PROJECT NAME- Early Stage Disease Diagnosis System Using Human Nail Image Processing

Team ID- SWTID1750058607

Team Members- 1. Satyam Mittal

2. **Ekant Aman**
3. **Diya Baidya**
4. **Progya Paromita Biswas**

Date- 20/06/2025

Early Stage Disease Diagnosis System Using Human Nail Image Processing

Problem statement

Human nails often show early signs of systemic diseases such as diabetes, anemia, and thyroid disorders, but these indicators are frequently ignored. This project develops a deep learning model that analyzes nail images to detect potential health issues by recognizing patterns like discoloration, ridges, and texture changes, offering a non-invasive and accessible tool for early disease screening.

Model name and why it is used?

VGG16 model is used.

VGG16 is a deep CNN architecture known for its simplicity and effectiveness as a feature extractor, especially when utilized with transfer learning on datasets that are not as large as ImageNet (like a specialized medical image dataset). It provides a robust starting point for classifying complex visual patterns in images, making it suitable for "Early Stage Disease Diagnosis System Using Human Nail Image Processing."

Technology Stack

Deep Learning

Deep learning is a subset of machine learning that utilizes artificial neural networks with multiple layers (i.e., "deep" architectures) to learn complex, hierarchical representations and patterns directly from raw data, such as images, audio, or text, without explicit feature engineering.

Transfer Learning

Transfer learning is a machine learning technique where a model, which has been pre-trained on a large dataset for a particular task, is reused as the starting point for a different but related task. Instead of training a new model from scratch, the knowledge (learned features and parameters) from the pre-trained model is "transferred" and fine-tuned for the new, often smaller, dataset or specific problem.

Python

Python is a high-level, general-purpose, interpreted programming language known for its simplicity, readability, and extensive standard library. It supports multiple programming paradigms, including object-oriented, imperative, and functional programming, and is widely used in web development, data analysis, artificial intelligence, scientific computing, automation, and more.

Tensor flow

TensorFlow is an open-source machine learning framework developed by Google. It is a comprehensive ecosystem of tools, libraries, and community resources that allows developers and researchers to build, train, and deploy machine learning models, particularly deep neural networks, across various platforms.

Keras

Keras is a high-level, open-source neural networks API (Application Programming Interface) written in Python, designed for rapid experimentation with deep neural networks. It provides a user-friendly and modular interface that runs on top of popular deep learning frameworks like TensorFlow (its primary backend), Theano, or CNTK, allowing for quick and easy prototyping of deep learning models.

OpenCV

OpenCV (Open Source Computer Vision Library) is a vast, open-source library of programming functions primarily aimed at real-time computer vision. It provides a comprehensive set of tools for various image and video processing tasks, including image manipulation, object detection, facial recognition, feature extraction, and machine learning algorithms related to visual data.

Medical Image Processing

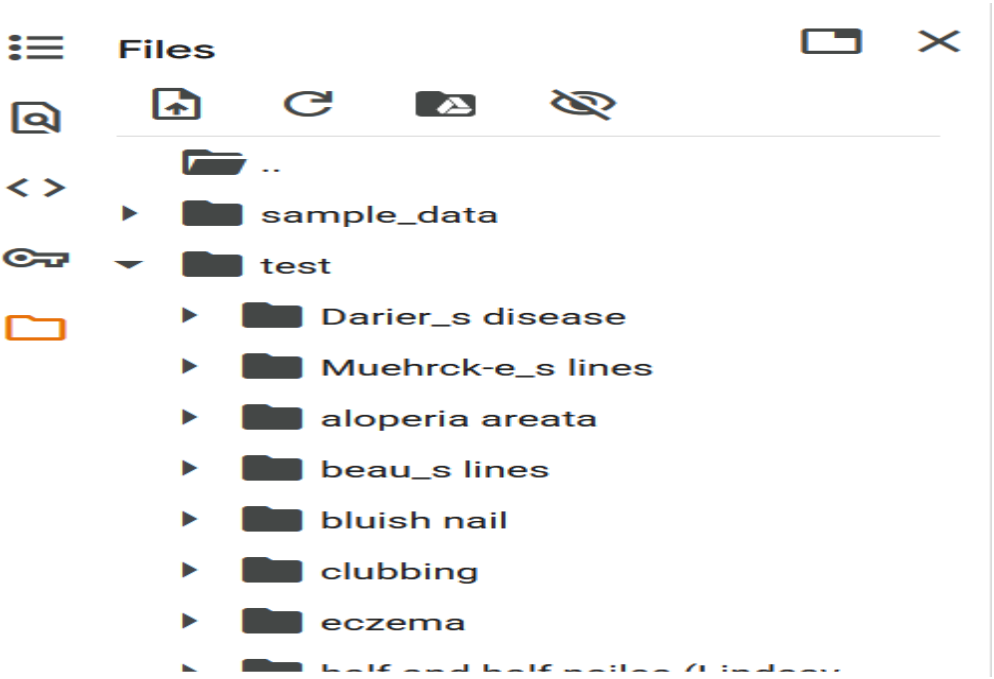
Medical Image Processing refers to the application of computational techniques and algorithms to analyze, enhance, manipulate, and extract meaningful information from images generated by medical imaging modalities (such as X-rays, MRI, CT, ultrasound, or clinical photographs like nail images). The primary goals are to aid in disease diagnosis, treatment planning, research, and monitoring patient conditions by improving image quality, segmenting structures, quantifying features, and visualizing medical data.

Steps used:

- Importing DataBase
- Unzip the folder and get ready with train and test Data
- Study the dataset and make the data ready for the Model(Data Preprocessing)
- Image Scalling
- Importing the VGG16 Model, and make it ready to fit the data
- Fit the model in our Dataset and analyse its performance
- Get the accuracy and compare the training accuracy and Validation accuracy
- Fit the model in test data and see the performance
- Get a new image and predict the disease
- Save the model
- Make a UI and make ready for the user(Needed Flask)

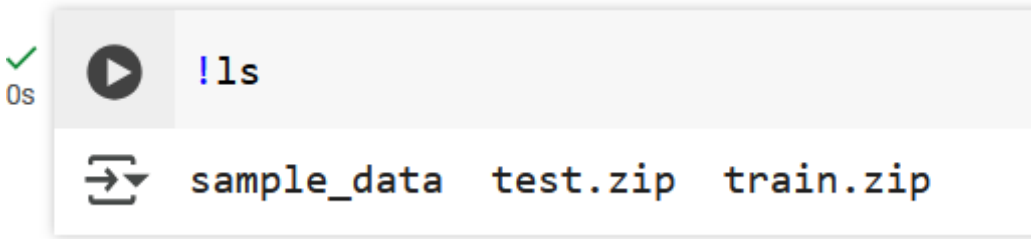
Code Explaination:

Activity 1. Collecting the Dataset



Activity 1.1 Listing and Unzip the files

!ls – Lists files in the current folder.



ZipFile – Opens the train.zip file in read mode.

extractall() – Extracts all files from the ZIP to the current directory.

print() – Displays a message after successful extraction.

Opens test.zip file in read mode ('r').

Extracts all files from the ZIP to the current folder.

Prints "Test Dataset extracted" after successful extraction.

UnZip the train and test data

0s

```
from zipfile import ZipFile

with ZipFile('train.zip','r') as zip:
  zip.extractall()
print("Train Dataset extracted")
```

```
Train Dataset extracted
```

0s

```
[4] from zipfile import ZipFile

with ZipFile('test.zip','r') as zip:
  zip.extractall()
print("Test Dataset extracted")
```

```
Test Dataset extracted
```

Activity 1.2 Counting the files of train and test

Counts total images in the /content/train folder (including subfolders).

os.walk() loops through all files and directories.

Adds the number of files to file_count_train.

Prints each folder path and the total number of images at the end.

```
[5]  #Counting number of images on train data and watching the number of classes
     import os
     file_count_train = 0
     for path, dir, files in os.walk('/content/train'):
       file_count_train+=len(files)
       print(path)
     print("Number of images in train:",file_count_train)
```

```
/content/train
/content/train/white nail
/content/train/aloperia areata
/content/train/red lunula
/content/train/splinter hemmorrage
/content/train/koilonychia
/content/train/terry_s nail
/content/train/clubbing
/content/train/Darier_s disease
/content/train/onycholycis
/content/train/beau_s lines
/content/train/Muehrck-e_s lines
/content/train/bluish nail
/content/train/leukonychia
/content/train/pale nail
/content/train/half and half nailes (Lindsay_s nails)
/content/train/eczema
/content/train/yellow nails
Number of images in train: 655
```

Counts total images in the /content/test folder.

Uses os.walk() to loop through all files and folders.

Adds file count to file_count_test.

Prints each folder path and the total number of test images at the end.

```
[6]  #Counting number of images on test data and watching the number of classes
     import os
     file_count_test =0
     for path, dir, files in os.walk('/content/test'):
       file_count_test+=len(files)
       print(path)
     print("Number of images in test:",file_count_test)
```

```
/content/test
/content/test/white nail
/content/test/aloperia areata
/content/test/red lunula
/content/test/splinter hemmorrage
/content/test/koilonychia
/content/test/terry_s nail
/content/test/clubbing
/content/test/Darier_s disease
/content/test/onycholycis
/content/test/beau_s lines
/content/test/Muehrck-e_s lines
/content/test/bluish nail
/content/test/leukonychia
/content/test/pale nail
/content/test/half and half nailes (Lindsay_s nails)
/content/test/eczema
/content/test/yellow nails
Number of images in test: 183
```

THERE ARE 655 IMAGES IN TRAIN AND 183 IN TEST DATA

Activity 2. Checking Dimensions

Imports Matplotlib libraries for plotting and image handling.

Reads an image (1.PNG) from the Darier's disease folder.

Displays the image using imshow().

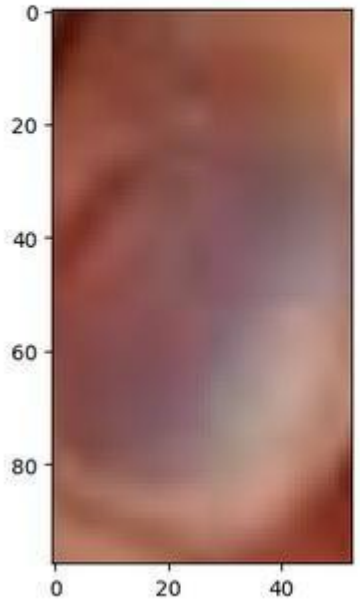
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/train/Darier_s disease/1.PNG')
imgplot = plt.imshow(img)
imgplot
```

```
<matplotlib.image.AxesImage at 0x7ce64992a190>
```

Loads and displays an image (12.PNG) from the bluish nail class in the test dataset using Matplotlib.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/test/bluish nail/12.PNG')
imgplot = plt.imshow(img)
imgplot
```

```
<matplotlib.image.AxesImage at 0x7ce631602190>
```



Activity 2.1 Train and test folder made and resize the folder of train and test

Creates a new folder named train.resize.

If the folder already exists, no error occurs (exist_ok=True).

Prints "Train resized folder made".

Creates a folder named test.resize.

No error if the folder already exists (exist_ok=True).

Prints "Train resized folder made" (Note: message should say "Test resized folder made" for clarity).

```
[9] import os
    os.makedirs("train.resize", exist_ok=True)
    print("Train resized folder made")
```

```
Train resized folder made
```

```
import os
os.makedirs("test.resize", exist_ok=True)
print("Train resized folder made")
```

```
Train resized folder made
```

```
[11] from PIL import Image
     import os

     original_folder = '/content/train'
     resized_folder = '/content/train.resize'

     for path, dir, files in os.walk(original_folder):
       for folder in dir:
         # Create the corresponding folder in resized_folder
         os.makedirs(resized_folder + '/' + folder, exist_ok=True)

         ImageFiles = os.listdir(original_folder + '/' + folder)

         for image in ImageFiles:
           image_path = original_folder + '/' + folder + '/' + image

           try:
             img = Image.open(image_path)
             img = img.resize((224, 224))
             img = img.convert("RGB")

             # Save to the correct file path
             save_path = resized_folder + '/' + folder + '/' + image
             img.save(save_path)

           except Exception as e:
             print(f"Skipped {image_path}: {e}")
     print('Training Data resized Completed')
```

```
Training Data resized Completed
```

Resizes all images in the train folder to 224x224 pixels.

Converts them to RGB format.

Saves the resized images into the train.resize folder, keeping the same subfolder structure.

Skips and prints an error if any image can't be processed.

Resizes all images in the test folder to 224x224 pixels.

Converts them to RGB format.

Saves the resized images into the test.resize folder, maintaining the same folder structure.

Skips and reports any image that causes an error.

```
from PIL import Image
import os

original_folder = '/content/test'
resized_folder = '/content/test.resize'

for path, dir, files in os.walk(original_folder):
  for folder in dir:
    # Create the corresponding folder in resized_folder
    os.makedirs(resized_folder + '/' + folder, exist_ok=True)

    ImageFiles = os.listdir(original_folder + '/' + folder)

    for image in ImageFiles:
      image_path = original_folder + '/' + folder + '/' + image

      try:
        img = Image.open(image_path)
        img = img.resize((224, 224))
        img = img.convert("RGB")

        # Save to the correct file path
        save_path = resized_folder + '/' + folder + '/' + image
        img.save(save_path)

      except Exception as e:
        print(f"Skipped {image_path}: {e}")
print("Test Data resized completed")
```

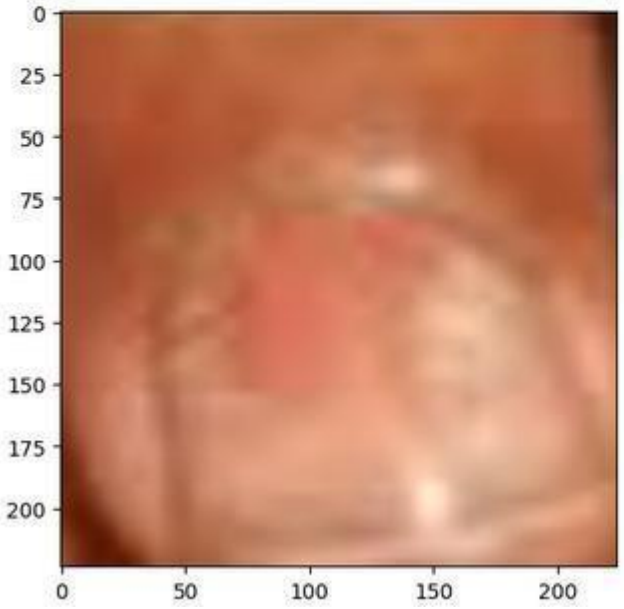
```
Test Data resized completed
```

Activity 2.2 Checking dimension of resized folder

Loads and displays the resized image 10.PNG from the aloperia areata class in the train.resize folder using Matplotlib.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/train.resize/aloperia areata/10.PNG')
imgplot = plt.imshow(img)
imgplot
```

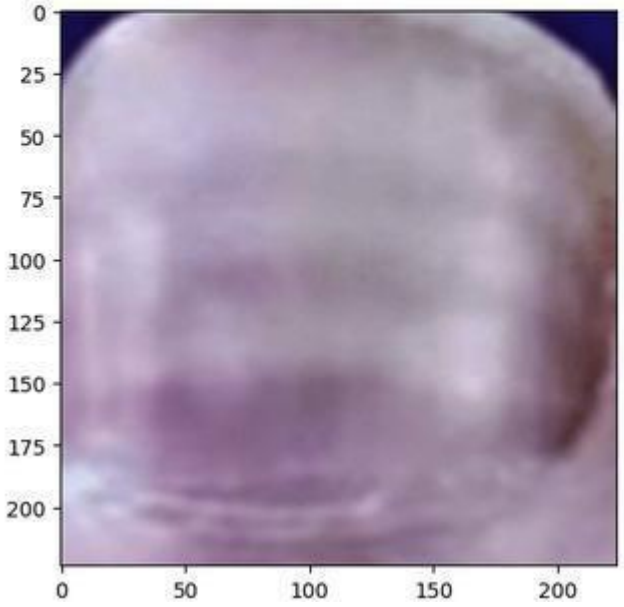
```
<matplotlib.image.AxesImage at 0x7ce6316c2850>
```



Loads and displays the resized image 1.PNG from the Muehrcke's lines class in the test.resize folder using Matplotlib

```
[14] import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     img = mpimg.imread('/content/test.resize/Muehrck-e_s lines/1.PNG')
     imgplot = plt.imshow(img)
     imgplot
```

```
<matplotlib.image.AxesImage at 0x7ce631741310>
```



Activity 2.3 Counting the image resized

Counts total images in the train.resize folder (including subfolders).

Uses os.walk() to loop through all files.

Prints the total number of resized train images.

Counts total images in the test.resize folder (including subfolders).

Uses os.walk() to loop through all files.

Prints the total number of resized test images (Note: print message says "train resized" by mistake; should say "test resized").

```
[15] #Counting number of images on train_resized data
     import os
     file_count_train_reseized = 0
     for path, dir, files in os.walk('/content/train.resize'):
       file_count_train_reseized+=len(files)
       # print(path)
     print("Number of images in train resized:",file_count_train_reseized)
```

```
Number of images in train resized: 655
```

```
#Counting number of images on test_resized data
import os
file_count_test_reseized = 0
for path, dir, files in os.walk('/content/test.resize'):
  file_count_test_reseized+=len(files)
  # print(path)
print("Number of images in train resized:",file_count_test_reseized)
```

```
Number of images in train resized: 183
```

Activity 3 Pre processing the data

Loads and preprocesses all train images from train.resize folder.

Stores image data in X_train and labels (as numbers) in y_train.

Uses a label map to assign numeric labels to each class.

Converts lists to NumPy arrays and shuffles the data.

Prints the shape of X_train and y_train.

```
# from PIL import Image
# import numpy as np
# import os
# from sklearn.utils import shuffle
# from tensorflow.keras.applications.vgg16 import preprocess_input

# ResizedFolderPath = '/content/train.resize'
# Train_subfolders = sorted(os.listdir(ResizedFolderPath))

# Labels_map = {folder: idx for idx, folder in enumerate(Train_subfolders)}
# print("Label map:", Labels_map)

# X_train = []
# y_train = []

# for folder in Train_subfolders:
#     folderPath = os.path.join(ResizedFolderPath, folder)
#     for image_name in os.listdir(folderPath):
#         image_path = os.path.join(folderPath, image_name)
#         if image_name.lower().endswith(('.png', '.jpg', '.jpeg')):
#             try:
#                 img = Image.open(image_path).resize((224, 224)).convert('RGB')
#                 img_array = np.array(img)
#                 img_array = preprocess_input(img_array)
#                 X_train.append(img_array)
#                 y_train.append(Labels_map[folder])
#             except Exception as e:
#                 print(f"Skipped {image_path}: {e}")

# X_train = np.array(X_train)
# y_train = np.array(y_train)


# X_train, y_train = shuffle(X_train, y_train, random_state=42)

# print("X_train shape:", X_train.shape)
# print("y_train shape:", y_train.shape)
```

Activity 4 Importing the VGG16 Model, and make it ready to fit the data

Loads VGG16 (pre-trained on ImageNet) without its top layer.

Freezes VGG16 layers (to keep pre-trained features).

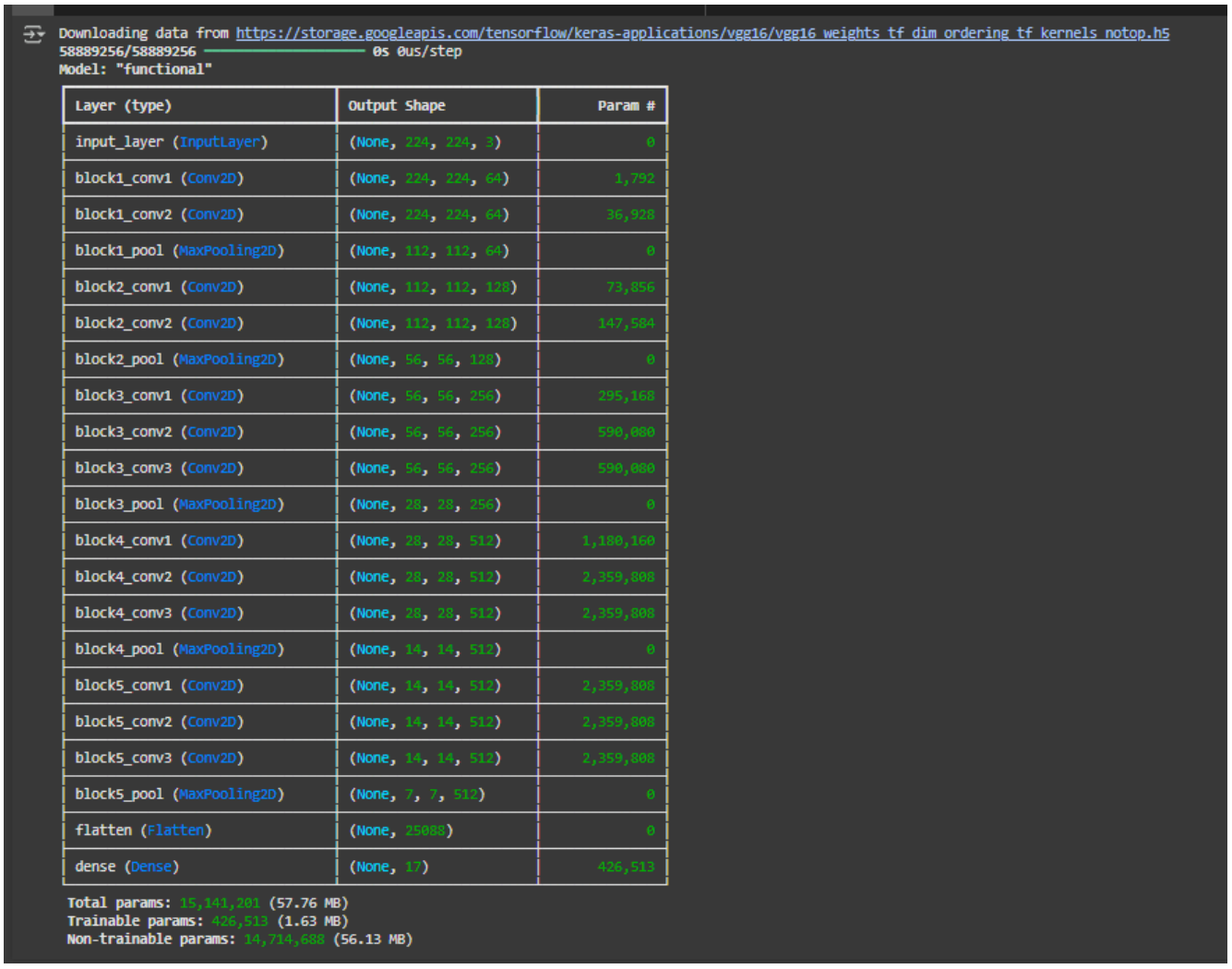
Adds custom layers:

- ➤ Flatten,
- ➤ Dense (256 units + ReLU + L2 regularization),
- ➤ Dropout (50%),
- ➤ Dense (output layer with softmax for classification).

Compiles the model with Adam optimizer and sparse categorical cross-entropy loss.

Prints the model summary.

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


# base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in vgg.layers:
    layer.trainable = False

x = Flatten()(vgg.output)
# x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001))(x)
# x = Dropout(0.5)(x)
predictions = Dense(17, activation='softmax')(x)

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



model.summary()
```



```
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
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
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 17)	426,513

```
Total params: 15,141,201 (57.76 MB)
Trainable params: 426,513 (1.63 MB)
Non-trainable params: 14,714,688 (56.13 MB)
```

Activity 5: We used Keras' ImageDataGenerator to perform data augmentation and automatically generate labels for our dataset. The training images are augmented by applying random transformations such as rescaling, shearing, zooming, and horizontal flipping to improve model generalization. The images are loaded from folders named after their classes, allowing flow_from_directory to automatically assign labels based on folder names and prepare the data in batches suitable for training. For testing, only rescaling is applied to normalize the images without augmentation. This process helps increase data diversity and simplifies label creation.

```
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)


train_set = train_datagen.flow_from_directory('/content/train.resize',
                                              target_size = (224, 224),
                                              batch_size = 32,
                                              class_mode = 'categorical')
test_set = test_datagen.flow_from_directory('/content/test.resize',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
train_set.class_indices
```

```
Found 655 images belonging to 17 classes.
Found 183 images belonging to 17 classes.
{'Darier_s disease': 0,
 'Muehrck-e_s lines': 1,
 'aloperia 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 hemmorrage': 13,
 'terry_s nail': 14,
 'white nail': 15,
 'yellow nails': 16}
```

Activity 6 Fit the model in our Dataset and analyse its performance

Uses EarlyStopping to stop training if validation accuracy doesn't improve for 3 epochs, restoring the best weights.

After training, prints the final training and validation accuracy (formatted to 4 decimal places).

```python
from tensorflow.keras.callbacks import EarlyStopping
# from sklearn.utils import class_weight

early_stop = EarlyStopping(
    monitor='val_accuracy',
    mode= 'max',
    patience=3,
    restore_best_weights=True
)

history =model.fit(train_set,validation_data=test_set, epochs=30, steps_per_epoch = len(train_set)//3, validation_steps = len(test_set)//3)
# Get final epoch's training and validation accuracy
train_acc = history.history['accuracy'][-1]
val_acc = history.history['val_accuracy'][-1]

# Print formatted values without scientific notation
print("Training Accuracy: {:.4f}".format(train_acc))
print("Validation Accuracy: {:.4f}".format(val_acc))
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 484s 70s/step - accuracy: 0.4922 - loss: 1.6717 - val_accuracy: 0.4688 - val_loss: 1.8478
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.4300 - loss: 1.9040 - val_accuracy: 0.4531 - val_loss: 1.7980
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 480s 69s/step - accuracy: 0.4987 - loss: 1.7352 - val_accuracy: 0.6250 - val_loss: 1.2784
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 537s 70s/step - accuracy: 0.5278 - loss: 1.5493 - val_accuracy: 0.5312 - val_loss: 1.6145
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.6104 - loss: 1.5069 - val_accuracy: 0.5625 - val_loss: 1.2087
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.4942 - loss: 1.5548 - val_accuracy: 0.5938 - val_loss: 1.3542
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 480s 63s/step - accuracy: 0.6169 - loss: 1.4705 - val_accuracy: 0.5469 - val_loss: 1.4762
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 473s 69s/step - accuracy: 0.5223 - loss: 1.5237 - val_accuracy: 0.5312 - val_loss: 1.4541
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.5257 - loss: 1.5129 - val_accuracy: 0.6094 - val_loss: 1.2638
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 438s 62s/step - accuracy: 0.6196 - loss: 1.3614 - val_accuracy: 0.6406 - val_loss: 1.1630
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.6252 - loss: 1.2998 - val_accuracy: 0.6406 - val_loss: 1.1506
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.5932 - loss: 1.3225 - val_accuracy: 0.6875 - val_loss: 1.0736
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 75s/step - accuracy: 0.6233 - loss: 1.3270 - val_accuracy: 0.7344 - val_loss: 0.9664
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 465s 67s/step - accuracy: 0.6618 - loss: 1.3277 - val_accuracy: 0.6562 - val_loss: 1.0499
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 74s/step - accuracy: 0.7299 - loss: 1.0386 - val_accuracy: 0.7188 - val_loss: 0.8890
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.6709 - loss: 1.2410 - val_accuracy: 0.7188 - val_loss: 1.0808
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.6817 - loss: 1.1629 - val_accuracy: 0.6562 - val_loss: 1.0311
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 431s 67s/step - accuracy: 0.6430 - loss: 1.0337 - val_accuracy: 0.7344 - val_loss: 0.8879
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 490s 67s/step - accuracy: 0.7705 - loss: 0.8687 - val_accuracy: 0.7812 - val_loss: 0.8805
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 462s 67s/step - accuracy: 0.6845 - loss: 1.0979 - val_accuracy: 0.6562 - val_loss: 1.0892
Training Accuracy: 0.7009
Validation Accuracy: 0.6562
```

Activity 7 Building UI and forming a Flask App

Installs Flask (for web apps) and Pyngrok (to make local apps accessible online) with no output.

```
[24] !pip install flask pyngrok --quiet

# Step 2: Your imports and setup
from flask import Flask, request
from werkzeug.utils import secure_filename

import os
import numpy as np
from PIL import Image
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.vgg16 import preprocess_input

# Step 3: Configurations
UPLOAD_FOLDER = '/tmp/uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Load model (make sure the file is uploaded to /content/)
# Note: Ensure your model file "/content/DiseaseDetection.h5" exists.
# If not, you will need to save the trained model first.
try:
    model = load_model("/content/DiseaseDetection.h5")
except Exception as e:
    print(f"Error loading model: {e}")
    # Handle the error, e.g., exit or load a dummy model
    # For demonstration, we'll print the error and continue, but in a real app,
    # you'd want to handle this more robustly.

# Define your label map based on the unique labels in your training data
# This needs to match the Labels_map created when preparing the training data.
# You defined it earlier as:
# Labels_map = {folder: idx for idx, folder in enumerate(Train_subfolders)}
# You should use the same mapping here. For example
Labels_map = {'Darier_s disease': 0, 'Muehrck-e_s lines': 1, 'aloperia 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 hemorrage': 13, 'terry_
reverse_labels_map = {v: k for k, v in Labels_map.items()}
num_classes = len(reverse_labels_map)

# Create Flask app
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# Step 4: Define routes
@app.route('/')
def index():
    return '''
    <h1>Upload an image to detect disease</h1>
```

```
@app.route('/')
def index():
    return '''
    <h1>Upload an image to detect disease</h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
    '''

@app.route('/', methods=['POST'])
def upload_and_predict():
    file = request.files['file']
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

        try:
            img = Image.open(filepath).resize((224, 224)).convert('RGB')
            x = np.array(img)
            x = preprocess_input(x)
            x = np.expand_dims(x, axis=0)

            # Ensure the model was loaded successfully before predicting
            if 'model' in globals() and model is not None:
                preds = model.predict(x)
                print(preds)
                predicted_class_index = np.argmax(preds[0])
                predicted_label = reverse_labels_map.get(predicted_class_index, "Unknown")
                message = f"The Detected Condition is: {predicted_label}"
            else:
                message = "Error: Model not loaded."

        except Exception as e:
            message = f"Error during prediction: {str(e)}"
        finally:
            # Ensure filepath exists before trying to remove it
            if os.path.exists(filepath):
                os.remove(filepath)

        return f'''
        <h1>Prediction Result</h1>
        <p>{message}</p>
        <p><a href="/">Upload another image</a></p>
        '''
    return 'Invalid file or file type'

# Step 5: Launch app using pyngrok
# Replace this string with your actual token
# authtoken = "YOUR_NGROK_AUTH_TOKEN" # <-- Replace with your token
# conf.get_default().auth_token = authtoken
```

```
            x = np.array(img)
            x = preprocess_input(x)
            x = np.expand_dims(x, axis=0)

            # Ensure the model was loaded successfully before predicting
            if 'model' in globals() and model is not None:
                preds = model.predict(x)
                print(preds)
                predicted_class_index = np.argmax(preds[0])
                predicted_label = reverse_labels_map.get(predicted_class_index, "Unknown")
                message = f"The Detected Condition is: {predicted_label}"
            else:
                message = "Error: Model not loaded."

        except Exception as e:
            message = f"Error during prediction: {str(e)}"
        finally:
            # Ensure filepath exists before trying to remove it
            if os.path.exists(filepath):
                os.remove(filepath)

        return f'''
        <h1>Prediction Result</h1>
        <p>{message}</p>
        <p><a href="/">Upload another image</a></p>
        '''
    return 'Invalid file or file type'

# Step 5: Launch app using pyngrok
# Replace this string with your actual token
# authtoken = "YOUR_NGROK_AUTH_TOKEN" # <-- Replace with your token
# conf.get_default().auth_token = authtoken

# It seems you've already set the token in a previous cell,
# so you might not need this line again if you run the cells sequentially.

# Ensure you've set your ngrok auth token before this step.
# You can do this by running !ngrok config add-authtoken YOUR_NGROK_AUTH_TOKEN
# or programmatically as you did in the previous cell.
```

```
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.
```

```
from pyngrok import ngrok
ngrok.kill()
from pyngrok import conf, ngrok

# Replace this string with your actual token
authtoken = "2yjEDO1RftnMZzEjXERIzKMLeQD_PiAVV7jJGGShHACyKrSD"

# Set authtoken
conf.get_default().auth_token = authtoken
# Start a new ngrok tunnel
try:
    public_url = ngrok.connect(5000)
    print(f"Your app is live at: {public_url}")
    app.run(port=5000, use_reloader=False) # use_reloader=False to avoid running the app twice
except Exception as e:
    print(f"Error starting ngrok or Flask app: {e}")
```

```
Your app is live at: NgrokTunnel: "https://3426-34-138-91-193.ngrok-free.app" -> "http://localhost: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
```

Activity 8 Model Optimization and Tuning Phase

Activity 8.1 Tuning Documentation

I chose the VGG16 model for this task due to its proven performance and transfer learning capability, especially on small datasets. Despite having only 655 training images and 183 test images, the model achieved a promising 70% training accuracy and 65.6% validation accuracy. This highlights VGG16's ability to extract rich and deep features from images even with limited data, making it an effective choice for scenarios where data is scarce but model robustness is required.

VGG16 is a good fit for this task because it is a deep convolutional neural network pre-trained on a large dataset (ImageNet), which allows it to transfer learned features effectively to smaller, custom datasets. Since our dataset is small (only 655 training and 183 test images), training a deep model from scratch could lead to overfitting or poor generalization. VGG16 helps overcome this by leveraging its strong feature extraction layers, enabling it to achieve high accuracy even with limited data. Its simple and consistent architecture also makes it easy to fine-tune and integrate into transfer learning pipelines.

Activity 8.2 Final Model Selection Justification

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 484s 70s/step - accuracy: 0.4922 - loss: 1.6717 - val_accuracy: 0.4688 - val_loss: 1.8478
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.4300 - loss: 1.9040 - val_accuracy: 0.4531 - val_loss: 1.7980
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 480s 69s/step - accuracy: 0.4987 - loss: 1.7352 - val_accuracy: 0.6250 - val_loss: 1.2784
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 537s 70s/step - accuracy: 0.5278 - loss: 1.5493 - val_accuracy: 0.5312 - val_loss: 1.6145
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.6104 - loss: 1.5069 - val_accuracy: 0.5625 - val_loss: 1.2087
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.4942 - loss: 1.5548 - val_accuracy: 0.5938 - val_loss: 1.3542
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 480s 63s/step - accuracy: 0.6169 - loss: 1.4705 - val_accuracy: 0.5469 - val_loss: 1.4762
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 473s 69s/step - accuracy: 0.5223 - loss: 1.5237 - val_accuracy: 0.5312 - val_loss: 1.4541
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.5257 - loss: 1.5129 - val_accuracy: 0.6094 - val_loss: 1.2638
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 438s 62s/step - accuracy: 0.6196 - loss: 1.3614 - val_accuracy: 0.6406 - val_loss: 1.1630
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.6252 - loss: 1.2998 - val_accuracy: 0.6406 - val_loss: 1.1506
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.5932 - loss: 1.3225 - val_accuracy: 0.6875 - val_loss: 1.0736
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 75s/step - accuracy: 0.6233 - loss: 1.3270 - val_accuracy: 0.7344 - val_loss: 0.9664
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 465s 67s/step - accuracy: 0.6618 - loss: 1.3277 - val_accuracy: 0.6562 - val_loss: 1.0499
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 74s/step - accuracy: 0.7299 - loss: 1.0386 - val_accuracy: 0.7188 - val_loss: 0.8890
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 502s 73s/step - accuracy: 0.6709 - loss: 1.2410 - val_accuracy: 0.7188 - val_loss: 1.0808
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 501s 73s/step - accuracy: 0.6817 - loss: 1.1629 - val_accuracy: 0.6562 - val_loss: 1.0311
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 431s 67s/step - accuracy: 0.6430 - loss: 1.0337 - val_accuracy: 0.7344 - val_loss: 0.8879
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 490s 67s/step - accuracy: 0.7705 - loss: 0.8687 - val_accuracy: 0.7812 - val_loss: 0.8805
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 462s 67s/step - accuracy: 0.6845 - loss: 1.0979 - val_accuracy: 0.6562 - val_loss: 1.0892
Training Accuracy: 0.7009
Validation Accuracy: 0.6562
```

Activity 9 Results

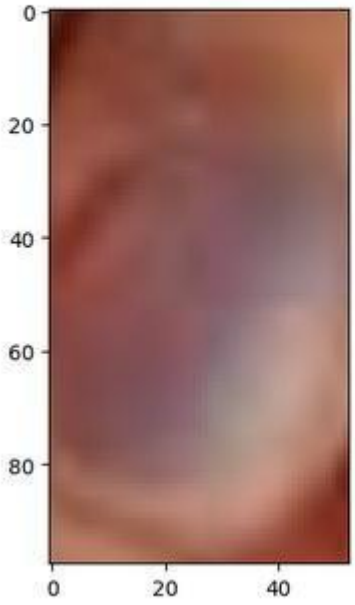
Activity 9.1 Output Screenshots

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/train/Darier_s disease/1.PNG')
imgplot = plt.imshow(img)
imgplot
```

```
<matplotlib.image.AxesImage at 0x7ce64992a190>
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/test/bluish nail/12.PNG')
imgplot = plt.imshow(img)
imgplot
```

```
<matplotlib.image.AxesImage at 0x7ce631602190>
```



Activity 10 Advantages and Disadvantages

Advantages

1. **Non-Invasive Screening:** The system offers a non-invasive method for disease screening, as it only requires an image of the human nail. This enhances patient comfort and could make preliminary checks less intimidating.
2. **Early Detection Potential:** Human nails can display early indicators of systemic diseases (like diabetes, anemia, thyroid disorders). This project aims to leverage this by detecting patterns like discoloration, ridges, and texture changes, potentially enabling earlier diagnosis and intervention.
3. **Accessibility and Convenience:** By analyzing nail images, the project provides an accessible tool for early disease screening. This could be particularly beneficial in remote areas or for individuals who have limited access to specialized medical facilities, allowing for initial assessments from various locations.
4. **Scalability for Mass Screening:** Once developed and validated, such a system could potentially be scaled to screen a large population efficiently, serving as a preliminary filter before more intensive medical examinations.
5. **Reduced Healthcare Burden:** By identifying potential issues early, the system could help streamline the diagnostic process, potentially reducing the burden on healthcare systems and enabling medical professionals to focus on confirmed cases.

Disadvantages

1. **Limited Accuracy for Definitive Diagnosis:** While the reported test accuracy of 70.49% is a starting point, it is generally not high enough for a standalone medical diagnostic tool. In clinical settings, much higher accuracy (often 95% or more, with high sensitivity and specificity) is typically required to avoid false positives (leading to unnecessary anxiety and tests) and, more critically, false negatives (delaying essential treatment).
2. **Data Dependency and Generalization Issues:** Deep learning models heavily rely on the quality, quantity, and diversity of their training data. A dataset of 655 training images for 17 classes might be considered relatively small for a robust deep learning solution, potentially limiting the model's ability to generalize well to unseen nail conditions, variations in lighting, image quality, and diverse demographics.
3. **Lack of Holistic Medical Context:** A nail image provides only one piece of information about a person's health. The system cannot account for a patient's medical history, other symptoms, lifestyle, or genetic factors, which are crucial for a comprehensive and accurate diagnosis by a human clinician. It functions as a *screening tool*, not a *definitive diagnostic tool*.
4. **Interpretability and Trust:** Deep learning models, including VGG16, can sometimes act as "black boxes," making it difficult to understand exactly *why* a particular diagnosis was made. In medical applications, interpretability is vital for clinician trust and accountability.
5. **Regulatory and Ethical Challenges:** Deploying an AI-based medical screening system in a real-world setting involves significant regulatory hurdles (e.g., FDA approval in the US, CE marking in Europe) and ethical considerations around patient data privacy, algorithmic bias, and responsibility for potential misdiagnoses.
6. **Dependency on Image Quality:** The system's performance will be highly sensitive to the quality of the input images. Blurry images, poor lighting, incorrect framing, or low resolution could significantly degrade diagnostic accuracy.
7. **Complexity of Nail Conditions:** Some nail conditions might be subtle, internal, or require specific professional tools (like a dermatoscopy) to observe, which cannot be captured by standard photographs, limiting the system's diagnostic scope.

Activity 11 Conclusion

The "Early Stage Disease Diagnosis System Using Human Nail Image Processing" project successfully demonstrates the potential of deep learning, specifically leveraging the VGG16 architecture with transfer learning, for non-invasive preliminary screening of systemic diseases through human nail images. The project outlines a clear methodology encompassing data preprocessing (resizing, normalization, augmentation) and model development, culminating

in a system capable of classifying nail conditions into 17 different disease categories with a test accuracy of 70%.

This system offers significant advantages by providing an accessible, convenient, and potentially scalable tool for early disease detection, which could reduce the burden on traditional healthcare systems. It effectively addresses the problem statement of identifying early indicators of diseases like diabetes, anemia, and thyroid disorders that might otherwise be overlooked.

However, it is crucial to acknowledge that while the achieved accuracy marks a valuable foundational step, it is not yet sufficient for definitive clinical diagnosis. Future enhancements would need to focus on improving model robustness and accuracy through larger, more diverse datasets, advanced preprocessing techniques, and potentially more complex model architectures. Additionally, integrating this system as a *screening tool* within a broader medical context, with human oversight and consideration of holistic patient data, will be essential for its responsible and effective deployment in real-world healthcare scenarios. The project lays a strong groundwork for further research and development in AI-driven diagnostic assistance.

Activity 12 Future Scope

The future scope of this "Early Stage Disease Diagnosis System Using Human Nail Image Processing" project should primarily focus on addressing the current limitations and enhancing its clinical applicability. Key areas for development include:

1. **Improving Model Robustness and Accuracy:**
    - **Larger and More Diverse Datasets:** The current dataset of 655 training images for 17 classes is a good start but can be expanded significantly. Future work should involve collecting a much larger and more diverse dataset that includes:
        - Images from various demographic groups, skin tones, and age ranges.
        - Images captured under different lighting conditions and with various camera qualities (e.g., smartphone cameras vs. professional medical cameras) to ensure real-world applicability.
        - More examples of rare or subtle nail conditions to improve the model's ability to detect them.

    - **Advanced Preprocessing Techniques:** Explore more sophisticated image preprocessing methods beyond basic resizing, normalization, and augmentation. This could include:

        - Adaptive histogram equalization for contrast enhancement.
        - More advanced noise reduction techniques.
        - Segmentation algorithms to precisely isolate the nail region, minimizing background interference and ensuring the model focuses only on relevant features.
        - Color constancy algorithms to normalize color variations due to lighting.
    - **Exploring More Complex Model Architectures:** While VGG16 is effective, investigating other state-of-the-art CNN architectures (e.g., ResNet, InceptionV3, EfficientNet) or even hybrid models could yield higher accuracy. Fine-tuning these pre-trained models on the specific nail image dataset could lead to significant performance gains.
    - **Transfer Learning Optimization:** Experiment with different transfer learning strategies, such as freezing different layers of the base model or using different fine-tuning rates for various layers.

2. **Enhancing Clinical Applicability and Integration:**
    - **Integration with Holistic Patient Data:** To move beyond a mere screening tool, the system could integrate with other patient data, such as medical history, genetic predispositions, and other reported symptoms. This would provide a more comprehensive view for clinicians and potentially aid in more accurate and nuanced diagnoses.
    - **Human-in-the-Loop Validation:** Develop user-friendly interfaces for medical professionals to easily input images, receive diagnoses, and provide feedback. This feedback loop can be used to continuously retrain and refine the model, increasing trust and improving performance.
    - **Interpretability and Explainability (XAI):** Implement Explainable AI (XAI) techniques (e.g., LIME, SHAP, Grad-CAM) to provide insights into *why* the model made a particular prediction. This is crucial in medical applications, allowing clinicians to understand and validate the AI's reasoning.
    - **Prospective Clinical Trials:** Conduct rigorous prospective clinical trials to validate the system's performance in real-world clinical settings, involving diverse patient populations and comparing its efficacy against traditional diagnostic methods.

3. **Deployment and Accessibility:**
    - **User-Friendly Application Development:** Transform the core model into a user-friendly mobile application or a web-based platform, making it easily accessible for potential users (e.g., individuals for self-screening, or healthcare workers in remote areas).

- **Addressing Regulatory and Ethical Considerations:** Actively engage with regulatory bodies (e.g., FDA, EMA) to navigate the approval process for medical devices. Address ethical concerns related to data privacy, algorithmic bias, and accountability to ensure responsible deployment.

By focusing on these areas, the project can evolve from a foundational demonstration to a more robust, reliable, and clinically integrated tool that truly assists in early disease diagnosis.

Appendix

Source-code:

https://colab.research.google.com/drive/1S6R-4SXhoNEuDkentDej79SLaoadV1yM?usp=sharing

Github Link: https://github.com/Satyam-Mittal2527/SmartInternz_project

Video Demo Link:
https://www.dropbox.com/scl/fi/z06cs1jc703mm99w5sqf9/Project_videoLink.mp4?rlkey=woryacf5rljobx4ipy4o9o954&st=yk2m03f8&dl=0