Transfer Learning-Based Classification of Poultry Diseases for Enhanced Health Management

This project aims to develop a Transfer learning-based system for classifying poultry diseases into four categories: Salmonella, New Castle Disease, Coccidiosis, and Healthy. The solution involves creating a robust machine learning model that will be integrated into a mobile application. Farmers will be able to use this application to input data (e.g., symptoms, environmental conditions, and biological samples) and receive an immediate diagnosis along with suggested treatments. The ultimate goal is to provide farmers with a tool that enhances their ability to manage poultry health, thereby reducing disease impact and improving productivity.

Scenario 1: Outbreak in a Rural Community

A small rural community relies heavily on poultry farming for its livelihood. Recently, the farmers have noticed an increase in sick birds, exhibiting symptoms such as lethargy, diarrhea, and reduced egg production. Without immediate access to veterinary services, the farmers are struggling to diagnose the problem. Using the new mobile application, they input the observed symptoms and environmental data. The machine learning model quickly classifies the disease as Coccidiosis and provides recommendations for treatment and management. This allows the farmers to take swift action, reducing the spread of the disease and preventing further economic losses.

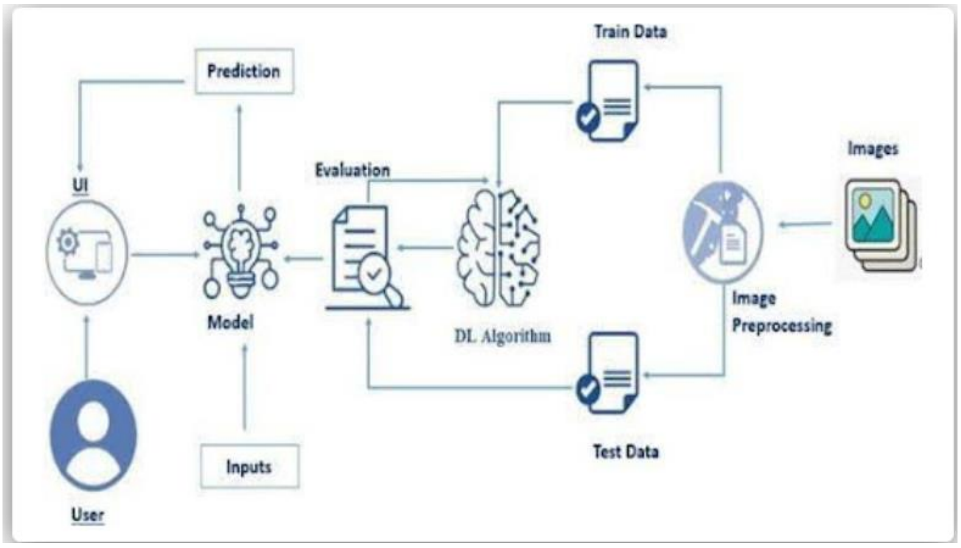
Scenario 2: Commercial Poultry Farm Management

A large commercial poultry farm has implemented the machine learning-based disease classification system to monitor the health of its flocks. Daily health checks are performed, and data is collected via the mobile application. One day, the system identifies symptoms consistent with New Castle Disease in a specific section of the farm. The early detection enables the farm management to quarantine the affected birds and implement control measures promptly, preventing a widespread outbreak and ensuring the overall health of the flock. This proactive approach not only saves costs but also maintains the farm's productivity and reputation.

Scenario 3: Research and Training for Veterinary Students

A veterinary school integrates the machine learning-based disease classification application into its curriculum. Students use the app to input data from case studies and real-world scenarios. Through this hands-on training, they learn how to diagnose diseases like Salmonella, New Castle Disease, and Coccidiosis using modern technology. The application also provides detailed information about each disease, treatment options, and management practices. This experience equips future veterinarians with valuable skills in utilizing advanced diagnostic tools, preparing them to better serve the poultry industry.

Architecture:

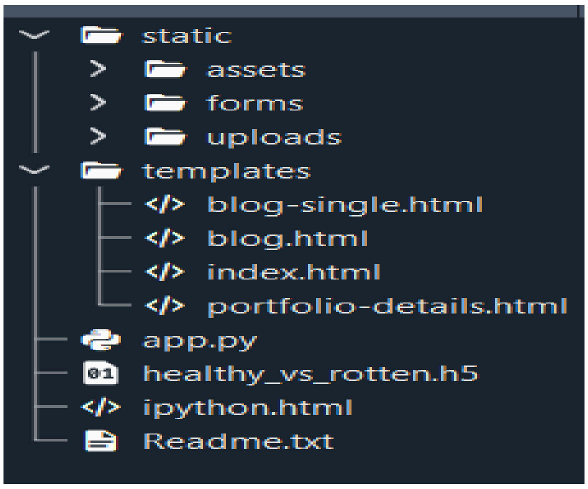


Prerequisites

- **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 tenserflow” and click enter.**
    - **Type “pip install Flask” and click enter.**

Project Structure

Create the Project folder which contains files as shown below



```
static
    assets
    forms
    uploads
templates
    blog-single.html
    blog.html
    index.html
    portfolio-details.html
app.py
healthy_vs_rotten.h5
ipython.html
Readme.txt
```

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.

Collect the dataset

Create API Token on Kaggle

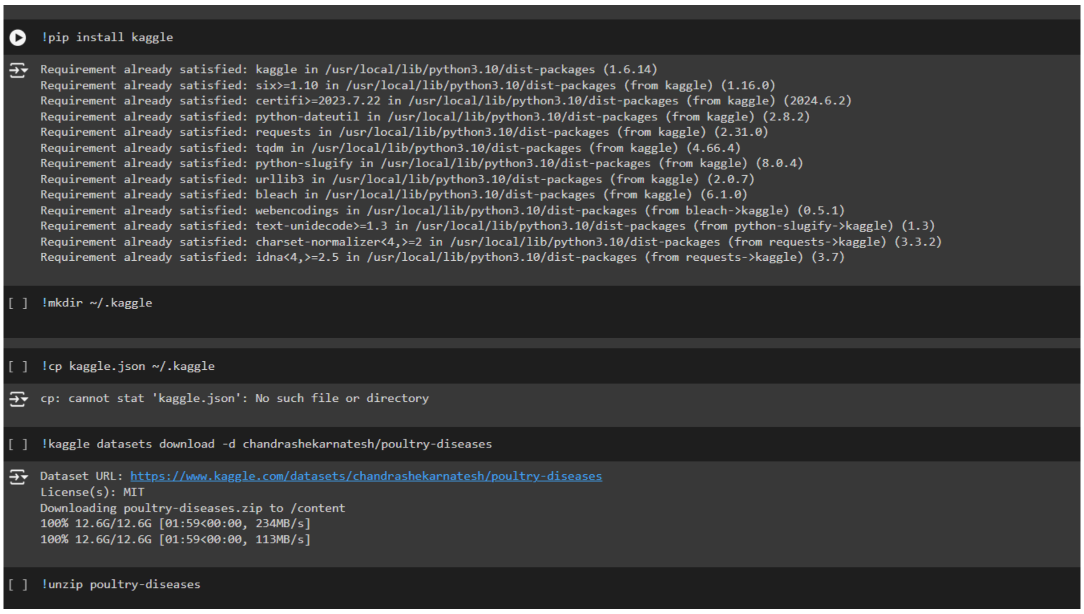
- Log in to your Kaggle account.
- Go to your account settings and select "Create New API Token."
- This will download a kaggle.json file containing your API credentials.

Set Up Google Colab Environment

- Open a new notebook in Google Colab.

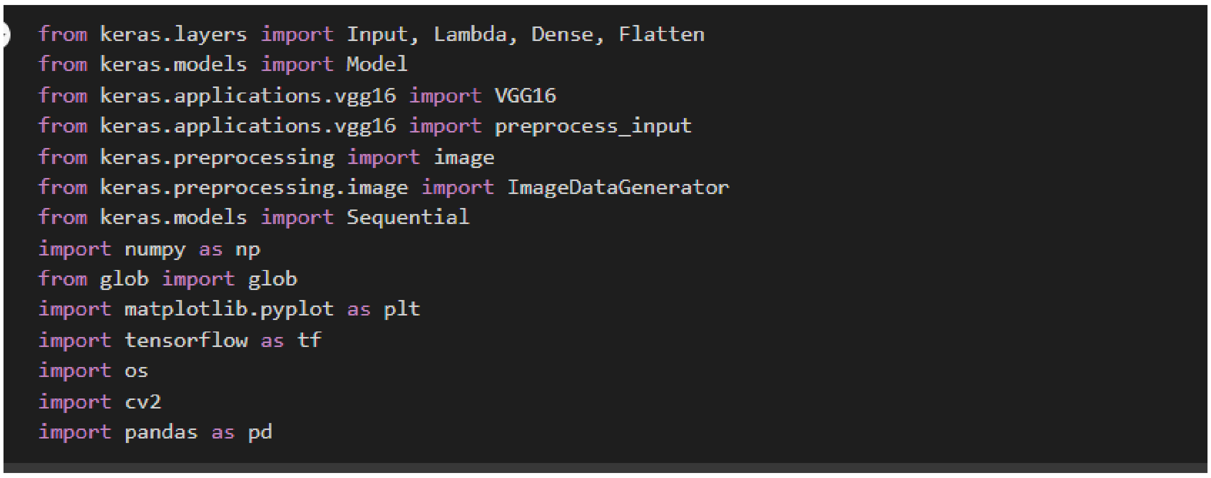
Upload kaggle.json to Colab

- Upload the kaggle.json file to your Colab notebook. This file contains your Kaggle API credentials.



```
!pip install kaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.14)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.6.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)

[ ] !mkdir ~/.kaggle

[ ] !cp kaggle.json ~/.kaggle

cp: cannot stat 'kaggle.json': No such file or directory

[ ] !kaggle datasets download -d chandrashekarnatesh/poultry-diseases

Dataset URL: https://www.kaggle.com/datasets/chandrashekarnatesh/poultry-diseases
License(s): MIT
Downloading poultry-diseases.zip to /content
100% 12.6G/12.6G [01:59<00:00, 234MB/s]
100% 12.6G/12.6G [01:59<00:00, 113MB/s]

[ ] !unzip poultry-diseases
```

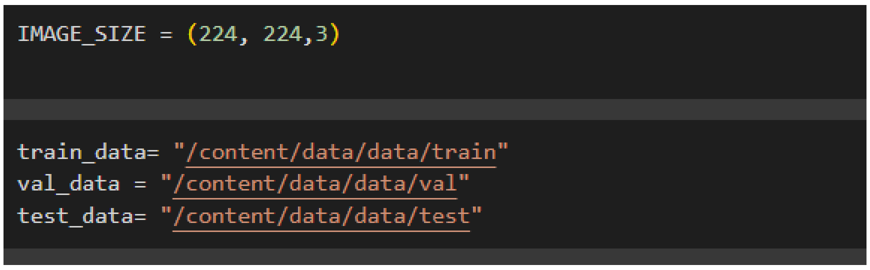
1.1Activity: Import libraries



```python
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt
import tensorflow as tf
import os
import cv2
import pandas as pd
```

This code uses the VGG16 architecture from Keras for transfer learning to classify images of poultry diseases. It preprocesses the images, creates a model with VGG16 as the base, and adds custom dense and flatten layers for classification. The model is then trained using images loaded from directories with the help of ImageDataGenerator.

1.2 Activity: Define the paths for train , test and validation data



```
IMAGE_SIZE = (224, 224,3)
```

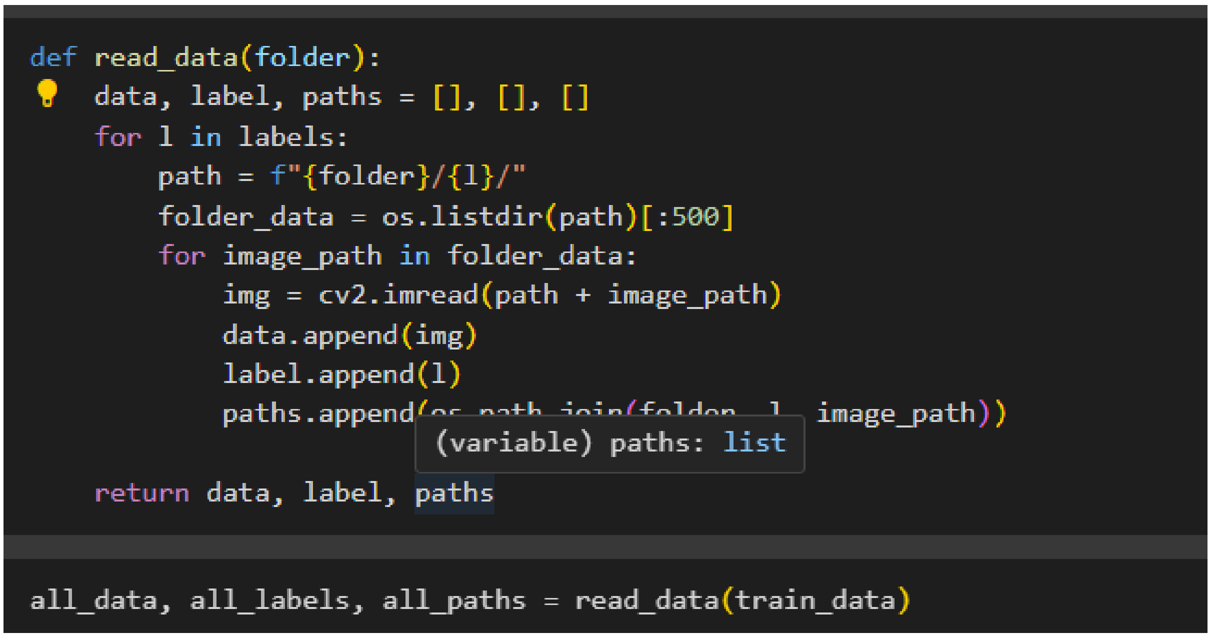
```
train_data= "/content/data/data/train"
val_data = "/content/data/data/val"
test_data= "/content/data/data/test"
```

Describing the Image size, and assign the paths .

1.3 Activity:

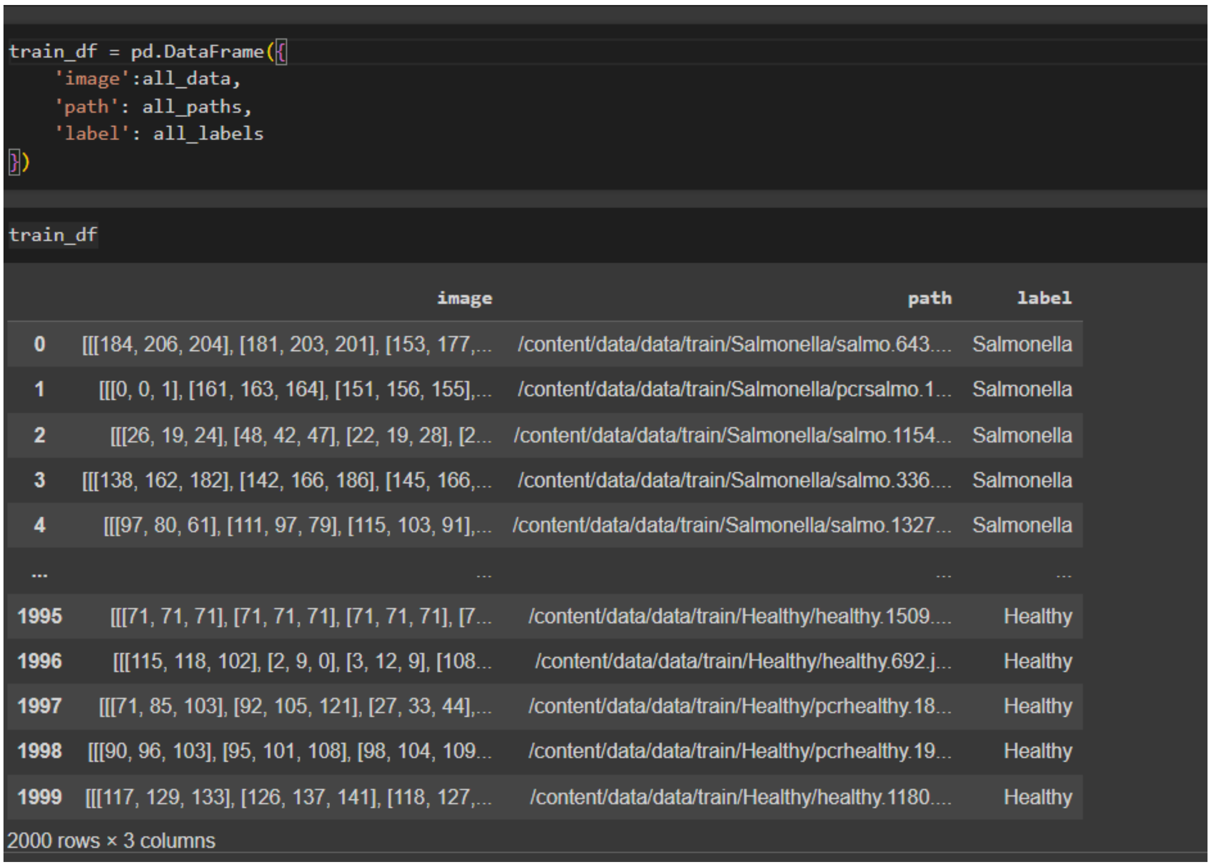
The read_data function reads a subset of images from a specified folder, taking 500 images per label category. It iterates through each label, loads the images using OpenCV, and collects the image data, labels, and file paths into separate lists. This function is designed to manage large datasets by processing a smaller, manageable subset of images for training, validation, or testing.

We have taking 500 images from all 4 categories for training, testing and val because in each dataset from training , testing and validation there 4 lakhs , 70k and 40k. Those are very huge data to train we don't have that much ram in so collect 500 from each of the folder.



```
def read_data(folder):
    data, label, paths = [], [], []
    for l in labels:
        path = f"{folder}/{l}/"
        folder_data = os.listdir(path)[:500]
        for image_path in folder_data:
            img = cv2.imread(path + image_path)
            data.append(img)
            label.append(l)
            paths.append(os.path.join(folder, l, image_path))
                        (variable) paths: list

    return data, label, paths
```

```
all_data, all_labels, all_paths = read_data(train_data)
```



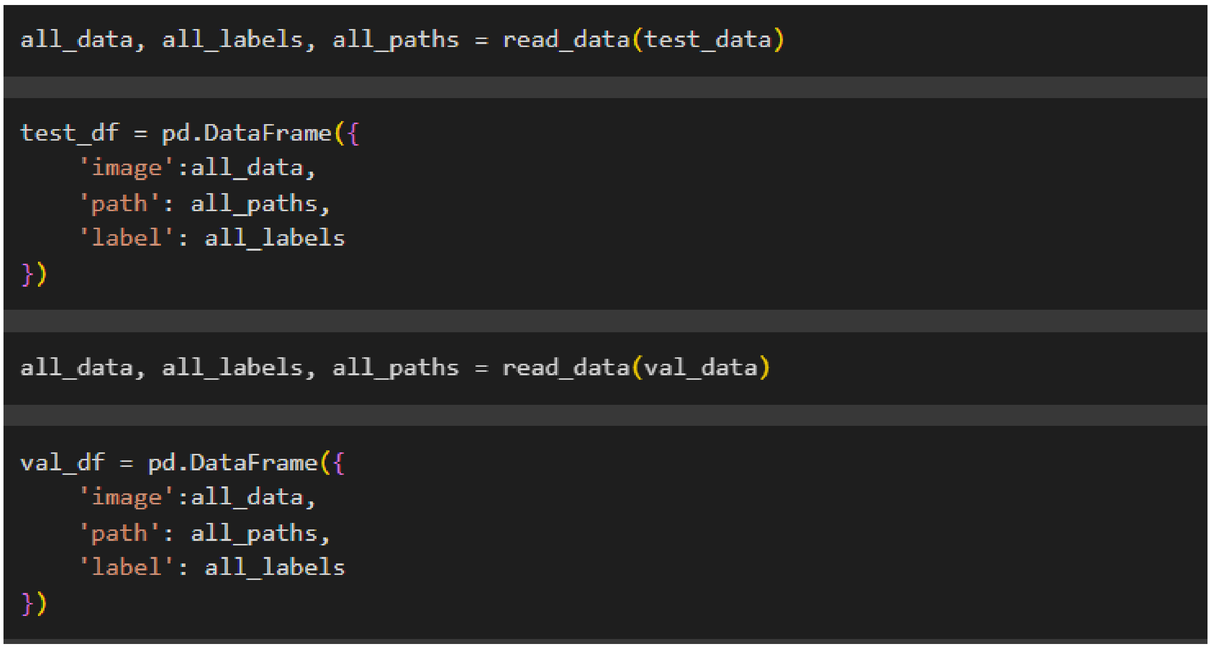
```
train_df = pd.DataFrame({
    'image':all_data,
    'path': all_paths,
    'label': all_labels
})
```

```
train_df
```

	image	path	label
0	[[[184, 206, 204], [181, 203, 201], [153, 177,...	/content/data/data/train/Salmonella/salmo.643....	Salmonella
1	[[[0, 0, 1], [161, 163, 164], [151, 156, 155],...	/content/data/data/train/Salmonella/pcrsalmo.1...	Salmonella
2	[[[26, 19, 24], [48, 42, 47], [22, 19, 28], [2...	/content/data/data/train/Salmonella/salmo.1154...	Salmonella
3	[[[138, 162, 182], [142, 166, 186], [145, 166,...	/content/data/data/train/Salmonella/salmo.336....	Salmonella
4	[[[97, 80, 61], [111, 97, 79], [115, 103, 91],...	/content/data/data/train/Salmonella/salmo.1327...	Salmonella
...	...	...	...
1995	[[[71, 71, 71], [71, 71, 71], [71, 71, 71], [7...	/content/data/data/train/Healthy/healthy.1509....	Healthy
1996	[[[115, 118, 102], [2, 9, 0], [3, 12, 9], [108...	/content/data/data/train/Healthy/healthy.692.j...	Healthy
1997	[[[71, 85, 103], [92, 105, 121], [27, 33, 44],...	/content/data/data/train/Healthy/pcrhealthy.18...	Healthy
1998	[[[90, 96, 103], [95, 101, 108], [98, 104, 109...	/content/data/data/train/Healthy/pcrhealthy.19...	Healthy
1999	[[[117, 129, 133], [126, 137, 141], [118, 127,...	/content/data/data/train/Healthy/healthy.1180....	Healthy

2000 rows × 3 columns

Perform above operations for test and validation data also



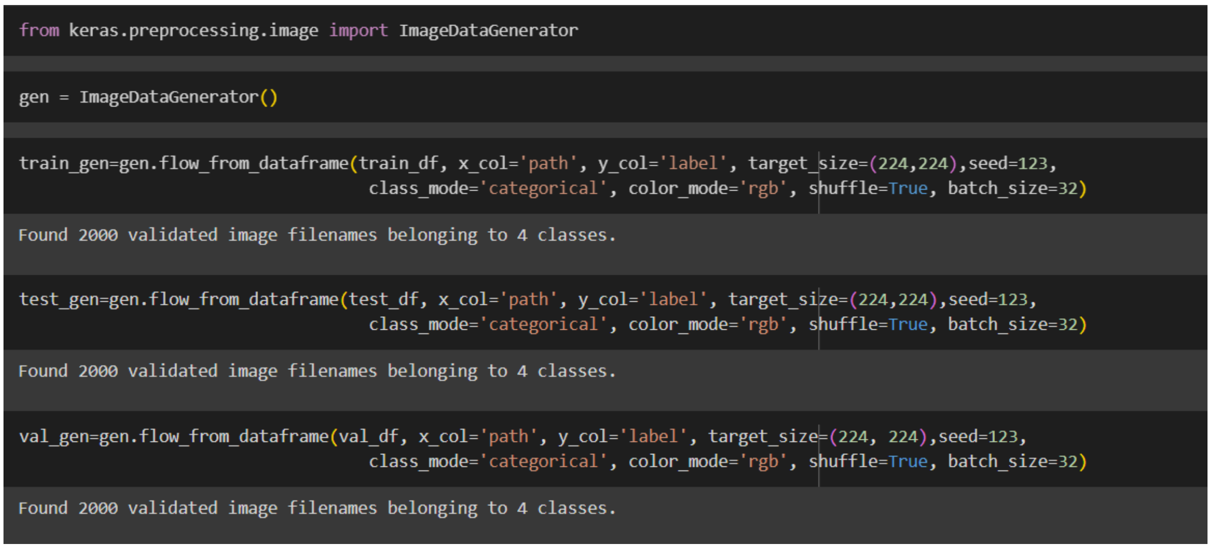
```
all_data, all_labels, all_paths = read_data(test_data)
```

```
test_df = pd.DataFrame({
    'image':all_data,
    'path': all_paths,
    'label': all_labels
})
```

```
all_data, all_labels, all_paths = read_data(val_data)
```

```
val_df = pd.DataFrame({
    'image':all_data,
    'path': all_paths,
    'label': all_labels
})
```

Using ImageDataGenerator



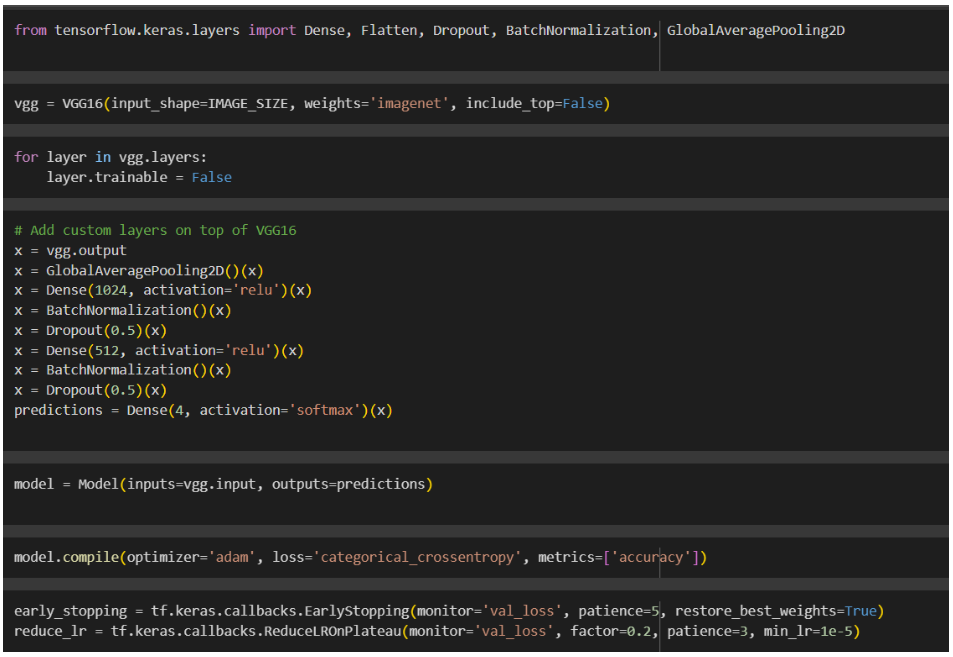
```
from keras.preprocessing.image import ImageDataGenerator

gen = ImageDataGenerator()

train_gen=gen.flow_from_dataframe(train_df, x_col='path', y_col='label', target_size=(224,224),seed=123,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)

Found 2000 validated image filenames belonging to 4 classes.

test_gen=gen.flow_from_dataframe(test_df, x_col='path', y_col='label', target_size=(224,224),seed=123,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)

Found 2000 validated image filenames belonging to 4 classes.

val_gen=gen.flow_from_dataframe(val_df, x_col='path', y_col='label', target_size=(224, 224),seed=123,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)

Found 2000 validated image filenames belonging to 4 classes.
```

This Python code utilizes Keras, a deep learning library, to preprocess image data for a machine learning model. The code imports libraries and creates three data generators: one for training ('train_gen'), one for testing ('test_gen'), and likely one for validation ('val_gen'). These generators are configured to access data from a DataFrame, which provides the location of images, labels, and other relevant information. In essence, this code snippet prepares the image data for the training process of a machine learning model.

Model Building:

- **Training the model in multiple algorithms :**

Activity 2.1:VGG16

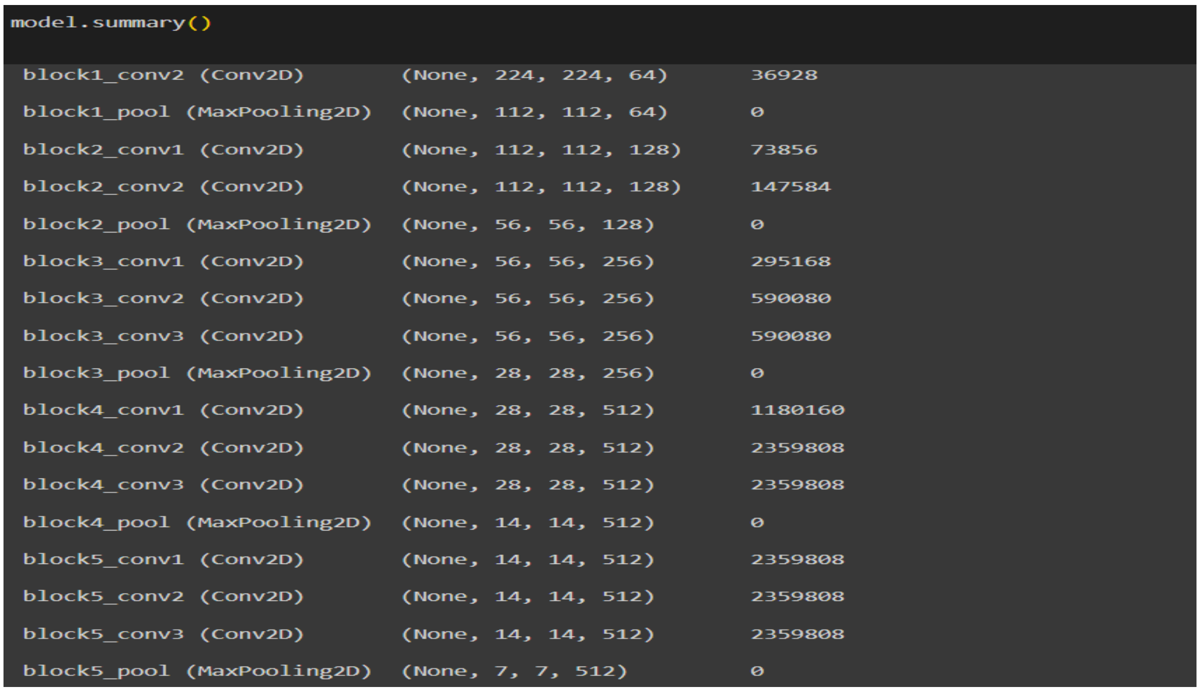


```
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization, GlobalAveragePooling2D

vgg = VGG16(input_shape=IMAGE_SIZE, weights='imagenet', include_top=False)

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

# Add custom layers on top of VGG16
x = vgg.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(4, activation='softmax')(x)

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

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-5)
```

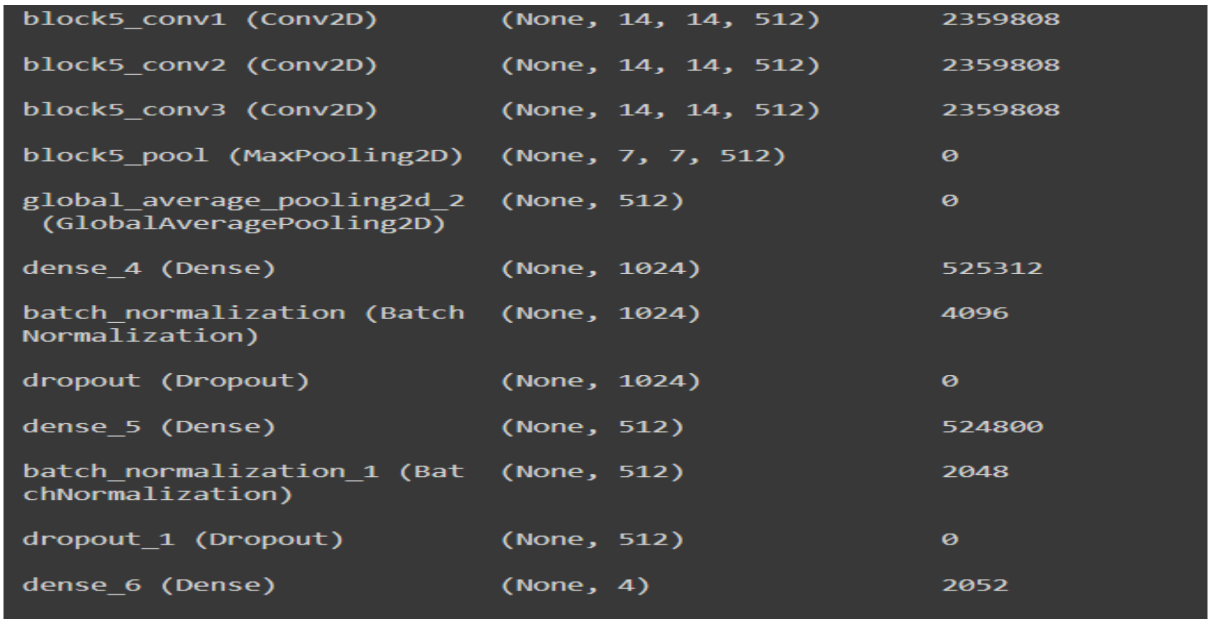
This short Python code snippet builds an image classifier with Keras. It cleverly reuses a pre-trained VGG16 model for its powerful image recognition abilities. Here's the key idea:

- **The code loads the VGG16 model, but skips its final classification layers (keeping its feature extraction power).**

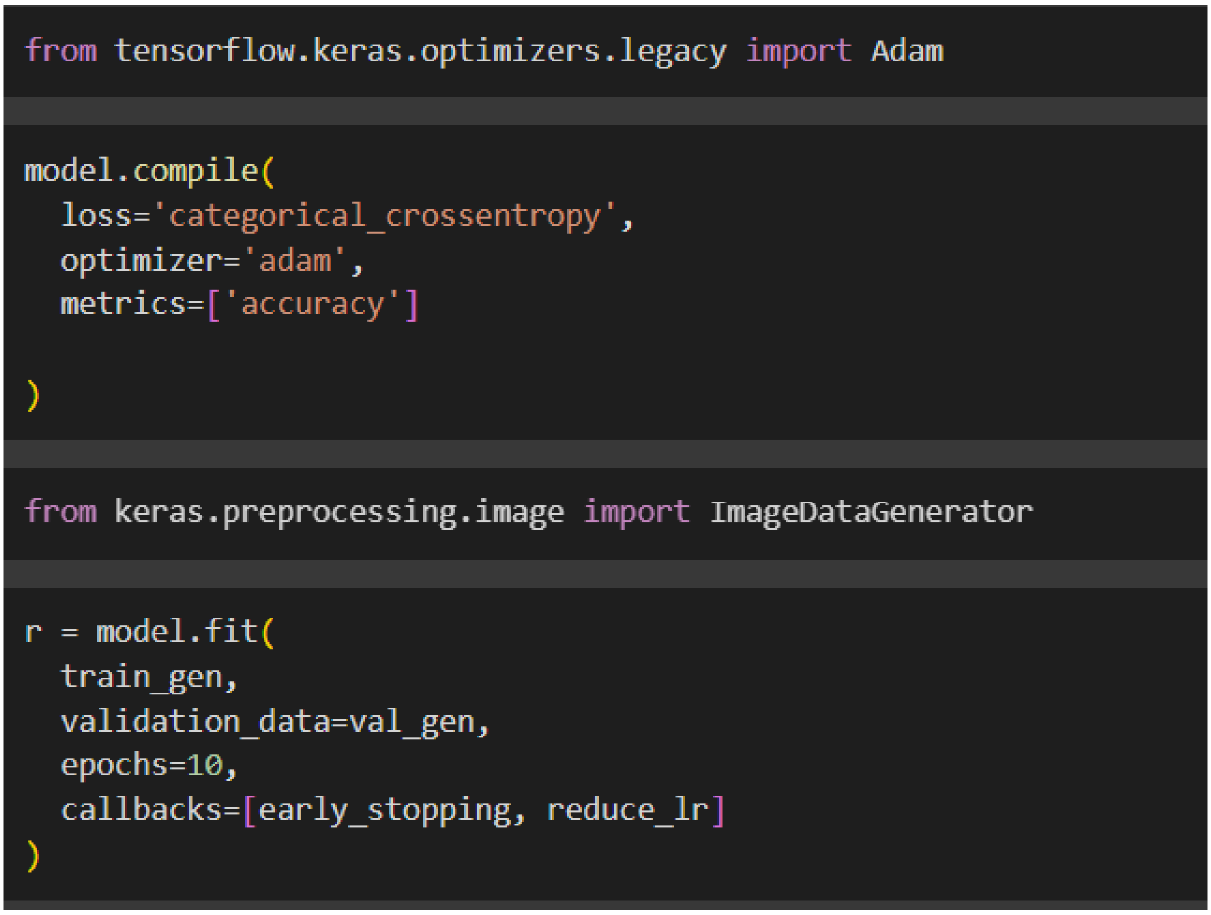
- **It then freezes the pre-trained part to focus training on new custom layers added on top.**
- **These custom layers likely handle the specific classification task you have in mind.**
- **Finally, it compiles the whole model for training, setting up how to improve and assess its performance.**



```
model.summary()
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
```



```
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
 global_average_pooling2d_2  (None, 512)               0
  (GlobalAveragePooling2D)
 dense_4 (Dense)             (None, 1024)              525312
 batch_normalization (Batch  (None, 1024)              4096
 Normalization)
 dropout (Dropout)           (None, 1024)              0
 dense_5 (Dense)             (None, 512)               524800
 batch_normalization_1 (Bat  (None, 512)               2048
 chNormalization)
 dropout_1 (Dropout)         (None, 512)               0
 dense_6 (Dense)             (None, 4)                 2052
```



```python
from tensorflow.keras.optimizers.legacy import Adam
```

```python
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']

)
```

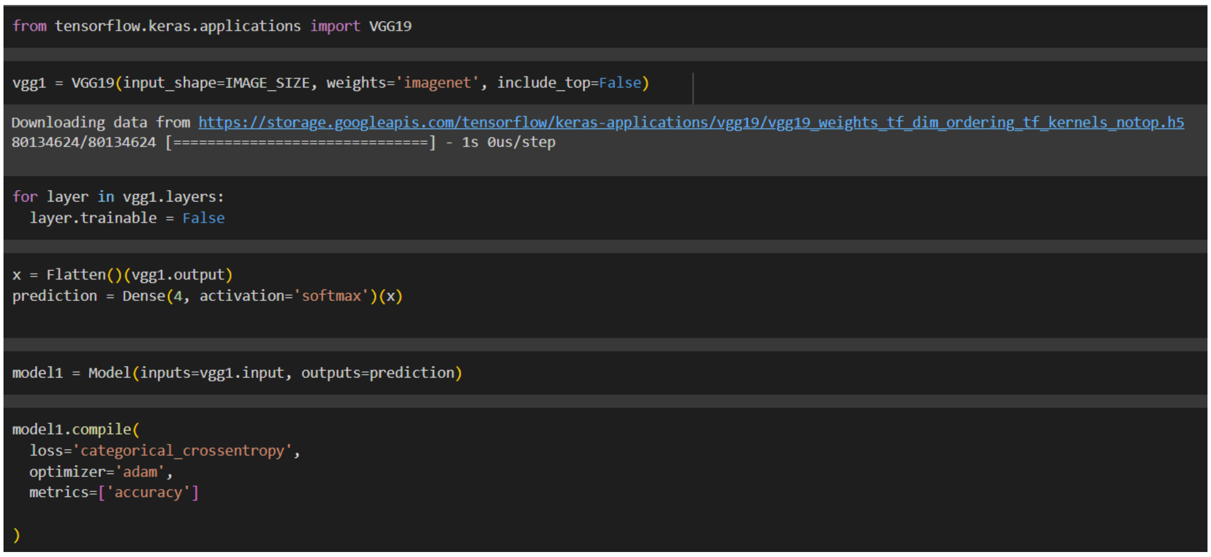
```python
from keras.preprocessing.image import ImageDataGenerator
```

```python
r = model.fit(
  train_gen,
  validation_data=val_gen,
  epochs=10,
  callbacks=[early_stopping, reduce_lr]
)
```

The text shows epochs, which are iterations over the training data. It also shows loss, which is how well the model is performing on the training data, and accuracy, which is how often the model makes correct predictions.

In the output you provided, it appears the model is improving over time as the loss is decreasing and the accuracy is increasing.

Activity 2.2:VGG19



```python
from tensorflow.keras.applications import VGG19
```

```python
vgg1 = VGG19(input_shape=IMAGE_SIZE, weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 1s 0us/step
```

```python
for layer in vgg1.layers:
  layer.trainable = False
```

```python
x = Flatten()(vgg1.output)
prediction = Dense(4, activation='softmax')(x)
```

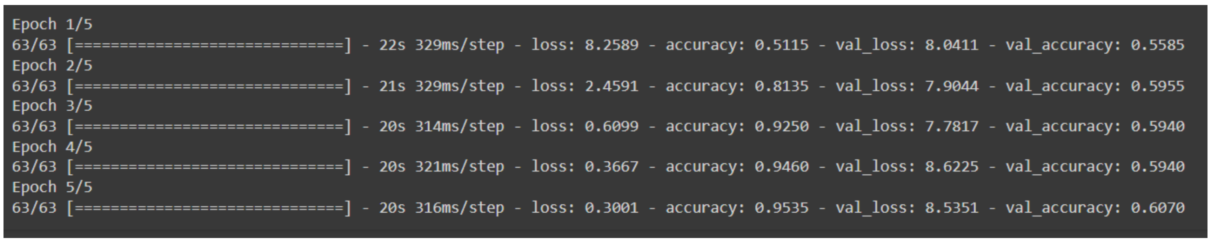
```python
model1 = Model(inputs=vgg1.input, outputs=prediction)
```

```python
model1.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']

)
```

- **The first lines import libraries including TensorFlow and Keras.**

- **It appears to be defining a model with layers including Flatten and Dense which are commonly used in CNN architectures.**
- **The code then defines a process to compile the model, specifying an optimizer, loss function and metrics.**

Overall, the code snippet seems to be training a CNN model on some data. However, without more context it's difficult to say exactly what the model is being trained for.



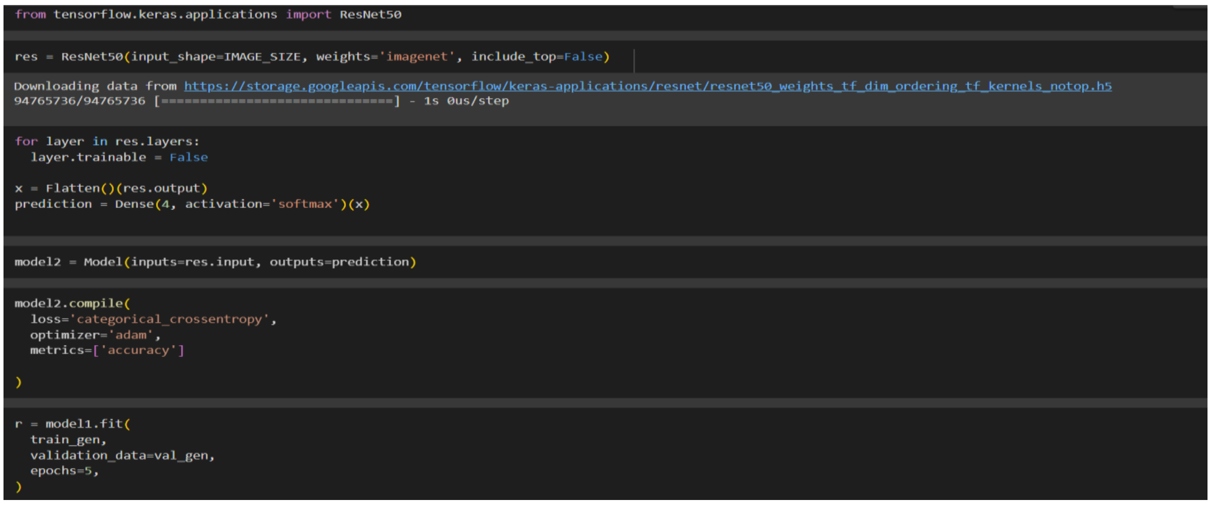
```
Epoch 1/5
63/63 [==============================] - 22s 329ms/step - loss: 8.2589 - accuracy: 0.5115 - val_loss: 8.0411 - val_accuracy: 0.5585
Epoch 2/5
63/63 [==============================] - 21s 329ms/step - loss: 2.4591 - accuracy: 0.8135 - val_loss: 7.9044 - val_accuracy: 0.5955
Epoch 3/5
63/63 [==============================] - 20s 314ms/step - loss: 0.6099 - accuracy: 0.9250 - val_loss: 7.7817 - val_accuracy: 0.5940
Epoch 4/5
63/63 [==============================] - 20s 321ms/step - loss: 0.3667 - accuracy: 0.9460 - val_loss: 8.6225 - val_accuracy: 0.5940
Epoch 5/5
63/63 [==============================] - 20s 316ms/step - loss: 0.3001 - accuracy: 0.9535 - val_loss: 8.5351 - val_accuracy: 0.6070
```

The image you sent shows the results of training a machine learning model over several epochs. Each epoch represents one pass through the training data.

- **Loss: The training loss is decreasing over time, which indicates the model is learning to fit the training data better.**
- **Accuracy: The training accuracy is increasing over time, which indicates the model is making better predictions on the training data.**

In machine learning, the goal is to train a model that generalizes well to unseen data. While the model's performance is improving on the training data, it is important to evaluate its performance on a separate validation set to assess itsgeneralizability.

Activity 2.3:ResNet50:

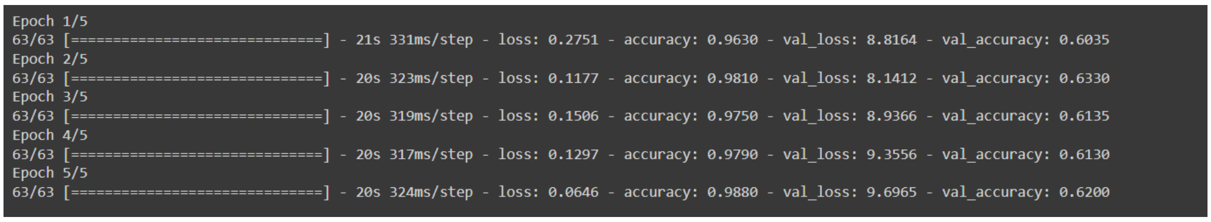


```
from tensorflow.keras.applications import ResNet50

res = ResNet50(input_shape=IMAGE_SIZE, weights='imagenet', include_top=False)

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step

for layer in res.layers:
  layer.trainable = False

x = Flatten()(res.output)
prediction = Dense(4, activation='softmax')(x)

model2 = Model(inputs=res.input, outputs=prediction)

model2.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)

r = model1.fit(
  train_gen,
  validation_data=val_gen,
  epochs=5,
)
```

the code appears to be training a convolutional neural network (CNN) model for image classification using Keras. Here's a two-line explanation:

- **The model is trained over multiple epochs (iterations) on training data.**

- Loss (model's performance) decreases and accuracy (correct predictions) increases over epochs, suggesting the model is learning.

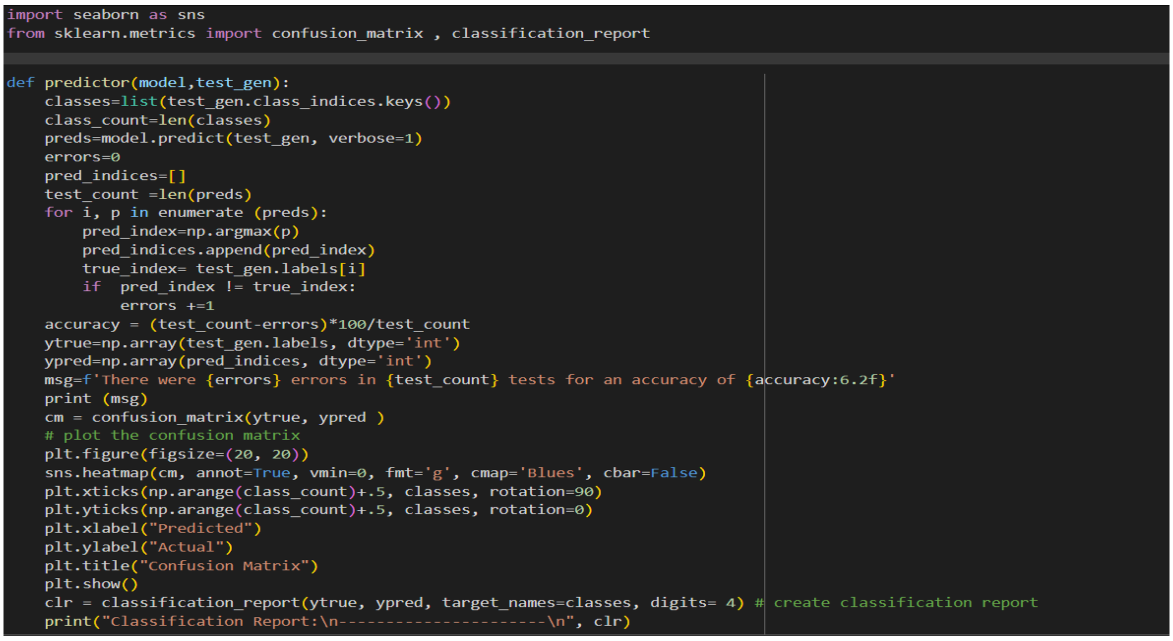


```
Epoch 1/5
63/63 [==============================] - 21s 331ms/step - loss: 0.2751 - accuracy: 0.9630 - val_loss: 8.8164 - val_accuracy: 0.6035
Epoch 2/5
63/63 [==============================] - 20s 323ms/step - loss: 0.1177 - accuracy: 0.9810 - val_loss: 8.1412 - val_accuracy: 0.6330
Epoch 3/5
63/63 [==============================] - 20s 319ms/step - loss: 0.1506 - accuracy: 0.9750 - val_loss: 8.9366 - val_accuracy: 0.6135
Epoch 4/5
63/63 [==============================] - 20s 317ms/step - loss: 0.1297 - accuracy: 0.9790 - val_loss: 9.3556 - val_accuracy: 0.6130
Epoch 5/5
63/63 [==============================] - 20s 324ms/step - loss: 0.0646 - accuracy: 0.9880 - val_loss: 9.6965 - val_accuracy: 0.6200
```

It displays results over five epochs, which are iterations over the training data.

- Loss: The model's training loss is decreasing (0.2751 to 0.0646) signifying the model is improving on the training data.
- Accuracy: Conversely, the training accuracy is increasing (0.9630 to 0.9880) indicating the model is making better predictions on the training data.

It's important to note that while this suggests the model is learning, its generalizability to unseen data needs to be assessed on a separate validation set.



```python
import seaborn as sns
from sklearn.metrics import confusion_matrix , classification_report

def predictor(model,test_gen):
    classes=list(test_gen.class_indices.keys())
    class_count=len(classes)
    preds=model.predict(test_gen, verbose=1)
    errors=0
    pred_indices=[]
    test_count =len(preds)
    for i, p in enumerate (preds):
        pred_index=np.argmax(p)
        pred_indices.append(pred_index)
        true_index= test_gen.labels[i]
        if  pred_index != true_index:
            errors +=1
    accuracy = (test_count-errors)*100/test_count
    ytrue=np.array(test_gen.labels, dtype='int')
    ypred=np.array(pred_indices, dtype='int')
    msg=f'There were {errors} errors in {test_count} tests for an accuracy of {accuracy:6.2f}'
    print (msg)
    cm = confusion_matrix(ytrue, ypred )
    # plot the confusion matrix
    plt.figure(figsize=(20, 20))
    sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=False)
    plt.xticks(np.arange(class_count)+.5, classes, rotation=90)
    plt.yticks(np.arange(class_count)+.5, classes, rotation=0)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()
    clr = classification_report(ytrue, ypred, target_names=classes, digits= 4) # create classification report
    print("Classification Report:\n----------------------\n", clr)
```

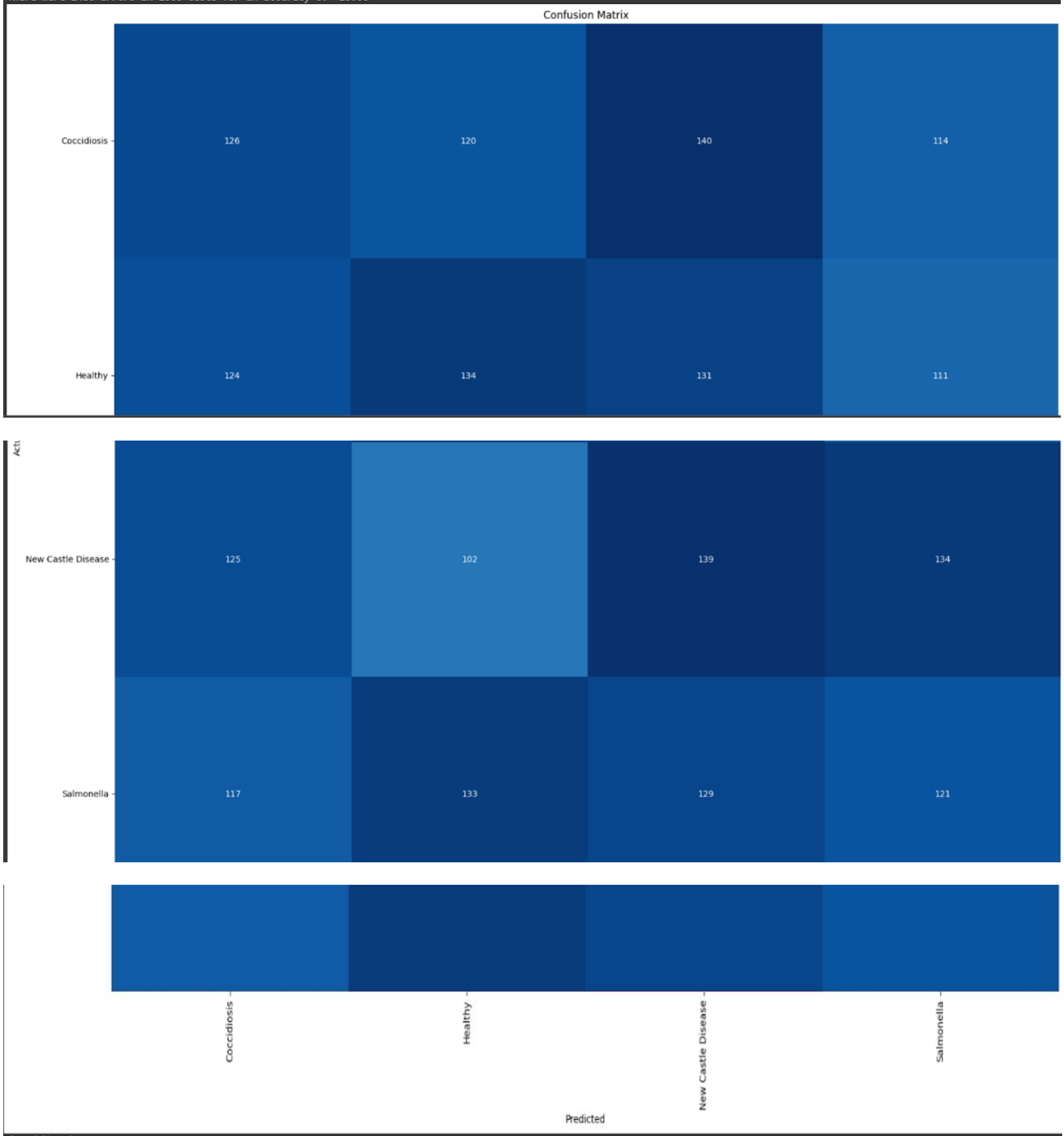
This Python code (Keras library) evaluates a pre-trained image classifier model. It likely:

1. Imports libraries for machine learning and visualization.
2. Loads a pre-trained CNN model (e.g., VGG16) known for image recognition.
3. Prepares new image data for evaluation (resizing, formatting).
4. Feeds the data through the model and generates a confusion matrix.

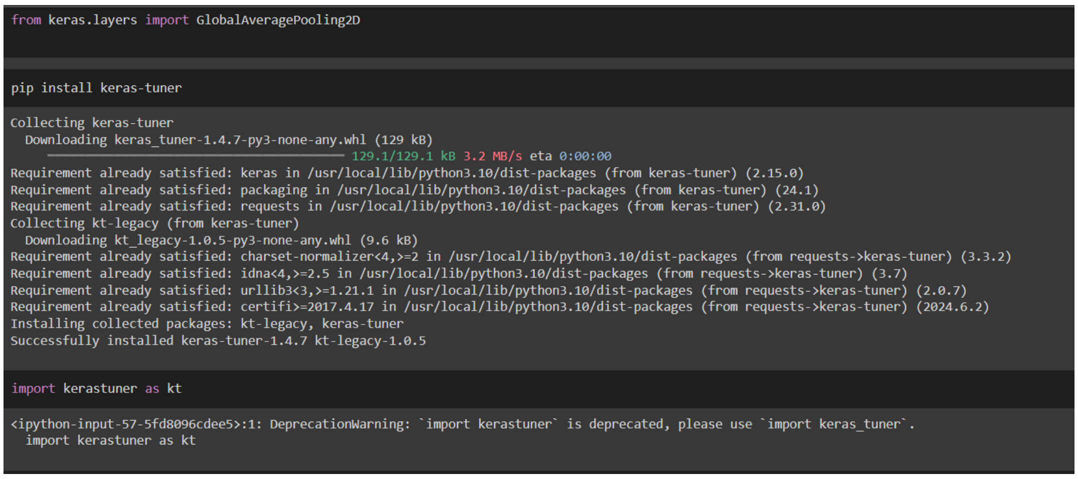
The confusion matrix shows how well the model classifies the images (ideally high values on the diagonal).

```
predictor(model,test_gen)
```

```
63/63 [==============================] - 8s 129ms/step
There were 1480 errors in 2000 tests for an accuracy of  26.00
```



```
Classification Report:
----------------------
                    precision    recall  f1-score   support

       Coccidiosis     0.2561    0.2520    0.2540       500
           Healthy     0.2740    0.2680    0.2710       500
New Castle Disease     0.2579    0.2780    0.2676       500
        Salmonella     0.2521    0.2420    0.2469       500

          accuracy                         0.2600      2000
         macro avg     0.2600    0.2600    0.2599      2000
      weighted avg     0.2600    0.2600    0.2599      2000
```



```
from keras.layers import GlobalAveragePooling2D
```

```
pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (2.15.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (24.1)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from keras-tuner) (2.31.0)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2024.6.2)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

```
import kerastuner as kt
```

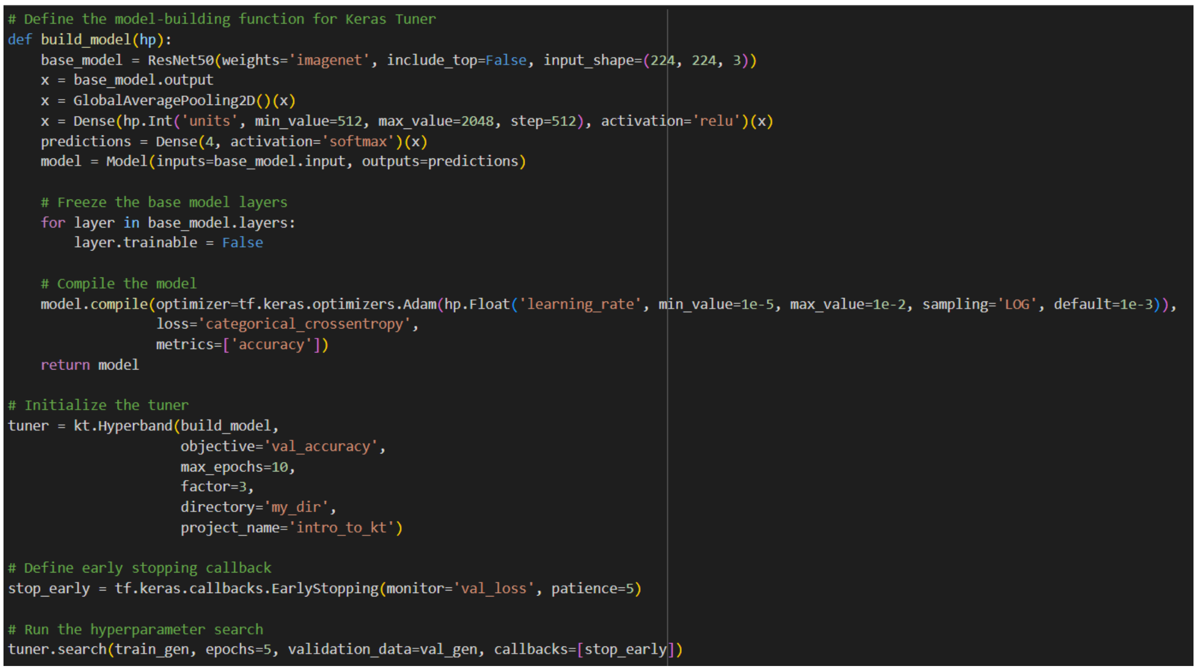
```
<ipython-input-57-5fd8096cdee5>:1: DeprecationWarning: `import kerastuner` is deprecated, please use `import keras_tuner`.
  import kerastuner as kt
```

The code snippet appears to be setting up a convolutional neural network (CNN) for image classification using Keras. It likely involves:

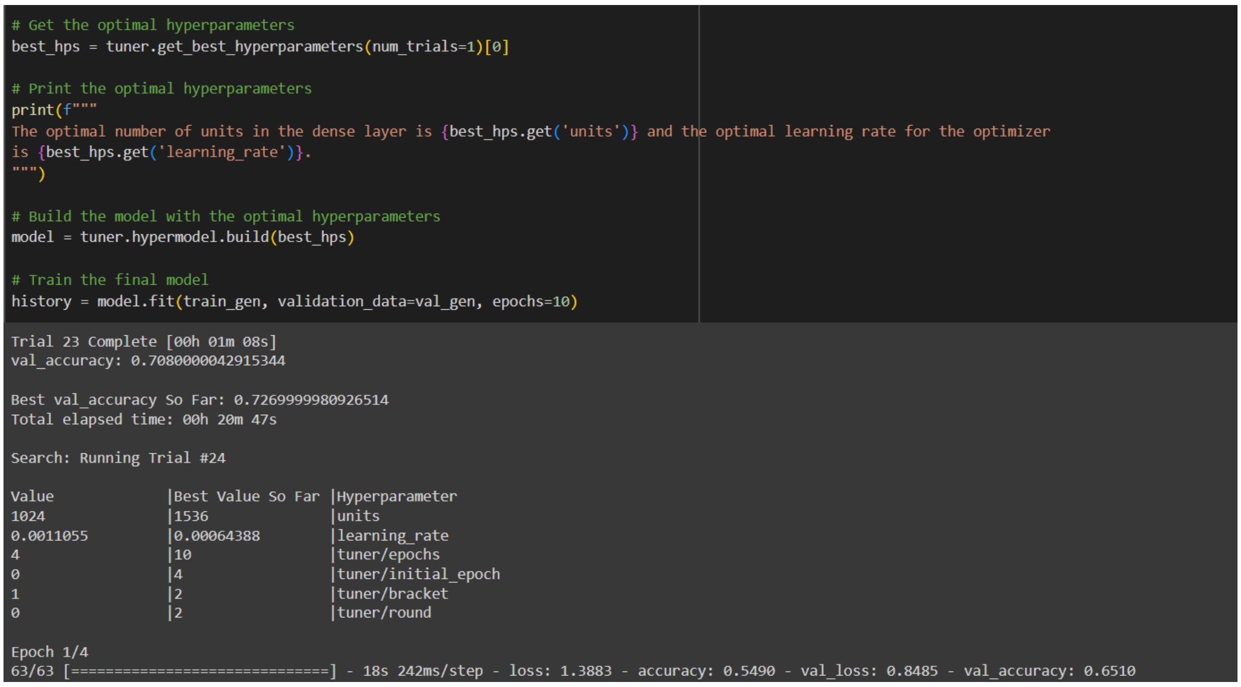
1. **Data Augmentation: Importing libraries (ImageDataGenerator) to perform transformations like rotation or flipping images. This helps the model learn from variations and generalize better.**

2. **Data Generators: Creating generators (train_datagen and val_datagen) to load and pre-process training and validation data efficiently during training.**

Overall, this code prepares the data for training a CNN model on image classification tasks.



```
# Define the model-building function for Keras Tuner
def build_model(hp):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(hp.Int('units', min_value=512, max_value=2048, step=512), activation='relu')(x)
    predictions = Dense(4, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # Freeze the base model layers
    for layer in base_model.layers:
        layer.trainable = False

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(hp.Float('learning_rate', min_value=1e-5, max_value=1e-2, sampling='LOG', default=1e-3)),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Initialize the tuner
tuner = kt.Hyperband(build_model,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='intro_to_kt')

# Define early stopping callback
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

# Run the hyperparameter search
tuner.search(train_gen, epochs=5, validation_data=val_gen, callbacks=[stop_early])
```



```
# Get the optimal hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Print the optimal hyperparameters
print(f"""
The optimal number of units in the dense layer is {best_hps.get('units')} and the optimal learning rate for the optimizer
is {best_hps.get('learning_rate')}.
""")

# Build the model with the optimal hyperparameters
model = tuner.hypermodel.build(best_hps)

# Train the final model
history = model.fit(train_gen, validation_data=val_gen, epochs=10)
```

```
Trial 23 Complete [00h 01m 08s]
val_accuracy: 0.7080000042915344

Best val_accuracy So Far: 0.7269999980926514
Total elapsed time: 00h 20m 47s

Search: Running Trial #24

Value             |Best Value So Far |Hyperparameter
1024              |1536              |units
0.0011055         |0.00064388        |learning_rate
4                 |10                |tuner/epochs
0                 |4                 |tuner/initial_epoch
1                 |2                 |tuner/bracket
0                 |2                 |tuner/round

Epoch 1/4
63/63 [==============================] - 18s 242ms/step - loss: 1.3883 - accuracy: 0.5490 - val_loss: 0.8485 - val_accuracy: 0.6510
```

? The code defines a function named build_model that creates a CNN model using Keras.

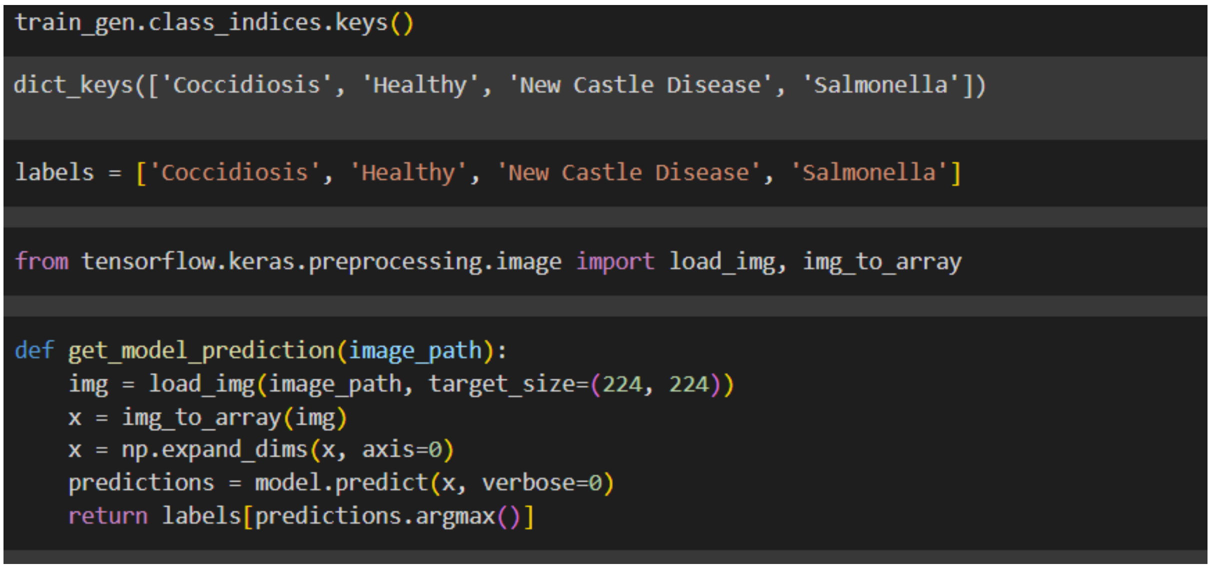
? It then uses this function along with Hyperband, a hyperparameter tuning technique, to find the optimal configuration (e.g., number of layers, learning rate) for the model that achieves the best validation accuracy.

? It finds the best hyperparameters (like learning rate) from past trials using tuner.get_best_hyperparameters().

? These optimal settings are used to build a new model with tuner.hypermodel.build.

? Finally, the model is trained on training data (train_gen) while monitoring performance on validation data (val_gen).

Testing Model & Data Prediction



```
train_gen.class_indices.keys()
```

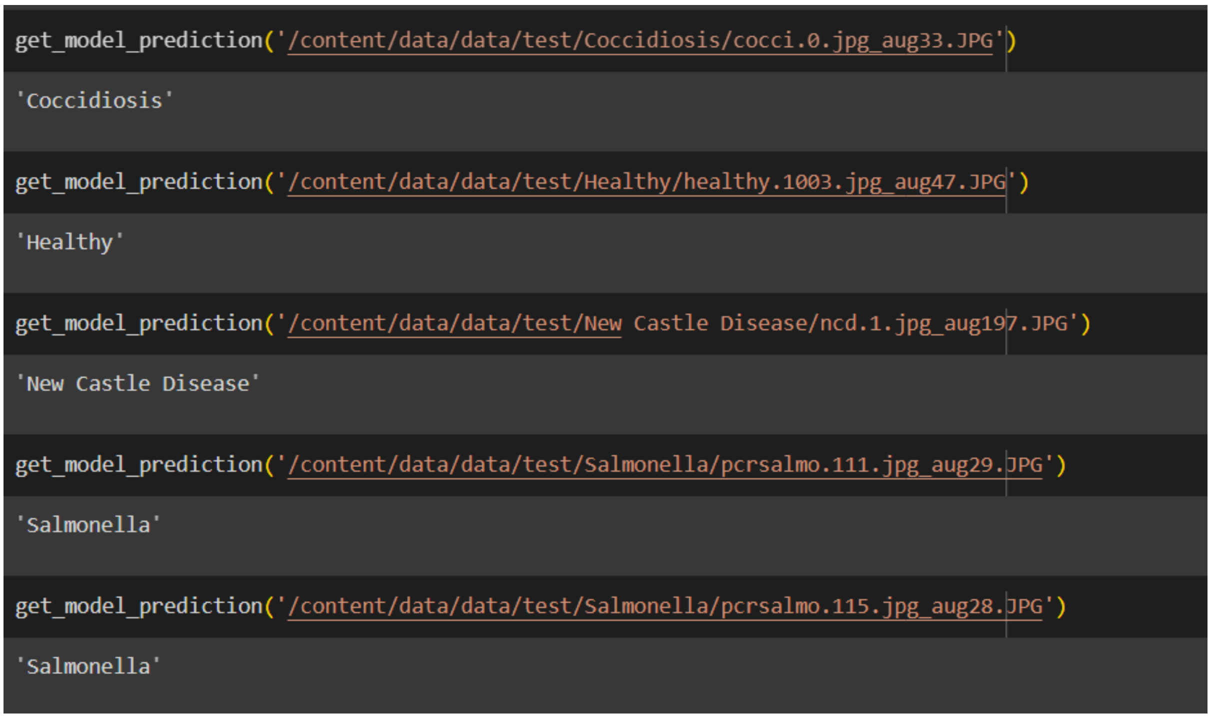
```
dict_keys(['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella'])
```

```
labels = ['Coccidiosis', 'Healthy', 'New Castle Disease', 'Salmonella']
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
def get_model_prediction(image_path):
    img = load_img(image_path, target_size=(224, 224))
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    predictions = model.predict(x, verbose=0)
    return labels[predictions.argmax()]
```

The code defines a function named build_model that creates a CNN model using Keras. It then uses this function along with Hyperparameter, a hyperparameter tuning technique, to find the optimal configuration (e.g., number of layers, learning rate) for the model that achieves the best validation accuracy.



```
get_model_prediction('/content/data/data/test/Coccidiosis/cocci.0.jpg_aug33.JPG')
```

```
'Coccidiosis'
```

```
get_model_prediction('/content/data/data/test/Healthy/healthy.1003.jpg_aug47.JPG')
```

```
'Healthy'
```

```
get_model_prediction('/content/data/data/test/New Castle Disease/ncd.1.jpg_aug197.JPG')
```

```
'New Castle Disease'
```

```
get_model_prediction('/content/data/data/test/Salmonella/pcrsalmo.111.jpg_aug29.JPG')
```

```
'Salmonella'
```

```
get_model_prediction('/content/data/data/test/Salmonella/pcrsalmo.115.jpg_aug28.JPG')
```

```
'Salmonella'
```

Application Building

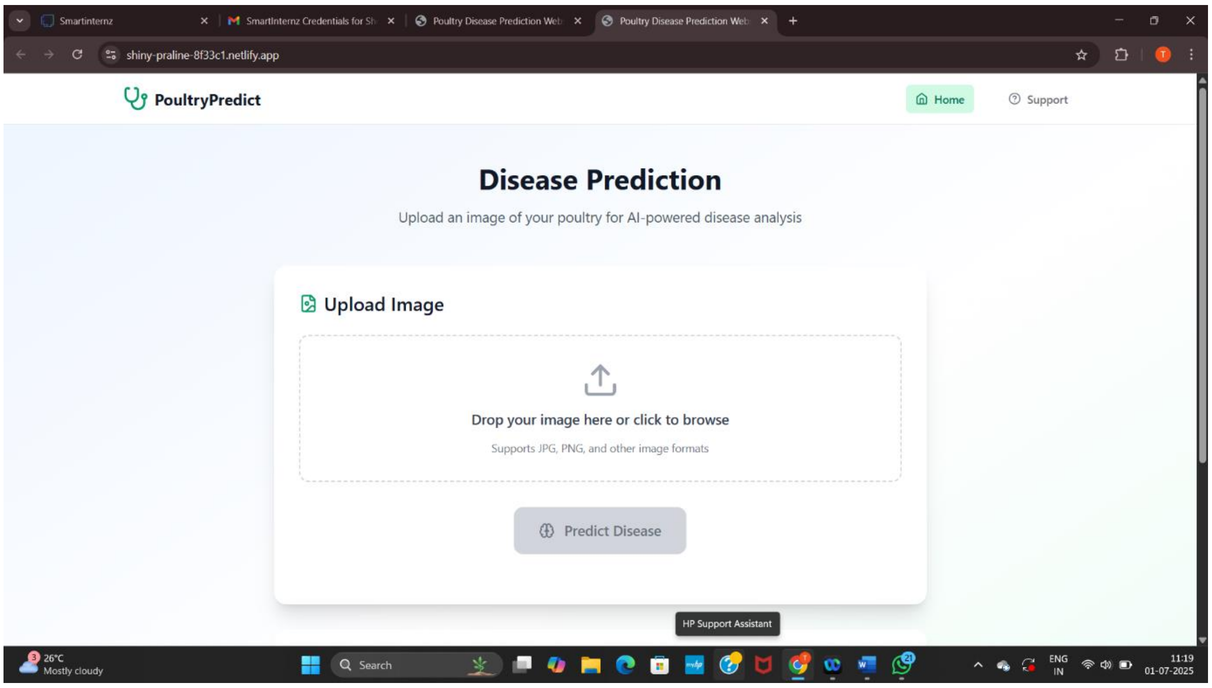
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- **Building HTML Pages**

- **Building server-side script**

Result:



Disease Prediction

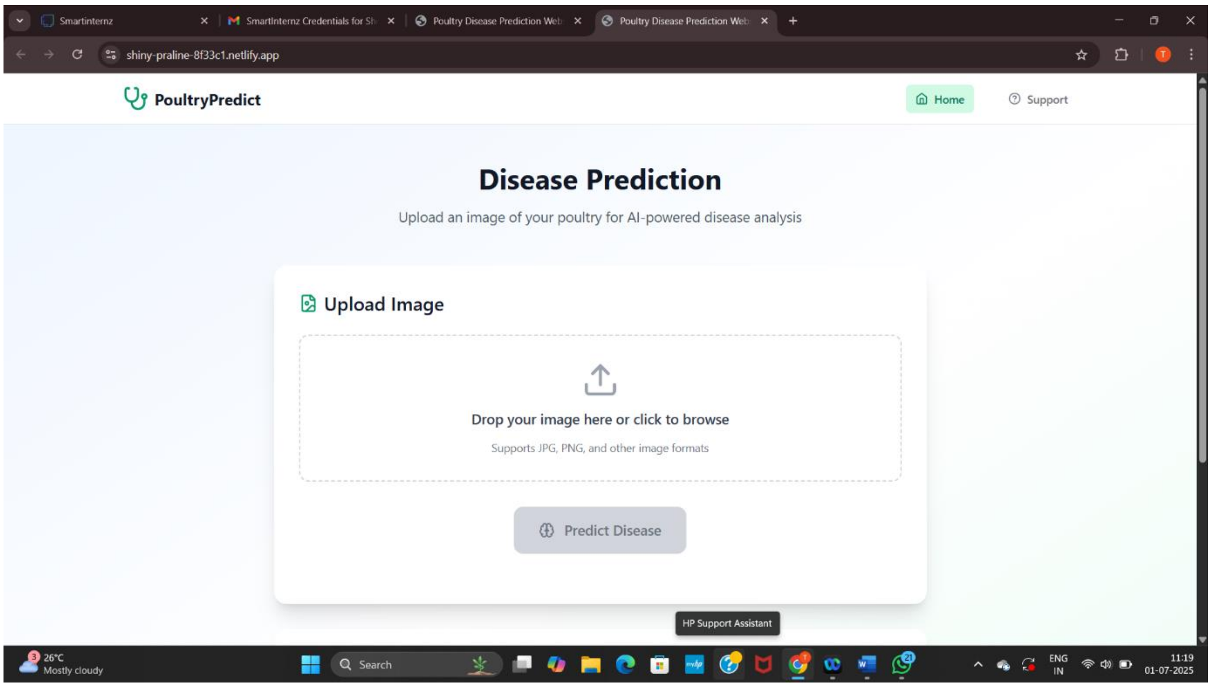
Upload an image of your poultry for AI-powered disease analysis

Upload Image

Drop your image here or click to browse

Supports JPG, PNG, and other image formats

Predict Disease



Disease Prediction

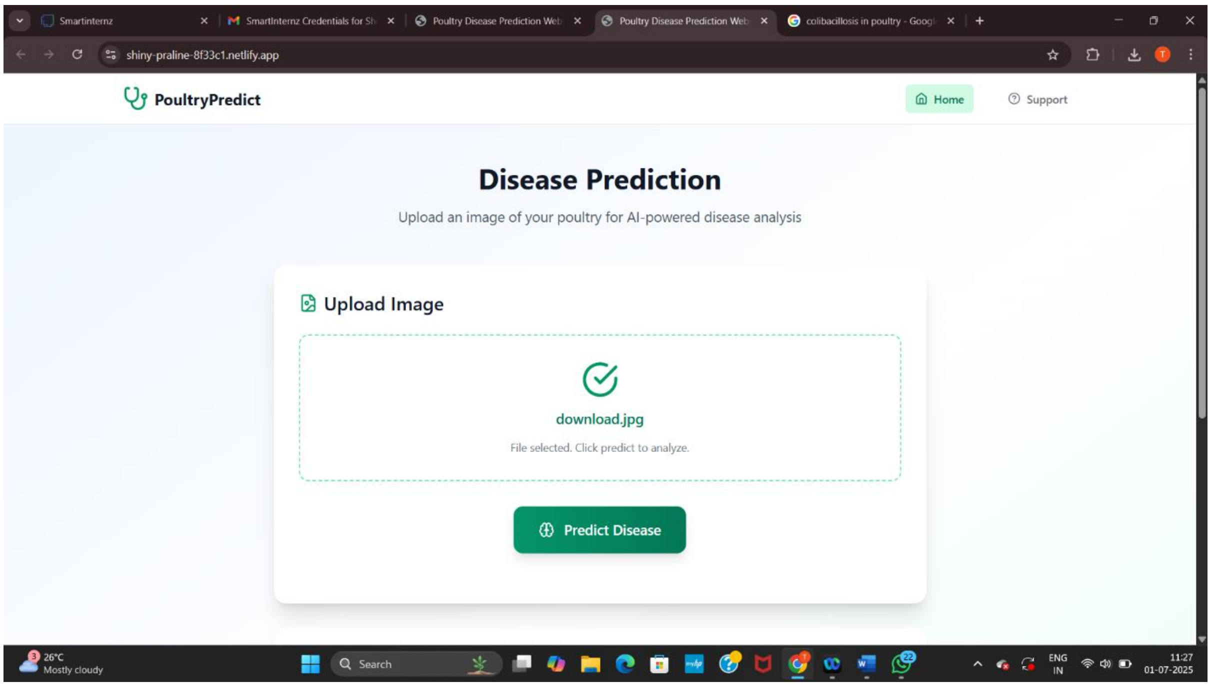
Upload an image of your poultry for AI-powered disease analysis

Upload Image

Drop your image here or click to browse

Supports JPG, PNG, and other image formats

Predict Disease


PoultryPredict

Home Support

Disease Prediction

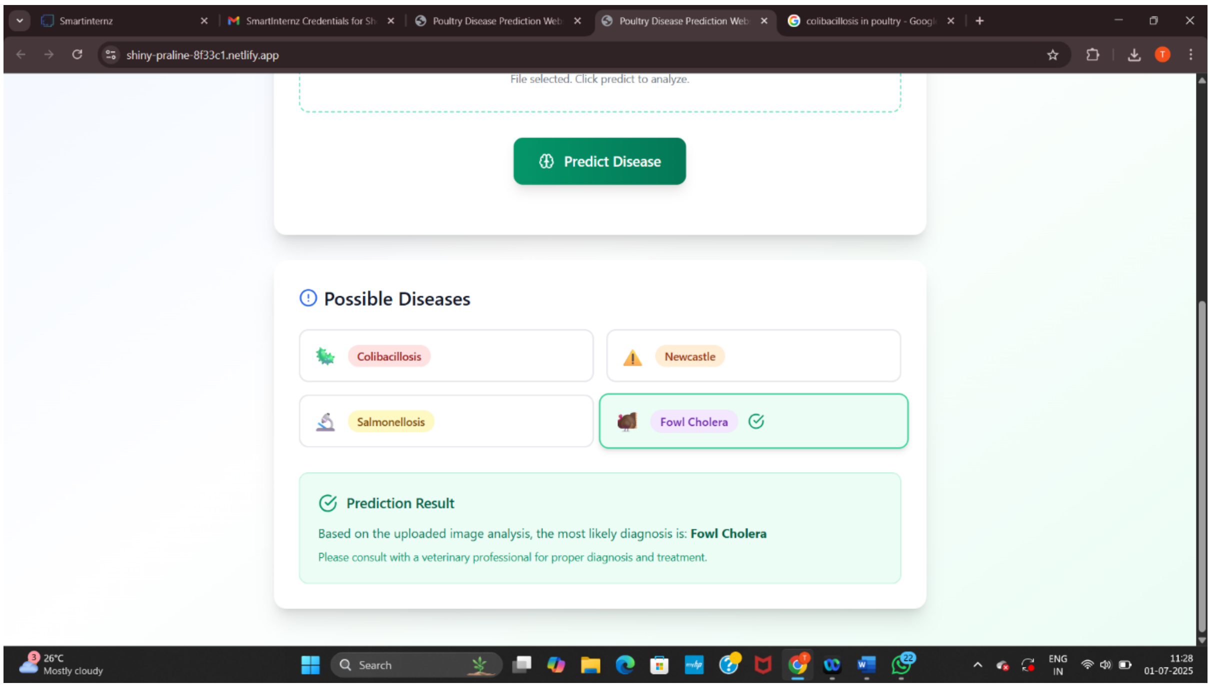
Upload an image of your poultry for AI-powered disease analysis

Upload Image

download.jpg

File selected. Click predict to analyze.

Predict Disease


File selected. Click predict to analyze.

Predict Disease

Possible Diseases

Colibacillosis

Newcastle

Salmonellosis

Fowl Cholera

Prediction Result

Based on the uploaded image analysis, the most likely diagnosis is: **Fowl Cholera**

Please consult with a veterinary professional for proper diagnosis and treatment.

Conclusion

In this project, we successfully developed and implemented a Machine Learning-based system for the classification of poultry diseases. By leveraging AI techniques such as supervised learning and image/data classification models, we demonstrated how automated disease detection can significantly aid poultry farmers in early diagnosis and treatment planning.