File: finalclassifierwith3attribute.py

Documented code for finalclassifierwith3attribute.py:

```
# -*- coding: utf-8 -*-
"""Finalclassifierwith3Attribute.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vsRq0vBb_lmABIzov2E4wlx0uOPFt824
"""

# Commented out IPython magic to ensure Python compatibility.
# %pip install numpy pandas

import time

# Record the start time
start_time = time.time()

import numpy as np
import pandas as pd
#import seaborn as sns
#import matplotlib.pyplot as plt
#Python 3.11.2

# Commented out IPython magic to ensure Python compatibility.
# %pip install tensorflow


# Commented out IPython magic to ensure Python compatibility.
# %pip install opencv-python


#from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras import applications
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Flatten, Dense, Dropout
from keras.preprocessing import image


import cv2


import warnings
warnings.filterwarnings('ignore')


import os




"""<h1>Dataset</h1>"""


from google.colab import drive
```

```
drive.mount("/content/drive")


# datasets

#labels = pd.read_csv("./labels.csv")

labels = pd.read_csv("/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture 3Attribute/KaranjN

#sample = pd.read_csv('/content/drive/My Drive/dog/sample_submission.csv')


# folders paths

train_path = "/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture 3Attribute/Allinone3Attribu

#test_path = "./test"


#import os.listdir() to os.pathjoin to the files, import files , print name of files, label files

#loading images dataset of documents : labels of documents:img,doc , train dataset:img of doc, test datase


labels.head()


labels = labels.sample(frac=1)


# invoicebank invoice

# insurance   car

# insurance   bike


labels["id"] = labels["image"]


#what is validation split

# Data agumentation and pre-processing using tensorflow
```

```
gen = ImageDataGenerator(
                  rescale=1./255.,
                  horizontal_flip = True,
                  validation_split=0.2 # training: 80% data, validation: 20% data
                 )


train_generator = gen.flow_from_dataframe(
    labels, # dataframe
    directory = train_path, # images data path / folder in which images are there
    x_col = 'image',
    y_col = 'type',
    subset="training",
    color_mode="rgb",
    target_size = (331,331), # image height , image width
    class_mode="categorical",
    batch_size=32,
    shuffle=True,
    seed=42,
)



validation_generator = gen.flow_from_dataframe(
    labels, # dataframe
    directory = train_path, # images data path / folder in which images are there
    x_col = 'image',
    y_col = 'type',
    subset="validation",

    color_mode="rgb",

    target_size = (331,331), # image height , image width

    class_mode="categorical",

    batch_size=32,

    shuffle=True,

    seed=42,

)


import sys

import PIL

from PIL import Image


x,y = next(train_generator)

x.shape # input shape of one record is (331,331,3) , 32: is the batch size


#x.shape , (32,331,331,3)


y.shape #y.shape (32,3)


y[3]


# Commented out IPython magic to ensure Python compatibility.

# %pip install matplotlib


import matplotlib.pyplot as plt
```

```
a = train_generator.class_indices
class_names = list(a.keys())  # storing class/breed names in a list
# a is dictionary with each breed assigned number , a.keys is dictionary of only keys, list(a.keys()) making

def plot_images(img, labels):
    plt.figure(figsize=[15, 10])
    for i in range(25):
        plt.subplot(5, 5, i+1)
        plt.imshow(img[i])
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis('off')

plot_images(x,y)

class_names

a.keys()

"""<h1>Model Build</h1>"""

# load the InceptionResNetV2 architecture with imagenet weights as base
base_model = tf.keras.applications.InceptionResNetV2(
                     include_top=False,
           weights='imagenet',
           input_shape=(331,331,3)
           )

base_model.trainable=False
# For freezing the layer we make use of layer.trainable = False
# means that its internal state will not change during training.
# model's trainable weights will not be updated during fit(),
# and also its state updates will not run.

model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.BatchNormalization(renorm=True),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(6, activation='softmax')
  ])

model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
# categorical cross entropy is taken since its used as a loss function for
# multi-class classification problems where there are two or more output labels.
# using Adam optimizer for better performance
# other optimizers such as sgd can also be used depending upon the model
```

```
model.summary()

early = tf.keras.callbacks.EarlyStopping( patience=10,
                            min_delta=0.001,
                            restore_best_weights=True)
# early stopping call back

"""<h1>Train Model</h1>"""

print(train_generator.batch_size)
train_generator.n//train_generator.batch_size

print(validation_generator.batch_size)
validation_generator.n//validation_generator.batch_size

batch_size=32
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = validation_generator.n//validation_generator.batch_size

# fit model
history = model.fit(train_generator,
             steps_per_epoch=STEP_SIZE_TRAIN,
             validation_data=validation_generator,
             validation_steps=STEP_SIZE_VALID,
             epochs=5,
                    callbacks=[early])


"""<h1>Save Model</h1>"""


model.save("/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture/3Attribute3AttributePlantM


"""# @title Default title text

from keras.models import load_model

import os

model.save(os.path.join('models','/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture/3Attrib

new_model = load_model('/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture/3Attribute3At

#yhatnew = new_model.predict(np.expand_dims(resize/255,0))


<h1>Model Performance</h1>

"""


# store results

acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

loss = history.history['loss']

val_loss = history.history['val_loss']



# plot results

# accuracy

plt.figure(figsize=(10, 16))
plt.rcParams['figure.figsize'] = [16, 9]
plt.rcParams['font.size'] = 14
plt.rcParams['axes.grid'] = True
plt.rcParams['figure.facecolor'] = 'white'
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.title(f'\nTraining and Validation Accuracy. \nTrain Accuracy:{str(acc[-1])}\nValidation Accuracy: {str(val_a

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.title(f'Training and Validation Loss. \nTrain Loss:{str(loss[-1])}\nValidation Loss: {str(val_loss[-1])}')
plt.xlabel('epoch')
plt.tight_layout(pad=3.0)
plt.show()

accuracy_score = model.evaluate(validation_generator)
print(accuracy_score)
print("Accuracy: {:.4f}%".format(accuracy_score[1] * 100))

print("Loss: ",accuracy_score[0])
```

```
"""<h1>Test Model</h1>"""


test_img_path = "/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture 3Attribute/Allinone3Att

 #test_img_path = "./home/Documents/Documents/Project/Invoice Insurance Images/train/1.jpg"

img = cv2.imread(test_img_path)

plt.imshow(img)

resized_img = cv2.resize(img, (331, 331)).reshape(-1, 331, 331, 3)/255


#plt.figure(figsize=(6,6))

#plt.title("TEST IMAGE")

#plt.imshow(resized_img[0])

prediction = model.predict(resized_img)

print(class_names[np.argmax(prediction)])


"""## **Now we list the medicinal properties of the plant detected**"""


class_prediction=class_names[np.argmax(prediction)]


if class_prediction == 'Karanj Trunk' or class_prediction == 'Karanj Leaf' or class_prediction == 'Karanj See

    print('Karanj Popularly known as Indian Beech in outside India is a medicinal herb used mainly for skin d


if class_prediction == 'Neem Trunk' or class_prediction == 'Neem Leaf' or class_prediction == 'Neem Seed

    print('Neem is a versatile medicinal tree. Neem oil and neem leaves are used for various medicinal purpo


if class_prediction == 'Peeple Trunk' or class_prediction == 'Peeple Leaf' or class_prediction == 'Peeple Se
```

```
    print('Peepal: The bark of the Peeple tree, rich in vitamin K, is an effective complexion corrector and pres

end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time

print(f"Time taken: {elapsed_time:.2f} seconds")

Time_in_Minute = elapsed_time / 60
print(f"Time taken: {Time_in_Minute:.2f} minutes")
```

Folder: plant

Folder: app

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: admin.py

Documented code for admin.py:

```
from django.contrib import admin

# Register your models here.
```

File: apps.py

Documented code for apps.py:

```
from django.apps import AppConfig



class AppConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'app'
```

File: forms.py

Documented code for forms.py:

```
# forms.py
from django import forms
from .models import UploadedImage

class ImageUploadForm(forms.ModelForm):
    class Meta:
        model = UploadedImage
        fields = ['image']
```

Folder: migrations

File: 0001_initial.py

Documented code for 0001_initial.py:

```
# Generated by Django 4.2.8 on 2023-12-29 14:16
```

```
from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='UploadedImage',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_nam
                ('image', models.ImageField(upload_to='uploaded_images/')),
            ],
        ),
    ]
```

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: ml_model.py

Documented code for ml_model.py:

```
# ml_model.py

import cv2

from keras.models import load_model

import numpy as np


# Load the pre-trained model

model = load_model('C:\\Users\\kyath\\OneDrive\\Desktop\\plantrecognition\\plant_recognition_model.h5')

print(model)

def plant_recognition_model(image_instance):

    img = cv2.imdecode(np.frombuffer(image_instance.read(), np.uint8), cv2.IMREAD_COLOR)

    resized_img = cv2.resize(img, (331, 331)).reshape(-1, 331, 331, 3) / 255.0

    prediction = model.predict(resized_img)

    class_names = ['Karanj Trunk', 'Karanj Leaf', 'Karanj Seed', 'Neem Trunk', 'Neem Leaf', 'Neem Seed', 'Pe

    class_prediction = class_names[np.argmax(prediction)]


    if class_prediction in ['Karanj Trunk', 'Karanj Leaf', 'Karanj Seed']:

        output_text = 'Karanj: Popularly known as Indian Beech in outside India is a medicinal herb used mair

    elif class_prediction in ['Neem Trunk', 'Neem Leaf', 'Neem Seed']:

        output_text = 'Neem: A versatile medicinal tree. Neem oil and neem leaves are used for various medic

    elif class_prediction in ['Peeple Trunk', 'Peeple Leaf', 'Peeple Seed']:

        output_text = 'Peepal: The bark of the Peepal tree, rich in vitamin K, is an effective complexion correc

    else:

        output_text = 'Unknown Plant'
    return class_prediction, output_text
```

File: models.py

Documented code for models.py:

```
from django.db import models


class UploadedImage(models.Model):
    image = models.ImageField(upload_to='uploaded_images/')
```

File: tests.py

Documented code for tests.py:

```
from django.test import TestCase


# Create your tests here.
```

File: urls.py

Documented code for urls.py:

```
# your_app/urls.py
from django.urls import path
from .views import upload_image


app_name = 'your_app'
```

```
urlpatterns = [
    path('upload/', upload_image, name='upload_image'),
    # Add other URL patterns as needed
]
```

File: views.py

Documented code for views.py:

```
from django.shortcuts import render
from .forms import ImageUploadForm
from .ml_model import plant_recognition_model

def upload_image(request):
    if request.method == 'POST':
        form = ImageUploadForm(request.POST, request.FILES)
        if form.is_valid():
            # Save the form to get the uploaded image instance
            uploaded_image = form.save(commit=False)

            # Pass the image URL to the recognition model
            result = plant_recognition_model(uploaded_image.image)

            return render(request, 'result.html', {'result': result, 'uploaded_image': uploaded_image.image.url})
    else:
        form = ImageUploadForm()
    return render(request, 'upload.html', {'form': form})
```

File: manage.py

Documented code for manage.py:

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'plant.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Folder: media

Folder: plant

Folder: templates