

```
In [4]: import zipfile
with zipfile.ZipFile('eye-diseases-classification.zip', 'r') as zip_ref:
    zip_ref.extractall('dataset_original')
```

```
In [5]: from PIL import Image
import os

input_root = 'dataset_original/dataset'
output_root = 'output'
resize_size = (256, 256)

os.makedirs(output_root, exist_ok=True)

for class_name in os.listdir(input_root):
    class_path = os.path.join(input_root, class_name)
    if not os.path.isdir(class_path) or class_name == 'masks':
        continue
    output_class_path = os.path.join(output_root, class_name)
    os.makedirs(output_class_path, exist_ok=True)

    for image_name in os.listdir(class_path):
        if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
            input_image_path = os.path.join(class_path, image_name)
            output_image_name = os.path.splitext(image_name)[0] + '.jpg'
            output_image_path = os.path.join(output_class_path, output_image_name)
            try:
                with Image.open(input_image_path) as img:
                    img = img.convert('RGB')
                    img = img.resize(resize_size)
                    img.save(output_image_path, 'JPEG')
            except Exception as e:
                print(f"خطأ في معالجة {input_image_path}: {e}")
```

```
In [6]: import os, shutil
from sklearn.model_selection import train_test_split

# إذا لم تطبقه 'dataset_original/dataset' إذا طبقت التصغير، أو 'output' استخدم
original_dataset_dir = 'output' # غير المسار حسب ما استخدمت
base_dir = 'new_dataset'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

for dir_path in [train_dir, validation_dir, test_dir]:
    os.makedirs(dir_path, exist_ok=True)

train_ratio, val_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir)
                   if os.path.isfile(os.path.join(class_dir, f))]
    train_imgs, temp_imgs = train_test_split(all_images, test_size=(1 - train_ratio))
    val_imgs, test_imgs = train_test_split(temp_imgs,
                                           test_size=test_ratio/(val_ratio + test_r
```

```

                                random_state=42)
    for imgs, folder in zip([train_imgs, val_imgs, test_imgs], ['train', 'validation']):
        dest_folder = os.path.join(dest_base_dir, folder, os.path.basename(class_dir))
        os.makedirs(dest_folder, exist_ok=True)
        for file in imgs:
            shutil.copy(file, dest_folder)

# التطبيق على كل فئة
for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy(full_class_dir, base_dir)

```

```

In [ ]: import os, shutil
        from sklearn.model_selection import train_test_split

original_dataset_dir = 'output'
base_dir = 'new_dataset'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

for dir_path in [train_dir, validation_dir, test_dir]:
    os.makedirs(dir_path, exist_ok=True)

train_ratio, val_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir)
                  if os.path.isfile(os.path.join(class_dir, f))]
    train_imgs, temp_imgs = train_test_split(all_images, test_size=(1-train_ratio),
                                             random_state=42)
    val_imgs, test_imgs = train_test_split(temp_imgs,
                                           test_size=test_ratio/(val_ratio+test_ratio),
                                           random_state=42)
    for imgs, folder in zip([train_imgs, val_imgs, test_imgs], ['train', 'validation']):
        dest_folder = os.path.join(dest_base_dir, folder, os.path.basename(class_dir))
        os.makedirs(dest_folder, exist_ok=True)
        for file in imgs:
            shutil.copy(file, dest_folder)

# تطبيق التقسيم
for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy(full_class_dir, base_dir)

```

```

In [7]: from tensorflow.keras.preprocessing.image import ImageDataGenerator

batch_size = 32
img_size = (256, 256)

# مولد التدريب مع التحسينات
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,

```

```

horizontal_flip=True,
vertical_flip=True,
width_shift_range=0.1,
height_shift_range=0.1,
brightness_range=[0.8, 1.2],
zoom_range=0.2
)

# مولدات التحقق والاختبار (بدون تحسينات)
val_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'new_dataset/train',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)

validation_generator = val_test_datagen.flow_from_directory(
    'new_dataset/validation',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

test_generator = val_test_datagen.flow_from_directory(
    'new_dataset/test',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

```

Found 2949 images belonging to 4 classes.

Found 422 images belonging to 4 classes.

Found 846 images belonging to 4 classes.

```

In [10]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,

def build_simple_cnn(input_shape, num_classes):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

```

```

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

# استخدام الدالة
input_shape = (256, 256, 3)
num_classes = 4 # cataract, diabetic_retinopathy, glaucoma, normal
model = build_simple_cnn(input_shape, num_classes)
model.summary()

```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_7 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_8 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_8 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten_2 (Flatten)	(None, 115200)	0
dense_4 (Dense)	(None, 256)	29,491,456
dropout_2 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 4)	1,028

Total params: 29,585,732 (112.86 MB)

Trainable params: 29,585,732 (112.86 MB)

Non-trainable params: 0 (0.00 B)

```

In [11]: history = model.fit(
            train_generator,
            epochs=15,
            validation_data=validation_generator
        )

```

```

Epoch 1/15
93/93 ————— 207s 2s/step - accuracy: 0.3900 - loss: 1.3436 - val_accu
racy: 0.4431 - val_loss: 1.1612
Epoch 2/15
93/93 ————— 218s 2s/step - accuracy: 0.4564 - loss: 1.1865 - val_accu
racy: 0.4882 - val_loss: 1.1146
Epoch 3/15
93/93 ————— 269s 3s/step - accuracy: 0.5053 - loss: 1.0968 - val_accu
racy: 0.4739 - val_loss: 1.1179
Epoch 4/15
93/93 ————— 291s 3s/step - accuracy: 0.5171 - loss: 1.0836 - val_accu
racy: 0.5261 - val_loss: 1.0693
Epoch 5/15
93/93 ————— 248s 3s/step - accuracy: 0.5029 - loss: 1.0788 - val_accu
racy: 0.4692 - val_loss: 1.1366
Epoch 6/15
93/93 ————— 242s 3s/step - accuracy: 0.5453 - loss: 1.0372 - val_accu
racy: 0.5237 - val_loss: 1.0681
Epoch 7/15
93/93 ————— 221s 2s/step - accuracy: 0.5388 - loss: 1.0146 - val_accu
racy: 0.5118 - val_loss: 1.0580
Epoch 8/15
93/93 ————— 214s 2s/step - accuracy: 0.5527 - loss: 1.0206 - val_accu
racy: 0.4905 - val_loss: 1.1327
Epoch 9/15
93/93 ————— 208s 2s/step - accuracy: 0.5510 - loss: 1.0306 - val_accu
racy: 0.5450 - val_loss: 1.0063
Epoch 10/15
93/93 ————— 210s 2s/step - accuracy: 0.5514 - loss: 1.0130 - val_accu
racy: 0.5190 - val_loss: 1.0321
Epoch 11/15
93/93 ————— 257s 2s/step - accuracy: 0.5544 - loss: 1.0077 - val_accu
racy: 0.5047 - val_loss: 1.0616
Epoch 12/15
93/93 ————— 210s 2s/step - accuracy: 0.5707 - loss: 0.9767 - val_accu
racy: 0.5664 - val_loss: 0.9538
Epoch 13/15
93/93 ————— 208s 2s/step - accuracy: 0.5856 - loss: 0.9524 - val_accu
racy: 0.5545 - val_loss: 0.9792
Epoch 14/15
93/93 ————— 266s 2s/step - accuracy: 0.5934 - loss: 0.9324 - val_accu
racy: 0.6019 - val_loss: 0.9244
Epoch 15/15
93/93 ————— 274s 2s/step - accuracy: 0.6056 - loss: 0.9144 - val_accu
racy: 0.5664 - val_loss: 0.9473

```

```

In [12]: from tensorflow.keras.applications import InceptionV3
         from tensorflow.keras.layers import GlobalAveragePooling2D
         from tensorflow.keras.models import Model

         base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(256,256,3))
         base_model.trainable = False # تجميد الطبقات الأساسية

         x = base_model.output
         x = GlobalAveragePooling2D()(x)
         x = Dense(128, activation='relu')(x)

```

```

x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968  **13s** 0us/step

```

In [13]: history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

```

Epoch 1/10

93/93  **339s** 4s/step - accuracy: 0.5914 - loss: 0.9516 - val_accuracy: 0.6730 - val_loss: 0.7872

Epoch 2/10

93/93  **339s** 4s/step - accuracy: 0.6680 - loss: 0.7892 - val_accuracy: 0.6967 - val_loss: 0.7525

Epoch 3/10

93/93  **338s** 4s/step - accuracy: 0.7063 - loss: 0.7369 - val_accuracy: 0.6991 - val_loss: 0.7498

Epoch 4/10

93/93  **377s** 4s/step - accuracy: 0.6952 - loss: 0.7312 - val_accuracy: 0.7346 - val_loss: 0.6881

Epoch 5/10

93/93  **375s** 4s/step - accuracy: 0.7111 - loss: 0.7061 - val_accuracy: 0.6848 - val_loss: 0.7419

Epoch 6/10

93/93  **363s** 3s/step - accuracy: 0.6985 - loss: 0.7144 - val_accuracy: 0.7014 - val_loss: 0.6743

Epoch 7/10

93/93  **325s** 3s/step - accuracy: 0.7057 - loss: 0.6875 - val_accuracy: 0.7488 - val_loss: 0.6691

Epoch 8/10

93/93  **330s** 4s/step - accuracy: 0.7152 - loss: 0.6737 - val_accuracy: 0.7180 - val_loss: 0.6681

Epoch 9/10

93/93  **330s** 4s/step - accuracy: 0.7257 - loss: 0.6684 - val_accuracy: 0.7251 - val_loss: 0.6841

Epoch 10/10

93/93  **331s** 4s/step - accuracy: 0.7152 - loss: 0.6878 - val_accuracy: 0.7251 - val_loss: 0.6529

```

In [14]: import numpy as np
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    import matplotlib.pyplot as plt
    import seaborn as sns

```

توقع الفئات

```

test_generator.reset()
predictions = model.predict(test_generator, steps=len(test_generator))
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes

```

```

class_labels = list(test_generator.class_indices.keys())

# الدقة الكلية
accuracy = accuracy_score(true_classes, predicted_classes)
print(f"Test Accuracy: {accuracy:.4f}")

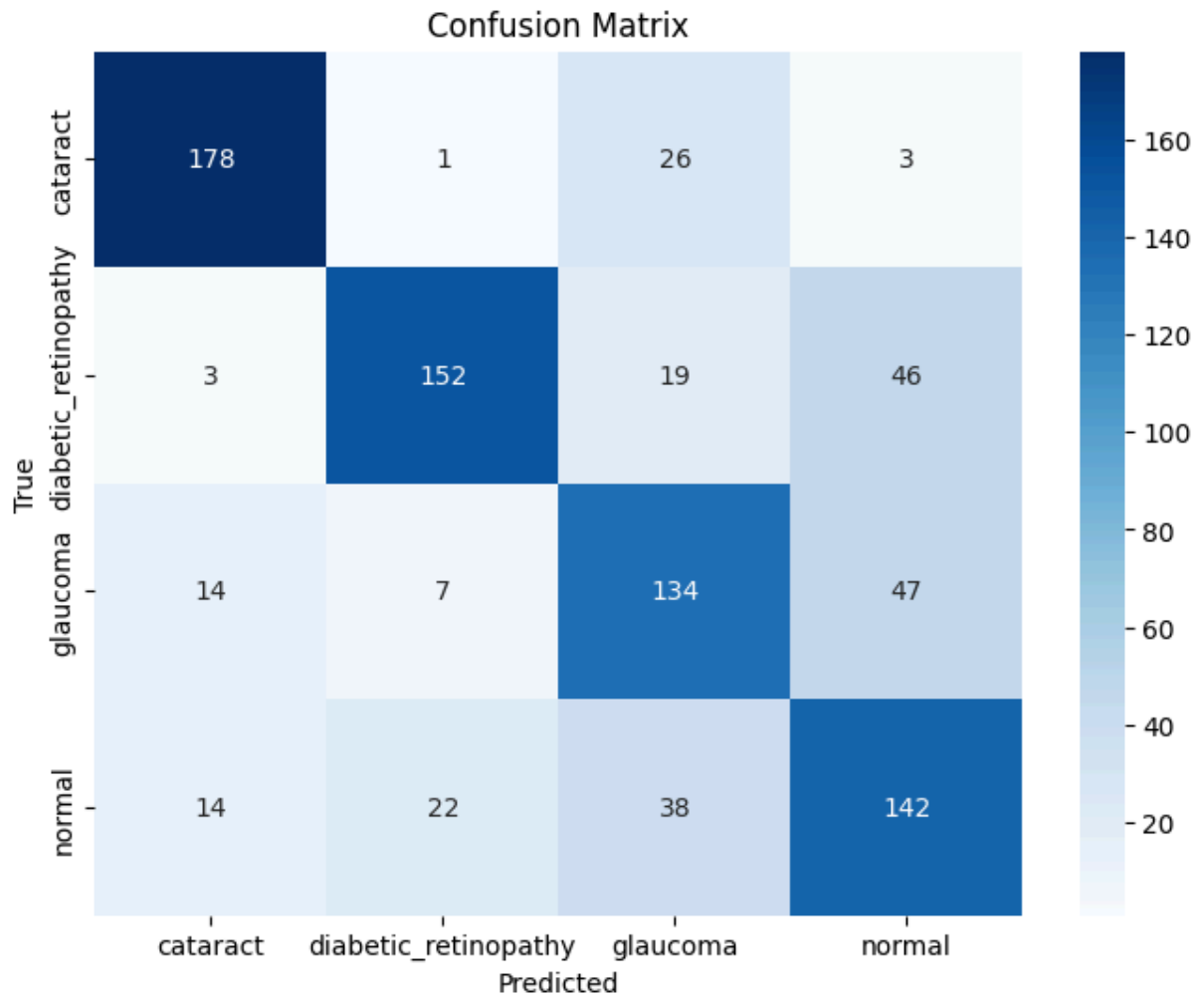
# مصفوفة الارتباك
cm = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# تقرير التصنيف (لكل فئة precision, recall, f1-score)
print(classification_report(true_classes, predicted_classes, target_names=class_labels))

```

27/27 ————— 71s 3s/step

Test Accuracy: 0.7163



	precision	recall	f1-score	support
cataract	0.85	0.86	0.85	208
diabetic_retinopathy	0.84	0.69	0.76	220
glaucoma	0.62	0.66	0.64	202
normal	0.60	0.66	0.63	216
accuracy			0.72	846
macro avg	0.73	0.72	0.72	846
weighted avg	0.73	0.72	0.72	846

```
In [18]: from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np

# 1. إعادة تعيين مولد الاختبار (للتأكد من البداية الصحيحة)
test_generator.reset()

# 2. التنبؤ بالاحتمالات لجميع الصور في مجموعة الاختبار
predictions = model.predict(test_generator, steps=len(test_generator))

# 3. الحصول على الفئات الحقيقية
y_true = test_generator.classes
y_scores = predictions
# الفئات كأرقام (0,1,2,3)
# عدد الصور × 4 (shape: 4 × عدد الصور)

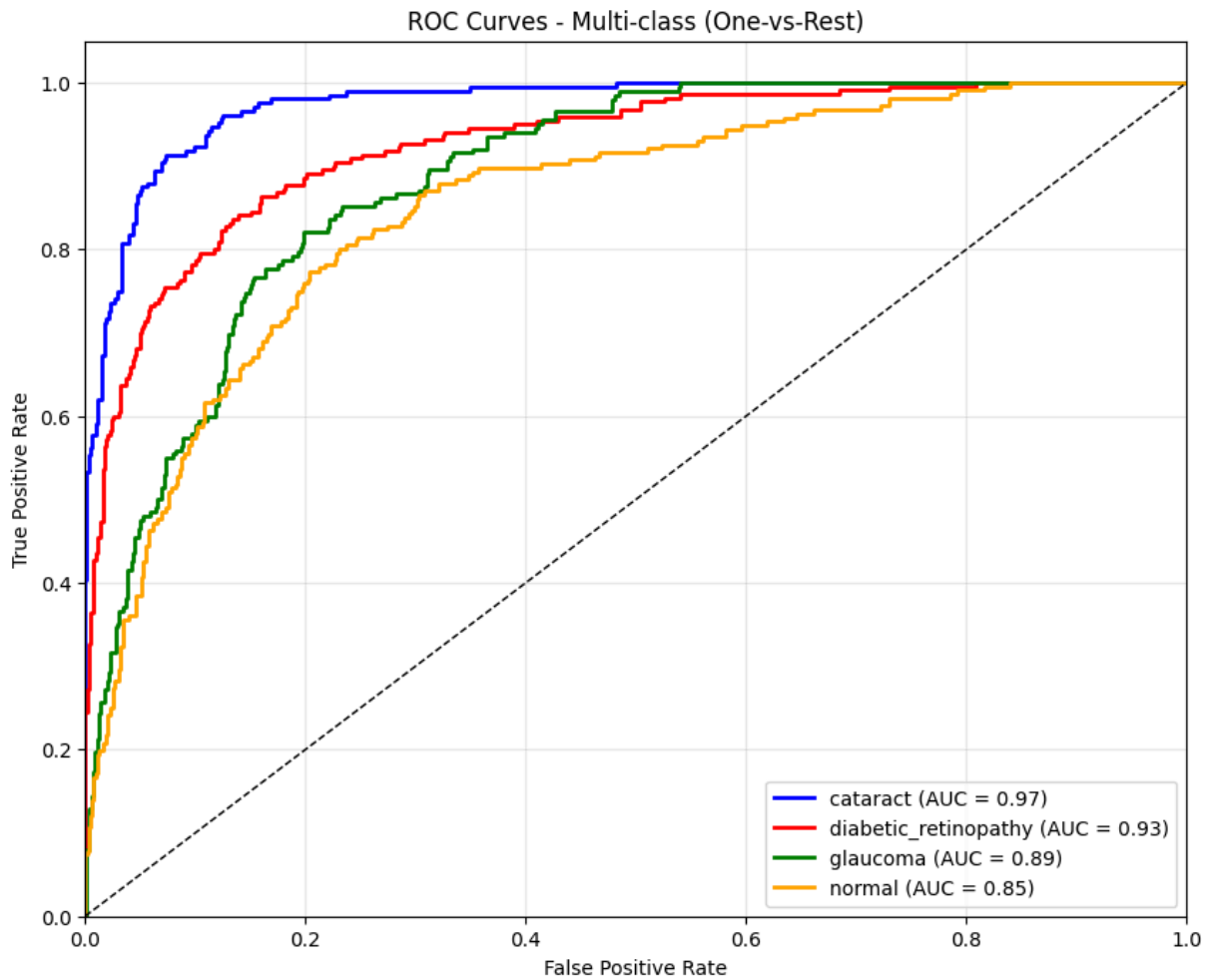
# 4. ROC لتناسب (One-hot encoding) تحويل الفئات إلى صيغة ثنائية
y_true_bin = label_binarize(y_true, classes=[0, 1, 2, 3])
n_classes = y_true_bin.shape[1]

# 5. لكل فئة ROC رسم منحنى
plt.figure(figsize=(10, 8))
colors = ['blue', 'red', 'green', 'orange']
class_labels = list(test_generator.class_indices.keys()) # أسماء الفئات من المولد

for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_true_bin[:, i], y_scores[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color=colors[i], lw=2,
             label=f'{class_labels[i]} (AUC = {roc_auc:.2f})')

# تمثيل النموذج العشوائي diagonal خط
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves - Multi-class (One-vs-Rest)')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.show()
```

27/27 ————— 76s 3s/step



```
In [20]: import matplotlib.pyplot as plt

# model.fit يحتوي على سجل التدريب بعد تنفيذ history افترض أن متغير
# history = model.fit(...)

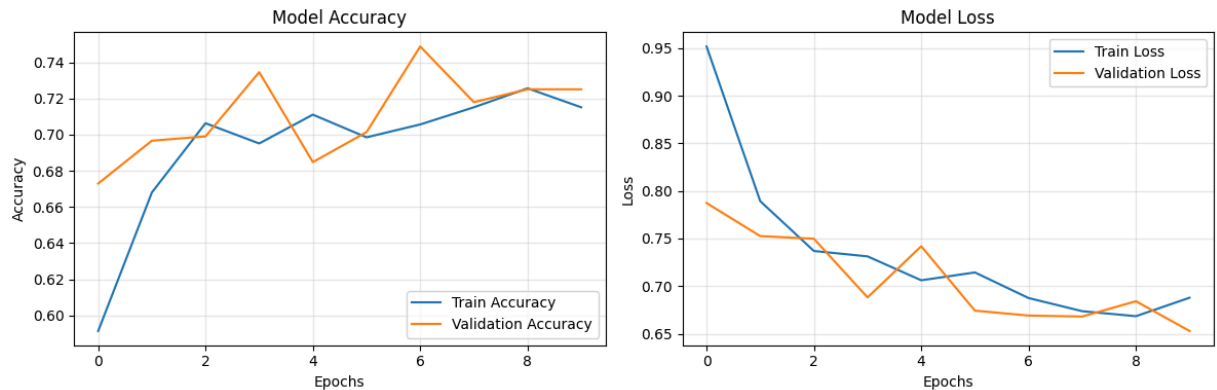
# رسم دقة التدريب والتحقق
plt.figure(figsize=(12, 4))

# منحنى الدقة
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(alpha=0.3)

# منحنى الخسارة
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.grid(alpha=0.3)

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



```
In [22]: import os
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_
import numpy as np

# البحث عن أول صورة في مجلد التدريب (أي فئة)
train_folder = 'new_dataset/train'
found_image = None
for class_name in os.listdir(train_folder):
    class_path = os.path.join(train_folder, class_name)
    if os.path.isdir(class_path):
        for file in os.listdir(class_path):
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                found_image = os.path.join(class_path, file)
                break
        if found_image:
            break

if found_image is None:
    raise FileNotFoundError("لم يتم العثور على أي صورة في مجلد التدريب")

print(f"تم العثور على الصورة: {found_image}")

# تحميل الصورة
img = load_img(found_image, target_size=(256, 256))
img_array = img_to_array(img)
img_array = img_array.reshape((1,) + img_array.shape)

# إنشاء مولد البيانات بنفس التحسينات
datagen = ImageDataGenerator(
    rotation_range=40,
    horizontal_flip=True,
    vertical_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=[0.8, 1.2],
    zoom_range=0.2
```

```

)

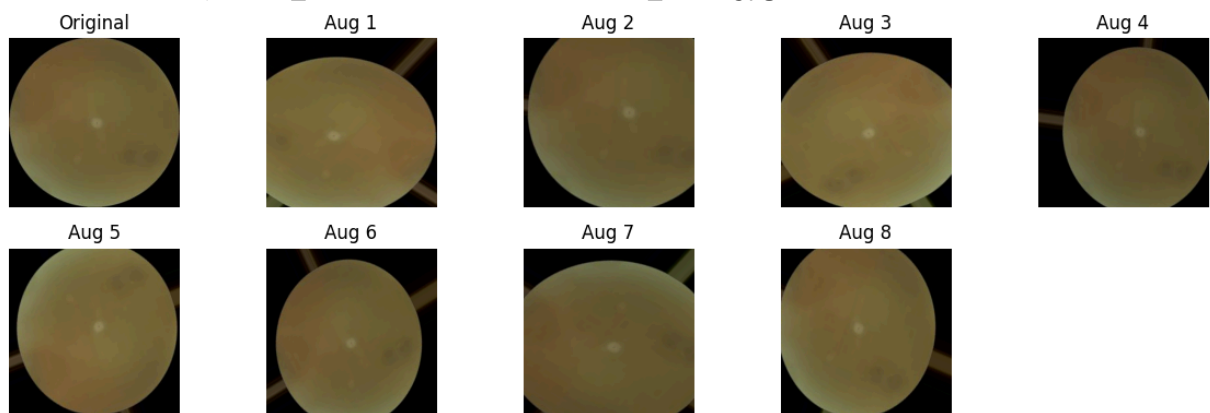
# عرض الصورة الأصلية و 8 صور محسنة
plt.figure(figsize=(12, 4))
plt.subplot(2, 5, 1)
plt.imshow(img)
plt.title('Original')
plt.axis('off')

for i, batch in enumerate(datagen.flow(img_array, batch_size=1)):
    if i >= 8:
        break
    plt.subplot(2, 5, i+2)
    aug_img = batch[0].astype('uint8')
    plt.imshow(aug_img)
    plt.title(f'Aug {i+1}')
    plt.axis('off')

plt.tight_layout()
plt.show()

```

تم العثور على الصورة new_dataset/train\cataract\0_left.jpg



In []: