OCULAR DISEASE RECOGNITION

Data Sciense Tools

Moahmmed Abdallah Abdelhafith	20221460214
Habiba Mohamed Bassouny	20221462320
Mostafa Abdelalieem Nasr	20221459995
Abu Bakr Mohamed Mahmoud	20221458962
Mostafa Mohamed Kamel	20221469440

INTRODUCTION

Ocular Disease Intelligent Recognition (ODIR) is a structured ophthalmic database of 5,000 patients with age, color fundus photographs from left and right eyes and doctors' diagnostic keywords from doctors.
This dataset is meant to represent ‘‘real-life’’ set of patient information.

IMPORT LIBRARIES

- Data Manipulation and Computations

numpy and pandas: These are fundamental packages for numerical computations and data manipulation. numpy provides support for arrays and matrices, while pandas is used for handling data structures like DataFrames.

- Visualization

matplotlib.pyplot: A module of the Matplotlib library used for creating static, interactive, and animated visualizations in Python.

- File Operations

os: Provides a way of using operating system dependent functionality like reading or writing to the file system.

- Image Processing

cv2 (OpenCV): A library of Python bindings designed to solve computer vision problems. It includes functionalities for image processing and manipulation.

- Progress Bar

tqdm: A fast, extensible progress bar for loops and other processes, which can help in visualizing the progress of long-running tasks.

- Random Number Generation

random: Provides tools for making random selections, a useful feature in many machine learning tasks, such as dividing data into training and testing sets.

IMPORT LIBRARIES

- Data Splitting and Categorization

sklearn.model_selection.train_test_split: A utility function to split datasets into training and testing subsets.
tensorflow.keras.utils.to_categorical: Converts a class vector (integers) to a binary class matrix, which is often required for classification tasks in machine learning.

- Reproducibility

Setting a random seed (42): Ensures that the script's output is deterministic, meaning it produces the same results each time it's run.

- Machine Learning Models and Layers

tensorflow.keras.applications.vgg19.VGG19: Imports the VGG19 model, a pre-trained deep learning model for image classification.
tensorflow, Sequential, Flatten, Dense: Components from TensorFlow and Keras for building and training neural network models. Sequential creates a linear stack of layers, while Flatten and Dense are types of layers used in neural networks.

- Image Data Preparation

ImageDataGenerator, load_img, img_to_array: These functions from Keras are used for image data augmentation (like resizing, normalization), loading images, and converting them into numpy arrays for model processing.

IMPORT LIBRARIES

- **Model Performance Assessment**

confusion_matrix, classification_report, accuracy_score: Functions from scikit-learn used to evaluate the performance of the machine learning model.

IMPORT DATA

1. Installing and Configuring Kaggle API

!pip install kaggle: This line installs the Kaggle API, a tool that allows you to interact with Kaggle (a platform for data science competitions and datasets) directly from your script.

!mkdir -p ~/.kaggle: Creates a directory named .kaggle in the home directory of the current user. The -p flag ensures that no error is thrown if the directory already exists.

!mv kaggle.json ~/.kaggle/: Moves the kaggle.json file into the .kaggle directory. The kaggle.json file typically contains your Kaggle API credentials.

!chmod 600 ~/.kaggle/kaggle.json: Changes the file permission of kaggle.json to 600. This means that the file is only readable and writable by the file's owner, which is a security measure to protect your credentials.

IMPORT DATA

2. Downloading the Dataset

!mkdir -p ocular-disease-recognition: Creates a directory named ocular-disease-recognition to store the downloaded dataset.

!kaggle datasets download -d andrewmvd/ocular-disease-recognition-odir5k -p ocular-disease-recognition: Uses the Kaggle API to download the dataset ocular-disease-recognition-odir5k by the user andrewmvd into the previously created ocular-disease-recognition directory.

3. Extracting the Dataset

!unzip -q ocular-disease-recognition/ocular-disease-recognition-odir5k.zip: Extracts the contents of the downloaded ZIP file (ocular-disease-recognition-odir5k.zip) quietly (without logging the extraction details) into the ocular-disease-recognition directory.

IMPORT DATA

4. Loading and Previewing the Data

data = pd.read_csv("full_df.csv"): Reads the CSV file full_df.csv into a pandas DataFrame. This file is assumed to be a part of the extracted dataset and contains the data necessary for further analysis or machine learning tasks.

data.head(3): Displays the first three rows of the DataFrame data. This is typically done to get a quick glimpse of the dataset structure and contents.

	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	labels	target	filename
0	0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	0	0	0	1	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	0_right.jpg
1	1	57	Male	1_left.jpg	1_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	['N']	[1, 0, 0, 0, 0, 0, 0, 0]	1_right.jpg
2	2	42	Male	2_left.jpg	2_right.jpg	laser spot，moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	1	../input/ocular-disease-recognition-odir5k/ODI...	['D']	[0, 1, 0, 0, 0, 0, 0, 0]	2_right.jpg

EXPLORATION

missing_values = data.isnull().sum():

data.isnull(): This function creates a new DataFrame where each cell is either True or False depending on whether the corresponding cell in the original data DataFrame is NaN (not a number, which is pandas' way of indicating missing or null values) or not.

.sum(): This function is then called on the resulting DataFrame. Since True is treated as 1 and False as 0, summing over a column effectively counts how many True (or missing) values are in that column. The result is a pandas Series where each element corresponds to a column in data and the value is the count of missing values in that column.

EXPLORATION

```python
# Checking for missing data

missing_values = data.isnull().sum()

print("Missing values in the entire DataFrame:")
print(missing_values)
```

```
Missing values in the entire DataFrame:
ID                          0
Patient Age                 0
Patient Sex                 0
Left-Fundus                 0
Right-Fundus                0
Left-Diagnostic Keywords    0
Right-Diagnostic Keywords   0
N                           0
D                           0
G                           0
C                           0
A                           0
H                           0
M                           0
O                           0
filepath                    0
labels                      0
target                      0
filename                    0
dtype: int64
```

EXPLORATION

data.duplicated(): This function returns a boolean Series indicating whether each row is a duplicate (has been observed in a previous row) or not. Rows are considered duplicates if all their elements match those of a previous row.

```
# Check for duplicates

any_duplicates = data.duplicated().any()
any_duplicates
```

```
False
```

EXPLORATION

data.duplicated(): This function returns a boolean Series indicating whether each row is a duplicate (has been observed in a previous row) or not. Rows are considered duplicates if all their elements match those of a previous row.

```
# Check for duplicates

any_duplicates = data.duplicated().any()
any_duplicates
```

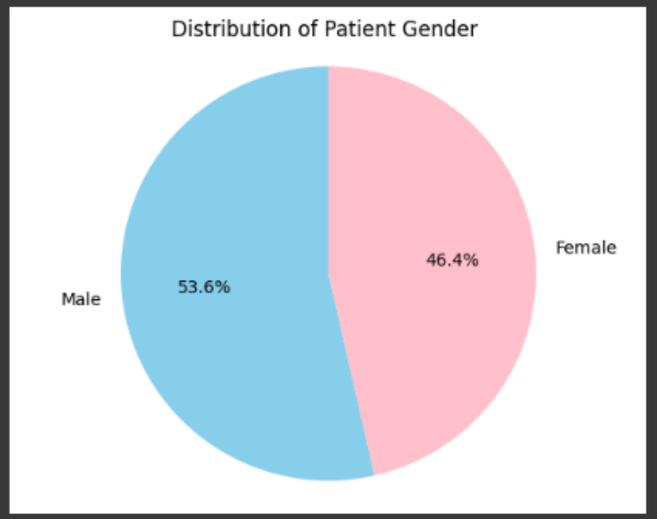
```
False
```

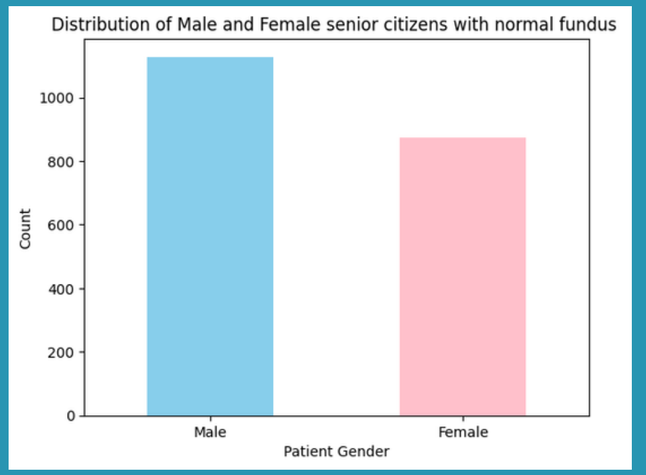
EXPLORATION

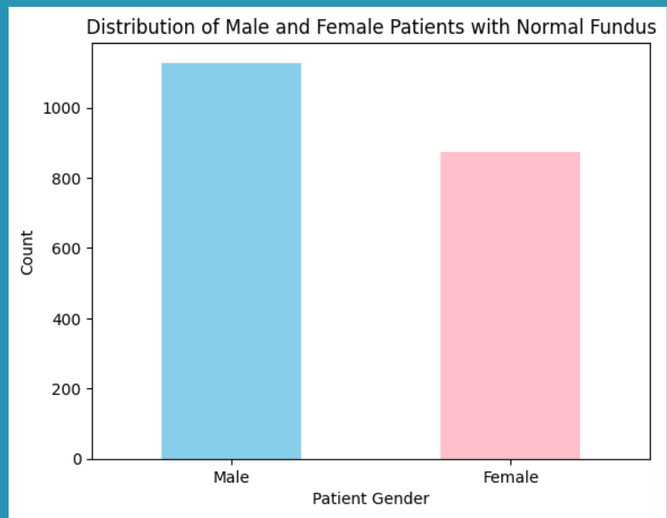
gender_counts = data['Patient Sex'].value_counts():

- **This line calculates the frequency of each unique value in the 'Patient Sex' column of the DataFrame data.**
- **value_counts() returns a Series containing counts of unique values, sorted in descending order by default.**

```
# Distribution of patient gender

gender_counts = data['Patient Sex'].value_counts()
colors=['skyblue','pink']
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=90,colors=colors)

plt.title('Distribution of Patient Gender ')

plt.axis('equal')
plt.show()
```







PREPARATION

HELPER FUNCTION: HAS_CONDN(TERM, TEXT)

THIS FUNCTION CHECKS IF A SPECIFIC TERM (A DISEASE OR CONDITION) IS PRESENT IN A TEXT (PRESUMABLY A STRING CONTAINING DIAGNOSTIC KEYWORDS). IT RETURNS 1 IF THE TERM IS PRESENT, INDICATING THE PRESENCE OF THE CONDITION, AND 0 OTHERWISE.

PREPARATION

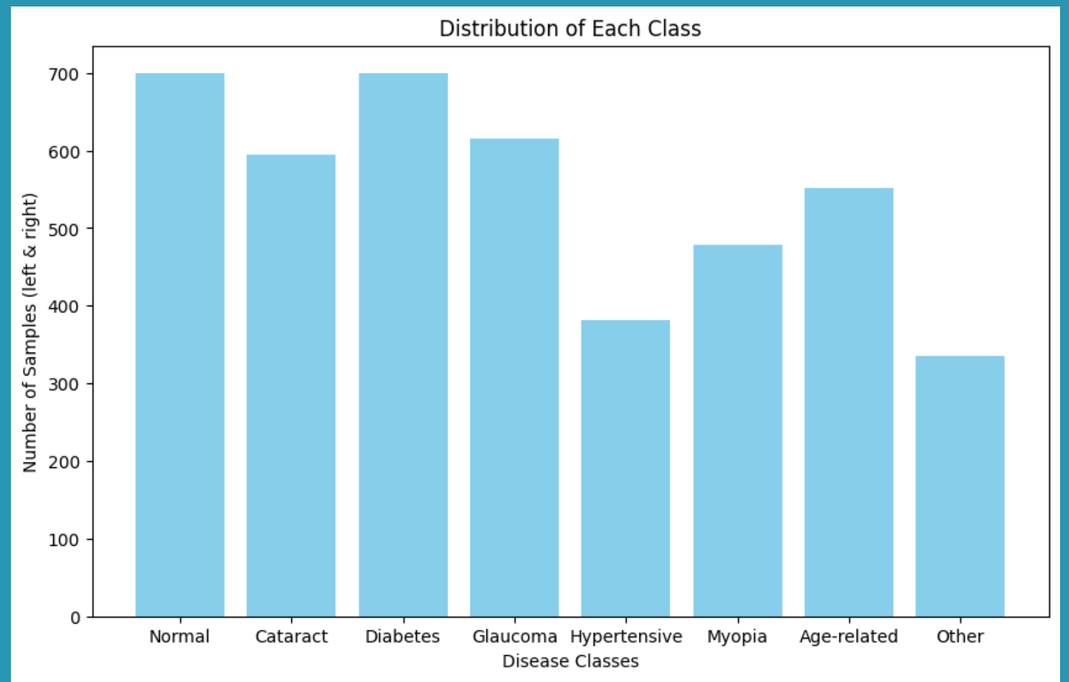
MAIN FUNCTION: PROCESS_DATASET(DATA)

THIS FUNCTION PERFORMS SEVERAL KEY STEPS IN PROCESSING THE OCULAR DISEASE DATASET:

1. CREATING INDICATORS FOR DISEASES:
   - FOR EACH DISEASE (LIKE CATARACT, GLAUCOMA, MYOPIA, ETC.), TWO NEW COLUMNS ARE ADDED TO THE DATA DATAFRAME.
   - THESE COLUMNS INDICATE WHETHER THE DISEASE IS MENTIONED IN THE DIAGNOSTIC KEYWORDS FOR THE LEFT AND RIGHT EYES (LEFT-DIAGNOSTIC KEYWORDS AND RIGHT-DIAGNOSTIC KEYWORDS COLUMNS, RESPECTIVELY).
   - THE APPLY METHOD IS USED WITH THE HAS_CONDN FUNCTION TO POPULATE THESE COLUMNS.
2. SAMPLING IMAGE IDS:
   - THE FUNCTION IDENTIFIES IMAGE FILE NAMES (OR IDS) FOR VARIOUS CONDITIONS FROM BOTH THE LEFT AND RIGHT EYES.
   - FOR EACH CONDITION, IT SELECTS IMAGES FROM PATIENTS DIAGNOSED WITH THAT CONDITION, BASED ON THE NEWLY CREATED INDICATORS.
   - IN SOME CASES, A RANDOM SAMPLE OF IMAGES IS TAKEN (E.G., FOR NORMAL, DIABETIC RETINOPATHY CASES) TO POSSIBLY BALANCE THE DATASET.

PREPARATION

3. COMBINING LEFT AND RIGHT EYE IMAGES:
   - ARRAYS OF IMAGE IDS FOR THE LEFT AND RIGHT EYES FOR EACH CONDITION ARE CONCATENATED. THIS RESULTS IN A COMBINED LIST OF IMAGE IDS FOR EACH CONDITION.
   - THE CONCATENATED ARRAYS FOR EACH CONDITION (NORMAL, CATARACT, DIABETIC RETINOPATHY, ETC.) REPRESENT A COLLECTION OF IMAGE IDS THAT CAN BE USED LATER TO FETCH IMAGES FROM A FOLDER OR DATABASE.
4. RETURN STATEMENT:
   - THE FUNCTION RETURNS THESE ARRAYS OF IMAGE IDS FOR EACH OCULAR CONDITION, WHICH CAN BE USED IN SUBSEQUENT STEPS OF THE PROJECT, SUCH AS LOADING IMAGES FOR MODEL TRAINING OR ANALYSIS.




```
Dataset stats:-

Normal :: 700
Cataract :: 594
Diabetes :: 700
Glaucoma :: 616
Hypertension :: 382
Myopia :: 479
Age Issues :: 551
Other :: 336
```

PREPARATION

THE DATASET_GENERATOR FUNCTION
PURPOSE: TO PROCESS AND APPEND IMAGES TO THE DATASET LIST ALONG WITH THEIR CORRESPONDING LABELS.

PARAMETERS:

IMAGECATEGORY: A LIST OF IMAGE FILE NAMES TO BE PROCESSED.
LABEL: THE LABEL ASSOCIATED WITH THESE IMAGES (INDICATIVE OF THE OCULAR CONDITION THEY REPRESENT).

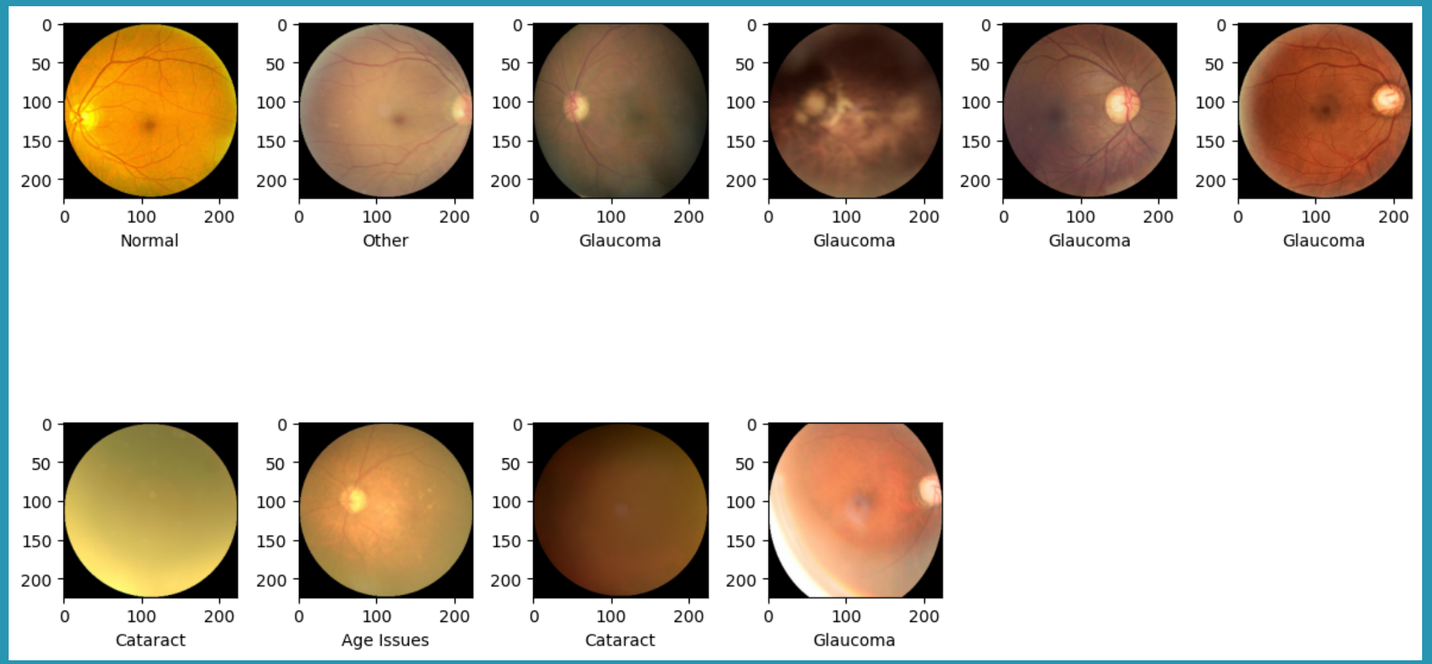
PROCESS INSIDE FUNCTION:

- ITERATING OVER EACH IMAGE NAME IN IMAGECATEGORY USING A FOR LOOP.
- CONSTRUCTING THE PATH TO THE IMAGE FILE (IMGPATH).
- USING OPENCV (CV2) TO READ, RESIZE, AND CONVERT THE IMAGE TO RGB COLOR SPACE.
- IN CASE OF AN ERROR (E.G., IMAGE FILE NOT FOUND), THE TRY-EXCEPT BLOCK SKIPS THE CURRENT IMAGE AND CONTINUES WITH THE NEXT.
- APPENDING THE PROCESSED IMAGE AND ITS LABEL AS A LIST TO THE DATASET.
- RANDOMLY SHUFFLING DATASET AFTER PROCESSING ALL IMAGES IN IMAGECATEGORY.
- RETURNING THE UPDATED DATASET.

PREPARATION

GENERATING THE DATASET

THE DATASET_GENERATOR FUNCTION IS CALLED FOR EACH CLASS OF OCULAR DISEASE (NORMAL, CATARACT, DIAB, ETC.). THE IMAGE NAMES FOR EACH CLASS AND THEIR CORRESPONDING LABELS (0 TO 7) ARE PASSED AS ARGUMENTS. THE DATASET GETS PROGRESSIVELY UPDATED WITH IMAGES FROM EACH CLASS, ALONG WITH THEIR LABELS.




```
100%|██████████| 700/700 [00:03<00:00, 221.76it/s]
100%|██████████| 594/594 [00:02<00:00, 263.26it/s]
100%|██████████| 700/700 [00:04<00:00, 164.38it/s]
100%|██████████| 616/616 [00:06<00:00, 95.17it/s]
100%|██████████| 382/382 [00:03<00:00, 107.97it/s]
100%|██████████| 479/479 [00:04<00:00, 101.51it/s]
100%|██████████| 551/551 [00:05<00:00, 97.34it/s]
100%|██████████| 336/336 [00:01<00:00, 276.08it/s]
Number of images we have for all classes: 4324
```

CNN MODEL - FIRST MODEL

LOADING THE PRE-TRAINED VGG19 MODEL

VGG = VGG19(WEIGHTS="IMAGENET", INCLUDE_TOP=FALSE, INPUT_SHAPE=(IMAGE_SIZE, IMAGE_SIZE, 3)):
THE VGG19 MODEL, KNOWN FOR ITS EFFECTIVENESS IN IMAGE CLASSIFICATION TASKS, IS LOADED WITH WEIGHTS PRE-TRAINED ON THE IMAGENET DATASET.
INCLUDE_TOP=FALSE INDICATES THAT THE TOP (FULLY CONNECTED) LAYERS OF VGG19 ARE NOT INCLUDED. THIS IS TYPICAL IN TRANSFER LEARNING SCENARIOS, WHERE THE FULLY CONNECTED LAYERS ARE SPECIFIC TO THE ORIGINAL CLASSIFICATION TASK (IMAGENET IN THIS CASE).
INPUT_SHAPE=(IMAGE_SIZE, IMAGE_SIZE, 3) SETS THE INPUT SIZE OF THE IMAGES. HERE, IMAGE_SIZE IS DEFINED EARLIER IN YOUR CODE AND 3 REFERS TO THE NUMBER OF COLOR CHANNELS (RGB).

CNN MODEL - FIRST MODEL

2. FREEZING THE LAYERS OF VGG19

FOR LAYER IN VGG.LAYERS: LAYER.TRAINABLE = FALSE:
THIS LOOP ITERATES THROUGH EACH LAYER IN THE VGG19 MODEL AND SETS ITS TRAINABLE ATTRIBUTE TO FALSE.
FREEZING THE LAYERS MEANS THEIR WEIGHTS WILL NOT BE UPDATED DURING TRAINING. THIS IS A COMMON PRACTICE IN TRANSFER LEARNING, ALLOWING THE MODEL TO RETAIN THE KNOWLEDGE IT HAS LEARNED FROM IMAGENET.

CNN MODEL - FIRST MODEL

3. CREATING A SEQUENTIAL MODEL

MODEL = SEQUENTIAL([...]):
A NEW SEQUENTIAL MODEL IS CREATED, WHICH MEANS THAT THE LAYERS ARE ARRANGED IN A LINEAR STACK.
THE VGG19 BASE MODEL IS ADDED FIRST, FOLLOWED BY ADDITIONAL CUSTOM LAYERS:
FLATTEN(): THIS LAYER FLATTENS THE OUTPUT OF VGG19 TO A ONE-DIMENSIONAL ARRAY, MAKING IT SUITABLE FOR INPUT INTO THE FULLY CONNECTED LAYERS.
DENSE(256, ACTIVATION="RELU"): THESE ARE FULLY CONNECTED LAYERS WITH 256 NEURONS EACH, USING RELU (RECTIFIED LINEAR UNIT) ACTIVATION.
TF.KERAS.LAYERS.BATCHNORMALIZATION(): BATCH NORMALIZATION LAYERS ARE ADDED TO NORMALIZE THE INPUTS OF EACH LAYER. THEY CAN IMPROVE TRAINING EFFICIENCY AND REDUCE THE CHANCE OF OVERFITTING.
DENSE(8, ACTIVATION="SOFTMAX"): THE FINAL LAYER HAS 8 NEURONS (PRESUMABLY FOR 8 DIFFERENT CLASSES IN YOUR TASK) AND USES SOFTMAX ACTIVATION, WHICH IS COMMON IN MULTI-CLASS CLASSIFICATION PROBLEMS.

CNN MODEL - FIRST MODEL

MODEL.COMPILE(OPTIMIZER="ADAM", LOSS="CATEGORICAL_CROSSENTROPY", METRICS=["ACCURACY"]):

OPTIMIZER="ADAM": THIS SPECIFIES THE OPTIMIZER USED FOR TRAINING. ADAM IS A POPULAR CHOICE DUE TO ITS EFFECTIVE HANDLING OF SPARSE GRADIENTS ON NOISY PROBLEMS.

LOSS="CATEGORICAL_CROSSENTROPY": SINCE THIS IS A MULTI-CLASS CLASSIFICATION PROBLEM, CATEGORICAL CROSSENTROPY IS USED AS THE LOSS FUNCTION. IT MEASURES THE PERFORMANCE OF THE MODEL WHOSE OUTPUT IS A PROBABILITY VALUE BETWEEN 0 AND 1.

METRICS=["ACCURACY"]: THE MODEL WILL TRACK ACCURACY DURING TRAINING. ACCURACY IS A COMMON METRIC FOR CLASSIFICATION TASKS AND INDICATES THE PROPORTION OF CORRECTLY CLASSIFIED INSTANCES.

CNN MODEL - FIRST MODEL

```
Epoch 1/15
109/109 [==============================] - 28s 154ms/step - loss: 1.4618 - accuracy: 0.4929
Epoch 2/15
109/109 [==============================] - 16s 144ms/step - loss: 0.7712 - accuracy: 0.7317
Epoch 3/15
109/109 [==============================] - 16s 147ms/step - loss: 0.5421 - accuracy: 0.8303
Epoch 4/15
109/109 [==============================] - 17s 152ms/step - loss: 0.3929 - accuracy: 0.8907
Epoch 5/15
109/109 [==============================] - 17s 158ms/step - loss: 0.3403 - accuracy: 0.8994
Epoch 6/15
109/109 [==============================] - 17s 160ms/step - loss: 0.2516 - accuracy: 0.9344
Epoch 7/15
109/109 [==============================] - 18s 162ms/step - loss: 0.2314 - accuracy: 0.9300
Epoch 8/15
109/109 [==============================] - 17s 160ms/step - loss: 0.2262 - accuracy: 0.9352
Epoch 9/15
109/109 [==============================] - 17s 156ms/step - loss: 0.2371 - accuracy: 0.9324
Epoch 10/15
109/109 [==============================] - 17s 158ms/step - loss: 0.2166 - accuracy: 0.9332
Epoch 11/15
109/109 [==============================] - 18s 161ms/step - loss: 0.1966 - accuracy: 0.9384
Epoch 12/15
109/109 [==============================] - 18s 161ms/step - loss: 0.1610 - accuracy: 0.9474
Epoch 13/15
109/109 [==============================] - 18s 161ms/step - loss: 0.1553 - accuracy: 0.9480
Epoch 14/15
109/109 [==============================] - 17s 159ms/step - loss: 0.1701 - accuracy: 0.9448
Epoch 15/15
109/109 [==============================] - 17s 160ms/step - loss: 0.1645 - accuracy: 0.9451
<keras.src.callbacks.History object at 0x7effd816f160>
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 7, 7, 512)         20024384

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 batch_normalization (Batch  (None, 256)               1024
 Normalization)

 dense_1 (Dense)             (None, 256)               65792

 batch_normalization_1 (Bat  (None, 256)               1024
 chNormalization)

 dense_2 (Dense)             (None, 8)                 2056

=================================================================
Total params: 26517064 (101.15 MB)
Trainable params: 6491656 (24.76 MB)
Non-trainable params: 20025408 (76.39 MB)
_________________________________________________________________
```

```
28/28 [==============================] - 6s 186ms/step - loss: 1.1176 - accuracy: 0.7676
Accuracy: 0.7676300406455994
```

ARE THERE ANY QUESTIONS?