In [ ]:
```
import os
import glob
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split      # dividing the dat
from sklearn.preprocessing import LabelEncoder            # for converting s
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from PIL import Image
from collections import defaultdict
import matplotlib.pyplot as plt
import seaborn as sn
import random


os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # 0 = all messages, 1 = filter INF
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping , ModelCheckpoint ,Call
from tensorflow.keras import layers, models ,optimizers # type: ignore
from tensorflow.keras.models import load_model # type: ignore
from tensorflow.keras.layers import GlobalAveragePooling2D # type: ignore
from tensorflow.keras.applications import EfficientNetB0 # type: ignore
import pathlib

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
! pip install kaggle
```

Loading [MathJax]/extensions/Safe.js

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-pack
ages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-p
ackages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.
10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/
dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pa
ckages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packag
es (from kaggle) (4.66.6)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/d
ist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-pac
kages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-pack
ages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dis
t-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python
3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests->kaggle) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests->kaggle) (3.10)
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
! mkdir ~/.kaggle
```

In [ ]:
```
!cp /content/drive/MyDrive/kaggle/kaggle.json ~/.kaggle/kaggle.json
```

In [ ]:
```
#Change the file permissions to read/write to the owner only
! chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:
```
! kaggle datasets download gunavenkatdoddi/eye-diseases-classification
```

```
Dataset URL: https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-cl
assification
License(s): ODbL-1.0
Downloading eye-diseases-classification.zip to /content
100% 735M/736M [00:35<00:00, 23.9MB/s]
100% 736M/736M [00:35<00:00, 22.0MB/s]
```

In [ ]:
```
! unzip eye-diseases-classification.zip
```

Loading [MathJax]/extensions/Safe.js

```
 inflating: dataset/normal/3446_right.jpg
 inflating: dataset/normal/3448_right.jpg
 inflating: dataset/normal/3485_left.jpg
 inflating: dataset/normal/4290_right.jpg
 inflating: dataset/normal/4637_left.jpg
 inflating: dataset/normal/4637_right.jpg
 inflating: dataset/normal/530_left.jpg
 inflating: dataset/normal/530_right.jpg
 inflating: dataset/normal/695_left.jpg
 inflating: dataset/normal/695_right.jpg
 inflating: dataset/normal/84_left.jpg
 inflating: dataset/normal/84_right.jpg
 inflating: dataset/normal/8_left.jpg
 inflating: dataset/normal/8_right.jpg
 inflating: dataset/normal/939_left.jpg
 inflating: dataset/normal/939_right.jpg
 inflating: dataset/normal/951_left.jpg
 inflating: dataset/normal/951_right.jpg
```

To downloaad specific files, instead of the netire data set

`! kaggle datasets download gunavenkatdoddi/eye-diseases-classification -f cataract/0_left.jpg`

In [ ]:

```
# Basic review of the photos directory

def dataset_analysis(path):
    subfolders = os.listdir(path)

    for subfolder in subfolders:
        subfolder_path = os.path.join(path, subfolder)
        if os.path.isdir(subfolder_path):
            files = os.listdir(subfolder_path)
            format_dimensions_counts = defaultdict(lambda: defaultdict(lambd

            for file in files:
                try:
                    file_path = os.path.join(subfolder_path, file)
                    with Image.open(file_path) as img:
                        image_type = img.format.upper()  # Format (e.g., JPE
                        image_dimensions = img.size  # (width, height)
                        image_mode = img.mode  # Mode (e.g., RGB, L)


                        # Calculate bit depth
                        if image_mode == "1":  # 1-bit pixels, black and whi
                            bit_depth = 1
                        elif image_mode == "L":  # 8-bit pixels, grayscale
                            bit_depth = 8
                        elif image_mode == "P":  # 8-bit pixels, mapped to a
                            bit_depth = 8
                        elif image_mode == "RGB":  # 8-bit pixels, true colo
                            bit_depth = 24  # 8 bits per channel
                        elif image_mode == "RGBA":  # 8-bit pixels, true col
                            bit_depth = 32  # 8 bits per channel
```

Loading [MathJax]/extensions/Safe.js

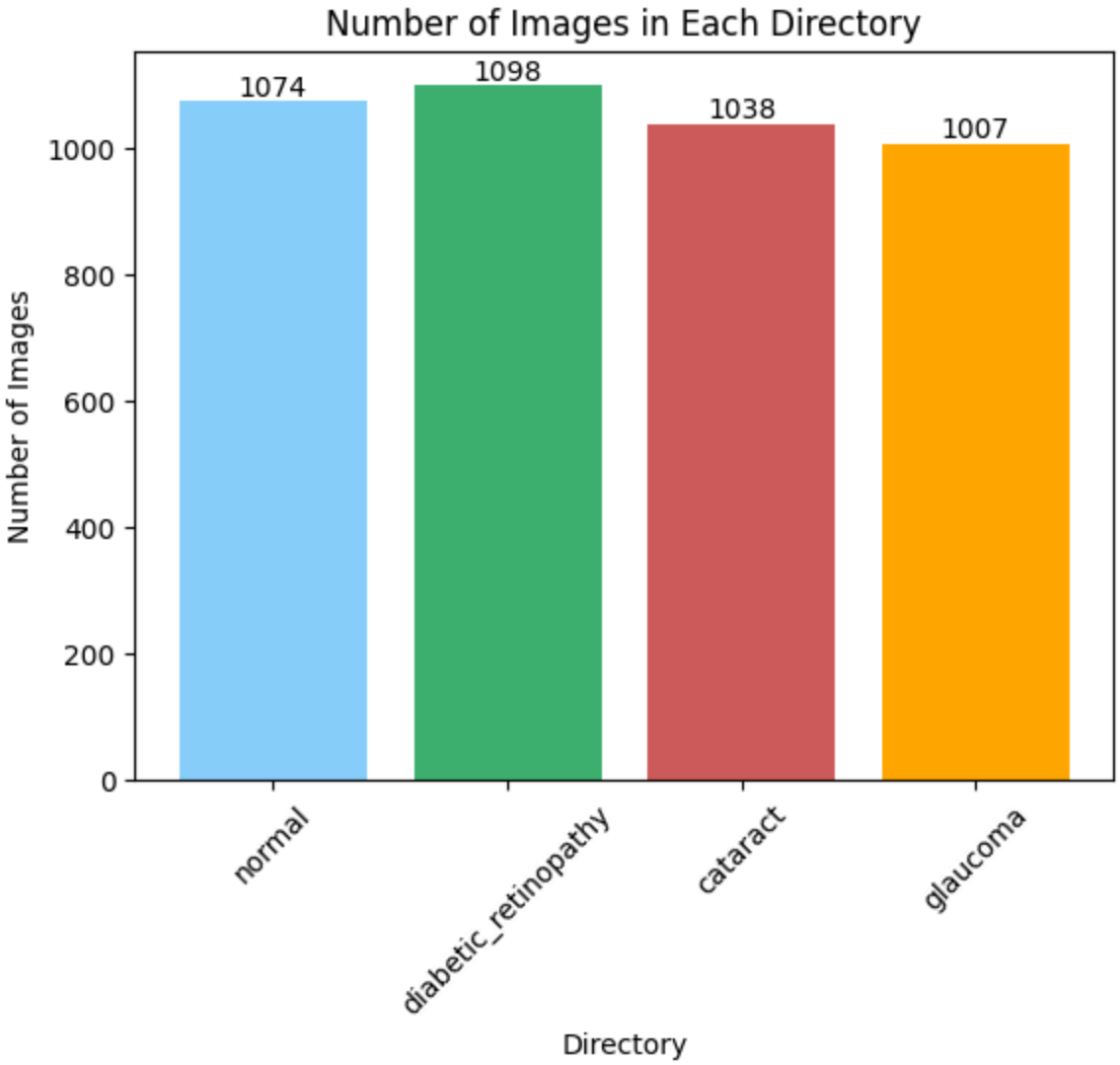
```
                        elif image_mode == "CMYK":  # 8-bit pixels, color se
                            bit_depth = 32  # 8 bits per channel
                        else:
                            bit_depth = "Unknown"

                        format_dimensions_counts[image_type][(image_dimensic
                    except Exception as e:
                        print(f"Exception processing '{file}' in '{subfolder}':

            print('--------'*10)
            print(f"Subfolder '{subfolder}' contains ({len(files)} files):")
            for format, dimensions_counts in format_dimensions_counts.items(
                print(f"- {sum(sum(counts.values()) for counts in dimensions
                for (dimensions, bit_depth), counts in dimensions_counts.ite
                    for mode, count in counts.items():
                        print(f"  - {count} images with dimensions {dimensic



path = r'/content/dataset'
dataset_path = path
dataset_analysis(dataset_path)
```

```
--------------------------------------------------------------------------------
----
Subfolder 'normal' contains (1074 files):
- 1074 images of format JPEG:
  - 1074 images with dimensions (512, 512), bit depth 24, mode RGB
--------------------------------------------------------------------------------
----
Subfolder 'diabetic_retinopathy' contains (1098 files):
- 1098 images of format JPEG:
  - 1098 images with dimensions (512, 512), bit depth 24, mode RGB
--------------------------------------------------------------------------------
----
Subfolder 'cataract' contains (1038 files):
- 938 images of format JPEG:
  - 640 images with dimensions (256, 256), bit depth 24, mode RGB
  - 298 images with dimensions (512, 512), bit depth 24, mode RGB
- 100 images of format PNG:
  - 76 images with dimensions (2592, 1728), bit depth 24, mode RGB
  - 19 images with dimensions (2464, 1632), bit depth 24, mode RGB
  - 5 images with dimensions (1848, 1224), bit depth 24, mode RGB
--------------------------------------------------------------------------------
----
Subfolder 'glaucoma' contains (1007 files):
- 906 images of format JPEG:
  - 600 images with dimensions (256, 256), bit depth 24, mode RGB
  - 306 images with dimensions (512, 512), bit depth 24, mode RGB
- 101 images of format PNG:
  - 22 images with dimensions (2464, 1632), bit depth 24, mode RGB
  - 5 images with dimensions (1848, 1224), bit depth 24, mode RGB
  - 74 images with dimensions (2592, 1728), bit depth 24, mode RGB
```

In [ ]:
```
# Count the number of images in each directory
subfolders = os.listdir(path)
```

Loading [MathJax]/extensions/Safe.js

```
image_counts = []
for directory in subfolders:
    sub_dir = os.path.join(path, directory)
    if os.path.isdir(sub_dir):
        file_count = len(os.listdir(sub_dir))
        image_counts.append(file_count)

#Add value counts on each bar
for i in range(len(subfolders)):
    plt.text(i, image_counts[i], str(image_counts[i]), ha='center', va='bott

#Set some colors
colors = ['lightskyblue', 'mediumseagreen', 'indianred', 'orange']

# Plotting the results
plt.bar(subfolders, image_counts, color=colors)
plt.xlabel('Directory')
plt.xticks(rotation = 45)
plt.ylabel('Number of Images')
plt.title('Number of Images in Each Directory')
plt.show()
```

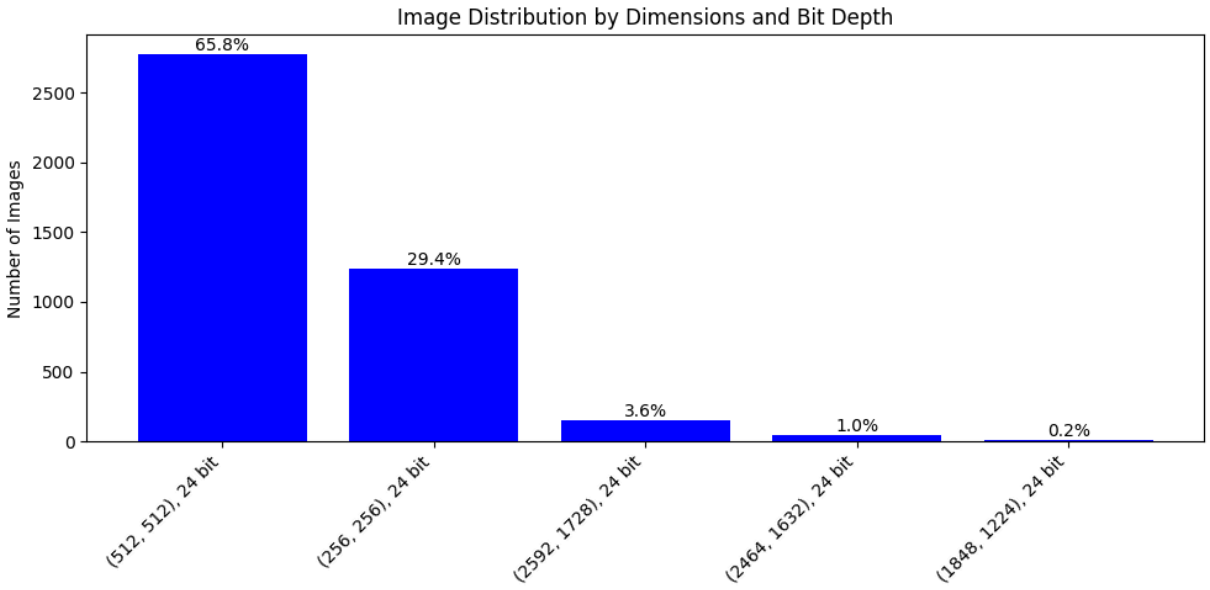


Loading [MathJax]/extensions/Safe.js

In [ ]:

```python
# Check the photos by size

def dataset_size_analysis(path):
    format_dimensions_counts = defaultdict(int)

    subfolders = os.listdir(path)
    for subfolder in subfolders:
        subfolder_path = os.path.join(path, subfolder)
        if os.path.isdir(subfolder_path):
            files = os.listdir(subfolder_path)

            for file in files:
                try:
                    file_path = os.path.join(subfolder_path, file)
                    with Image.open(file_path) as img:
                        image_dimensions = img.size
                        image_mode = img.mode

                        # Calculate bit depth
                        bit_depth = {
                            "1": 1,
                            "L": 8,
                            "P": 8,
                            "RGB": 24,
                            "RGBA": 32,
                            "CMYK": 32
                        }.get(image_mode, "Unknown")

                        # Update counts
                        format_dimensions_counts[(image_dimensions, bit_dept

                except Exception as e:
                    print(f"Exception processing '{file}' in '{subfolder}':

    # Plotting dimensions and bit depths
    plt.figure(figsize=(10, 5))
    labels = [f"{dims}, {depth} bit" for (dims, depth) in format_dimensions_
    sizes = list(format_dimensions_counts.values())
    total = sum(sizes)
    bars = plt.bar(labels, sizes, color='blue')
    plt.xticks(rotation=45, ha="right")
    plt.ylabel('Number of Images')
    plt.title('Image Distribution by Dimensions and Bit Depth')

    # Adding percentage labels above the bars
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, yval, f'{100 * yval/total:

    plt.tight_layout()
    plt.show()

# Set the path to the dataset directory
dataset_size_analysis(path)
```

Loading [MathJax]/extensions/Safe.js

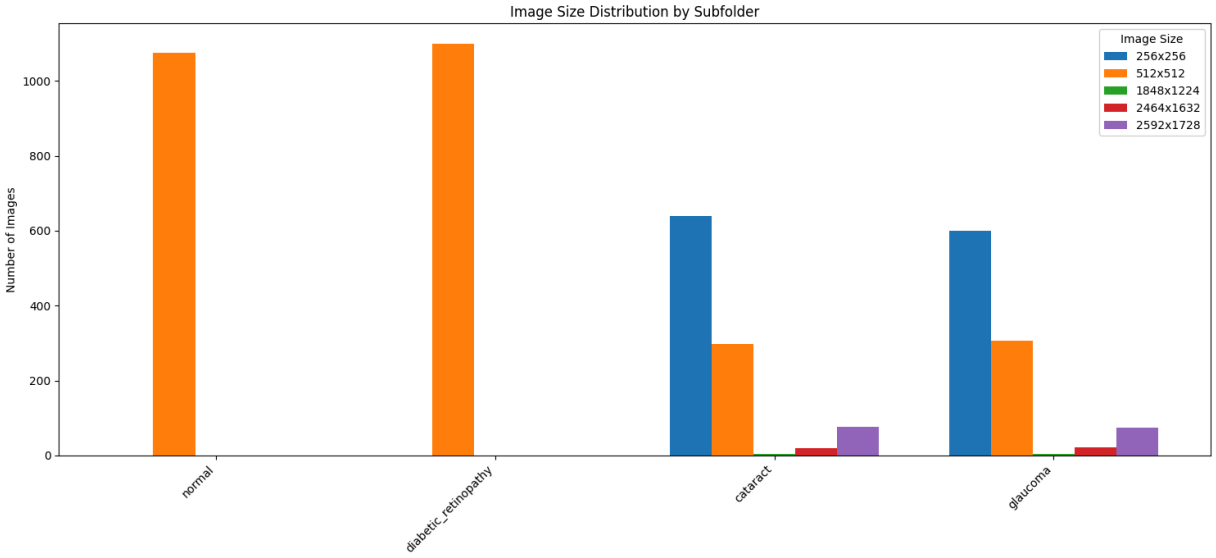


In [ ]:

```python
def dataset_size_analysis(path):
    # Dictionary to store counts: {subfolder: {image_size: count}}
    folder_size_counts = defaultdict(lambda: defaultdict(int))

    for subfolder in os.listdir(path):
        subfolder_path = os.path.join(path, subfolder)
        if os.path.isdir(subfolder_path):
            for file in os.listdir(subfolder_path):
                try:
                    file_path = os.path.join(subfolder_path, file)
                    with Image.open(file_path) as img:
                        dims = img.size
                        folder_size_counts[subfolder][dims] += 1
                except Exception as e:
                    print(f"Exception processing '{file}' in '{subfolder}':

    # Create a single plot
    plt.figure(figsize=(15, 7))

    # Determine unique image sizes across all folders for consistent coloring
    all_sizes = set(size for sizes in folder_size_counts.values() for size i
    all_sizes = sorted(all_sizes, key=lambda s: (s[0] * s[1]))  # Sort by ar

    subfolder_names = list(folder_size_counts.keys())
    bar_width = 0.15  # Width of bars
    indices = range(len(subfolder_names))

    for i, size in enumerate(all_sizes):
        counts = [folder_size_counts[subfolder].get(size, 0) for subfolder i
        plt.bar([index + i * bar_width for index in indices], counts, bar_wi

    plt.xticks([index + (len(all_sizes) - 1) * bar_width / 2 for index in ir
    plt.ylabel('Number of Images')
    plt.title('Image Size Distribution by Subfolder')
    plt.legend(title="Image Size")
    plt.tight_layout()
    plt.show()
```

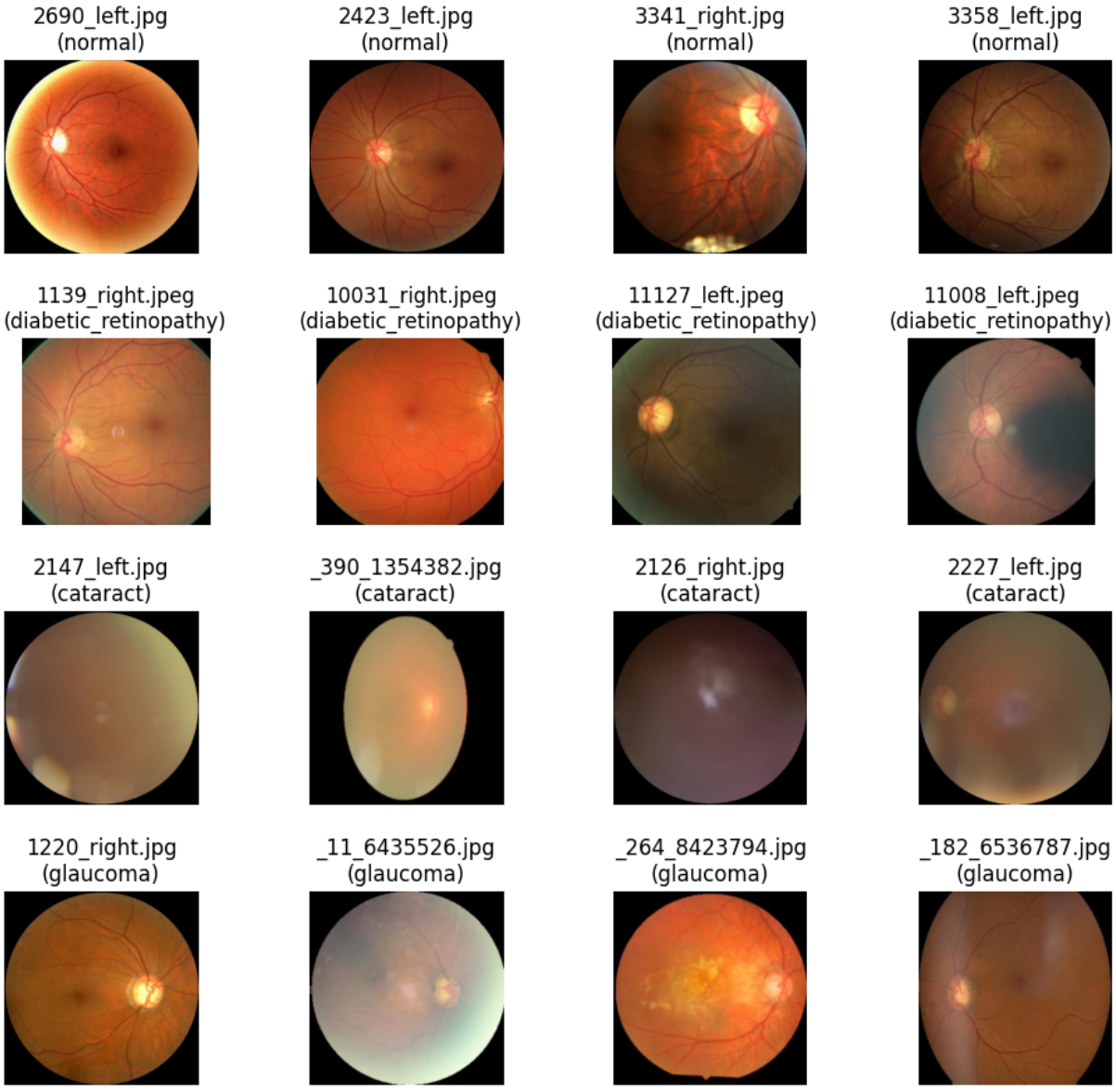
Loading [MathJax]/extensions/Safe.js

```
# Set the path to the dataset directory
dataset_size_analysis(path)
```



In [ ]:
```
# Preview photos

def random_photos_from_folders(base_folder):
    # Walk through all directories and files in the base_folder
    for root, dirs, files in os.walk(base_folder):
        # Filter to get only files that are images
        images = [file for file in files if file.lower().endswith(('.png', '

        if len(images) >= 4:  # Ensure there are at least 4 images
            selected_images = random.sample(images, 4)  # Randomly select 4

            # Display selected images
            fig, axs = plt.subplots(1, 4, figsize=(12, 2))  # Create a 1x4 g
            for idx, img_name in enumerate(selected_images):
                img_path = os.path.join(root, img_name)
                img = Image.open(img_path)
                axs[idx].imshow(img)
                axs[idx].axis('off')  # Hide axes

                # Extract sub-folder name from the root path
                subfolder_name = os.path.basename(root)
                # Set the title to include image name and sub-folder name
                axs[idx].set_title(f"{img_name}\n({subfolder_name})")

            plt.show()

# Path to the folder containing sub-folders with images

random_photos_from_folders(path)
```

Loading [MathJax]/extensions/Safe.js



In [ ]:
```
# Getting the names of classes
class_dirs = [d for d in os.listdir(path) if os.path.isdir(os.path.join(path

# create data path and their labeles
data = []
labels = []
extensions = ["jpg", "JPG", "jpeg", "JPEG", "png", "PNG", "bmp", "BMP", "gif

for i in class_dirs:
    class_label = i
    image_files = []
    for ext in extensions:
        # Search for files with each extension and extend the image_files li
        image_files.extend(glob.glob(os.path.join(path, i, f"*.{ext}")))
    data.extend(image_files)
    labels.extend([class_label] * len(image_files))

# Check if lists are still empty
if not data:
    raise ValueError("No files were found. Check your directory paths and file formats.
```

Loading [MathJax]/extensions/Safe.js

```
else:
    print("Files found and listed.")


# Create a DataFrame with the image paths and labels
df = pd.DataFrame({
'filename': data,
'class': labels
})


# Shuffle the dataset by rows
df = df.sample(frac=1)
```

```
Files found and listed.
```

In [ ]:
```
display(df)
```

	filename	class
2605	/content/dataset/cataract/_120_3067914.jpg	cataract
1844	/content/dataset/diabetic_retinopathy/102_righ...	diabetic_retinopathy
3750	/content/dataset/glaucoma/_326_7155812.jpg	glaucoma
1138	/content/dataset/diabetic_retinopathy/1017_rig...	diabetic_retinopathy
1493	/content/dataset/diabetic_retinopathy/11037_ri...	diabetic_retinopathy
...	...	...
3481	/content/dataset/glaucoma/_161_5855059.jpg	glaucoma
422	/content/dataset/normal/2418_right.jpg	normal
2587	/content/dataset/cataract/_246_8777337.jpg	cataract
2731	/content/dataset/cataract/_93_7584804.jpg	cataract
3463	/content/dataset/glaucoma/_123_6574789.jpg	glaucoma

4217 rows × 2 columns

In [ ]:
```
# Convert labels to one-hot encodings
label_encoder = LabelEncoder()
label = label_encoder.fit_transform(df['class'])
df['class'] = label

# check number assigned to each class
# Get the class names and corresponding integer encodings
class_names = label_encoder.classes_
class_numbers = label_encoder.transform(label_encoder.classes_)

# Print class names with the assigned numbers
class_dict = dict(zip(class_names, class_numbers))
print(class_dict)
```

```
{'cataract': 0, 'diabetic_retinopathy': 1, 'glaucoma': 2, 'normal': 3}
```

Loading [MathJax]/extensions/Safe.js

In [ ]:
```
display(df)
```

	filename	class
2605	/content/dataset/cataract/_120_3067914.jpg	0
1844	/content/dataset/diabetic_retinopathy/102_righ...	1
3750	/content/dataset/glaucoma/_326_7155812.jpg	2
1138	/content/dataset/diabetic_retinopathy/1017_rig...	1
1493	/content/dataset/diabetic_retinopathy/11037_ri...	1
...	...	...
3481	/content/dataset/glaucoma/_161_5855059.jpg	2
422	/content/dataset/normal/2418_right.jpg	3
2587	/content/dataset/cataract/_246_8777337.jpg	0
2731	/content/dataset/cataract/_93_7584804.jpg	0
3463	/content/dataset/glaucoma/_123_6574789.jpg	2

4217 rows × 2 columns

In [ ]:
```
# Check the balance of the classes
print(df['class'].value_counts(normalize=True))
print('------'*10)

# Split the data into train+validation and test sets
train_plus_val, test = train_test_split(df, test_size=0.2, stratify=df['clas

# Split the train+validation set into train and validation sets
train, val = train_test_split(train_plus_val, test_size=0.25, stratify=trair

# Now you have:
# train: 60% of the data
# val: 20% of the data
# test: 20% of the data

# Confirm the distribution across splits
print("Training set:")
print(train['class'].value_counts(normalize=True))
print('------'*10)


print("Validation set:")
print(val['class'].value_counts(normalize=True))
print('------'*10)


print("Test set:")
print(test['class'].value_counts(normalize=True))
```

Loading [MathJax]/extensions/Safe.js

```
class
1    0.260375
3    0.254683
0    0.246147
2    0.238795
Name: proportion, dtype: float64
------------------------------------------------------------
Training set:
class
1    0.260182
3    0.254646
0    0.245947
2    0.239225
Name: proportion, dtype: float64
------------------------------------------------------------
Validation set:
class
1    0.260664
3    0.254739
0    0.246445
2    0.238152
Name: proportion, dtype: float64
------------------------------------------------------------
Test set:
class
1    0.260664
3    0.254739
0    0.246445
2    0.238152
Name: proportion, dtype: float64
```

In [ ]:
```
train_links, train_labels = train['filename'].values , train['class'].values
val_links , val_labels = val['filename'].values , val['class'].values
test_links, test_labels = test['filename'].values , test['class'].values
```

In [ ]:
```
# Create a Function to Load and Preprocess Images
# tf.cond is a TensorFlow operation that allows for conditional execution ba


def load_and_preprocess_image(path, label, data_augmentation=True):
    # Read the image file
    image = tf.io.read_file(path)

    # Extract file extension
    file_extension = tf.strings.split(path, '.')[-1]

    # Decode based on file extension using tf.cond
    def decode_jpeg():
        return tf.image.decode_jpeg(image, channels=3)

    def decode_png():
        return tf.image.decode_png(image, channels=3)

    def decode_bmp():
        return tf.image.decode_bmp(image, channels=3)
```

Loading [MathJax]/extensions/Safe.js

```
    def decode_gif():
        # Decode GIF and take the first frame
        return tf.squeeze(tf.image.decode_gif(image), axis=0)

    # Handle each format
    image = tf.cond(tf.math.equal(file_extension, 'jpg'), decode_jpeg,
            lambda: tf.cond(tf.math.equal(file_extension, 'jpeg'), decode_jp
            lambda: tf.cond(tf.math.equal(file_extension, 'png'), decode_png
            lambda: tf.cond(tf.math.equal(file_extension, 'bmp'), decode_bmp
            lambda: tf.cond(tf.math.equal(file_extension, 'gif'), decode_gif
            decode_jpeg)))))

    # Resize and normalize
    image = tf.image.resize(image, [256, 256])
    image = image / 255.0  # Normalize to [0, 1] range

    # Apply data augmentation if in training mode
    if data_augmentation == True:
        # Randomly flip the image horizontally
        image = tf.image.random_flip_left_right(image)

        # Randomly flip the image vertically
        image = tf.image.random_flip_up_down(image)

        # Randomly rotate the image
        image = tf.image.rot90(image, k=tf.random.uniform(shape=[], minval=0

        # Randomly adjust brightness
        image = tf.image.random_brightness(image, max_delta=0.1)

        # Randomly zoom in
        image = tf.image.resize_with_crop_or_pad(image, 266, 266)  # Zoom in
        image = tf.image.random_crop(image, size=[256, 256, 3])

        # Randomly adjust contrast
        image = tf.image.random_contrast(image, lower=0.8, upper=1.2)

    return image, label
```

In [ ]:
```
# create TensorFlow datasets for each split
# When loading datasets, pass the data_augmentation flag True or False to ap
train_dataset = tf.data.Dataset.from_tensor_slices( (train_links , train_lab
train_dataset = train_dataset.map(lambda x, y: load_and_preprocess_image(x,

val_dataset = tf.data.Dataset.from_tensor_slices( (val_links , val_labels) )
val_dataset = val_dataset.map(lambda x, y: load_and_preprocess_image(x, y, d

test_dataset = tf.data.Dataset.from_tensor_slices( (test_links , test_labels
test_dataset = test_dataset.map(lambda x, y: load_and_preprocess_image(x, y,
```

In [ ]:
```
# Iterate over the dataset and print the first few elements
for data_element, label_element in train_dataset.take(1):  # Adjust the numb
    print(f"Data: {data_element.numpy()}, Label: {label_element.numpy()}")
```

Loading [MathJax]/extensions/Safe.js

```
Data: [[[-0.0467025  -0.02004027 -0.00261218]
  [-0.0467025  -0.02004027 -0.00261218]
  [-0.0467025  -0.02004027 -0.00261218]
  ...
  [-0.0467025  -0.02004027 -0.00261218]
  [-0.0467025  -0.02004027 -0.00261218]
  [-0.0467025  -0.02004027 -0.00261218]]

 [[-0.0467025  -0.02004027 -0.00261218]
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.16063404 -0.13397183 -0.11654373]
  ...
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.15862602 -0.1319638  -0.1145357 ]
  [-0.16063404 -0.13397183 -0.11654373]]

 [[-0.0467025  -0.02004027 -0.00261218]
  [-0.16063404 -0.13063706 -0.11654373]
  [-0.16063404 -0.13397183 -0.11654373]
  ...
  [-0.16063404 -0.13081634 -0.11338827]
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.15973762 -0.13307537 -0.11564729]]

 ...

 [[-0.0467025  -0.02004027 -0.00261218]
  [-0.15633115 -0.12941793 -0.11198983]
  [-0.15891287 -0.13225065 -0.11482257]
  ...
  [-0.15661803 -0.12995578 -0.11252768]
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.16063404 -0.12938206 -0.11195397]]

 [[-0.0467025  -0.02004027 -0.00261218]
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.1560443  -0.12938206 -0.11195397]
  ...
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.15891287 -0.13225065 -0.11482257]
  [-0.16063404 -0.13397183 -0.11654373]]

 [[-0.0467025  -0.02004027 -0.00261218]
  [-0.16063404 -0.13214308 -0.11654373]
  [-0.15776545 -0.13110322 -0.11367513]
  ...
  [-0.1589846  -0.12938206 -0.11489429]
  [-0.16063404 -0.13397183 -0.11654373]
  [-0.16063404 -0.13110322 -0.11367513]]], Label: 2
```

In [ ]:
```python
# Iterate over the dataset and print the first few elements
for data_element, label_element in train_dataset.take(1):  # Adjust the numb
    print(f"Data: {data_element.numpy().shape}, Label: {label_element.numpy(
```

```
Data: (256, 256, 3), Label: 2
```

Loading [MathJax]/extensions/Safe.js

In [ ]:
```
#prepare your datasets for model training and evaluation

batch_size = 16

train_dataset = train_dataset.batch(batch_size)
train_dataset = train_dataset.prefetch(buffer_size= tf.data.AUTOTUNE)

val_dataset = val_dataset.batch(batch_size)
val_dataset = val_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)

test_dataset = test_dataset.batch(batch_size)
test_dataset = test_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [ ]:
```
model = models.Sequential([
    # First Block
    layers.Conv2D(128, (3, 3), activation='relu', input_shape=(256, 256, 3),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),


    # Second Block
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),

    # Second Block
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),

    # Second Block
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.Conv2D(256, (3, 3), activation='relu',padding='valid'),
    layers.MaxPooling2D((2, 2)),
    layers.BatchNormalization(),



    # Global Average Pooling instead of Flatten
    layers.GlobalAveragePooling2D(),

    # Fully Connected Layers
    layers.Dense(512,activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(len(class_names), activation='softmax')  # Output layer
])


model.summary()
```

Model: "sequential"

Loading [MathJax]/extensions/Safe.js

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 256, 256, 128)
conv2d_1 (Conv2D)	(None, 256, 256, 128)
max_pooling2d (MaxPooling2D)	(None, 128, 128, 128)
batch_normalization (BatchNormalization)	(None, 128, 128, 128)
conv2d_2 (Conv2D)	(None, 126, 126, 256)
conv2d_3 (Conv2D)	(None, 124, 124, 256)
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 256)
batch_normalization_1 (BatchNormalization)	(None, 62, 62, 256)
conv2d_4 (Conv2D)	(None, 60, 60, 256)
conv2d_5 (Conv2D)	(None, 58, 58, 256)
max_pooling2d_2 (MaxPooling2D)	(None, 29, 29, 256)
batch_normalization_2 (BatchNormalization)	(None, 29, 29, 256)
conv2d_6 (Conv2D)	(None, 27, 27, 256)
conv2d_7 (Conv2D)	(None, 25, 25, 256)
max_pooling2d_3 (MaxPooling2D)	(None, 12, 12, 256)
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 256)
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)
dense (Dense)	(None, 512)
dropout (Dropout)	(None, 512)
dense_1 (Dense)	(None, 4)

Total params: 3,533,956 (13.48 MB)

Trainable params: 3,532,164 (13.47 MB)

Non-trainable params: 1,792 (7.00 KB)

In [ ]:
```
# Define the EarlyStopping callback to monitor the validation accuracy
early_stopping = EarlyStopping(
    monitor='val_accuracy',  # Monitoring validation accuracy
    patience=12,  # Number of epochs with no improvement after which trainin
    verbose=1,
```

Loading [MathJax]/extensions/Safe.js

```
    mode='max',  # Stops training when the quantity monitored has stopped ir
    restore_best_weights=True  # Restores model weights from the epoch with
)


# Define the ModelCheckpoint callback
model_checkpoint = ModelCheckpoint(
    filepath=r'/kaggle/working/best_model_custom.keras',  # Path to save the
    monitor='val_loss',  # Change to val_loss to monitor the validation loss
    verbose=1,
    save_best_only=True,  # Save only the best model
    mode='min'  # Save the model when the monitored metric has minimized

)


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


# Fit the model
history = model.fit(
    x = train_dataset,
    validation_data = val_dataset,
    epochs = 20,
    callbacks=[early_stopping , model_checkpoint]  # Add the EarlyStopping c
)
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 1/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 0s 449ms/step - accuracy: 0.3237 - loss: 1.5304
Epoch 1: val_loss improved from inf to 1.36084, saving model to /kaggle/work
ing/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 156s 712ms/step - accuracy: 0.3238 - loss: 1.53
01 - val_accuracy: 0.3803 - val_loss: 1.3608
Epoch 2/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 401ms/step - accuracy: 0.3676 - loss: 1.3511
Epoch 2: val_loss improved from 1.36084 to 1.32952, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 121s 438ms/step - accuracy: 0.3679 - loss: 1.35
07 - val_accuracy: 0.3851 - val_loss: 1.3295
Epoch 3/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 414ms/step - accuracy: 0.4579 - loss: 1.2265
Epoch 3: val_loss improved from 1.32952 to 1.30120, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 85s 459ms/step - accuracy: 0.4580 - loss: 1.226
3 - val_accuracy: 0.4218 - val_loss: 1.3012
Epoch 4/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.4954 - loss: 1.1402
Epoch 4: val_loss improved from 1.30120 to 1.27796, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 81s 451ms/step - accuracy: 0.4955 - loss: 1.140
3 - val_accuracy: 0.4633 - val_loss: 1.2780
Epoch 5/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step - accuracy: 0.5034 - loss: 1.1096
Epoch 5: val_loss improved from 1.27796 to 1.24438, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 76s 476ms/step - accuracy: 0.5034 - loss: 1.109
6 - val_accuracy: 0.5047 - val_loss: 1.2444
Epoch 6/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 413ms/step - accuracy: 0.5092 - loss: 1.0820
Epoch 6: val_loss improved from 1.24438 to 1.18827, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 72s 454ms/step - accuracy: 0.5092 - loss: 1.082
1 - val_accuracy: 0.5190 - val_loss: 1.1883
Epoch 7/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step - accuracy: 0.5257 - loss: 1.0599
Epoch 7: val_loss improved from 1.18827 to 1.08230, saving model to /kaggle/
working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 72s 452ms/step - accuracy: 0.5257 - loss: 1.059
9 - val_accuracy: 0.5201 - val_loss: 1.0823
Epoch 8/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 415ms/step - accuracy: 0.5484 - loss: 1.0480
Epoch 8: val_loss did not improve from 1.08230
159/159 ━━━━━━━━━━━━━━━━━━━━ 82s 452ms/step - accuracy: 0.5484 - loss: 1.047
9 - val_accuracy: 0.5130 - val_loss: 1.1042
Epoch 9/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step - accuracy: 0.5642 - loss: 0.9887
Epoch 9: val_loss did not improve from 1.08230
159/159 ━━━━━━━━━━━━━━━━━━━━ 76s 475ms/step - accuracy: 0.5641 - loss: 0.988
8 - val_accuracy: 0.4526 - val_loss: 1.2079
Epoch 10/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 415ms/step - accuracy: 0.6140 - loss: 0.9474
Epoch 10: val_loss improved from 1.08230 to 1.03608, saving model to /kaggl
e/working/best_model_custom.keras
```

Loading [MathJax]/extensions/Safe.js

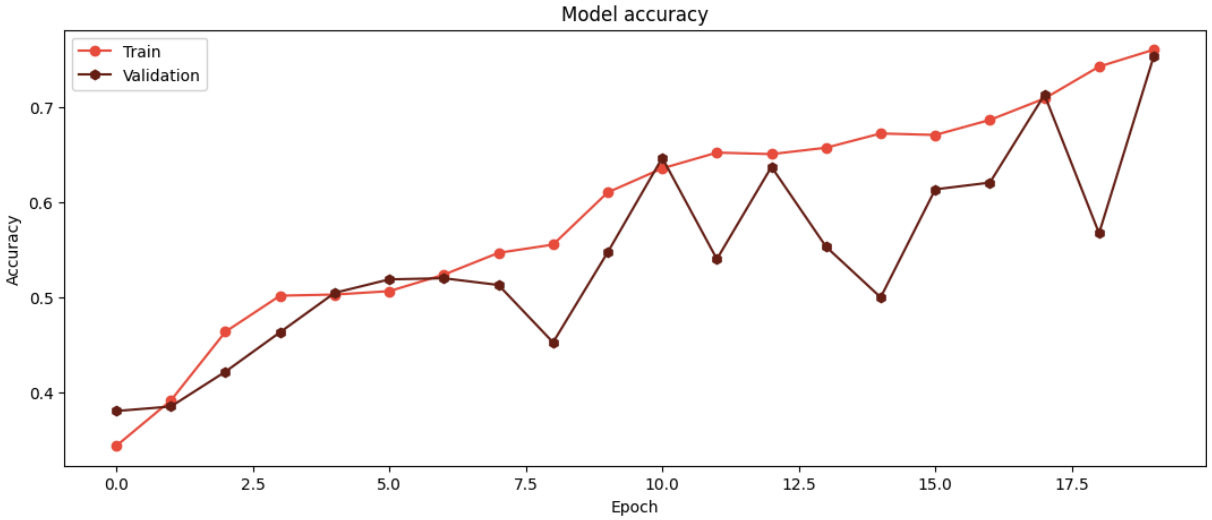
```
159/159 ━━━━━━━━━━━━━━━━━━━━ 83s 479ms/step - accuracy: 0.6139 - loss: 0.947
2 - val_accuracy: 0.5474 - val_loss: 1.0361
Epoch 11/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.6438 - loss: 0.8939
Epoch 11: val_loss improved from 1.03608 to 0.88659, saving model to /kaggl
e/working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 78s 452ms/step - accuracy: 0.6437 - loss: 0.893
9 - val_accuracy: 0.6469 - val_loss: 0.8866
Epoch 12/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 413ms/step - accuracy: 0.6577 - loss: 0.8755
Epoch 12: val_loss did not improve from 0.88659
159/159 ━━━━━━━━━━━━━━━━━━━━ 71s 448ms/step - accuracy: 0.6577 - loss: 0.875
5 - val_accuracy: 0.5403 - val_loss: 0.9910
Epoch 13/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 415ms/step - accuracy: 0.6502 - loss: 0.8561
Epoch 13: val_loss did not improve from 0.88659
159/159 ━━━━━━━━━━━━━━━━━━━━ 87s 478ms/step - accuracy: 0.6502 - loss: 0.856
0 - val_accuracy: 0.6374 - val_loss: 0.9086
Epoch 14/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.6563 - loss: 0.8347
Epoch 14: val_loss did not improve from 0.88659
159/159 ━━━━━━━━━━━━━━━━━━━━ 78s 450ms/step - accuracy: 0.6564 - loss: 0.834
7 - val_accuracy: 0.5533 - val_loss: 1.0052
Epoch 15/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 413ms/step - accuracy: 0.6762 - loss: 0.8113
Epoch 15: val_loss did not improve from 0.88659
159/159 ━━━━━━━━━━━━━━━━━━━━ 76s 475ms/step - accuracy: 0.6762 - loss: 0.811
3 - val_accuracy: 0.5000 - val_loss: 1.6304
Epoch 16/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 413ms/step - accuracy: 0.6697 - loss: 0.8125
Epoch 16: val_loss improved from 0.88659 to 0.86898, saving model to /kaggl
e/working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 76s 477ms/step - accuracy: 0.6697 - loss: 0.812
5 - val_accuracy: 0.6137 - val_loss: 0.8690
Epoch 17/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.6788 - loss: 0.7850
Epoch 17: val_loss improved from 0.86898 to 0.79307, saving model to /kaggl
e/working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 82s 480ms/step - accuracy: 0.6789 - loss: 0.784
9 - val_accuracy: 0.6209 - val_loss: 0.7931
Epoch 18/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.6911 - loss: 0.7289
Epoch 18: val_loss improved from 0.79307 to 0.71205, saving model to /kaggl
e/working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 82s 480ms/step - accuracy: 0.6914 - loss: 0.728
7 - val_accuracy: 0.7133 - val_loss: 0.7121
Epoch 19/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.7397 - loss: 0.6552
Epoch 19: val_loss did not improve from 0.71205
159/159 ━━━━━━━━━━━━━━━━━━━━ 77s 451ms/step - accuracy: 0.7398 - loss: 0.655
1 - val_accuracy: 0.5675 - val_loss: 1.0198
Epoch 20/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 415ms/step - accuracy: 0.7617 - loss: 0.6142
Epoch 20: val_loss improved from 0.71205 to 0.61925, saving model to /kaggl
e/working/best_model_custom.keras
159/159 ━━━━━━━━━━━━━━━━━━━━ 82s 451ms/step - accuracy: 0.7617 - loss: 0.614
```

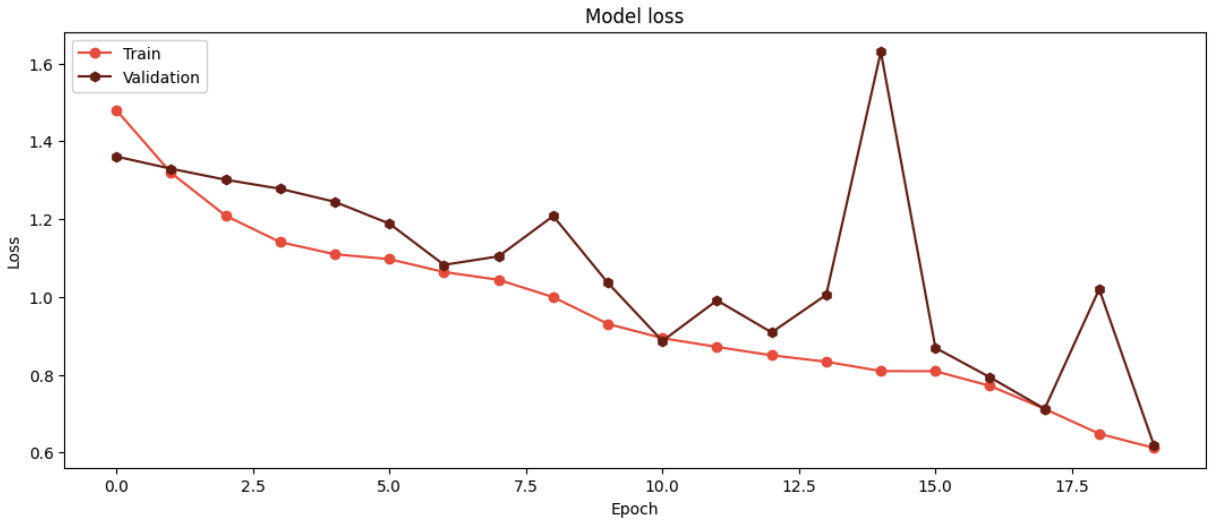
Loading [MathJax]/extensions/Safe.js

```
2 - val_accuracy: 0.7536 - val_loss: 0.6192
Restoring model weights from the end of the best epoch: 20.
```

In [ ]:

```
# Plot training & validation accuracy values
plt.figure(figsize=(13,5))
plt.plot(history.history['accuracy'],color="#E74C3C", marker='o')
plt.plot(history.history['val_accuracy'], color='#641E16', marker='h')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(13,5))
plt.plot(history.history['loss'],color="#E74C3C", marker='o')
plt.plot(history.history['val_loss'], color='#641E16', marker='h')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend( ['Train', 'Validation'], loc='upper left')
plt.show()
```





Loading [MathJax]/extensions/Safe.js

In [ ]:
```
# Load the saved model
# If the best model is captured by the early stopping mechanism then best_mo
# best_model = model

best_model = load_model(r'/kaggle/working/best_model_custom.keras')



# Evaluate the model
train_loss, train_accuracy = best_model.evaluate(train_dataset)
val_loss, val_accuracy = best_model.evaluate(val_dataset)
test_loss, test_accuracy = best_model.evaluate(test_dataset)

print(f"train loss: {train_loss}")
print(f"train accuracy: {train_accuracy}")
print('----'*6)
print(f"val loss: {val_loss}")
print(f"val accuracy: {val_accuracy}")
print('----'*6)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 26s 146ms/step - accuracy: 0.7099 - loss: 0.699
7
53/53 ━━━━━━━━━━━━━━━━━━━━ 7s 134ms/step - accuracy: 0.7577 - loss: 0.6065
53/53 ━━━━━━━━━━━━━━━━━━━━ 6s 117ms/step - accuracy: 0.7519 - loss: 0.6151
train loss: 0.68248450756073
train accuracy: 0.7133254408836365
------------------------
val loss: 0.61924809217453
val accuracy: 0.7535545229911804
------------------------
Test loss: 0.6317344903945923
Test accuracy: 0.7274881601333618
```

In [ ]:
```
# Assuming best_model is your trained Keras model

# Get the predicted labels from the model
y_pred = np.argmax( best_model.predict(test_dataset) , axis=1 ) # Convert pr
y_true = test_labels

# Compute confusion matrix
conf_mat = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_mat)

# Get the class labels from the LabelEncoder
class_labels = label_encoder.classes_

# Compute classification report
report = classification_report(y_true, y_pred, target_names=class_labels)

print("\nClassification Report:")
print(report)
```

Loading [MathJax]/extensions/Safe.js

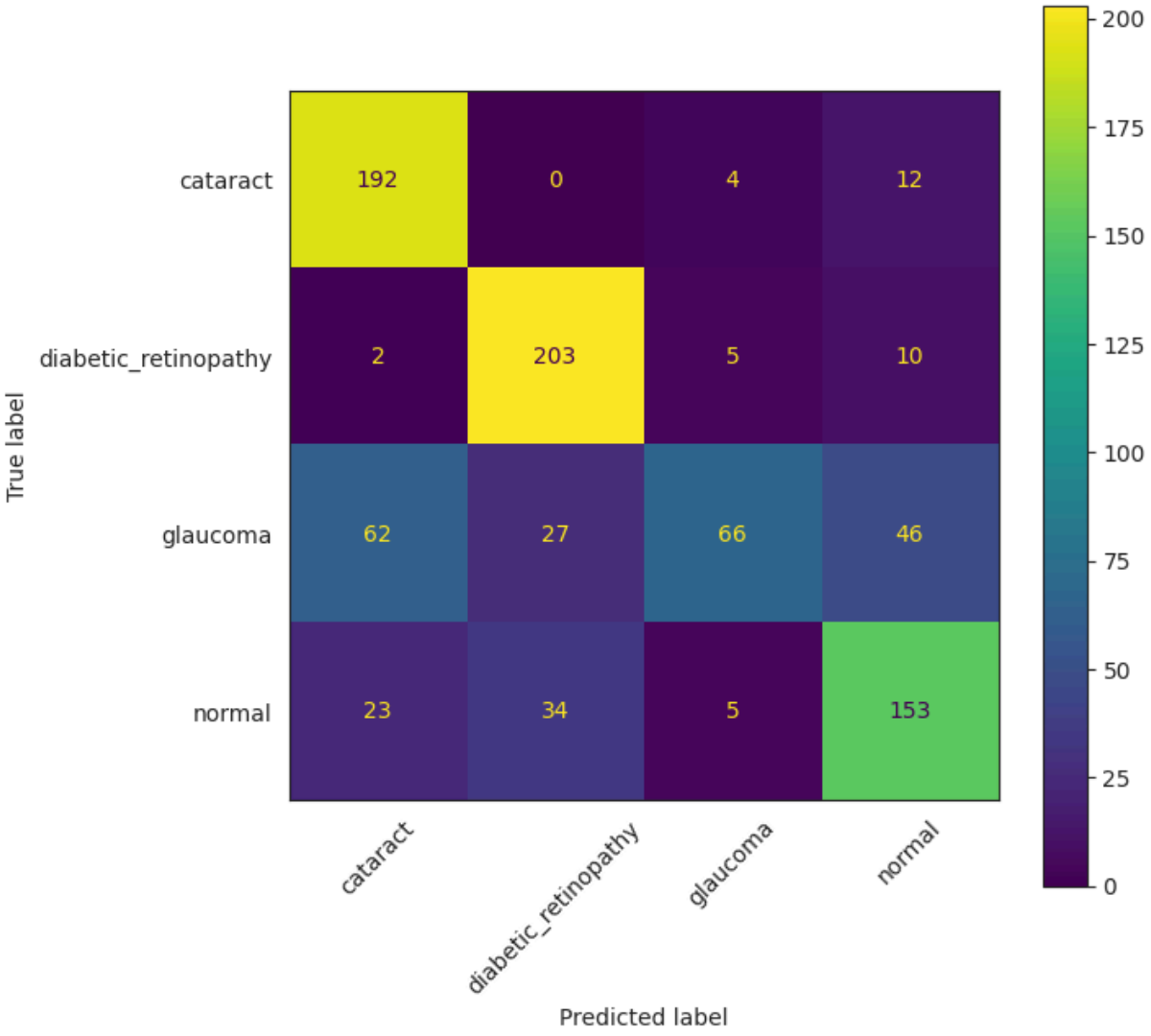
```
53/53 ━━━━━━━━━━━━━━━━━━━━ 12s 215ms/step
Confusion Matrix:
[[192   0   4  12]
 [  2 203   5  10]
 [ 62  27  66  46]
 [ 23  34   5 153]]

Classification Report:
                      precision    recall  f1-score   support

            cataract       0.69      0.92      0.79       208
diabetic_retinopathy       0.77      0.92      0.84       220
            glaucoma       0.82      0.33      0.47       201
              normal       0.69      0.71      0.70       215

            accuracy                           0.73       844
           macro avg       0.74      0.72      0.70       844
        weighted avg       0.74      0.73      0.70       844
```

In [ ]:

```python
#Confusion matrix
print('Total Number Of Test data: ', len(test_labels))

sn.set_style("white")
def plot_confusion_matrix(conf_mat, classes):
    """
    This function prints and plots the confusion matrix.
    """
    fig, ax = plt.subplots(figsize=(7,7)) # change the plot size
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat, display_labels=
    disp = disp.plot(include_values=True,cmap='viridis', ax=ax, xticks_rotat
    plt.show()

# Get your confusion matrix
conf_mat = conf_mat

# Using label_encoder.classes_ guarantees that class_names matches
# the order that was used during the one-hot encoding process
class_names = label_encoder.classes_

# Now plot using the function
plot_confusion_matrix(conf_mat, class_names)
```

```
Total Number Of Test data:  844
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

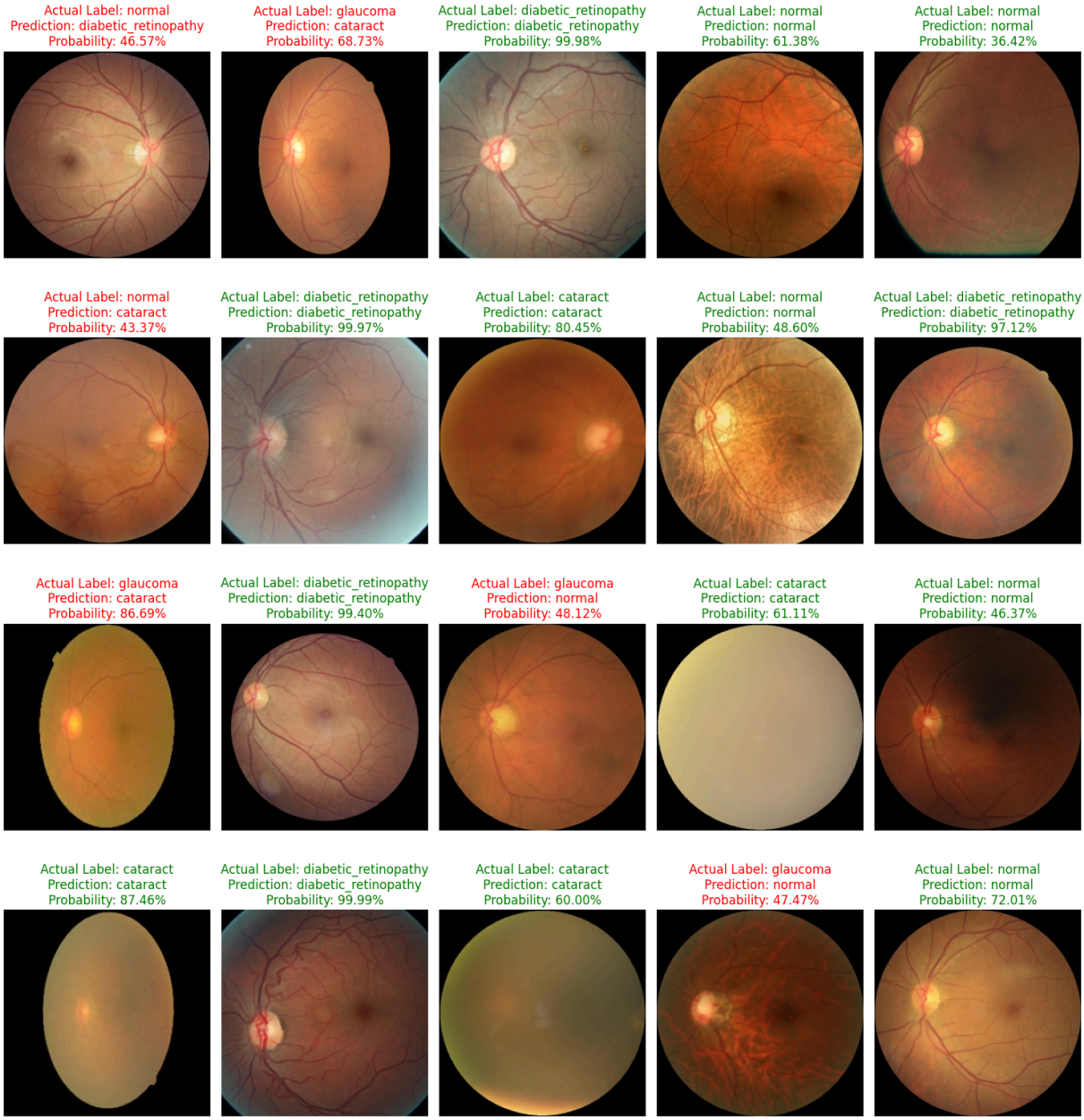
```python
# probability explanation in below function:
# For example, if the model predicts an image as class B with a probability
# the plot will show "Probability: 70%".
# This means the model is 70% confident that the image belongs to class B.


def plot_test_predictions(model, test_dataset, class_labels, num_images=20):
    """
    Plots the predictions of a model on the test dataset.

    Parameters:
    - model: Trained Keras model to be used for prediction.
    - test_dataset: TensorFlow dataset containing the test images and labels
    - class_labels: List of class labels.
    - num_images: Number of test images to plot (default is 20).
    """

    # Initialize lists to accumulate images and labels
    images = []
    true_labels = []
    pred_labels = []
    pred_probs = []
```

Loading [MathJax]/extensions/Safe.js

```
    for batch_images, batch_labels in test_dataset:
        # Predict on the batch
        batch_pred_probs = model.predict(batch_images)
        batch_pred_labels = np.argmax(batch_pred_probs, axis=1)

        # Accumulate images and labels
        images.extend(batch_images)
        true_labels.extend(batch_labels)
        pred_labels.extend(batch_pred_labels)
        pred_probs.extend(np.max(batch_pred_probs, axis=1) * 100)

        if len(images) >= num_images:
            break

    # Plot the images with predictions
    plt.figure(figsize=(15, 20))
    for i in range(num_images):
        plt.subplot(5, 5, i + 1)
        plt.imshow(images[i])
        actual_label = class_labels[true_labels[i]]
        predicted_label = class_labels[pred_labels[i]]
        probability = pred_probs[i]  # Probability of the predicted class

        color = 'green' if actual_label == predicted_label else 'red'
        plt.title(f"Actual Label: {actual_label}\nPrediction: {predicted_lab
                  color=color, fontsize=12)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Use the function
plot_test_predictions(best_model, test_dataset, class_labels=class_names, nu
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

```
# Load the model pre-trained on EfficientNetB0, without the top layers
base_model = EfficientNetB0(input_shape=(256, 256, 3), include_top=False, we

# set True >>> makes all the layers in the base model trainable
base_model.trainable = True


# Create your own model on top of the pre-trained model
model = models.Sequential([
    base_model,
    layers.BatchNormalization(),
    layers.MaxPool2D((2,2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(len(class_names), activation='softmax')
])
```

Loading [MathJax]/extensions/Safe.js

```
model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/effi
cientnetb0_notop.h5
16705208/16705208 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

Model: "sequential_1"

Layer (type)	Output Shape
efficientnetb0 (Functional)	(None, 8, 8, 1280)
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 1280)
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 1280)
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1280)
dense_2 (Dense)	(None, 512)
dropout_1 (Dropout)	(None, 512)
dense_3 (Dense)	(None, 4)

Total params: 4,712,615 (17.98 MB)

Trainable params: 4,668,032 (17.81 MB)

Non-trainable params: 44,583 (174.16 KB)

In [ ]:
```
#-------------------------------------------------------------------------

class CustomEarlyStoppingAndCheckpoint(Callback):
    def __init__(self, save_path, patience=12, verbose=1, save_best_only=Tru
        super(CustomEarlyStoppingAndCheckpoint, self).__init__()
        self.save_path = save_path
        self.patience = patience
        self.verbose = verbose
        self.save_best_only = save_best_only
        self.best_weights = None
        self.best_acc = -np.Inf
        self.best_loss = np.Inf
        self.wait = 0
        self.stopped_epoch = 0

    def on_epoch_end(self, epoch, logs=None):
        val_loss = logs.get('val_loss')
        val_acc = logs.get('val_accuracy')
        if val_acc is None or val_loss is None:
            return

        # Checkpoint for best loss
        if val_loss < self.best_loss:
            self.best_loss = val_loss
            if self.save_best_only:
                self.model.save(self.save_path, include_optimizer=True)
```

Loading [MathJax]/extensions/Safe.js

```
                if self.verbose > 0:
                    print(f"\nEpoch {epoch + 1}: val_loss improved to {val_l

        # Early stopping for best accuracy
        if val_acc > self.best_acc:
            self.best_acc = val_acc
            self.best_weights = self.model.get_weights()
            self.wait = 0
        else:
            self.wait += 1
            if self.wait >= self.patience:
                self.stopped_epoch = epoch
                self.model.stop_training = True
                if self.verbose > 0:
                    print(f"\nEpoch {epoch + 1}: early stopping")
                if self.best_weights is not None:
                    self.model.set_weights(self.best_weights)
                    if self.verbose > 0:
                        print("Restoring model weights from the end of the b

    def on_train_end(self, logs=None):
        if self.stopped_epoch > 0 and self.verbose > 0:
            print(f"Epoch {self.stopped_epoch + 1}: early stopping triggered


#--------------------------------------------------------------------------

# Usage of the custom callback
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr

callbacks = [
    CustomEarlyStoppingAndCheckpoint(
        save_path=r'/kaggle/working/best_model_transfer.h5',
        patience=12,
        verbose=1,
        save_best_only=True
    )
]



# Fit the model
history = model.fit(
    x = train_dataset,
    validation_data = val_dataset,
    epochs = 20,
    callbacks= callbacks # Add the EarlyStopping callback
)
```

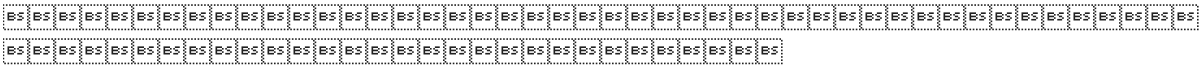
```
Epoch 1/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 0s 360ms/step - accuracy: 0.6445 - loss: 1.1540
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 1: val_loss improved to 5.6709, saving model to /kaggle/working/best_m
odel_transfer.h5
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 163s 483ms/step - accuracy: 0.6452 - loss: 1.15
17 - val_accuracy: 0.2464 - val_loss: 5.6709
Epoch 2/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 0.8105 - loss: 0.5227
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 2: val_loss improved to 2.7260, saving model to /kaggle/working/best_m
odel_transfer.h5
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 39s 153ms/step - accuracy: 0.8107 - loss: 0.522
2 - val_accuracy: 0.2405 - val_loss: 2.7260
Epoch 3/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 38s 136ms/step - accuracy: 0.8693 - loss: 0.372
6 - val_accuracy: 0.2476 - val_loss: 5.5512
Epoch 4/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step - accuracy: 0.8860 - loss: 0.3235
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 4: val_loss improved to 1.7219, saving model to /kaggle/working/best_m
odel_transfer.h5
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 43s 152ms/step - accuracy: 0.8859 - loss: 0.323
7 - val_accuracy: 0.4763 - val_loss: 1.7219
Epoch 5/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step - accuracy: 0.8913 - loss: 0.3110
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 5: val_loss improved to 0.8746, saving model to /kaggle/working/best_m
odel_transfer.h5
```

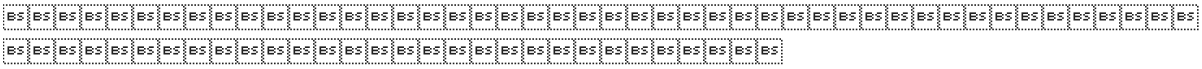
```
159/159 ━━━━━━━━━━━━━━━━━━━━ 47s 191ms/step - accuracy: 0.8914 - loss: 0.310
9 - val_accuracy: 0.6943 - val_loss: 0.8746
Epoch 6/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.8899 - loss: 0.3136
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

Loading [MathJax]/extensions/Safe.js

```
Epoch 6: val_loss improved to 0.4133, saving model to /kaggle/working/best_m
odel_transfer.h5
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 24s 148ms/step - accuracy: 0.8898 - loss: 0.313
5 - val_accuracy: 0.8626 - val_loss: 0.4133
Epoch 7/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.9113 - loss: 0.2610
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 7: val_loss improved to 0.3247, saving model to /kaggle/working/best_m
odel_transfer.h5
```

```
159/159 ━━━━━━━━━━━━━━━━━━━━ 46s 181ms/step - accuracy: 0.9113 - loss: 0.261
0 - val_accuracy: 0.9040 - val_loss: 0.3247
Epoch 8/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 28s 177ms/step - accuracy: 0.9051 - loss: 0.263
4 - val_accuracy: 0.8768 - val_loss: 0.3386
Epoch 9/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.8971 - loss: 0.2831
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save('my_model.
keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 9: val_loss improved to 0.3097, saving model to /kaggle/working/best_m
odel_transfer.h5
```



```
159/159 ━━━━━━━━━━━━━━━━━━━━ 24s 147ms/step - accuracy: 0.8972 - loss: 0.282
8 - val_accuracy: 0.9052 - val_loss: 0.3097
Epoch 10/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 40s 139ms/step - accuracy: 0.9170 - loss: 0.222
4 - val_accuracy: 0.8495 - val_loss: 0.4871
Epoch 11/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 42s 143ms/step - accuracy: 0.9379 - loss: 0.190
2 - val_accuracy: 0.8614 - val_loss: 0.4570
Epoch 12/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 22s 138ms/step - accuracy: 0.9337 - loss: 0.164
1 - val_accuracy: 0.8412 - val_loss: 0.4457
Epoch 13/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 28s 176ms/step - accuracy: 0.9231 - loss: 0.229
3 - val_accuracy: 0.7820 - val_loss: 0.8331
Epoch 14/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 36s 142ms/step - accuracy: 0.9348 - loss: 0.191
6 - val_accuracy: 0.8602 - val_loss: 0.4556
Epoch 15/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 46s 178ms/step - accuracy: 0.9370 - loss: 0.170
0 - val_accuracy: 0.8898 - val_loss: 0.4365
Epoch 16/20
158/159 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step - accuracy: 0.9205 - loss: 0.2344
```

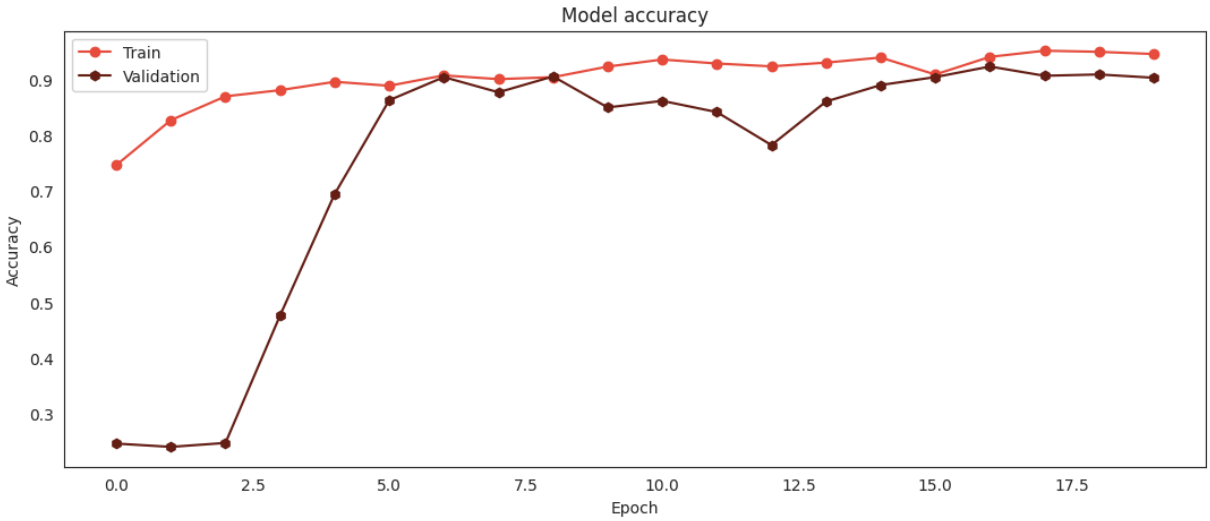
Loading [MathJax]/extensions/Safe.js

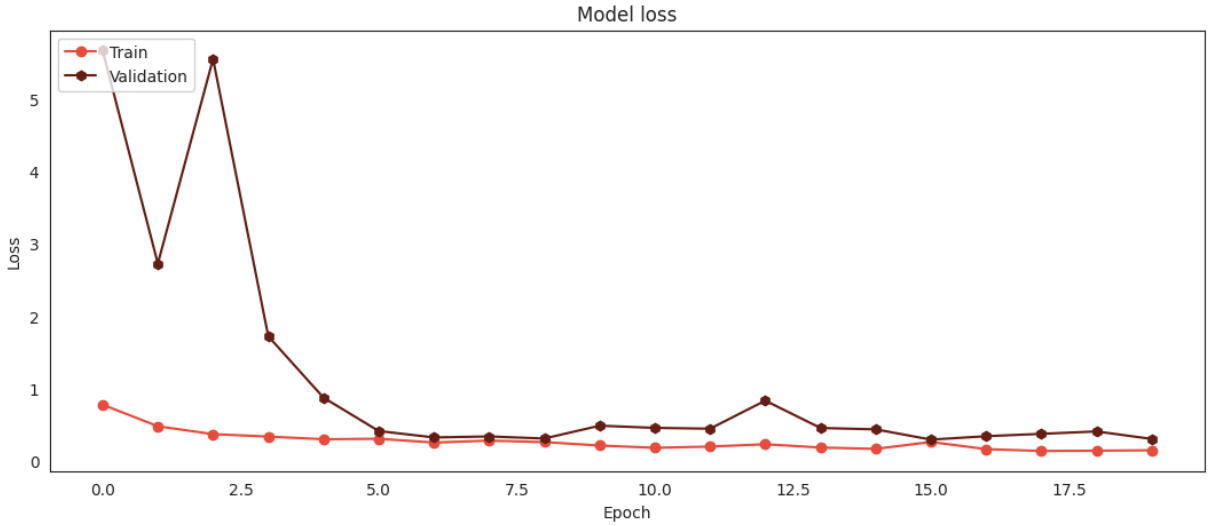
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Epoch 16: val_loss improved to 0.2946, saving model to /kaggle/working/best_model_transfer.h5
159/159 ━━━━━━━━━━━━━━━━━━━━ 37s 150ms/step - accuracy: 0.9204 - loss: 0.2347 - val_accuracy: 0.9040 - val_loss: 0.2946
Epoch 17/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 40s 146ms/step - accuracy: 0.9403 - loss: 0.1722 - val_accuracy: 0.9230 - val_loss: 0.3419
Epoch 18/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 24s 148ms/step - accuracy: 0.9535 - loss: 0.1309 - val_accuracy: 0.9064 - val_loss: 0.3746
Epoch 19/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 28s 176ms/step - accuracy: 0.9520 - loss: 0.1262 - val_accuracy: 0.9088 - val_loss: 0.4079
Epoch 20/20
159/159 ━━━━━━━━━━━━━━━━━━━━ 41s 175ms/step - accuracy: 0.9420 - loss: 0.1680 - val_accuracy: 0.9028 - val_loss: 0.3036
```

In [ ]:

```python
# Plot training & validation accuracy values
plt.figure(figsize=(13,5))
plt.plot(history.history['accuracy'],color="#E74C3C", marker='o')
plt.plot(history.history['val_accuracy'], color='#641E16', marker='h')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(13,5))
plt.plot(history.history['loss'],color="#E74C3C", marker='o')
plt.plot(history.history['val_loss'], color='#641E16', marker='h')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend( ['Train', 'Validation'], loc='upper left')
plt.show()
```

Loading [MathJax]/extensions/Safe.js





In [ ]:
```
# Load the saved model
# If the best model is captured by the early stopping mechanism then best_mo
# best_model = model

best_model = load_model(r'/kaggle/working/best_model_transfer.h5')
best_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',


# Evaluate the model
train_loss, train_accuracy = best_model.evaluate(train_dataset)
val_loss, val_accuracy = best_model.evaluate(val_dataset)
test_loss, test_accuracy = best_model.evaluate(test_dataset)

print(f"train loss: {train_loss}")
print(f"train accuracy: {train_accuracy}")
print('----'*6)
print(f"val loss: {val_loss}")
print(f"val accuracy: {val_accuracy}")
print('----'*6)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

Loading [MathJax]/extensions/Safe.js

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
159/159 ━━━━━━━━━━━━━━━━━━━━ 30s 137ms/step - accuracy: 0.9098 - loss: 0.2378
53/53 ━━━━━━━━━━━━━━━━━━━━ 7s 137ms/step - accuracy: 0.9067 - loss: 0.2736
53/53 ━━━━━━━━━━━━━━━━━━━━ 6s 104ms/step - accuracy: 0.8763 - loss: 0.3474
train loss: 0.22449705004692078
train accuracy: 0.9157769680023193
------------------------
val loss: 0.294607013463974
val accuracy: 0.9040284156799316
------------------------
Test loss: 0.35813987255096436
Test accuracy: 0.8803317546844482
```

In [ ]:

```python
# Assuming best_model is your trained Keras model

# Get the predicted labels from the model
y_pred = np.argmax( best_model.predict(test_dataset) , axis=1 ) # Convert pr
y_true = test_labels

# Compute confusion matrix
conf_mat = confusion_matrix(y_true, y_pred)

print("Confusion Matrix:")
print(conf_mat)

# Get the class labels from the LabelEncoder
class_labels = label_encoder.classes_

# Compute classification report
report = classification_report(y_true, y_pred, target_names=class_labels)

print("\nClassification Report:")
print(report)
```

```
53/53 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step
Confusion Matrix:
[[192   0   5  11]
 [  1 219   0   0]
 [ 26   0 131  44]
 [  4   3   7 201]]

Classification Report:
                     precision    recall  f1-score   support

           cataract       0.86      0.92      0.89       208
diabetic_retinopathy       0.99      1.00      0.99       220
           glaucoma       0.92      0.65      0.76       201
             normal       0.79      0.93      0.85       215

           accuracy                           0.88       844
          macro avg       0.89      0.88      0.87       844
       weighted avg       0.89      0.88      0.88       844
```

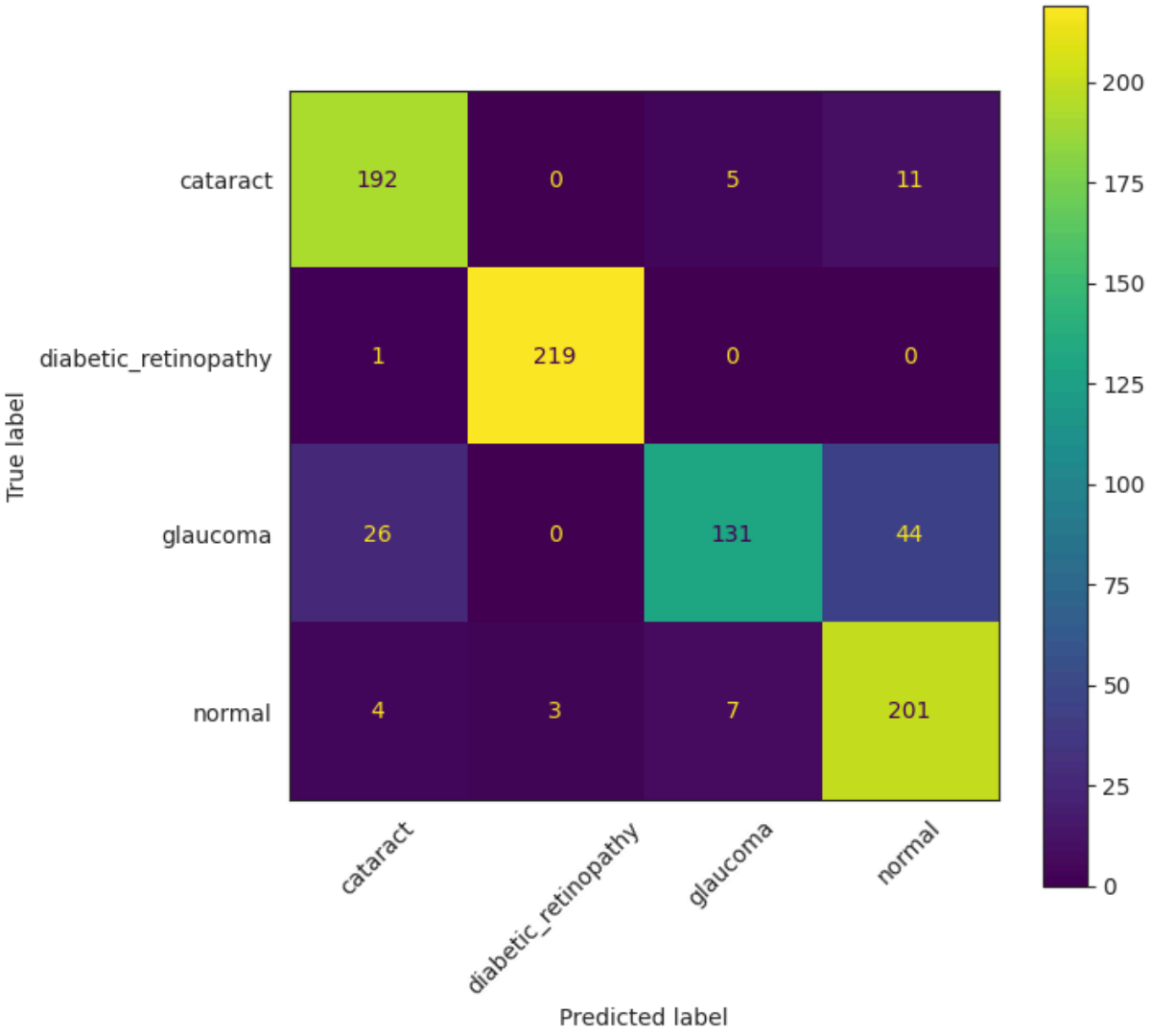
Loading [MathJax]/extensions/Safe.js

In [ ]:

```
#Confusion matrix
print('Total Number Of Test data: ', len(test_labels))

sn.set_style("white")
def plot_confusion_matrix(conf_mat, classes):
    """
    This function prints and plots the confusion matrix.
    """
    fig, ax = plt.subplots(figsize=(7,7)) # change the plot size
    disp = ConfusionMatrixDisplay(confusion_matrix=conf_mat, display_labels=
    disp = disp.plot(include_values=True,cmap='viridis', ax=ax, xticks_rotat
    plt.show()

# Get your confusion matrix
conf_mat = conf_mat

# Using label_encoder.classes_ guarantees that class_names matches
# the order that was used during the one-hot encoding process
class_names = label_encoder.classes_

# Now plot using the function
plot_confusion_matrix(conf_mat, class_names)
```

```
Total Number Of Test data:  844
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

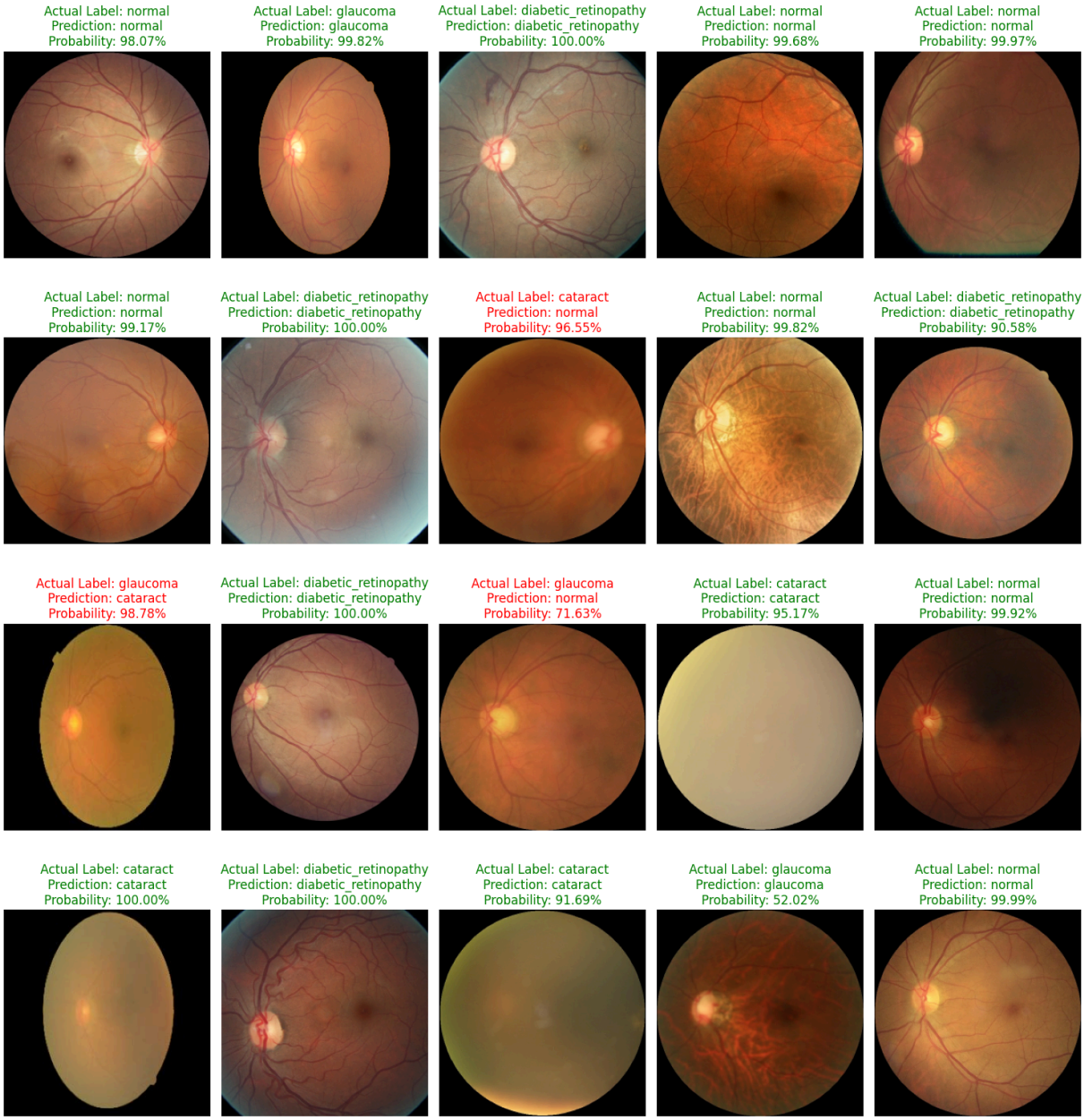
```python
# probability explanation in below function:
# For example, if the model predicts an image as class B with a probability
# the plot will show "Probability: 70%".
# This means the model is 70% confident that the image belongs to class B.

def plot_test_predictions(model, test_dataset, class_labels, num_images=20):
    """
    Plots the predictions of a model on the test dataset.

    Parameters:
    - model: Trained Keras model to be used for prediction.
    - test_dataset: TensorFlow dataset containing the test images and labels
    - class_labels: List of class labels.
    - num_images: Number of test images to plot (default is 20).
    """

    # Initialize lists to accumulate images and labels
    images = []
    true_labels = []
    pred_labels = []
    pred_probs = []
```

Loading [MathJax]/extensions/Safe.js

```
    for batch_images, batch_labels in test_dataset:
        # Predict on the batch
        batch_pred_probs = model.predict(batch_images)
        batch_pred_labels = np.argmax(batch_pred_probs, axis=1)

        # Accumulate images and labels
        images.extend(batch_images)
        true_labels.extend(batch_labels)
        pred_labels.extend(batch_pred_labels)
        pred_probs.extend(np.max(batch_pred_probs, axis=1) * 100)

        if len(images) >= num_images:
            break

    # Plot the images with predictions
    plt.figure(figsize=(15, 20))
    for i in range(num_images):
        plt.subplot(5, 5, i + 1)
        plt.imshow(images[i])
        actual_label = class_labels[true_labels[i]]
        predicted_label = class_labels[pred_labels[i]]
        probability = pred_probs[i]  # Probability of the predicted class

        color = 'green' if actual_label == predicted_label else 'red'
        plt.title(f"Actual Label: {actual_label}\nPrediction: {predicted_lab
                  color=color, fontsize=12)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Use the function
plot_test_predictions(best_model, test_dataset, class_labels=class_names, nu
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 6s 6s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

This notebook was converted with convert.ploomber.io

Loading [MathJax]/extensions/Safe.js