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/KaranjNeemDataset (1).csv")
#sample = pd.read_csv('/content/drive/My Drive/dog/sample_submission.csv')

# folders paths
train_path = "/content/drive/MyDrive/PhD Work/InceptionResNetV2 architecture 3Attribute/Allinone3Attribute"
#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 dataset:img of doc

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 dictionary to list

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/3Attribute3AttributePlantModel.h5")

"""# @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/3Attribute3AttributePlantModel.h5')) #model directory
new_model          =          load_model('/content/drive/MyDrive/PhD          Work/InceptionResNetV2
architecture/3Attribute3AttributePlantModel.h5')
#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_acc[-1])}')

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/Allinone3Attribute/1456.jpg"
 #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 Seed':
    print('Karanj Popularly known as Indian Beech in outside India is a medicinal herb used mainly for skin disorders. Karanja oil is applied to the skin to manage boils, rashes, and eczema as well as heal wounds due to its antimicrobial properties. The oil can also be useful in arthritis due to its anti-inflammatory activities.')
```

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 purposes. It has anti-inflammatory, antifungal, and antibacterial properties, making it beneficial for skin care, hair care, and managing various health conditions.')

if class_prediction == 'Peeple Trunk' or class_prediction == 'Peeple Leaf' or class_prediction == 'Peeple Seed':

print('Peepal: The bark of the Peeple tree, rich in vitamin K, is an effective complexion corrector and preserver. It also helps in various ailments such as Strengthening blood capillaries, minimising inflammation, Healing skin bruises faster, increasing skin resilience, treating pigmentation issues, wrinkles, dark circles, lightening surgery marks, scars, and stretch marks.')

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_name='ID')),
                ('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', 'Peeple Trunk', 'Peeple Leaf', 'Peeple Seed']
    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 mainly for skin disorders. Karanja oil is applied to the skin to manage boils, rashes, and eczema as well as heal wounds due to its antimicrobial properties. The oil can also be useful in arthritis due to its anti-inflammatory activities.'
    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 medicinal purposes. It has anti-inflammatory, antifungal, and antibacterial properties, making it beneficial for skin care, hair care, and managing various health conditions.'
    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 corrector and preserver. It also helps in various ailments such as strengthening blood capillaries, minimizing inflammation, healing skin bruises faster, increasing skin resilience, treating pigmentation issues, wrinkles, dark circles, lightening surgery marks, scars, and stretch marks.'
    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

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: asgi.py

Documented code for asgi.py:

```
"""
ASGI config for plant project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'plant.settings')
```

```
application = get_asgi_application()
```

File: settings.py

Documented code for settings.py:

```
"""
Django settings for plant project.

Generated by 'django-admin startproject' using Django 4.2.8.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-0dkmbb0paa08u1u56@_j9tkq6ruf*#jsi&=g((n6xu*ftf2trc'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'plant.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
```

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'plant.wsgi.application'
import os
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}



# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]



# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True



# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

File: urls.py

Documented code for urls.py:

```
"""
URL configuration for plant project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('app.urls')),
]
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

File: wsgi.py

Documented code for wsgi.py:

```
"""
WSGI config for plant project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'plant.settings')

application = get_wsgi_application()
```

Folder: templates
File: result.html
Documented code for result.html:

```
<!-- result.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Recognition Result</title>
    <style>
        body {
            background-color: #f8f9fa;
            color: #343a40;
            height: 100vh;
            display: flex;
```

```
            align-items: center;
            justify-content: center;
            margin: 0;
        }
        .result-box {
            max-width: 400px;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            background-color: #fff;
            text-align: center;
        }
        img {
            width: 100%;
            max-height: 200px;
            object-fit: cover; /* Preserve aspect ratio while covering the box */
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <div class="result-box">
        <h2>Recognition Result:</h2>
        <img src="{{ uploaded_image }}" alt="Uploaded Image">
        <p>Class Prediction: {{ result.0 }}</p>
        <p>{{ result.1 }}</p>
    </div>
</body>
</html>
```

File: upload.html

Documented code for upload.html:

```
<!-- upload.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Upload</title>
    <!-- Bootstrap CSS CDN -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8sh+WytkY3Y5CcH25PLOiMZ95ES9B3xUZUn6N" crossorigin="anonymous">
    <style>
        body {
            background-color: #f8f9fa;
            color: #343a40;
            height: 100vh;
            display: flex;
            align-items: center;
            justify-content: center;
            margin: 0;
        }
        .container {
            max-width: 400px;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            background-color: #fff;
        }
        h2 {
            text-align: center;
```

```
            margin-bottom: 30px;
        }
        form {
            padding: 20px;
        }
        button {
            background-color: #007bff;
            color: #fff;
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Upload Image</h2>
        <form method="post" enctype="multipart/form-data">
            {% csrf_token %}
            {{ form }}
            <button type="submit" class="btn btn-primary btn-block">Upload Image</button>
        </form>
    </div>

    <!-- Bootstrap JS and Popper.js CDN (required for Bootstrap) -->
                                        <script            src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkf
j" crossorigin="anonymous"></script>
                <script    src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"
integrity="sha384-dQVlziZ8DEghGHiP99R8Tz9DJks7Cd7P3MZweelF+0GnyDwr6bVEwH+WrGzprY
" crossorigin="anonymous"></script>
                    <script      src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"
integrity="sha384-B4gt1jrGC7Jh4AgTPSdUtOBvfO8sh+WytkY3Y5CcH25PLOiMZ95ES9B3xUZUn6
N" crossorigin="anonymous"></script>
</body>
</html>
```