Fungi Edibility Predictive Image Classification Model - Final Notebook

PROGRAMMER: Alex Karadjov

-

Imports

In [18]:

```
import csv
import os
os.environ["PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION"] = "python"
import yaml
import joblib

import pandas as pd
import numpy as np
import pickle
import streamlit as st
import matplotlib.pyplot as plt
import tensorflow as tf
from keras import models
from PIL import Image


from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import img_to_array, load_
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import preprocess_input

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

-

Loading The Images into a Dataset

Because I only have a folder of all my images labeled by Genus of Mushroom, I will write a function to label my Images either Poisonous or Edible for my Dataset, this is also optimal for efficiency.

In [2]:

```
def label_images(image_folder):
    data = []

    for root, dirs, files in os.walk(image_folder):
        folder_name = os.path.basename(root)

        if folder_name.lower() == 'exidia':
            label = 'edible'

        elif folder_name.lower() == 'inocybe':
            label = 'poisonous'

        elif folder_name.lower() == 'agaricus':
            label = 'poisonous'

        elif folder_name.lower() == 'amanita':
            label = 'poisonous'

        elif folder_name.lower() == 'boletus':
            label = 'poisonous'

        elif folder_name.lower() == 'cortinarius':
            label = 'poisonous'

        elif folder_name.lower() == 'entoloma':
            label = 'poisonous'

        elif folder_name.lower() == 'hygrocybe':
            label = 'edible'

        elif folder_name.lower() == 'lactarius':
            label = 'poisonous'

        elif folder_name.lower() == 'russula':
            label = 'poisonous'

        elif folder_name.lower() == 'suillus':
            label = 'edible'

        else:
            continue

        for file in files:
            if file.endswith('.jpg'):
                image_path = os.path.join(root, file)
                data.append({'Image': image_path, 'Label': label, 'F

    df = pd.DataFrame(data)
    return df
```

-

Calling the Function and Loading the Dataset

Setting image folder file path and calling the label_images function to prep a dataset.

In [3]:

```
image_folder = 'archive/Mushrooms'
labeled_data = label_images(image_folder)
```

-

Viewing dataset.

Changing column names and cleaning values.

In [4]:

```
labeled_data.head()
```

Out[4]:

	Image	Label	Folder
0	archive/Mushrooms\Agaricus\000_ePQknW8cTp8.jpg	poisonous	Agaricus
1	archive/Mushrooms\Agaricus\001_2jP9N_ipAo8.jpg	poisonous	Agaricus
2	archive/Mushrooms\Agaricus\002_hNh3aQSH-ZM.jpg	poisonous	Agaricus
3	archive/Mushrooms\Agaricus\003_4AurAO4Jil8.jpg	poisonous	Agaricus
4	archive/Mushrooms\Agaricus\004_Syi3NxxviC0.jpg	poisonous	Agaricus

In [5]:

```
labeled_data = labeled_data.rename(columns={'Image': 'image_path', '
```

-

In [6]:

```
labeled_data.head()
```

Out[6]:

	image_path	label	genus_type
0	archive/Mushrooms\Agaricus\000_ePQknW8cTp8.jpg	poisonous	Agaricus
1	archive/Mushrooms\Agaricus\001_2jP9N_ipAo8.jpg	poisonous	Agaricus
2	archive/Mushrooms\Agaricus\002_hNh3aQSH-ZM.jpg	poisonous	Agaricus
3	archive/Mushrooms\Agaricus\003_4AurAO4Jil8.jpg	poisonous	Agaricus
4	archive/Mushrooms\Agaricus\004_Syi3NxxviC0.jpg	poisonous	Agaricus

In [7]:

```
labeled_data['genus_type'] = labeled_data['genus_type'].str.lower()
```

-

Checking class balances

In [8]:

```
labeled_data['genus_type'].value_counts()
```

Out[8]:

```
lactarius      1563
russula        1148
boletus        1073
cortinarius     836
inocybe         618
exidia          435
entoloma        364
agaricus        353
hygrocybe       316
suillus         311
amanita          46
Name: genus_type, dtype: int64
```

The classes are extremely unbalanced, I am going to apply a SMOTE resampler to balance the datasets classes.

In [9]:

```
smote = SMOTE(sampling_strategy = 0.5, k_neighbors = 3, random_state
```

-

Setting a final image size for all images when preprocessing.

In [10]:

```
image_width, image_height = 128, 128
```

-

Writing a function to load all the images as an array and reshape all images too the same size.

There are also a significantly small amount of truncated images in my dataset, so I have written the function to address the truncated images as a NONE type as removing the truncated images will not affect my data.

In [11]:

```
def preprocess_images(image_path):
    try:
        img = load_img(image_path, target_size=(image_width, image_he
        img = img_to_array(img)
        img = img / 255.0

        return img

    except OSError as e:
        print(f"Skipping truncated image: {image_path}")
        return None

```

-

Loading the final dataset

Loading the images and labels into empty dictionaries to then convert and apply SMOTE to an X and Y variable being images and labels.

In [12]:

```
def loading_dataset(labeled_data):
    images = []
    labels = []

    for _, row in labeled_data.iterrows():
        image_path = row['Image']
        label = row['Label']

        img = preprocess_images(image_path)
        if img is not None:
            images.append(img)
            labels.append(label)

    images = np.array(images)
    labels = np.array(labels)

    num_samples = images.shape[0]
    images_flattened = images.reshape(num_samples, -1)


    smote = SMOTE()
    images_resampled, labels_resampled = smote.fit_resample(images_f


    image_width, image_height = images.shape[1], images.shape[2]
    images_resampled = images_resampled.reshape(-1, image_width, imag

    return images_resampled, labels_resampled
```

-

Calling all functions to get X and Y

In [13]:

```
image_folder = 'archive/Mushrooms'
labeled_data = label_images(image_folder)
images, labels = loading_dataset(labeled_data)
```

```
Skipping truncated image: archive/Mushrooms\Russula\092_43B354vYxm8.jpg
```

In [14]:

```
images.shape
```

Out[14]:

```
(12000, 128, 128, 3)
```

In [15]:

```
labels.shape
```

Out[15]:

```
(12000,)
```

Saving Images and Labels Into Files to be Used in Seperate Notebooks

In [19]:

```
np.savetxt('images.csv', images.flatten(), delimiter=',')

label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
np.savetxt('labels.csv', labels_encoded, delimiter=',')
```

-

In [ ]:

```

```