Eye Disease Classifier

April 6, 2025

[2]:
```
# Import necessary libraries
import os
import numpy as np
import pandas as pd
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt

# Import other libraries
import keras
from keras import layers, models, regularizers, optimizers, callbacks
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import image_dataset_from_directory
```

[3]:
```
# Physical device configuration
physical_devices = tf.config.list_physical_devices('GPU')
for gpu in physical_devices:
    tf.config.experimental.set_memory_growth(gpu, True)

print("Num GPUs Available: ", len(physical_devices))  # 1 or 0 (1 if GPU is␣
↪available)
```

```
Num GPUs Available:  1
```

[4]:
```
# Constants Variables
EPOCHS = 100

DATA_SET = 'dataset'
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
SEED = 123
```

[5]:
```
# Load the dataset
dataset = image_dataset_from_directory(
    DATA_SET,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=None,
```

```
    shuffle=True,
)
```

```
Found 16242 files belonging to 10 classes.
```

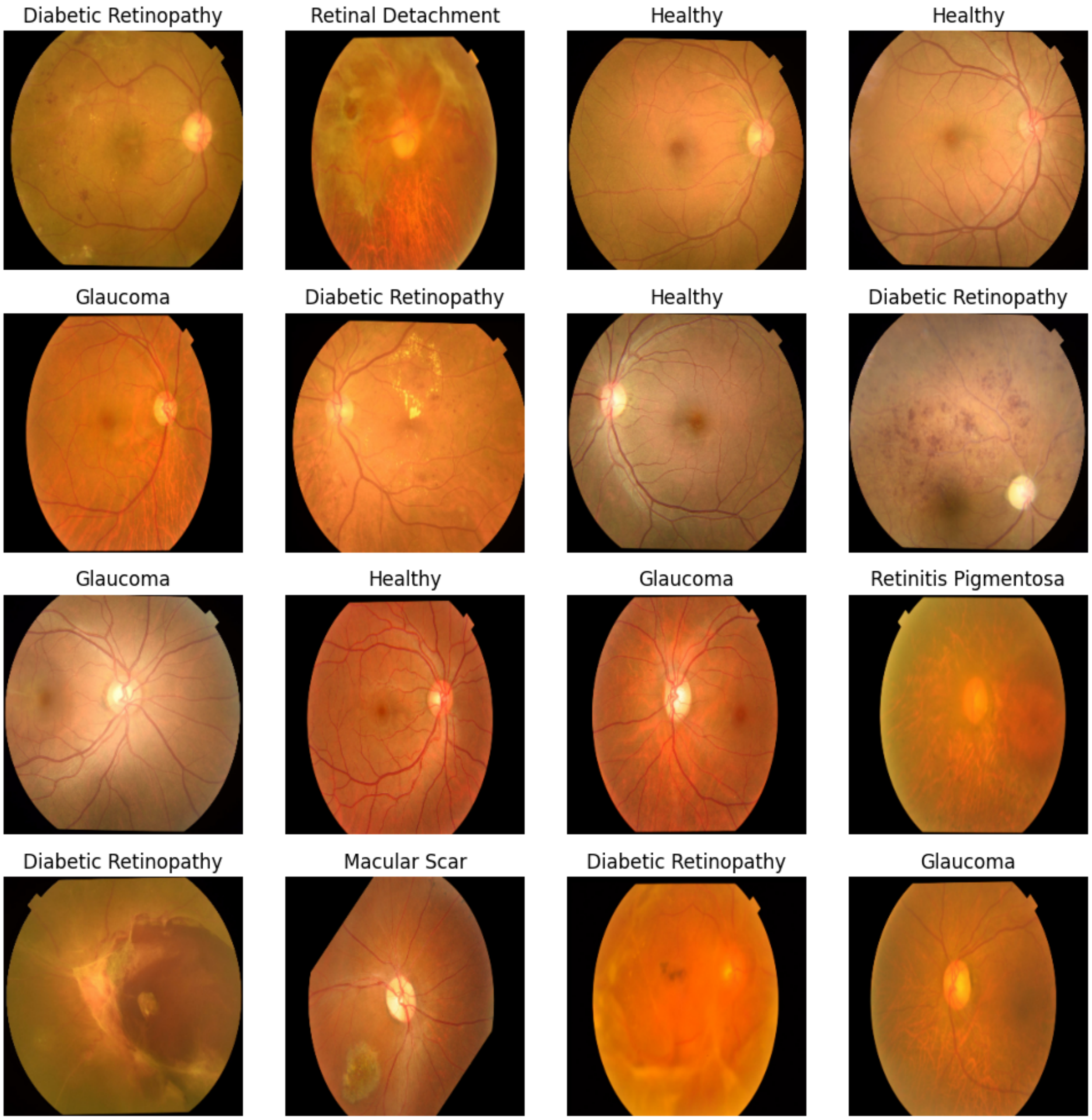
[7]:
```
# Print the class names
class_names = dataset.class_names

# Get the class names
print("Dataset Classes:")
for i, class_name in enumerate(class_names):
    print(f"{i + 1}. {class_name.replace('_', ' ')}")
```

```
Dataset Classes:
1. Central Serous Chorioretinopathy
2. Diabetic Retinopathy
3. Disc Edema
4. Glaucoma
5. Healthy
6. Macular Scar
7. Myopia
8. Pterygium
9. Retinal Detachment
10. Retinitis Pigmentosa
```

[8]:
```
# Dataset size
dataset_size = tf.data.experimental.cardinality(dataset).numpy()
print(f"Total samples: {dataset_size}")
```

```
Total samples: 16242
```

[42]:
```
# Set up figure
plt.figure(figsize=(10, 10))

# Take 16 individual samples
for i, (image, label) in enumerate(dataset.take(16)):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(image.numpy().astype("uint8"))
    plt.title(class_names[label.numpy()])
    plt.axis("off")

plt.tight_layout()
plt.show()
```



[9]:
```
# Split ratios
train_ratio, val_ratio, test_ratio  = 0.8, 0.1, 0.1

# Calculate sizes
dataset_size = len(dataset)  # or use cardinality
train_size = int(train_ratio * dataset_size)
val_size = int(val_ratio * dataset_size)
test_size = dataset_size - train_size - val_size
```

[10]:
```
# Shuffle entire dataset with buffer
dataset = dataset.cache().shuffle(buffer_size=dataset_size, seed=SEED)
```

[11]:
```
# Split into train, validation, and test sets
train_dataset = dataset.take(train_size)
val_test_dataset = dataset.skip(train_size)

val_dataset = val_test_dataset.take(val_size)
test_dataset = val_test_dataset.skip(val_size)
```

[12]:
```
# Data augmentation for training set only
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
    tf.keras.layers.RandomContrast(0.2),
    tf.keras.layers.RandomTranslation(0.1, 0.1),
    tf.keras.layers.Rescaling(1./255)
])
```

[13]:
```
# Apply augmentation to training data only
train_dataset = train_dataset.map(
    lambda x, y: (data_augmentation(x, training=True), y),
    num_parallel_calls=tf.data.AUTOTUNE
)

# Rescale val/test without augmentation
rescale_layer = tf.keras.layers.Rescaling(1./255)

val_dataset = val_dataset.map(lambda x, y: (rescale_layer(x), y),␣
 ↪num_parallel_calls=tf.data.AUTOTUNE)
test_dataset = test_dataset.map(lambda x, y: (rescale_layer(x), y),␣
 ↪num_parallel_calls=tf.data.AUTOTUNE)
```

[14]:
```
# Batch and prefetch all datasets
train_dataset = train_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
val_dataset = val_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
```

[15]:
```
print("Train samples:", tf.data.experimental.cardinality(train_dataset).numpy())
print("Validation samples:", tf.data.experimental.cardinality(val_dataset).
 ↪numpy())
print("Test samples:", tf.data.experimental.cardinality(test_dataset).numpy())
```

```
Train samples: 407
Validation samples: 51
Test samples: 51
```

[50]:
```
# # CNN Model Architecture
# def build_custom_cnn(input_shape, num_classes):
#     model = models.Sequential([
```

```
#         layers.Input(shape=input_shape),

#         # Block 1
#         layers.Conv2D(32, (3, 3), activation='relu', padding='same',
#                       kernel_regularizer=regularizers.l2(0.001)),
#         layers.BatchNormalization(),
#         layers.MaxPooling2D((2, 2)),

#         # Block 2
#         layers.Conv2D(64, (3, 3), activation='relu', padding='same',
#                       kernel_regularizer=regularizers.l2(0.001)),
#         layers.BatchNormalization(),
#         layers.MaxPooling2D((2, 2)),

#         # Block 3
#         layers.Conv2D(128, (3, 3), activation='relu', padding='same',
#                       kernel_regularizer=regularizers.l2(0.001)),
#         layers.BatchNormalization(),
#         layers.MaxPooling2D((2, 2)),
#         layers.Dropout(0.3),

#         # Block 4
#         layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
#         layers.BatchNormalization(),
#         layers.GlobalAveragePooling2D(),
#         layers.Dropout(0.4),

#         # Output
#         layers.Dense(128, activation='relu'),
#         layers.Dropout(0.5),
#         layers.Dense(num_classes, activation='softmax')
#     ])

#     return model
```

[19]:
```
def build_custom_cnn(input_shape, num_classes):
    model = models.Sequential([
        layers.Input(shape=input_shape),

        # Block 1
        layers.Conv2D(32, (5, 5), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),

        # Block 2
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
```

```
        layers.MaxPooling2D((2, 2)),

        # Block 3
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.3),

        # Block 4
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.BatchNormalization(),
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.4),

        # Optional Dense BottleNeck
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),

        # Output
        layers.Dense(num_classes, activation='softmax')
    ])
    return model
```

[20]:

```
# Build the model
model = build_custom_cnn(input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS),
 ↪num_classes=len(class_names))
```

[21]:

```
# Summary (optional)
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      2432

 batch_normalization (BatchN  (None, 256, 256, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 batch_normalization_1 (Batc  (None, 128, 128, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
```

```
 2D)

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      2432

 batch_normalization (BatchN  (None, 256, 256, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 128, 128, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 batch_normalization_1 (Batc  (None, 128, 128, 64)     256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856

 batch_normalization_2 (Batc  (None, 64, 64, 128)      512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 128)      0
 2D)

 dropout (Dropout)           (None, 32, 32, 128)       0

 conv2d_3 (Conv2D)           (None, 32, 32, 256)       295168

 batch_normalization_3 (Batc  (None, 32, 32, 256)      1024
 hNormalization)

 global_average_pooling2d (G  (None, 256)              0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 256)               0

 dense (Dense)               (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 426,058
```

```
Trainable params: 425,098
Non-trainable params: 960
_________________________________________________________________
```

[22]:
```
# Callbacks for early stopping and model checkpointing
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=6,
    verbose=1,
    min_lr=1e-6
)

callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=12,␣
 ↪restore_best_weights=True),
    tf.keras.callbacks.ModelCheckpoint('Best_CNN_Model.h5',␣
 ↪monitor='val_accuracy', save_best_only=True),
    reduce_lr
]
```

[23]:
```
# Compile the model
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

[56]:
```
# Train the model
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    callbacks=callbacks
)
```

```
Epoch 1/100
407/407 [==============================] - 47s 69ms/step - loss: 1.8862 -
accuracy: 0.3316 - val_loss: 1.8342 - val_accuracy: 0.3350 - lr: 0.0010
Epoch 2/100
407/407 [==============================] - 27s 67ms/step - loss: 1.6939 -
accuracy: 0.3886 - val_loss: 1.9115 - val_accuracy: 0.2808 - lr: 0.0010
Epoch 3/100
407/407 [==============================] - 27s 67ms/step - loss: 1.6301 -
accuracy: 0.4126 - val_loss: 1.5277 - val_accuracy: 0.4538 - lr: 0.0010
Epoch 4/100
407/407 [==============================] - 27s 67ms/step - loss: 1.5644 -
accuracy: 0.4412 - val_loss: 1.5848 - val_accuracy: 0.4089 - lr: 0.0010
```

```
Epoch 5/100
407/407 [==============================] - 27s 67ms/step - loss: 1.4975 -
accuracy: 0.4635 - val_loss: 1.4112 - val_accuracy: 0.4741 - lr: 0.0010
Epoch 6/100
407/407 [==============================] - 27s 67ms/step - loss: 1.4139 -
accuracy: 0.4844 - val_loss: 2.0267 - val_accuracy: 0.2956 - lr: 0.0010
Epoch 7/100
407/407 [==============================] - 27s 67ms/step - loss: 1.3549 -
accuracy: 0.5063 - val_loss: 1.3810 - val_accuracy: 0.4982 - lr: 0.0010
Epoch 8/100
407/407 [==============================] - 27s 67ms/step - loss: 1.2812 -
accuracy: 0.5274 - val_loss: 2.2363 - val_accuracy: 0.3319 - lr: 0.0010
Epoch 9/100
407/407 [==============================] - 27s 67ms/step - loss: 1.2072 -
accuracy: 0.5553 - val_loss: 1.1791 - val_accuracy: 0.5683 - lr: 0.0010
Epoch 10/100
407/407 [==============================] - 27s 67ms/step - loss: 1.1677 -
accuracy: 0.5721 - val_loss: 1.3248 - val_accuracy: 0.5154 - lr: 0.0010
Epoch 11/100
407/407 [==============================] - 27s 67ms/step - loss: 1.0894 -
accuracy: 0.6047 - val_loss: 1.6778 - val_accuracy: 0.4735 - lr: 0.0010
Epoch 12/100
407/407 [==============================] - 27s 67ms/step - loss: 1.0438 -
accuracy: 0.6186 - val_loss: 1.0868 - val_accuracy: 0.6102 - lr: 0.0010
Epoch 13/100
407/407 [==============================] - 27s 67ms/step - loss: 1.0090 -
accuracy: 0.6358 - val_loss: 2.0115 - val_accuracy: 0.4286 - lr: 0.0010
Epoch 14/100
407/407 [==============================] - 27s 67ms/step - loss: 0.9632 -
accuracy: 0.6542 - val_loss: 2.7561 - val_accuracy: 0.3374 - lr: 0.0010
Epoch 15/100
407/407 [==============================] - 27s 67ms/step - loss: 0.9493 -
accuracy: 0.6560 - val_loss: 1.0940 - val_accuracy: 0.5948 - lr: 0.0010
Epoch 16/100
407/407 [==============================] - 27s 67ms/step - loss: 0.9138 -
accuracy: 0.6614 - val_loss: 0.9481 - val_accuracy: 0.6410 - lr: 0.0010
Epoch 17/100
407/407 [==============================] - 27s 67ms/step - loss: 0.8811 -
accuracy: 0.6788 - val_loss: 1.3957 - val_accuracy: 0.4883 - lr: 0.0010
Epoch 18/100
407/407 [==============================] - 27s 67ms/step - loss: 0.8537 -
accuracy: 0.6839 - val_loss: 1.3660 - val_accuracy: 0.5413 - lr: 0.0010
Epoch 19/100
407/407 [==============================] - 27s 67ms/step - loss: 0.8442 -
accuracy: 0.6894 - val_loss: 0.8379 - val_accuracy: 0.6872 - lr: 0.0010
Epoch 20/100
407/407 [==============================] - 27s 67ms/step - loss: 0.8079 -
accuracy: 0.7013 - val_loss: 1.3511 - val_accuracy: 0.5844 - lr: 0.0010
```

```
Epoch 21/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7993 -
accuracy: 0.7077 - val_loss: 0.8347 - val_accuracy: 0.6847 - lr: 0.0010
Epoch 22/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7896 -
accuracy: 0.7103 - val_loss: 0.8507 - val_accuracy: 0.6786 - lr: 0.0010
Epoch 23/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7561 -
accuracy: 0.7168 - val_loss: 0.8327 - val_accuracy: 0.6940 - lr: 0.0010
Epoch 24/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7540 -
accuracy: 0.7209 - val_loss: 0.8871 - val_accuracy: 0.6656 - lr: 0.0010
Epoch 25/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7399 -
accuracy: 0.7322 - val_loss: 0.6510 - val_accuracy: 0.7654 - lr: 0.0010
Epoch 26/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7193 -
accuracy: 0.7332 - val_loss: 1.1544 - val_accuracy: 0.6422 - lr: 0.0010
Epoch 27/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7016 -
accuracy: 0.7424 - val_loss: 0.7178 - val_accuracy: 0.7118 - lr: 0.0010
Epoch 28/100
407/407 [==============================] - 27s 67ms/step - loss: 0.7074 -
accuracy: 0.7374 - val_loss: 1.0566 - val_accuracy: 0.6219 - lr: 0.0010
Epoch 29/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6953 -
accuracy: 0.7380 - val_loss: 0.5636 - val_accuracy: 0.7888 - lr: 0.0010
Epoch 30/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6747 -
accuracy: 0.7489 - val_loss: 5.5045 - val_accuracy: 0.3381 - lr: 0.0010
Epoch 31/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6739 -
accuracy: 0.7513 - val_loss: 0.8003 - val_accuracy: 0.6977 - lr: 0.0010
Epoch 32/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6683 -
accuracy: 0.7546 - val_loss: 1.7784 - val_accuracy: 0.6022 - lr: 0.0010
Epoch 33/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6557 -
accuracy: 0.7577 - val_loss: 1.5741 - val_accuracy: 0.5351 - lr: 0.0010
Epoch 34/100
407/407 [==============================] - 27s 67ms/step - loss: 0.6440 -
accuracy: 0.7612 - val_loss: 2.9963 - val_accuracy: 0.5000 - lr: 0.0010
Epoch 35/100
406/407 [============================>.] - ETA: 0s - loss: 0.6377 - accuracy:
0.7613
Epoch 35: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
407/407 [==============================] - 27s 67ms/step - loss: 0.6380 -
accuracy: 0.7613 - val_loss: 1.4964 - val_accuracy: 0.5388 - lr: 0.0010
```

```
Epoch 36/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5908 -
accuracy: 0.7772 - val_loss: 0.5840 - val_accuracy: 0.7765 - lr: 5.0000e-04
Epoch 37/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5792 -
accuracy: 0.7818 - val_loss: 0.5521 - val_accuracy: 0.7876 - lr: 5.0000e-04
Epoch 38/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5722 -
accuracy: 0.7793 - val_loss: 0.5353 - val_accuracy: 0.7956 - lr: 5.0000e-04
Epoch 39/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5921 -
accuracy: 0.7770 - val_loss: 0.6050 - val_accuracy: 0.7752 - lr: 5.0000e-04
Epoch 40/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5764 -
accuracy: 0.7809 - val_loss: 0.4644 - val_accuracy: 0.8153 - lr: 5.0000e-04
Epoch 41/100
407/407 [==============================] - 28s 68ms/step - loss: 0.5582 -
accuracy: 0.7862 - val_loss: 0.5438 - val_accuracy: 0.7851 - lr: 5.0000e-04
Epoch 42/100
407/407 [==============================] - 28s 69ms/step - loss: 0.5583 -
accuracy: 0.7894 - val_loss: 0.4881 - val_accuracy: 0.8122 - lr: 5.0000e-04
Epoch 43/100
407/407 [==============================] - 28s 69ms/step - loss: 0.5481 -
accuracy: 0.7867 - val_loss: 0.6226 - val_accuracy: 0.7728 - lr: 5.0000e-04
Epoch 44/100
407/407 [==============================] - 28s 68ms/step - loss: 0.5504 -
accuracy: 0.7885 - val_loss: 0.6268 - val_accuracy: 0.7525 - lr: 5.0000e-04
Epoch 45/100
407/407 [==============================] - 27s 67ms/step - loss: 0.5509 -
accuracy: 0.7918 - val_loss: 0.4781 - val_accuracy: 0.7925 - lr: 5.0000e-04
Epoch 46/100
406/407 [============================>.] - ETA: 0s - loss: 0.5337 - accuracy:
0.7937
Epoch 46: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
407/407 [==============================] - 28s 68ms/step - loss: 0.5337 -
accuracy: 0.7937 - val_loss: 0.6939 - val_accuracy: 0.7518 - lr: 5.0000e-04
Epoch 47/100
407/407 [==============================] - 28s 68ms/step - loss: 0.5314 -
accuracy: 0.7978 - val_loss: 0.4041 - val_accuracy: 0.8368 - lr: 2.5000e-04
Epoch 48/100
407/407 [==============================] - 28s 69ms/step - loss: 0.5109 -
accuracy: 0.8022 - val_loss: 0.4129 - val_accuracy: 0.8257 - lr: 2.5000e-04
Epoch 49/100
407/407 [==============================] - 28s 68ms/step - loss: 0.5013 -
accuracy: 0.8080 - val_loss: 0.4869 - val_accuracy: 0.8091 - lr: 2.5000e-04
Epoch 50/100
407/407 [==============================] - 28s 69ms/step - loss: 0.5047 -
accuracy: 0.8057 - val_loss: 0.4844 - val_accuracy: 0.8140 - lr: 2.5000e-04
```

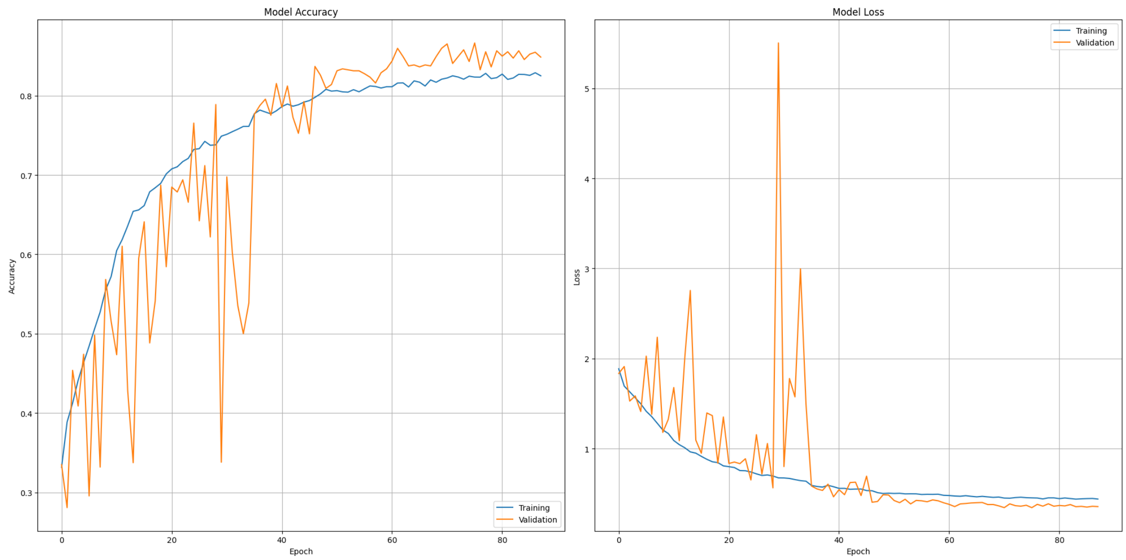
```
Epoch 51/100
407/407 [==============================] - 28s 69ms/step - loss: 0.5017 -
accuracy: 0.8063 - val_loss: 0.4235 - val_accuracy: 0.8313 - lr: 2.5000e-04
Epoch 52/100
407/407 [==============================] - 28s 68ms/step - loss: 0.5036 -
accuracy: 0.8047 - val_loss: 0.3993 - val_accuracy: 0.8337 - lr: 2.5000e-04
Epoch 53/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4972 -
accuracy: 0.8044 - val_loss: 0.4371 - val_accuracy: 0.8325 - lr: 2.5000e-04
Epoch 54/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4981 -
accuracy: 0.8075 - val_loss: 0.3853 - val_accuracy: 0.8313 - lr: 2.5000e-04
Epoch 55/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4975 -
accuracy: 0.8048 - val_loss: 0.4244 - val_accuracy: 0.8313 - lr: 2.5000e-04
Epoch 56/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4901 -
accuracy: 0.8086 - val_loss: 0.4201 - val_accuracy: 0.8276 - lr: 2.5000e-04
Epoch 57/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4922 -
accuracy: 0.8122 - val_loss: 0.4096 - val_accuracy: 0.8233 - lr: 2.5000e-04
Epoch 58/100
407/407 [==============================] - 28s 70ms/step - loss: 0.4916 -
accuracy: 0.8115 - val_loss: 0.4302 - val_accuracy: 0.8159 - lr: 2.5000e-04
Epoch 59/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4930 -
accuracy: 0.8096 - val_loss: 0.4201 - val_accuracy: 0.8288 - lr: 2.5000e-04
Epoch 60/100
406/407 [============================>.] - ETA: 0s - loss: 0.4807 - accuracy:
0.8113
Epoch 60: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
407/407 [==============================] - 28s 69ms/step - loss: 0.4809 -
accuracy: 0.8112 - val_loss: 0.3981 - val_accuracy: 0.8337 - lr: 2.5000e-04
Epoch 61/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4787 -
accuracy: 0.8112 - val_loss: 0.3804 - val_accuracy: 0.8436 - lr: 1.2500e-04
Epoch 62/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4733 -
accuracy: 0.8158 - val_loss: 0.3558 - val_accuracy: 0.8596 - lr: 1.2500e-04
Epoch 63/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4707 -
accuracy: 0.8161 - val_loss: 0.3853 - val_accuracy: 0.8491 - lr: 1.2500e-04
Epoch 64/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4768 -
accuracy: 0.8109 - val_loss: 0.3900 - val_accuracy: 0.8374 - lr: 1.2500e-04
Epoch 65/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4698 -
accuracy: 0.8186 - val_loss: 0.3963 - val_accuracy: 0.8387 - lr: 1.2500e-04
```

```
Epoch 66/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4640 -
accuracy: 0.8169 - val_loss: 0.3990 - val_accuracy: 0.8362 - lr: 1.2500e-04
Epoch 67/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4697 -
accuracy: 0.8121 - val_loss: 0.4015 - val_accuracy: 0.8387 - lr: 1.2500e-04
Epoch 68/100
406/407 [============================>.] - ETA: 0s - loss: 0.4636 - accuracy:
0.8197
Epoch 68: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
407/407 [==============================] - 27s 67ms/step - loss: 0.4637 -
accuracy: 0.8197 - val_loss: 0.3785 - val_accuracy: 0.8374 - lr: 1.2500e-04
Epoch 69/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4583 -
accuracy: 0.8168 - val_loss: 0.3789 - val_accuracy: 0.8491 - lr: 6.2500e-05
Epoch 70/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4617 -
accuracy: 0.8207 - val_loss: 0.3639 - val_accuracy: 0.8596 - lr: 6.2500e-05
Epoch 71/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4504 -
accuracy: 0.8222 - val_loss: 0.3442 - val_accuracy: 0.8651 - lr: 6.2500e-05
Epoch 72/100
407/407 [==============================] - 27s 67ms/step - loss: 0.4488 -
accuracy: 0.8250 - val_loss: 0.3860 - val_accuracy: 0.8405 - lr: 6.2500e-05
Epoch 73/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4560 -
accuracy: 0.8234 - val_loss: 0.3664 - val_accuracy: 0.8491 - lr: 6.2500e-05
Epoch 74/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4600 -
accuracy: 0.8207 - val_loss: 0.3604 - val_accuracy: 0.8578 - lr: 6.2500e-05
Epoch 75/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4548 -
accuracy: 0.8245 - val_loss: 0.3708 - val_accuracy: 0.8430 - lr: 6.2500e-05
Epoch 76/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4529 -
accuracy: 0.8234 - val_loss: 0.3440 - val_accuracy: 0.8664 - lr: 6.2500e-05
Epoch 77/100
407/407 [==============================] - 29s 70ms/step - loss: 0.4511 -
accuracy: 0.8233 - val_loss: 0.3810 - val_accuracy: 0.8325 - lr: 6.2500e-05
Epoch 78/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4410 -
accuracy: 0.8281 - val_loss: 0.3607 - val_accuracy: 0.8553 - lr: 6.2500e-05
Epoch 79/100
407/407 [==============================] - 28s 70ms/step - loss: 0.4529 -
accuracy: 0.8214 - val_loss: 0.3867 - val_accuracy: 0.8362 - lr: 6.2500e-05
Epoch 80/100
407/407 [==============================] - 28s 69ms/step - loss: 0.4528 -
accuracy: 0.8226 - val_loss: 0.3600 - val_accuracy: 0.8565 - lr: 6.2500e-05
```

```
Epoch 81/100
407/407 [==============================] - 28s 70ms/step - loss: 0.4440 -
accuracy: 0.8271 - val_loss: 0.3685 - val_accuracy: 0.8498 - lr: 6.2500e-05
Epoch 82/100
406/407 [============================>.] - ETA: 0s - loss: 0.4514 - accuracy:
0.8205
Epoch 82: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
407/407 [==============================] - 28s 68ms/step - loss: 0.4515 -
accuracy: 0.8204 - val_loss: 0.3636 - val_accuracy: 0.8553 - lr: 6.2500e-05
Epoch 83/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4451 -
accuracy: 0.8222 - val_loss: 0.3781 - val_accuracy: 0.8473 - lr: 3.1250e-05
Epoch 84/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4387 -
accuracy: 0.8268 - val_loss: 0.3546 - val_accuracy: 0.8565 - lr: 3.1250e-05
Epoch 85/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4420 -
accuracy: 0.8267 - val_loss: 0.3585 - val_accuracy: 0.8454 - lr: 3.1250e-05
Epoch 86/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4445 -
accuracy: 0.8255 - val_loss: 0.3504 - val_accuracy: 0.8522 - lr: 3.1250e-05
Epoch 87/100
407/407 [==============================] - 28s 68ms/step - loss: 0.4460 -
accuracy: 0.8288 - val_loss: 0.3592 - val_accuracy: 0.8547 - lr: 3.1250e-05
Epoch 88/100
406/407 [============================>.] - ETA: 0s - loss: 0.4394 - accuracy:
0.8250
Epoch 88: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
407/407 [==============================] - 28s 68ms/step - loss: 0.4396 -
accuracy: 0.8250 - val_loss: 0.3558 - val_accuracy: 0.8485 - lr: 3.1250e-05
```

[57]:
```python
# Create figure and axis objects with a single subplot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Plot training & validation accuracy values
ax1.plot(history.history['accuracy'], label='Training')
ax1.plot(history.history['val_accuracy'], label='Validation')
ax1.set_title('Model Accuracy')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
ax1.legend(loc='lower right')
ax1.grid(True)

# Plot training & validation loss values
ax2.plot(history.history['loss'], label='Training')
ax2.plot(history.history['val_loss'], label='Validation')
ax2.set_title('Model Loss')
```

```
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Loss')
ax2.legend(loc='upper right')
ax2.grid(True)

# Adjust the layout and display the plot
plt.tight_layout()
plt.show()
```



[58]:
```
from sklearn.metrics import classification_report

def evaluate_and_report_model(model, test_dataset, class_names):
    loss, acc = model.evaluate(test_dataset)
    print(f"Test Loss: {loss:.4f}")
    print(f"Test Accuracy: {acc:.4f}")

    y_true = []
    y_pred = []

    for images, labels in test_dataset:
        preds = model.predict(images)
        y_true.extend(labels.numpy())
        y_pred.extend(np.argmax(preds, axis=1))

    y_true = np.array(y_true)
    y_pred = np.array(y_pred)

    print("\n=== Classification Report ===")
```

```
    print(classification_report(y_true, y_pred, target_names=class_names))

evaluate_and_report_model(model, test_dataset, class_names)
```

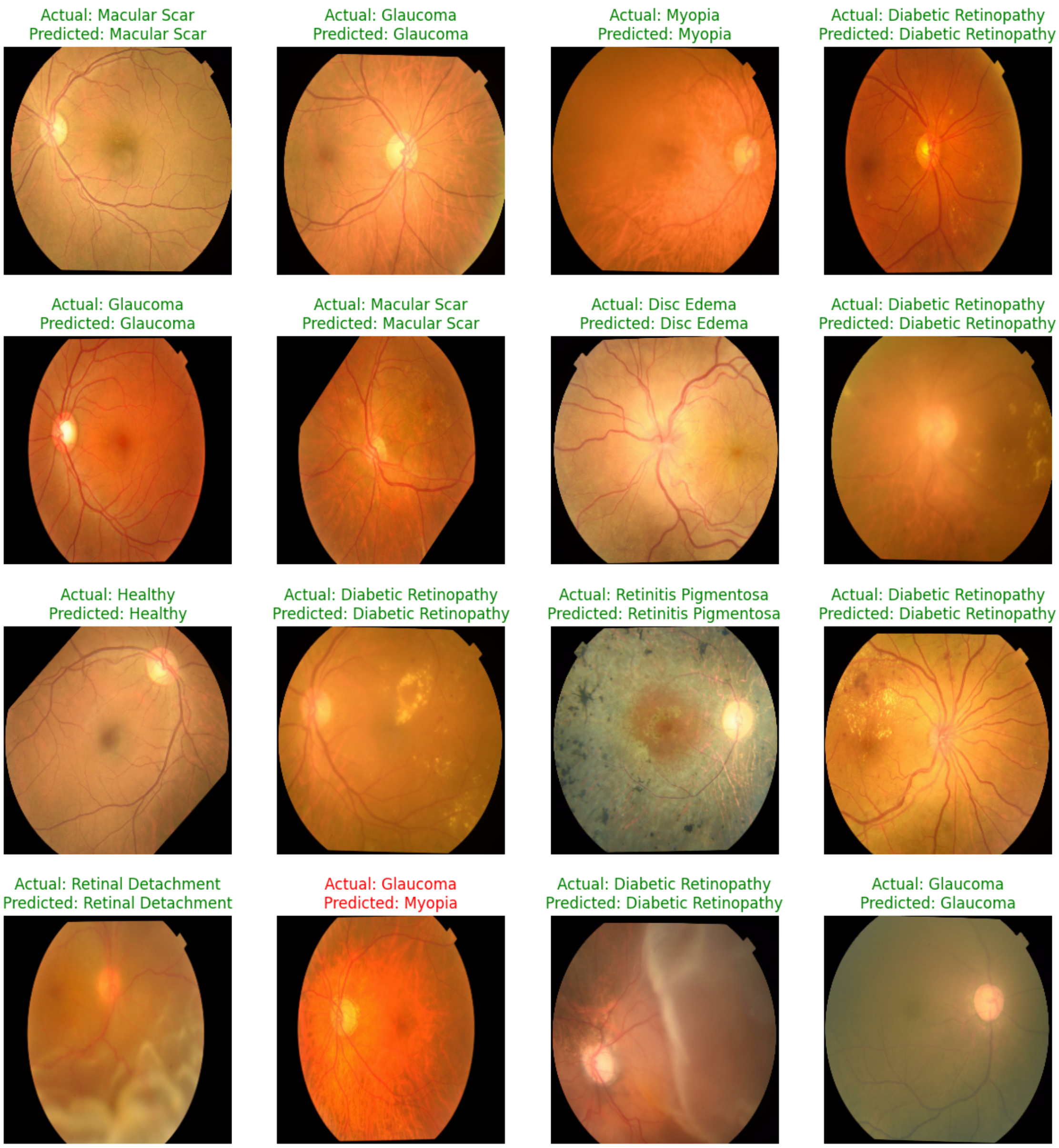
```
51/51 [==============================] - 1s 23ms/step - loss: 0.3806 - accuracy:
0.8418
Test Loss: 0.3806
Test Accuracy: 0.8418
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 71ms/step

=== Classification Report ===
                                  precision    recall  f1-score   support

Central Serous Chorioretinopathy       0.86      0.72      0.78        53
            Diabetic Retinopathy       0.96      0.95      0.96       377
                      Disc Edema       0.96      0.88      0.92        77
                        Glaucoma       0.76      0.77      0.77       291
                         Healthy       0.78      0.86      0.82       286
                    Macular Scar       0.84      0.74      0.79       178
                          Myopia       0.83      0.84      0.84       225
                       Pterygium       1.00      1.00      1.00         9
              Retinal Detachment       0.98      0.98      0.98        49
             Retinitis Pigmentosa       0.92      0.95      0.93        80

                        accuracy                           0.86      1625
                       macro avg       0.89      0.87      0.88      1625
                    weighted avg       0.86      0.86      0.86      1625
```

[30]:
```
# Predictions using the model
for image_batch, labels_batch in test_dataset.take(1):  # Take one batch from␣
 ↪test dataset
    ps = model.predict(image_batch)  # Get model predictions for the batch
    images = (image_batch.numpy() * 255).astype("uint8")  # Convert images to␣
 ↪numpy array and scale to [0, 255]
    labels = labels_batch.numpy()  # Convert labels to numpy array

    plt.figure(figsize=(15, 16))  # Create a figure with specified size
    for i in range(16):  # Loop through first 12 images
        ax = plt.subplot(4, 4, i + 1)  # Create a 4x4 subplot grid
        plt.imshow(images[i])  # Display the image

        # Get prediction confidence
        confidence = ps[i][ps[i].argmax()] * 100
```

```
        # Check if prediction is correct
        is_correct = labels[i] == ps[i].argmax()
        color = "green" if is_correct else "red"

        # Create title with actual label, predicted label, and confidence
        title = f"Actual: {class_names[labels[i]]}\nPredicted: {class_names[ps[i].argmax()]}"
        plt.title(title, color=color)  # Show title with color
        plt.axis("off")  # Hide axes
```

```
1/1 [==============================] - 0s 20ms/step
```



[ ]:
```
# Load the best model
model = keras.models.load_model("Best_CNN_Model.h5")
```

[81]:
```
# Save the model from the best checkpoint
model.save("model/EyeDiseaseClassifier_v1.h5")
```

[ ]:
```
model.save("model/EyeDiseaseClassifier_TF")
```