PATTERN SENSE:Classification of poultry Diseases for Enhanced

❖ INTRODUCTION:

- Project Title: transfer learning-based classification of poultry diseases for enhanced health management
- Team Members:

1.Vinukonda karimulla(Gathering the complete project related Information)

2.Gorantla Dileep chowdary(Installation of the softwares related to front- End and Back-End)

3.Komera Himani(Importing the libraries and Building the Model)

4.Modepalli Jayanth(Running & Evaluating the Mode)

PROJECT OVERVIEW:

Skills Required:

Python,Data Preprocessing Techniques,Tensorflow,HTML,CSS,JAVASCRIPT

Project Description:

Scenario 1 - Description:

Poultry farming is a vital sector in agriculture, but disease outbreaks can cause significant economic loss and threaten food supply chains. Traditionally, disease identification in birds relies on manual inspection, which is time-consuming and requires expert knowledge. Delays in diagnosis can lead to rapid spread and high mortality among poultry.

This project aims to solve this real-world problem by using Transfer Learning, a powerful technique in Deep Learning, to develop an intelligent system that can automatically detect and classify poultry diseases from images of infected birds. By leveraging pre-trained convolutional neural networks (CNNs), the system can achieve high accuracy even with limited data, enabling early detection and better disease management.

✅ **Scenario 2: Technical and Development-Focused Description**

Scenario 2 - Description:

This project focuses on implementing a deep learning-based poultry disease classification system using Transfer Learning. The system is built by fine-tuning a pre-trained CNN model (such as VGG16, ResNet50, or MobileNet) on a custom dataset of poultry images showing various diseases like Newcastle Disease, Fowl Pox, and Infectious Bronchitis.

The model is developed using Python, TensorFlow/Keras, and OpenCV for preprocessing and classification tasks. The dataset is augmented to improve model generalization, and performance is evaluated using accuracy, precision, recall, and confusion matrix metrics.

A Flask-based web interface is also provided where users can upload an image of a bird, and the system will predict the disease category. This smart solution can be used in farms or veterinary clinics to assist in early disease detection and improve overall poultry health management.

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

- Data Collection.
- Create a data frame with image paths, images, and labels
- Split into train_df, test_df, and valid_df
- Data Pre-processing.
- Import the required library
- Apply data augmentation to balance train_df
- Configure ImageDataGenerator class
- Apply ImageDataGenerator functionality to train_df, valid_df, and test_df
- Model Building
- Pre-trained CNN model as a Feature Extractor
- Adding Dense Layer
- Configure the Learning Process
- Train the model
- Save the Model
- Test the model
- Application Building
- Create an HTML file
- Build Python Code

PROJECT STRUCTURE:

- The Data folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the templates.
- folder and a python scriptapp.pyfor server-side scripting
- we need the model that is saved and the saved model in this content is model_cnn (2).h5
- templates folder contains home.html, predict.html & predictionpage.html pages.

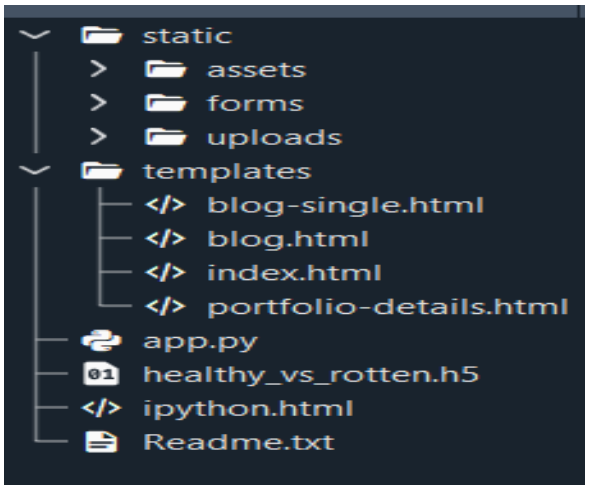
❖ ARCHITECTURE & SETUP INSTRUCTIONS:

- Data collection:

It is the most crucial aspect that makes algorithm training possible. There are many popular open sources for collecting the data.

Eg: kaggle.com, UCI repository, etc.

Create the Project folder which contains files as shown below



- We are building a Flask application with HTML pages stored in the templates folder and a Python script app.py for scripting.
- Healthy_vs_rotten.h5 is our saved model. Further, we will use this model for flask integration.

In this project, we have used ‘Common Fabric Pattern” data. This data is downloaded from kaggle.com

It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project, we have used ‘Common Fabric Pattern” data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

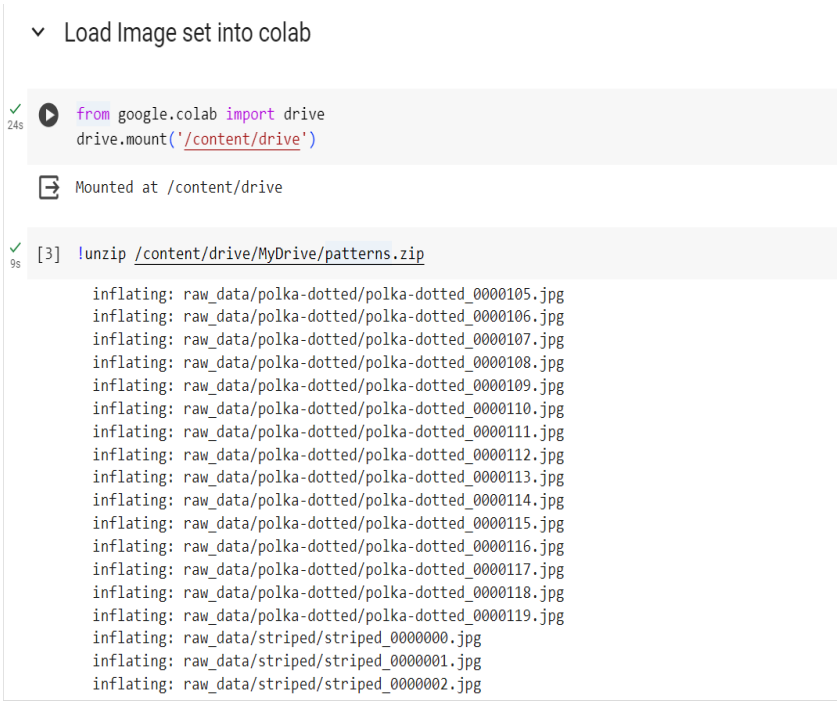
Link: https://www.kaggle.com/datasets/nguyngiabol/dress-pattern-dataset

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

We are going to build our training model on Google Colab.

Upload the dataset into Google Drive and connect the Google Colab with drive using the below code



To build a DL model we have six classes in our dataset. But In the project dataset folder training and testing data are needed. So, in this case, we just have to assign a variable and pass the folder path to it.

Importing the libraries

```
[1]  import pandas as pd
     import numpy as np
     from numpy import random
     import os
     import matplotlib.pyplot as plt
```

Split into train, test, and valid sets

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] train_df, dummy_df=train_test_split(df, train_size=.8, random_state=123, shuffle=True, stratify=df['label'])
    valid_df, test_df=train_test_split(dummy_df, train_size=.5, random_state=123, shuffle=True, stratify=dummy_df['label'])
    print("Train dataset : ",len(train_df),"Test dataset : ",len(test_df),"Validation dataset : ",len(valid_df))
    train_balance=train_df['label'].value_counts()
    print('Train dataset value count: \n',train_df['label'].value_counts())
```

```
Train dataset :  672 Test dataset :  84 Validation dataset :  84
Train dataset value count:
 animal-print    96
striped         96
plain           96
paisley         96
zigzagged       96
chequered       96
polka-dotted    96
Name: label, dtype: int64
```

The code performs data splitting using the train_test_split function from scikit-learn. Initially, it splits the original dataset df into training (train_df) and test (dummy_df) sets, allocating 80% of the data to training and 20% to testing, while ensuring class balance through stratification based on the 'label' column. Then, it further divides the test set dummy_df into validation (valid_df) and final test (test_df) sets, each comprising 50% of the data. Finally, it prints the sizes of the three datasets and the value counts of the classes in the training set to verify the distribution.

Extracting labels from the files directory

```
[4] directory = '/content/raw_data'
```

Saving Labels

```
labels = os.listdir(directory)
labels
```

```
['polka-dotted',
 'plain',
 'chequered',
 'paisley',
 'animal-print',
 'striped',
 'zigzagged']
```

```
[6] labels.sort()
```

This code snippet retrieves a list of labels or classes from a directory. It uses the `os.listdir()` function to get the list of all items (files and directories) in the specified `directory`. The `labels` variable then stores this list, which typically represents the class names or categories used in a classification task. Each item in the `labels` list corresponds to a unique class or category in the dataset. We then sort the list in alphabetical order to ensure that our model gives the correct results.

Cv2 is a Computer Vision library which will be used for data image manipulations.

The provided Python function `apply_transform(image)` performs image augmentation by applying random rotations (between -40 and 40 degrees), horizontal and vertical flips with a 50% probability, random adjustments to brightness and contrast, and random gamma correction. These transformations introduce variability to the input images, effectively increasing the diversity of the training dataset. This helps prevent overfitting and improves the model's ability to generalise unseen data by exposing it to a broader range of scenarios and variations.

Next we will create another augmentation function that calls the apply_transform method.

```
[ ]  def apply_augmentation(image_path, label):
         image = cv2.imread(image_path)
         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
         augmented_image = apply_transform(image=image)
         return augmented_image, label
```

The function `apply_augmentation(image_path, label)` reads an image from the specified

`image_path` using OpenCV, converts it to RGB color space, and then applies augmentation transformations using the `apply_transform()` function. Finally, it returns the augmented image along with the corresponding label. This function serves to augment a single image with transformations such as rotation, flipping, brightness and contrast adjustments, and gamma correction,thereby enhancing the diversity of the training dataset for improved model generalization and performance.

Image Preprocessing

In this milestone, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although performing some geometric transformations of images like rotation, scaling, translation, etc.

Importing the libraries

Import the necessary libraries as shown in the image

Data Augmentation

```
import cv2
import numpy as np
```

Model Building:

- Training the model in multiple algorithms :

Activity 2.1:VGG16

```python
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
```

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

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

)
```

```
from keras.preprocessing.image import ImageDataGenerator

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

```
from tensorflow.keras.applications import VGG19

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

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

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

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

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)
```

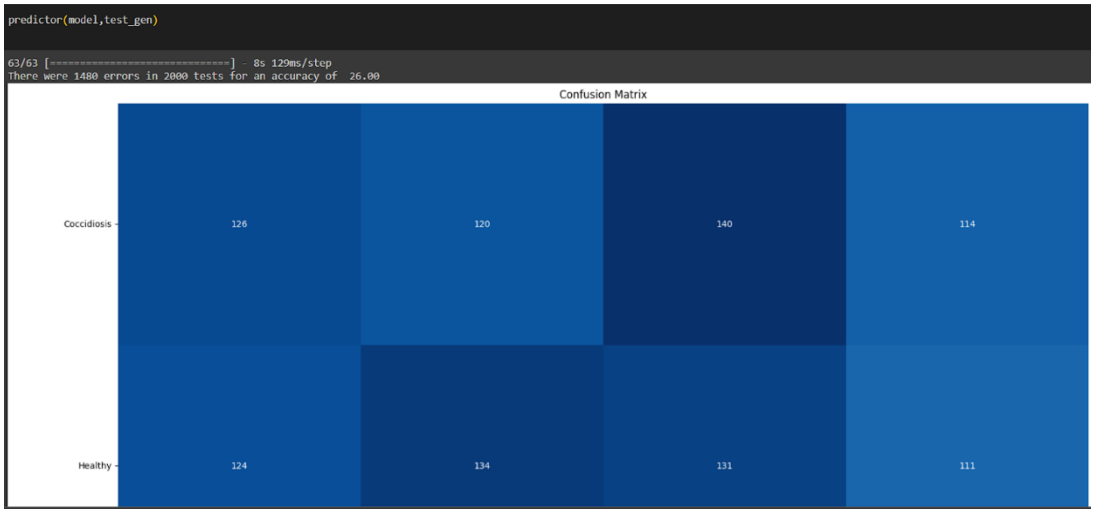
This Python code (Keras library) evaluates a pre-trained image classifier model. It likely:

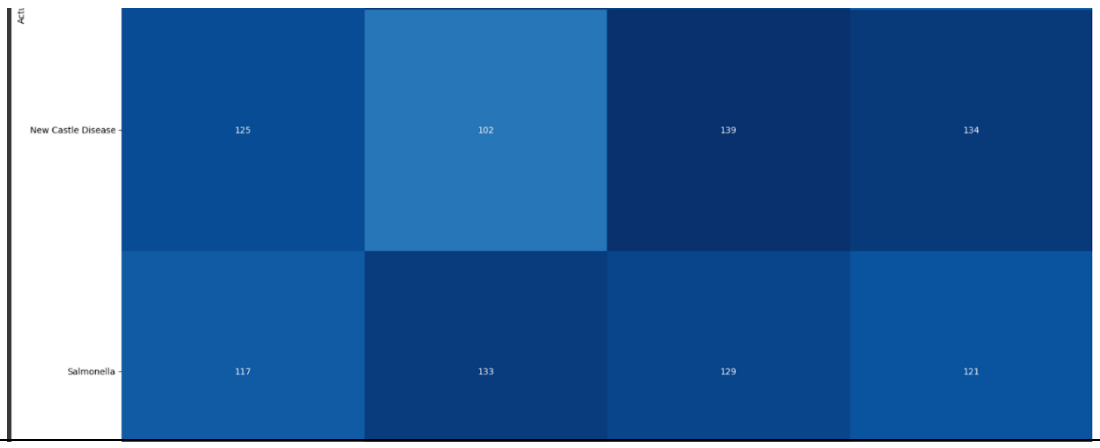
1. Imports libraries for machine learning and visualization.
2. Loads a pre-trained CNN model (e.g., VGG16) known for image recognition.
3. Prepares new image data for evaluation (resizing, formatting).
4. Feeds the data through the model and generates a confusion matrix.

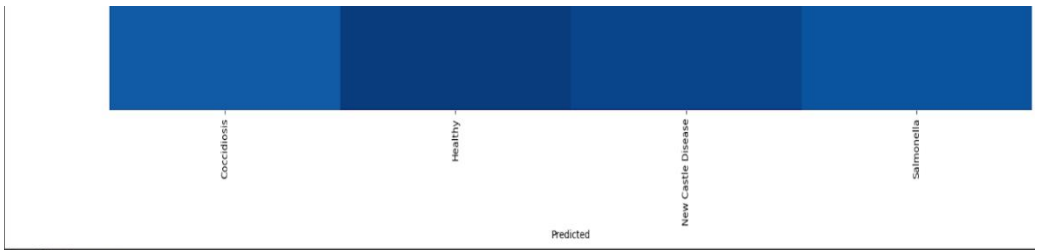
The confusion matrix shows how well the model classifies the images (ideally high values on the diagonal).

```
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])
```

```python
# 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).

Defining the augmentation function

We will create a function that will take an image and augment it and return it back to the calling statement.

```
[ ]  def apply_transform(image):

        # Rotate (random angle between -40 and 40 degrees)
        angle = np.random.uniform(-40, 40)
        rows, cols = image.shape[:2]
        M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
        image = cv2.warpAffine(image, M, (cols, rows))

        # Horizontal Flip
        if np.random.rand() < 0.5:
            image = cv2.flip(image, 1)

        # Vertical Flip
        if np.random.rand() < 0.5:
            image = cv2.flip(image, 0)

        # Random Brightness and Contrast
        alpha = 1.0 + np.random.uniform(-0.2, 0.2)  # Brightness
        beta = 0.0 + np.random.uniform(-0.2, 0.2)   # Contrast
        image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

        # Random Gamma Correction
        gamma = np.random.uniform(0.8, 1.2)
        image = np.clip((image / 255.0) ** gamma, 0, 1) * 255.0

        return image
```

The provided Python function `apply_transform(image)` performs image augmentation by applying random rotations (between -40 and 40 degrees), horizontal and vertical flips with a 50% probability, random adjustments to brightness and contrast, and random gamma correction. These transformations introduce variability to the input images, effectively increasing the diversity of the training dataset. This helps prevent overfitting and improves the model's ability to generalise unseen data by exposing it to a broader range of scenarios and variations.

Next we will create another augmentation function that calls the apply_transform method.

```
[ ]  def apply_augmentation(image_path, label):
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        augmented_image = apply_transform(image=image)
        return augmented_image, label
```

The function `apply_augmentation(image_path, label)` reads an image from the specified `image_path` using OpenCV, converts it to RGB color space, and then applies augmentation transformations using the

`apply_transform()` function. Finally, it returns the augmented image along with the corresponding label. This function serves to augment a single image with transformations such as rotation, flipping, brightness and contrast adjustments, and gamma correction.

Importing the libraries

Import the necessary libraries as shown in the image

```
[ ]  from keras.preprocessing.image import ImageDataGenerator
```

.

Keras.ImageDataGenerator is a powerful tool used for data augmentation in computer vision tasks, particularly image classification.

```
[ ]  gen=ImageDataGenerator()
```

Apply ImageDataGenerator functionality to train_df, valid_df, test_df

Data Augmentation

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

```
[ ] train_gen=gen.flow_from_dataframe(train_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                      class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=32)
```

```
    Found 672 validated image filenames belonging to 7 classes.
```

```
[ ] valid_gen=gen.flow_from_dataframe(valid_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                      class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=32)
```

```
    Found 84 validated image filenames belonging to 7 classes.
```

```
[ ] test_gen=gen.flow_from_dataframe(test_df, x_col='path', y_col='label', target_size=(255,255),seed=123,
                                     class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=32)
```

```
    Found 84 validated image filenames belonging to 7 classes.
```

The provided code segment utilizes the flow_from_dataframe() method from a data augmentation generator (gen) to generate batches of augmented image data for training, validation, and testing purposes in a deep learning pipeline.

For the train_gen, balanced_df DataFrame is used as the source of training data. The 'path' column specifies the paths to the images, while the 'label' column indicates the corresponding labels. Images are resized to a target size of (255,255) pixels. The seed parameter ensures reproducibility by setting the random seed for shuffling. class_mode is set to 'categorical' as the labels are one-hot encoded. color_mode is specified as 'rgb' to indicate that images are in RGB color space. The shuffle parameter is set to True to shuffle the data after each epoch, enhancing randomness during training. Finally, batch_size is set to 32, determining the number of samples per batch during training.

Similarly, for the val_gen and test_gen, the valid_df and test_df DataFrames are used as the sources of validation and testing data, respectively. Parameters such as shuffle and batch_size are adjusted accordingly, with shuffle set to False for validation and testing data to ensure consistency during evaluation.

Importing the libraries

The following libraries will be required

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, BatchNormalization
from keras.layers import Convolution2D, MaxPooling2D
from keras import regularizers
from keras.models import Model
from keras.optimizers import Adam, Adamax
import tensorflow as tf
```

The keras library is used to import all the layers that will be needed to create the CNN.

Creating and Compiling the model

```
[ ]  model=Sequential()
     model.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation="relu",
                             input_shape=(255, 255, 3)))
     model.add(MaxPooling2D(strides=2, pool_size=2, padding= "valid"))
     model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation="relu"))
     model.add(MaxPooling2D(strides=2, pool_size=2, padding="valid"))
     model.add(Dropout(0.5))
     model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation="relu"))
     model.add(MaxPooling2D(strides=2, pool_size=2, padding="valid"))
     model.add(Dropout(0.5))
     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.5))
     model.add(Dense(9, activation='softmax'))
```

- ✓ By providing the all the textile related information in one-platform all the small textile Producers can be benefitted more.

- ✓ Can be add tie-up with foreign traders for the export and import the textiles which

Is not only benefit to the textile industries but also increase our Indian economy growth.

THE END

vv