, 0.0000000e+00,
        0.0000000e+00]], dtype=float32)

output=np.argmax(model.predict(img_data), axis=1)

1/1 ————— 0s 48ms/step
```

4.2:Model Evaluation on Test Dataset

After the training process, the **VGG16-based nail disease classification model** was evaluated on the **test dataset** to measure its performance on **unseen images**.

This step helps determine how well the model generalizes beyond the training data and provides a quantitative assessment of its classification ability.

The **evaluate()** function from Keras was used to compute key performance metrics such as **loss** and **accuracy** on the test dataset. These metrics provide a clear understanding of how effectively the model distinguishes between different nail diseases like Beau's Lines, Clubbing, Koilonychia, Leukonychia, and Yellow Nails.

```
# Evaluate the trained VGG16 nail disease model
test_loss, test_accuracy = model.evaluate(test_set)

# Display the results
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

6/6 ————— 2s 230ms/step - accuracy: 0.9109 - loss: 0.4361
Test Loss: 0.4739299714565277
Test Accuracy: 0.8961748480796814

4.3: Training and Validation Performance Visualization

During the training of the **VGG16-based nail disease classification model**, the learning progress was continuously monitored using **training and validation accuracy and loss plots**. These visualizations help to understand how well the model learns over time and whether it is overfitting, underfitting, or converging effectively.

The **accuracy plot** displays how the model's classification accuracy improves on both the training and validation datasets across each epoch.

The **loss plot**, on the other hand, shows how the model's prediction error decreases as training progresses, providing insights into the optimization and learning behavior.

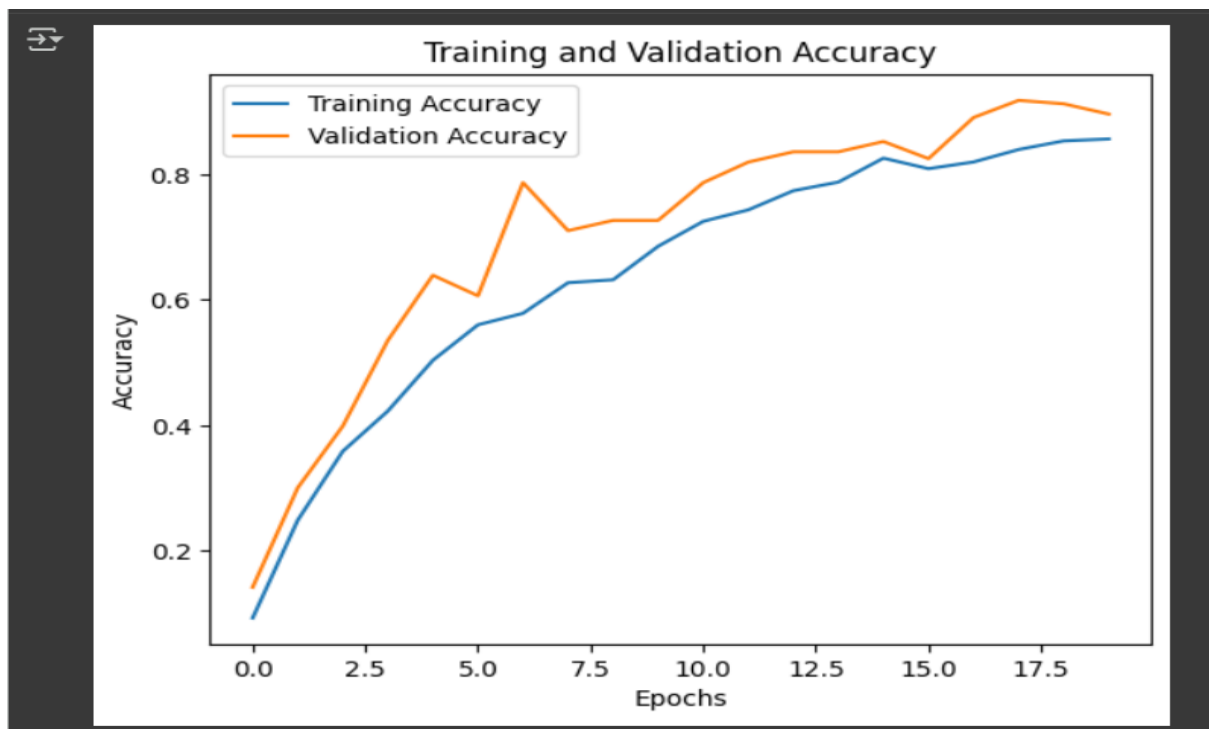
These plots provide a visual representation of the model's learning behavior, making it easier to assess convergence, evaluate performance, and make decisions about further tuning or training.

Default title text

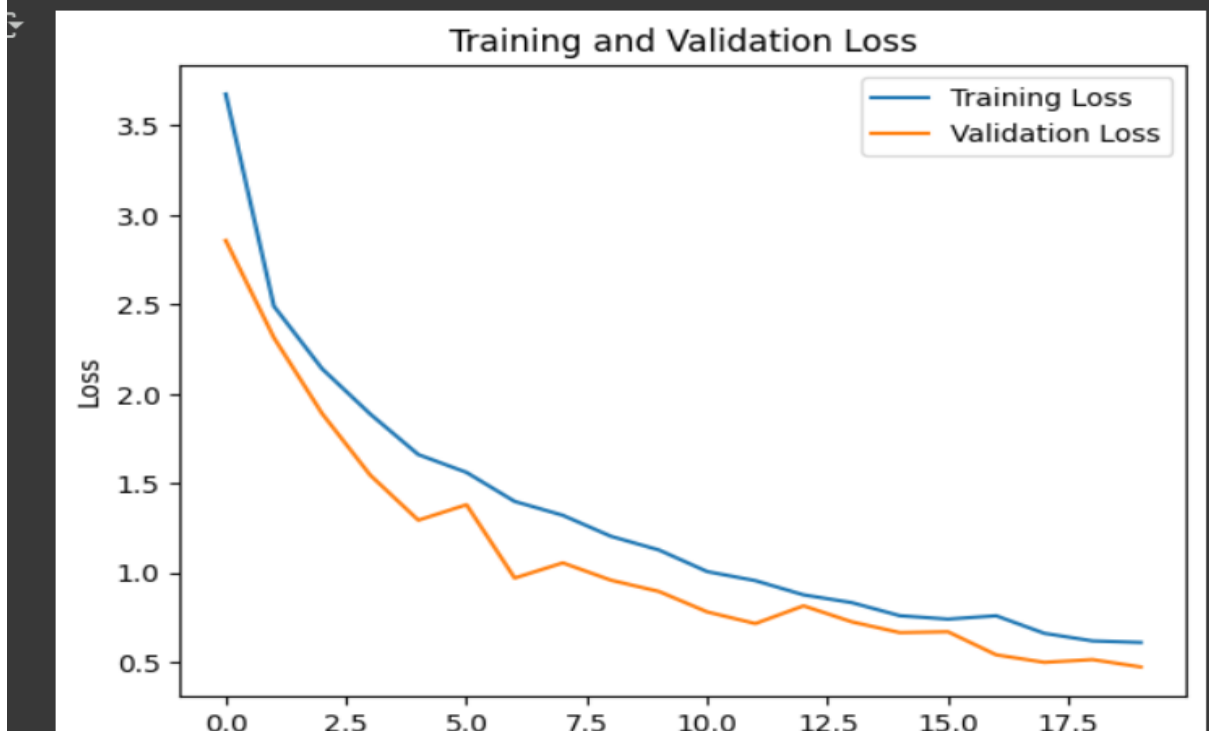
```
# @title Default title text
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(r.history['accuracy'], label='Training Accuracy')
plt.plot(r.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
```



```
plt.plot(r.history['loss'], label='Training Loss')
plt.plot(r.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



5.Saving the model:

After successfully training and evaluating the **VGG16-based nail disease classification model**, the trained model was saved for future use to avoid retraining.

Saving the model allows it to be reloaded and used for **real-time disease prediction** on new nail images without repeating the computationally expensive training process.

The model was stored in the **HDF5 format (.keras)**, which preserves both the **model architecture** and the **learned weights**. This ensures that the model can be easily reloaded later for deployment or additional evaluation.

Additionally, the **class indices** — which represent the mapping between disease class names (e.g., Clubbing, Beau's Lines, Leukonychia) and their corresponding numeric labels — were saved in a **JSON file**.

This mapping is essential during inference to translate the model's numerical predictions back into their actual disease labels.

```
import json

# Save the trained model
model.save('vgg16-nail-disease.keras')

# Save class indices for reference during prediction
class_indices = training_set.class_indices
with open('class_indices.json', 'w') as f:
    json.dump(class_indices, f)

print("Model and class indices have been saved successfully.")
```

Model and class indices have been saved successfully.

6.Application Building:

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the users where they can upload image for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

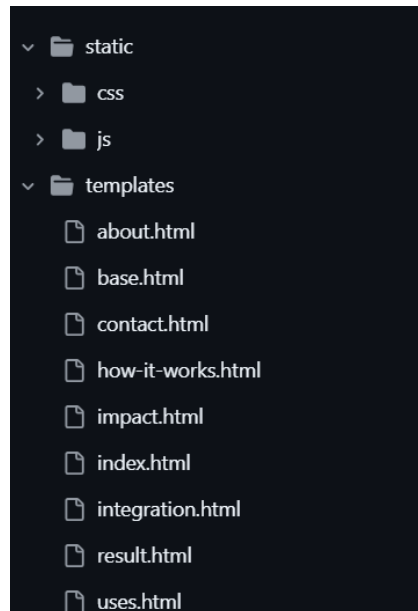
This section has the following tasks

- Building HTML Pages
- Building server-side script

6.1:Building HTML Pages:

For this project create HTML files namely

Building HTML Pages:



For this project create three HTML files namely

- index.html
- nailhome.html
- nailpred.html and all other supporting files

And save them in the templates folder.

6.2: Build Python code:

Import the libraries

```
1
2 import os
3 import json
4 import numpy as np
5 from flask import Flask, render_template, request, redirect, url_for, flash
6 from tensorflow.keras.models import load_model
7 from tensorflow.keras.preprocessing import image
8 from werkzeug.utils import secure_filename
9 # from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
10 from tensorflow.keras.applications.vgg16 import preprocess_input
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (`__name__`) as argument.

```

BASE_DIR = os.path.dirname(os.path.abspath(__file__)) # Project base folder
MODEL_PATH = os.path.join(BASE_DIR, 'Vgg-16-nail-disease.h5')
UPLOAD_FOLDER = os.path.join(BASE_DIR, 'static', 'uploads')

# Ensure upload folder exists
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# =====
# 2 Load the trained model
# =====
model = load_model(MODEL_PATH)

# Class labels
CLASS_NAMES = [
    '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 hemmorrhage',
    'terry_s nail', 'white nail', 'yellow nails'
]

```

```

# Flask app
app = Flask(__name__)
app.config["UPLOAD_FOLDER"] = "static/uploads"
app.secret_key = "cleantech_secret_key" # for flash messages

# Load model & class indices
model = load_model("waste_classifier.h5")
with open("class_indices.json") as f:
    class_indices = json.load(f)
class_labels = [class_indices[str(i)] for i in range(len(class_indices))]

```

Here we will be using the declared constructor to route to the HTML page which we have created earlier.

```

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return "No file uploaded"

    file = request.files['file']
    if file.filename == '':
        return "No selected file"

    # Save uploaded image
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
    file.save(filepath)

    # Preprocess image
    img = image.load_img(filepath, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.0


```

Retrieves the value from UI:

```
# Helper function: predict uploaded image
def predict_image(img_path):
    img = image.load_img(img_path, target_size=(224,224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)

    preds = model.predict(img_array)
    idx = np.argmax(preds[0])
    return class_labels[idx], float(np.max(preds[0]))
```

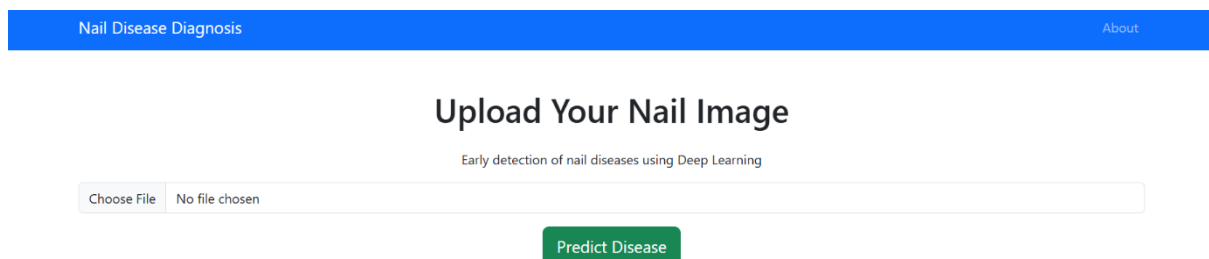
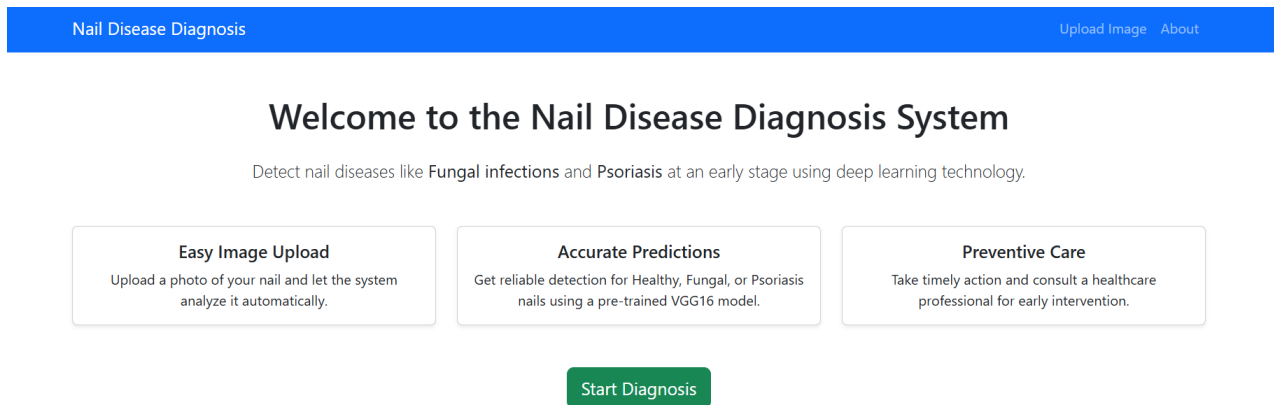
Main Function:

```
# =====
#  Run app
# =====
if __name__ == '__main__':
    app.run(debug=True)
```

- **Run the web application**
- Run the application
- Open Anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type the “python app.py” command
- Navigate to the local host where you can view your web page.
- Click on the inspect button from the top right corner, enter the inputs, click on the classify image button, and see the result/prediction on the web.

```
NOTE: If you're using GPUs in performance-critical operations:
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow
with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
you train or evaluate the model.
* Serving Flask app 'app'
* Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server inst
ad.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.146.22.133:5000
INFO:werkzeug:Press CTRL+C to quit
```


UI Image preview:



Prediction Result



Predicted Disease: **bluish nail**

[Analyze Another Image](#)

Prediction Result



Predicted Disease: **leukonychia**

[Analyze Another Image](#)