

```
# ربط كاتل في قوغل كولاب

import kagglehub

# Download latest version
path = kagglehub.dataset_download("andrewmvd/lung-and-colon-cancer-histopathological-images")

print("Path to dataset files:", path)
```

Using Colab cache for faster access to the 'lung-and-colon-cancer-histopathological-images' dataset.
Path to dataset files: /kaggle/input/lung-and-colon-cancer-histopathological-images

```
# استيراد المكتبات الأساسية

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

# الموجودة في الملف augmentation مكتبات الـ
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
```

```
# المسار الرئيسي للداتا سيت
dataset_path = "/kaggle/input/lung-and-colon-cancer-histopathological-images/lung_colon_image_set"

# اختيار صورة واحدة للتجربة
img_path = os.path.join(
    dataset_path,
    "colon_image_sets",
    "colon_aca",
    "colonca1.jpeg"
)

print("مسار الصورة:", img_path)
print("هل الصورة موجودة", os.path.exists(img_path))
```

مسار الصورة: /kaggle/input/lung-and-colon-cancer-histopathological-images/lung_colon_image_set/colon_image_sets/colon_aca/colonca1.jpeg
هل الصورة موجودة True

```
# قراءة الصورة باستخدام
image = cv2.imread(img_path)

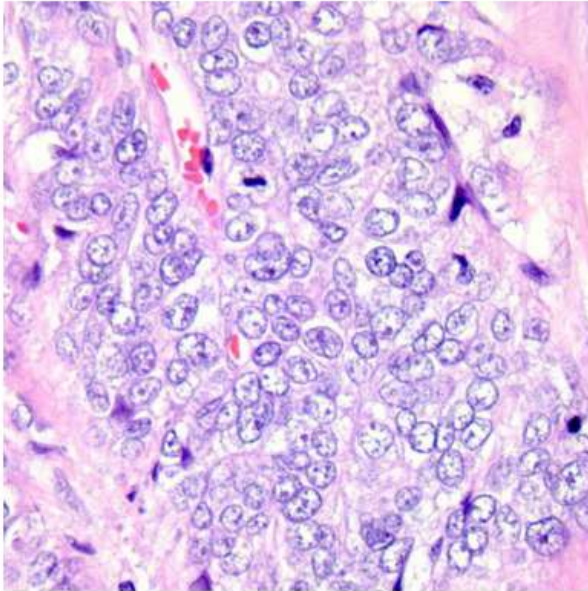
# التأكد أن الصورة انقراأت بشكل صحيح
if image is None:
    raise ValueError("تعذر قراءة الصورة تأكد من المسار")

# تحويل الصورة من BGR إلى RGB

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# عرض الصورة
plt.figure(figsize=(5, 5))
plt.imshow(image_rgb)
plt.title("Loaded Image")
plt.axis('off')
plt.show()
```

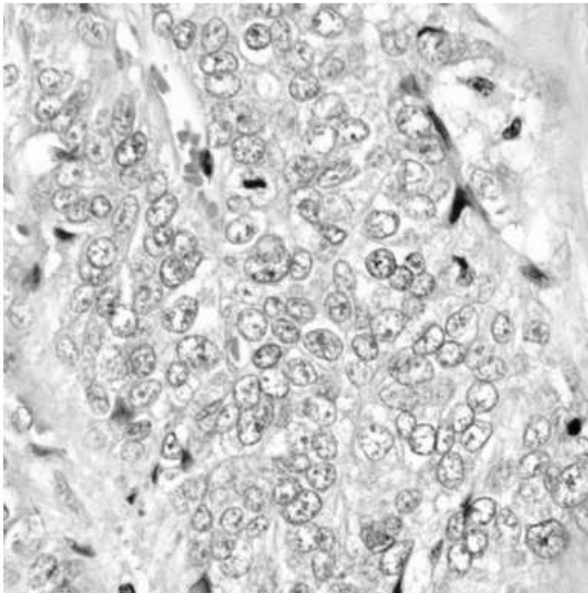
Loaded Image



```
# تحويل الصورة إلى Grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# عرض الصورة
plt.figure(figsize=(5, 5))
plt.imshow(gray_image, cmap="gray")
plt.title("Grayscale Image")
plt.axis("off")
plt.show()
```

Grayscale Image



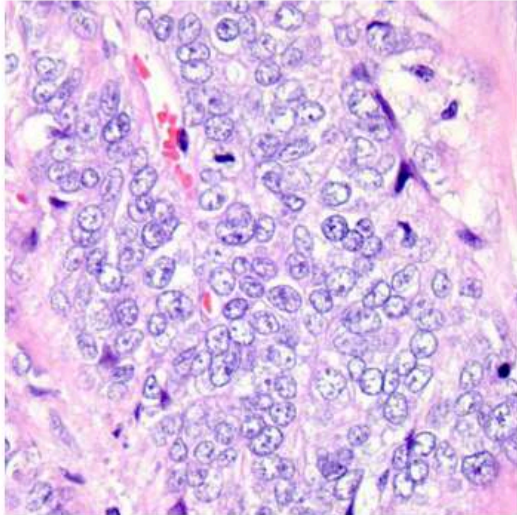
```
# مقارنة بين الصورة الأصلية و Grayscale
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.imshow(image_rgb)
plt.title("Original")
plt.axis("off")

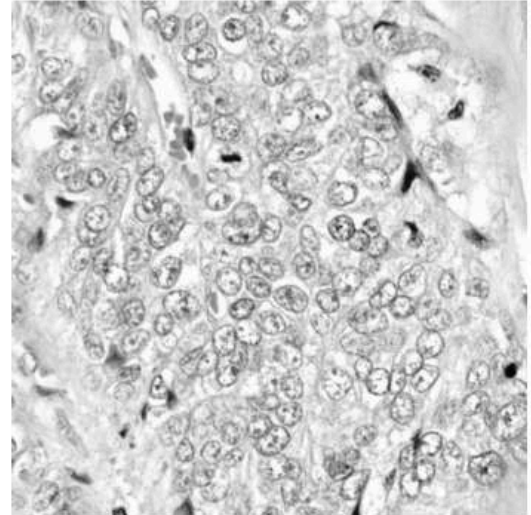
plt.subplot(1, 2, 2)
plt.imshow(gray_image, cmap="gray")
plt.title("Grayscale")
plt.axis("off")
```

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

Original



Grayscale



```
# قراءة الصورة
image_color = cv2.imread(img_path, cv2.IMREAD_COLOR)

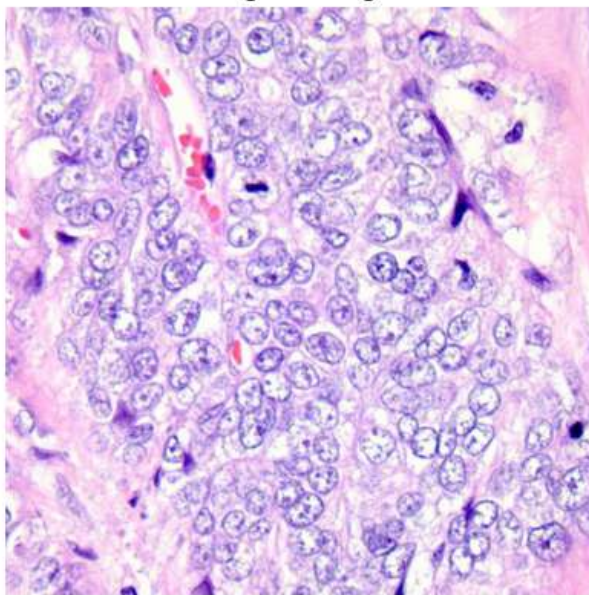
# عرض الصورة
image_color_rgb = cv2.cvtColor(image_color, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(5, 5))
plt.imshow(image_color_rgb)
plt.title("Original Image")
plt.axis("off")
plt.show()

# حفظ الصورة في ملف جديد
saved_path = "/kaggle/working/saved_image.jpg"
cv2.imwrite(saved_path, image_color)

print("تم حفظ الصورة في:", saved_path)
```

Original Image



تم حفظ الصورة في: /kaggle/working/saved_image.jpg

```
# استخراج أبعاد الصورة
height, width, channels = image.shape
```

```
print(f"Width: {width}, Height: {height}, Channels: {channels}")
```

```
Width: 768, Height: 768, Channels: 3
```

```
# دالة لعرض الصورة الأصلية مع صور augmented
def display_augmented_images(img_path, datagen, num_augmented=3):

    # ImageDataGenerator عرض الصورة الأصلية مع عدة نسخ معدلة باستخدام

    # التحقق من وجود الصورة
    if not os.path.exists(img_path):
        raise FileNotFoundError(f"الصورة غير موجودة: {img_path}")

    # تحميل الصورة وتحويلها إلى مصفوفة
    img = load_img(img_path)
    img_array = img_to_array(img)

    # إضافة بعد إضافي
    img_array = np.expand_dims(img_array, axis=0)

    # توليد الصور المعدلة
    augmented_images = datagen.flow(img_array, batch_size=1)

    # إنشاء نافذة عرض
    plt.figure(figsize=(4 * (num_augmented + 1), 5))

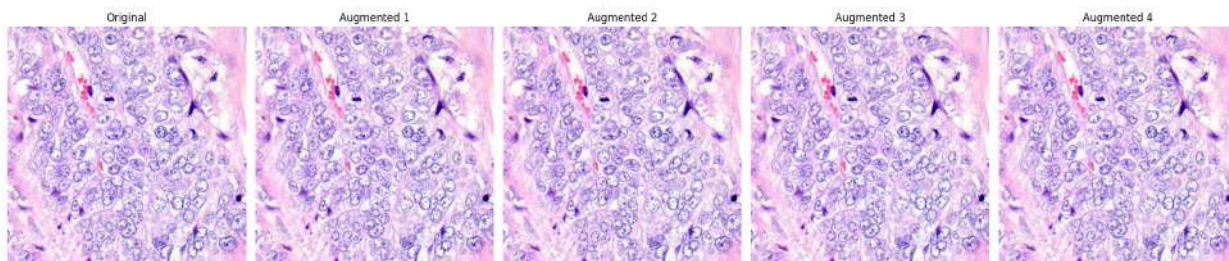
    # عرض الصورة الأصلية
    plt.subplot(1, num_augmented + 1, 1)
    plt.title("Original")
    plt.imshow(img)
    plt.axis("off")

    # عرض الصور المعدلة
    for i in range(num_augmented):
        batch = next(augmented_images)
        aug_img = batch[0].astype("uint8")

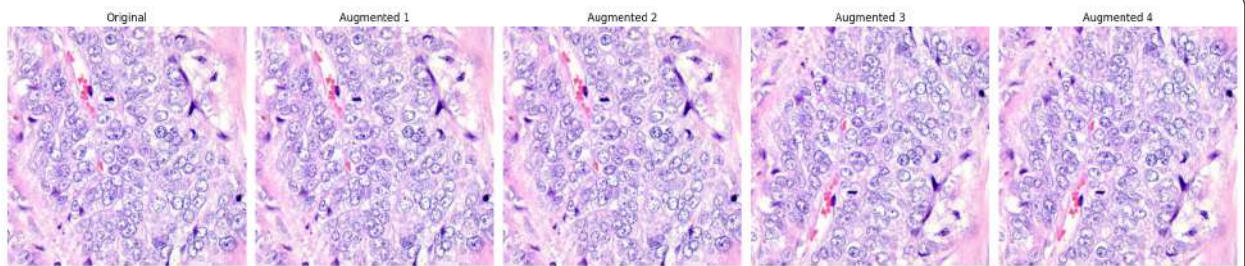
        plt.subplot(1, num_augmented + 1, i + 2)
        plt.title(f"Augmented {i+1}")
        plt.imshow(aug_img)
        plt.axis("off")

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

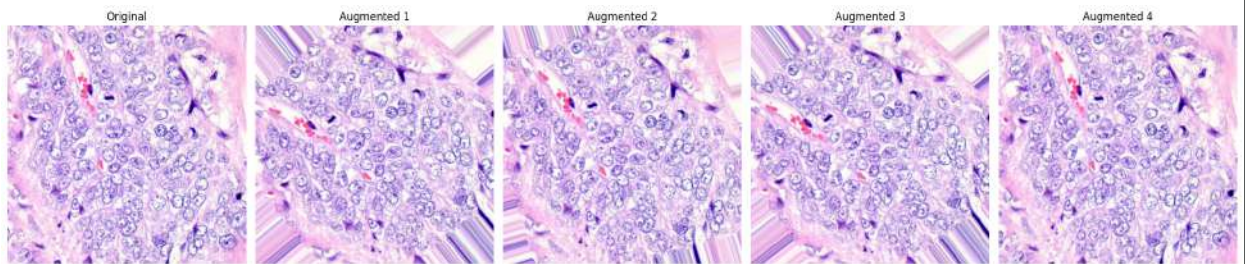
```
# تجربة العرض الأفقي
datagen = ImageDataGenerator(horizontal_flip=True)
display_augmented_images(img_path, datagen, num_augmented=4)
```



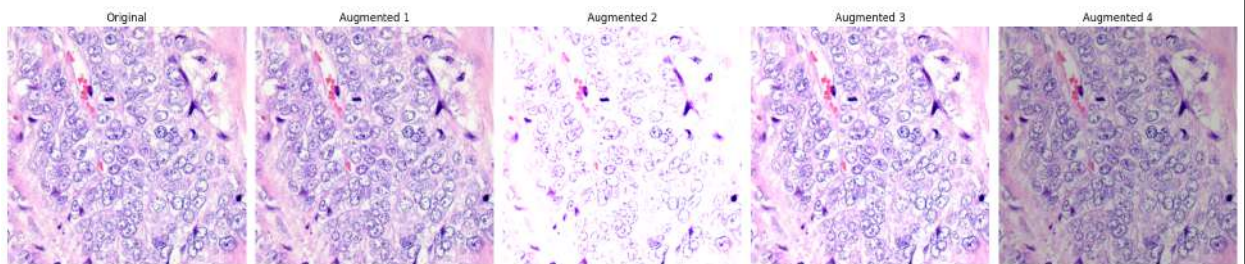
```
# تجربة العرض العمودي
datagen = ImageDataGenerator(vertical_flip=True)
display_augmented_images(img_path, datagen, num_augmented=4)
```



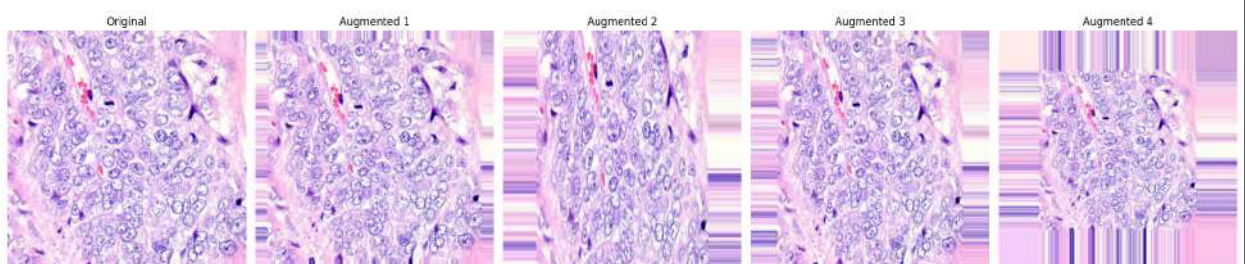
```
# تجربة التدوير العشوائي
datagen = ImageDataGenerator(rotation_range=40)
display_augmented_images(img_path, datagen, num_augmented=4)
```



```
# تجربة تغيير الإضاءة
datagen = ImageDataGenerator(brightness_range=[0.5, 1.5])
display_augmented_images(img_path, datagen, num_augmented=4)
```



```
# تجربة الزوم العشوائي
datagen = ImageDataGenerator(zoom_range=0.6)
display_augmented_images(img_path, datagen, num_augmented=4)
```



```
# مكتبات التقسيم والنسخ
import shutil
from sklearn.model_selection import train_test_split

# جمع كل الفئات في مكان واحد
# يحتوي جميع الكلاسات تحت مجلد واحد working سننشئ مجلد جديد داخل
source_base = dataset_path
combined_dir = "/kaggle/working/all_classes"

os.makedirs(combined_dir, exist_ok=True)

# أسماء المجلدات الرئيسية داخل الداتا سيت
top_level_dirs = ["colon_image_sets", "lung_image_sets"]
```

```
# ربط الفئات إلى مجلد موحد
for top_dir in top_level_dirs:
    top_path = os.path.join(source_base, top_dir)

    for class_name in os.listdir(top_path):
        class_path = os.path.join(top_path, class_name)

        if os.path.isdir(class_path):
            target_class_path = os.path.join(combined_dir, class_name)
            os.makedirs(target_class_path, exist_ok=True)

            for file_name in os.listdir(class_path):
                src_file = os.path.join(class_path, file_name)
                dst_file = os.path.join(target_class_path, file_name)

                # نسخ الملفات إلى المجلد الموحد
                if os.path.isfile(src_file) and not os.path.exists(dst_file):
                    shutil.copy2(src_file, dst_file)

print("تم تجهيز مجلد موحد للفئات في", combined_dir)
print("الفئات الموجودة:", os.listdir(combined_dir))
```

تم تجهيز مجلد موحد للفئات في /kaggle/working/all_classes
الفئات الموجودة: ['lung_scc', 'lung_aca', 'colon_n', 'lung_n', 'colon_aca']

```
# إنشاء مجلدات التقسيم
original_dataset_dir = combined_dir
base_dir = "/kaggle/working/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 folder in [train_dir, validation_dir, test_dir]:
    os.makedirs(folder, exist_ok=True)

# نسب التقسيم
train_ratio = 0.7
validation_ratio = 0.1
test_ratio = 0.2

print("تم إنشاء مجلدات التقسيم")
```

تم إنشاء مجلدات التقسيم

```
# تقسيم الصور لكل فئة
def split_and_copy_files(class_dir, dest_base_dir):
    # جمع جميع الصور داخل الفئة
    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))
    ]

    # تقسيم أولي: train و temp
    train_images, temp_images = train_test_split(
        all_images,
        test_size=(1 - train_ratio),
        random_state=42
    )

    # تقسيم temp إلى validation و test
    validation_images, test_images = train_test_split(
        temp_images,
        test_size=test_ratio / (validation_ratio + test_ratio),
        random_state=42
    )

    # نسخ الصور إلى مجلداتها
    for images, folder_name in zip(
        [train_images, validation_images, test_images],
```

```

        ["train", "validation", "test"]
    ):
        dest_dir = os.path.join(dest_base_dir, folder_name, os.path.basename(class_dir))
        os.makedirs(dest_dir, exist_ok=True)

        for file in images:
            shutil.copy2(file, dest_dir)

```

```

# تطبيق التقسيم على جميع الفئات
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_files(full_class_dir, base_dir)

print("تم تقسيم الداتا بنجاح")

```

تم تقسيم الداتا بنجاح

```

# التحقق من عدد الصور في كل جزء
for split_name in ["train", "validation", "test"]:
    split_path = os.path.join(base_dir, split_name)
    print(f"\n===== {split_name.upper()} =====")

    for class_name in os.listdir(split_path):
        class_path = os.path.join(split_path, class_name)
        if os.path.isdir(class_path):
            print(class_name, ":", len(os.listdir(class_path)))

```

```

===== TRAIN =====
lung_scc : 3499
lung_aca : 3499
colon_n : 3499
lung_n : 3499
colon_aca : 3499

===== VALIDATION =====
lung_scc : 500
lung_aca : 500
colon_n : 500
lung_n : 500
colon_aca : 500

===== TEST =====
lung_scc : 1001
lung_aca : 1001
colon_n : 1001
lung_n : 1001
colon_aca : 1001

```

```

# استيراد المكتبات

import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D,
    MaxPooling2D,

```

```

        Flatten,
        Dense,
        Dropout,
        GlobalAveragePooling2D
    )
from tensorflow.keras.applications import ResNet50, EfficientNetB0, InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

تحديد مسارات البيانات بعد التقسيم

```

train_dir = "/kaggle/working/dataset/train"
val_dir = "/kaggle/working/dataset/validation"
test_dir = "/kaggle/working/dataset/test"

```

```

print("Train path:", train_dir)
print("Validation path:", val_dir)
print("Test path:", test_dir)

```

```

print("\nClasses in train folder:")
print(os.listdir(train_dir))

```

```

Train path: /kaggle/working/dataset/train
Validation path: /kaggle/working/dataset/validation
Test path: /kaggle/working/dataset/test

```

```

Classes in train folder:
['lung_scc', 'lung_aca', 'colon_n', 'lung_n', 'colon_aca']

```

دالة تجهيز الـ Data Generators

```
def create_data_generators(train_dir, val_dir, test_dir, img_size=(240, 240), batch_size=64):
```

```

    # Augmentation
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        horizontal_flip=True,
        vertical_flip=False,
        width_shift_range=0.05,
        height_shift_range=0.05
    )

```

```

    # normalization نكتفي فقط بعمل test والـ validation
    val_test_datagen = ImageDataGenerator(rescale=1./255)

```

```

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

```

```

    val_generator = val_test_datagen.flow_from_directory(
        val_dir,
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

```

```

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

```

```

    return train_generator, val_generator, test_generator

```

```
# generators

train_gen, val_gen, test_gen = create_data_generators(
    train_dir,
    val_dir,
    test_dir,
    img_size=(240, 240),
    batch_size=64
)

# شكل الصور الداخلة للنموذج
input_shape = (240, 240, 3)

# عدد الفئات من الداتا نفسها
num_classes = len(train_gen.class_indices)

print("عدد الفئات:", num_classes)
print("الفئات:", train_gen.class_indices)
```

```
Found 17495 images belonging to 5 classes.
Found 2500 images belonging to 5 classes.
Found 5005 images belonging to 5 classes.
عدد الفئات: 5
الفئات: {'colon_aca': 0, 'colon_n': 1, 'lung_aca': 2, 'lung_n': 3, 'lung_scc': 4}
```

```
# دالة تقييم النماذج

def evaluate_model(model_name, y_true, y_pred_classes, labels):
    accuracy = accuracy_score(y_true, y_pred_classes)
    precision = precision_score(y_true, y_pred_classes, average='weighted')
    recall = recall_score(y_true, y_pred_classes, average='weighted')
    f1 = f1_score(y_true, y_pred_classes, average='weighted')
    cm = confusion_matrix(y_true, y_pred_classes)

    print(f"==== {model_name} =====")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

    print("\nClassification Report:")
    print(classification_report(y_true, y_pred_classes, target_names=labels))

    plt.figure(figsize=(8, 6))
    sns.heatmap(
        cm,
        annot=True,
        fmt='d',
        cmap='Blues',
        xticklabels=labels,
        yticklabels=labels
    )
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()
```

```
# دالة رسم نتائج التدريب

def plot_training_history(history, title="Training History"):
    plt.figure(figsize=(12, 5))

    # رسم الـ Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{title} - Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
# رسم الـ Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title(f'{title} - Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

```
print("عدد صور التدريب:", train_gen.samples)
print("Batch size:", train_gen.batch_size)
print("epoch: عدد الخطوات في كل", len(train_gen))
```

```
عدد صور التدريب: 17495
Batch size: 64
epoch: عدد الخطوات في كل 274
```

```
# بناء نموذج CNN

def build_cnn_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(2, 2),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

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

    return model
```

```
# تدريب نموذج CNN

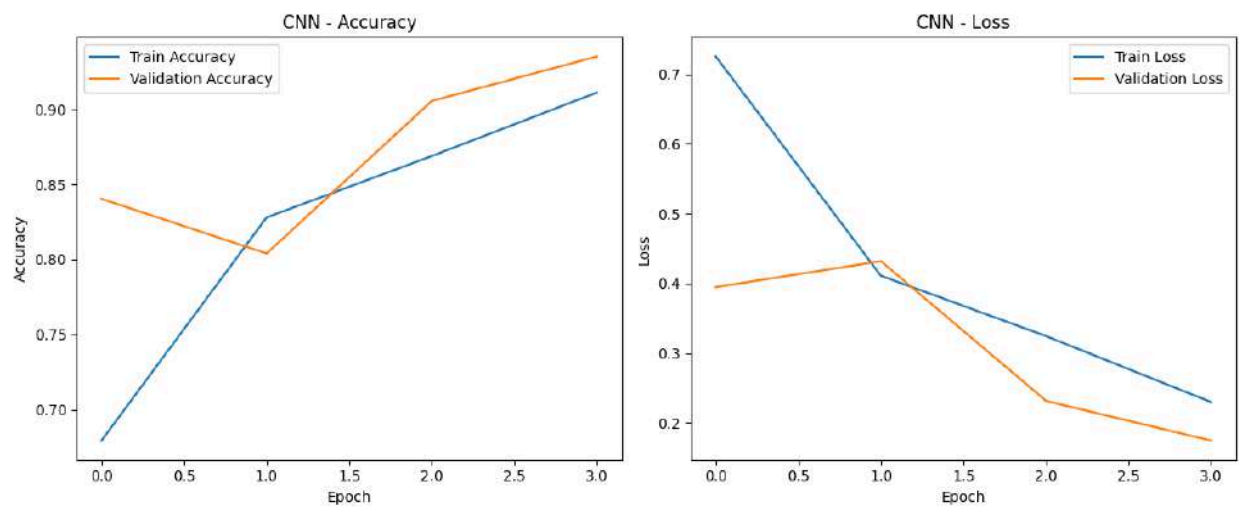
cnn_model = build_cnn_model(input_shape, num_classes)

history_cnn = cnn_model.fit(
    train_gen,
    epochs=4, # تم تقليلها لتقليل وقت العملية
    validation_data=val_gen
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning:
  self._warn_if_super_not_called()
Epoch 1/4
274/274 ━━━━━━━━━━━ 269s 947ms/step - accuracy: 0.5600 - loss: 1.1627 - val_accuracy: 0.8404 - val_loss: 0.4498
Epoch 2/4
274/274 ━━━━━━━━━━━ 256s 933ms/step - accuracy: 0.8074 - loss: 0.4498 - val_accuracy: 0.8040 - val_loss: 0.4498
Epoch 3/4
274/274 ━━━━━━━━━━━ 252s 918ms/step - accuracy: 0.8579 - loss: 0.3467 - val_accuracy: 0.9056 - val_loss: 0.2358
Epoch 4/4
274/274 ━━━━━━━━━━━ 257s 938ms/step - accuracy: 0.9092 - loss: 0.2358 - val_accuracy: 0.9352 - val_loss: 0.2358
```

```
# رسم نتائج تدريب CNN
```

```
plot_training_history(history_cnn, title="CNN")
```



```
# تقييم نموذج CNN
```

```
y_pred = cnn_model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)

labels = list(test_gen.class_indices.keys())

evaluate_model("CNN", y_true, y_pred_classes, labels)
```

79/79 ————— 21s 255ms/step

==== CNN =====

Accuracy: 0.9330669330669331

Precision: 0.9353034813196605

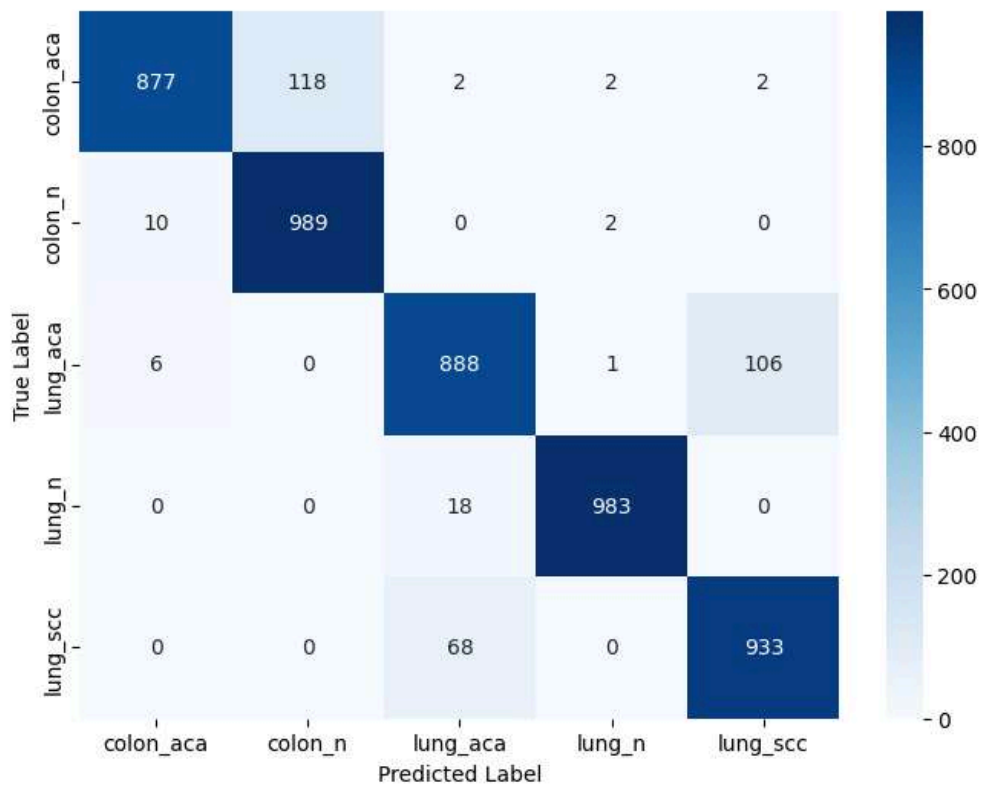
Recall: 0.9330669330669331

F1-score: 0.9329979461595891

Classification Report:

	precision	recall	f1-score	support
colon_aca	0.98	0.88	0.93	1001
colon_n	0.89	0.99	0.94	1001
lung_aca	0.91	0.89	0.90	1001
lung_n	0.99	0.98	0.99	1001
lung_scc	0.90	0.93	0.91	1001
accuracy			0.93	5005
macro avg	0.94	0.93	0.93	5005
weighted avg	0.94	0.93	0.93	5005

Confusion Matrix - CNN



بناء نموذج ResNet50

```
def build_resnet50_model(input_shape, num_classes):
    base_model = ResNet50(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )

    # Freeze the base model
    base_model.trainable = False

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(
        optimizer='adam',
```

```

        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model

```

تدريب نموذج ResNet50

```
resnet50_model = build_resnet50_model(input_shape, num_classes)
```

```

history_resnet50 = resnet50_model.fit(
    train_gen,
    epochs=2,      # تم تقليل وقت العملية
    validation_data=val_gen
)

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_data_format.h5: 94765736/94765736 ————— 1s 0us/step

Epoch 1/2

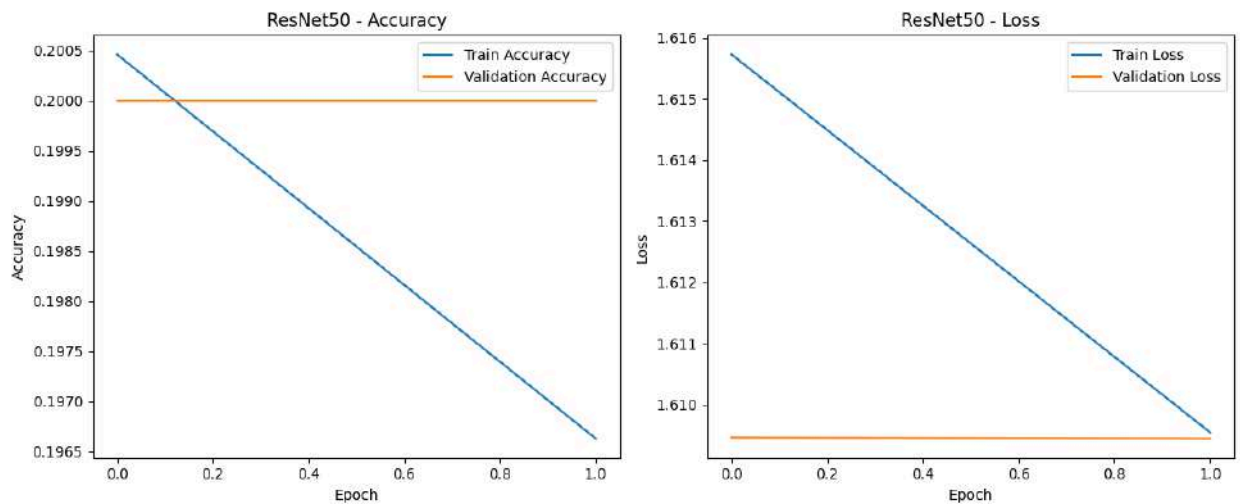
274/274 ————— 285s 984ms/step - accuracy: 0.2090 - loss: 1.6375 - val_accuracy: 0.2000 - val_loss: 1.6157

Epoch 2/2

274/274 ————— 253s 925ms/step - accuracy: 0.2023 - loss: 1.6094 - val_accuracy: 0.2000 - val_loss: 1.6157

رسم نتائج تدريب ResNet50

```
plot_training_history(history_resnet50, title="ResNet50")
```



تقييم نموذج ResNet50

```

y_pred = resnet50_model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)

labels = list(test_gen.class_indices.keys())

evaluate_model("ResNet50", y_true, y_pred_classes, labels)

```

79/79 ————— 29s 321ms/step

==== ResNet50 =====

Accuracy: 0.2

Precision: 0.04000799360511591

Recall: 0.2

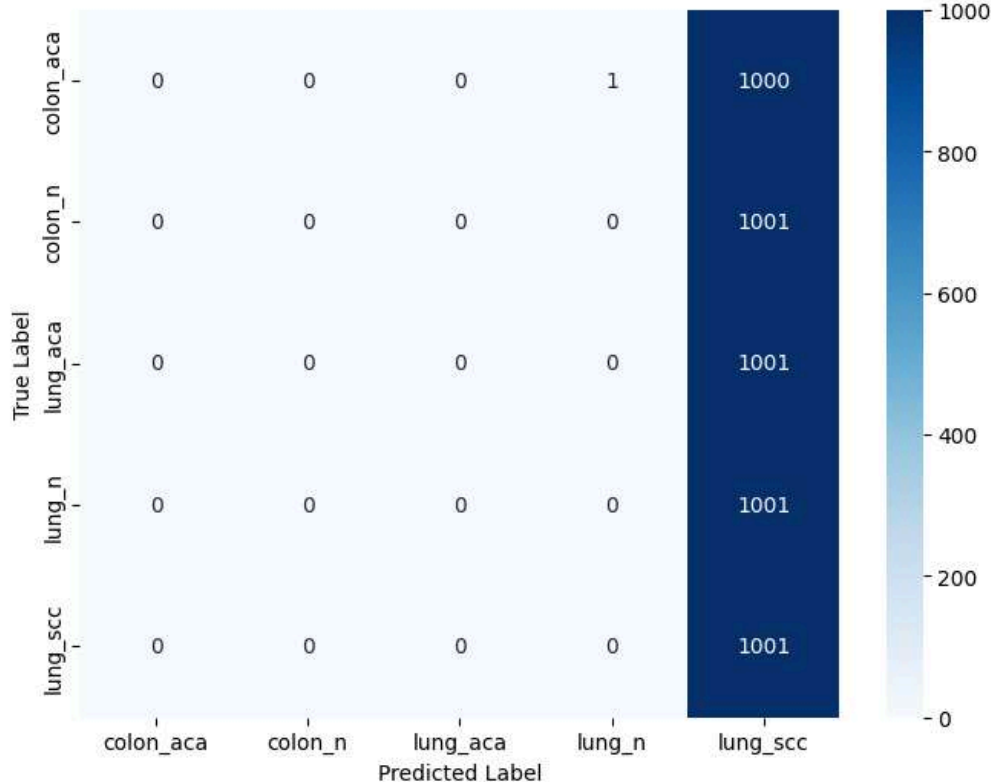
F1-score: 0.06667776852622814

Classification Report:

	precision	recall	f1-score	support
colon_aca	0.00	0.00	0.00	1001
colon_n	0.00	0.00	0.00	1001
lung_aca	0.00	0.00	0.00	1001
lung_n	0.00	0.00	0.00	1001
lung_scc	0.20	1.00	0.33	1001
accuracy			0.20	5005
macro avg	0.04	0.20	0.07	5005
weighted avg	0.04	0.20	0.07	5005

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
 /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
 /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
 /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
 /usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Confusion Matrix - ResNet50



بناء نموذج EfficientNetB0

```
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )

    # Freeze the base model
    base_model.trainable = False

    model = Sequential([
        base_model,
```

```

        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

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

    return model

```

تدريب نموذج EfficientNetB0

```
efficientnet_model = build_efficientnetb0_model(input_shape, num_classes)
```

```

history_efficientnet = efficientnet_model.fit(
    train_gen,
    epochs=2,      # تم تقليلها لتقليل وقت العملية
    validation_data=val_gen
)

```

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 ————— 0s 0us/step

Epoch 1/2

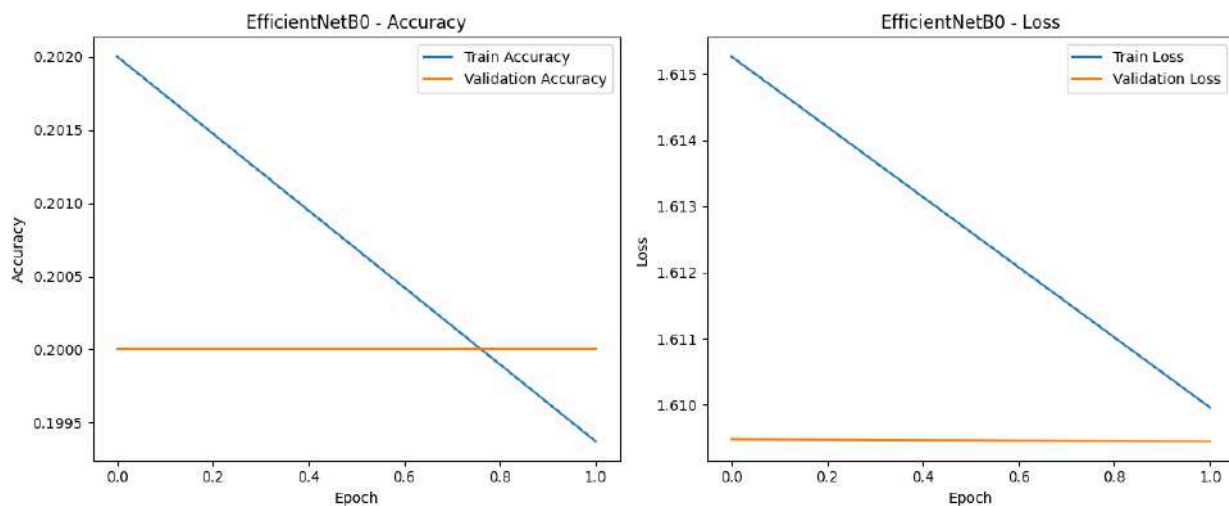
274/274 ————— 346s 1s/step - accuracy: 0.2089 - loss: 1.6271 - val_accuracy: 0.2000 - val_loss: 1.6102

Epoch 2/2

274/274 ————— 253s 925ms/step - accuracy: 0.2017 - loss: 1.6102 - val_accuracy: 0.2000 - val_loss: 1.6102

رسم نتائج تدريب EfficientNetB0

```
plot_training_history(history_efficientnet, title="EfficientNetB0")
```



تقييم نموذج EfficientNetB0

```

y_pred = efficientnet_model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)

labels = list(test_gen.class_indices.keys())

evaluate_model("EfficientNetB0", y_true, y_pred_classes, labels)

```

79/79 ————— 47s 519ms/step

==== EfficientNetB0 ====

Accuracy: 0.2

Precision: 0.04

Recall: 0.2

F1-score: 0.06666666666666667

Classification Report:

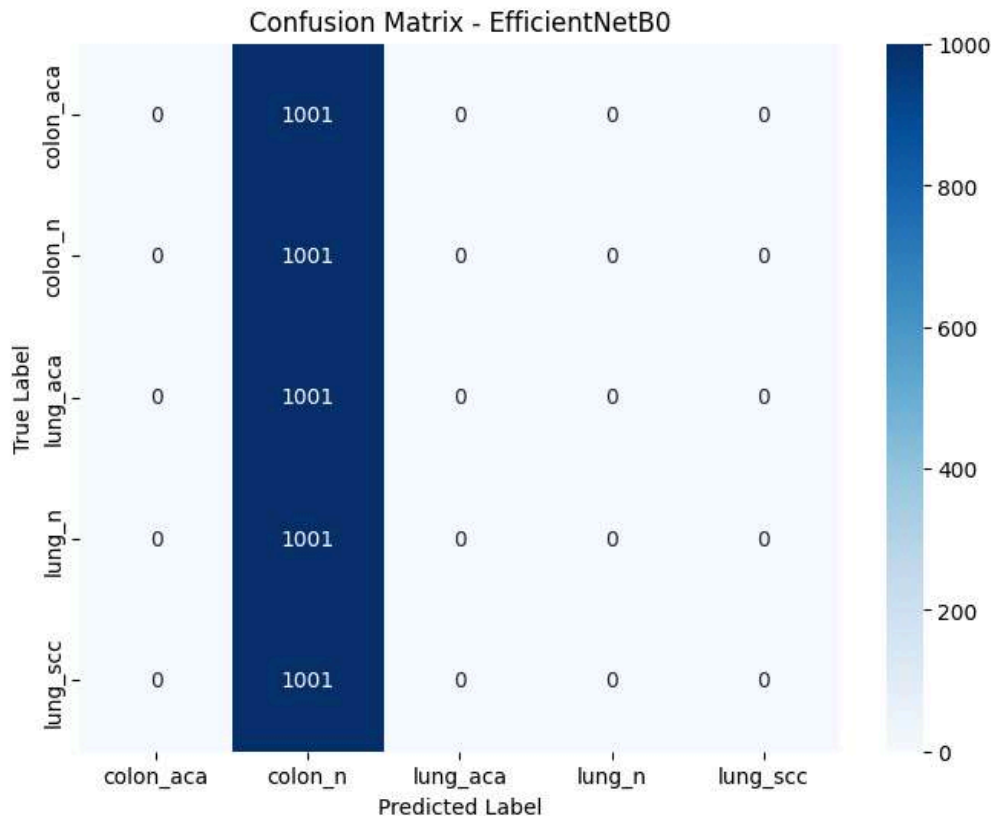
	precision	recall	f1-score	support
colon_aca	0.00	0.00	0.00	1001
colon_n	0.20	1.00	0.33	1001
lung_aca	0.00	0.00	0.00	1001
lung_n	0.00	0.00	0.00	1001
lung_scc	0.00	0.00	0.00	1001
accuracy			0.20	5005
macro avg	0.04	0.20	0.07	5005
weighted avg	0.04	0.20	0.07	5005

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))



بناء نموذج InceptionV3

```
def build_inceptionv3_model(num_classes):
```

```
    base_model = InceptionV3(
        weights='imagenet',
        include_top=False,
        input_shape=(240, 240, 3)
    )
```

```
    # Freeze the base model
    base_model.trainable = False
```

```
    model = Sequential([
        base_model,
```

```

        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

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

    return model

```

تدريب نموذج InceptionV3

```
inceptionv3_model = build_inceptionv3_model(num_classes)
```

```

history_inceptionv3 = inceptionv3_model.fit(
    train_gen,
    epochs=2,      # تم تقليلها لتقليل وقت العملية
    validation_data=val_gen
)

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_2016_08_16/inception_v3_2016_08_16.tar.gz 87910968/87910968 ————— 0s 0us/step

Epoch 1/2

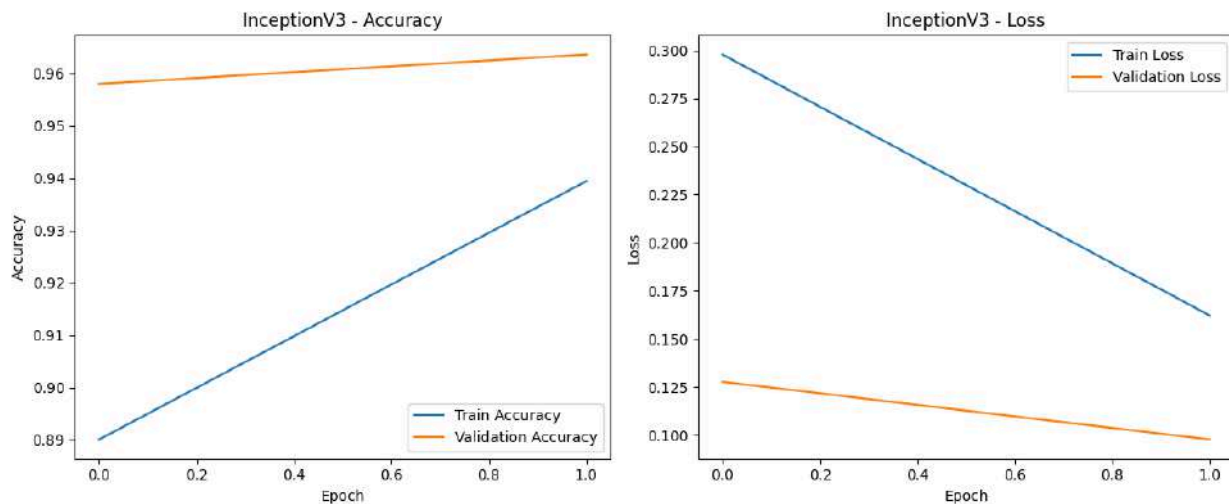
274/274 ————— 292s 1s/step - accuracy: 0.8095 - loss: 0.5102 - val_accuracy: 0.9580 - val_loss: 0.1250

Epoch 2/2

274/274 ————— 254s 925ms/step - accuracy: 0.9381 - loss: 0.1669 - val_accuracy: 0.9636 - val_loss: 0.1000

رسم نتائج تدريب InceptionV3

```
plot_training_history(history_inceptionv3, title="InceptionV3")
```



تقييم نموذج InceptionV3

```

y_pred = inceptionv3_model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)

```

```
labels = list(test_gen.class_indices.keys())
```

```
evaluate_model("InceptionV3", y_true, y_pred_classes, labels)
```

79/79 ————— 34s 355ms/step

===== InceptionV3 =====

Accuracy: 0.9644355644355644

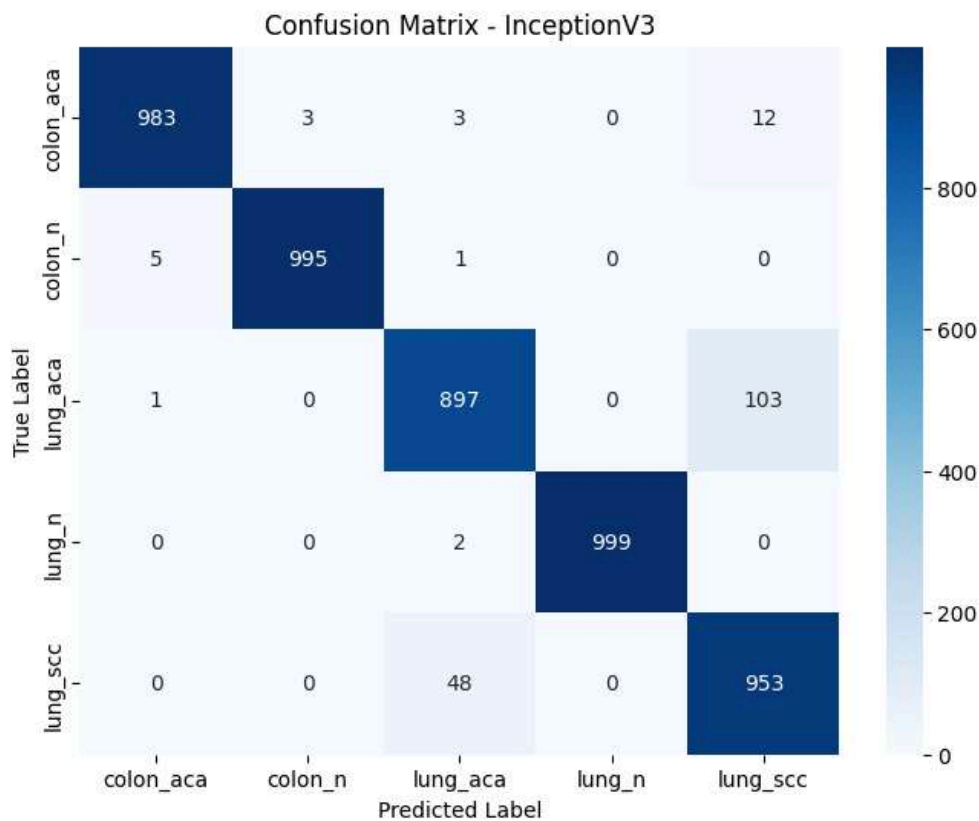
Precision: 0.9652934033789096

Recall: 0.9644355644355644

F1-score: 0.9645425608232837

Classification Report:

	precision	recall	f1-score	support
colon_aca	0.99	0.98	0.99	1001
colon_n	1.00	0.99	1.00	1001
lung_aca	0.94	0.90	0.92	1001
lung_n	1.00	1.00	1.00	1001
lung_scc	0.89	0.95	0.92	1001
accuracy			0.96	5005
macro avg	0.97	0.96	0.96	5005
weighted avg	0.97	0.96	0.96	5005



مقارنة دقة النماذج

```
models_accuracy = {
    "CNN": max(history_cnn.history["val_accuracy"]),
    "ResNet50": max(history_resnet50.history["val_accuracy"]),
    "EfficientNetB0": max(history_efficientnet.history["val_accuracy"]),
    "InceptionV3": max(history_inceptionv3.history["val_accuracy"])
}
```

```
print("لكل Model Validation Accuracy أفضل:")
for model_name, acc in models_accuracy.items():
    print(f"{model_name}: {acc:.4f}")
```

```
لكل Model Validation Accuracy أفضل:
CNN: 0.9352
ResNet50: 0.2000
EfficientNetB0: 0.2000
InceptionV3: 0.9636
```

استيراد المكتبات الأساسية الخاصة بـ PyTorch

```
import os
```

```

import time
import copy
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import torch
from torch import nn, optim
from torch.utils.data import DataLoader

from torchvision import datasets, transforms as T, models

from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

from tqdm import tqdm

```

```

# استخدام نفس البيانات الجاهزة من

dataset_path = "/kaggle/working/dataset"

train_dir = os.path.join(dataset_path, "train")
val_dir   = os.path.join(dataset_path, "validation")
test_dir  = os.path.join(dataset_path, "test")

print("Train path:", train_dir)
print("Validation path:", val_dir)
print("Test path:", test_dir)

print("\nClasses:")
print(sorted(os.listdir(train_dir)))

Train path: /kaggle/working/dataset/train
Validation path: /kaggle/working/dataset/validation
Test path: /kaggle/working/dataset/test

Classes:
['colon_aca', 'colon_n', 'lung_aca', 'lung_n', 'lung_scc']

```

```

# قراءة أسماء الفئات من مجلد التدريب

def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

classes = get_classes(train_dir)
num_classes = len(classes)

print("Classes:", classes)
print("Number of classes:", num_classes)

Classes: ['colon_aca', 'colon_n', 'lung_aca', 'lung_n', 'lung_scc']
Number of classes: 5

```

```

# تجهيز DataLoaders

def get_data_loaders(data_dir, batch_size, train=False):
    if train:
        # Augmentation لتدريب
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),

```

```

        T.RandomRotation(10),
        T.Resize((224, 224)), # خففناها من 299 إلى 224 لتسريع التدريب
        T.ToTensor(),
        T.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

train_data = datasets.ImageFolder(
    os.path.join(data_dir, "train"),
    transform=transform
)

train_loader = DataLoader(
    train_data,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2
)

return train_loader, len(train_data)

else:
    # بدون augmentation و validation تجهيز
    transform = T.Compose([
        T.Resize((224, 224)), # نفس المقياس المخفض
        T.ToTensor(),
        T.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])

    val_data = datasets.ImageFolder(
        os.path.join(data_dir, "validation"),
        transform=transform
    )

    test_data = datasets.ImageFolder(
        os.path.join(data_dir, "test"),
        transform=transform
    )

    val_loader = DataLoader(
        val_data,
        batch_size=batch_size,
        shuffle=False,
        num_workers=2
    )

    test_loader = DataLoader(
        test_data,
        batch_size=batch_size,
        shuffle=False,
        num_workers=2
    )

    return val_loader, test_loader, len(val_data), len(test_data)

```

```
# تشغيل الـ DataLoaders
```

```
batch_size = 64
```

```
train_loader, train_data_len = get_data_loaders(dataset_path, batch_size, train=True)
val_loader, test_loader, valid_data_len, test_data_len = get_data_loaders(dataset_path, batch_size, train=False)
```

```
dataloaders = {
    "train": train_loader,
    "val": val_loader
}
```

```
dataset_sizes = {
    "train": train_data_len,
    "val": valid_data_len
}

print("Train images:", train_data_len)
print("Validation images:", valid_data_len)
print("Test images:", test_data_len)
print("Batch size:", batch_size)
print("Train steps:", len(train_loader))
```

```
Train images: 17495
Validation images: 2500
Test images: 5005
Batch size: 64
Train steps: 274
```

```
# استخدام GPU
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cuda
```

```
# تحميل نموذج EfficientNet-B0
```

```
model = models.efficientnet_b0(pretrained=True)
```

```
# استبدال الطبقة الأخيرة بعدد الفئات الحالية
```

```
model.classifier[1] = nn.Linear(
    model.classifier[1].in_features,
    num_classes
)
```

```
model = model.to(device)
```

```
print(model.classifier)
```

```
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /root/.cache,
100%|██████████| 20.5M/20.5M [00:00<00:00, 144MB/s]
```

```
Sequential(
  (0): Dropout(p=0.2, inplace=True)
  (1): Linear(in_features=1280, out_features=5, bias=True)
)
```

```
# تعريف دالة scheduler
```

```
criterion = nn.CrossEntropyLoss().to(device)
```

```
optimizer = optim.AdamW(model.parameters(), lr=0.001)
```

```
scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)
```

```
# دالة تدريب النموذج
```

```
# val accuracy تحفظ أفضل نسخة بناءً على
```

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=3):
    best_model_wts = copy.deepcopy(model.state_dict())
    best_val_acc = 0.0
```

```
    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch + 1}/{num_epochs}")
        print("-" * 30)
```

```
        for phase in ["train", "val"]:
```

```

    if phase == "train":
        model.train()
    else:
        model.eval()

    running_loss = 0.0
    running_corrects = 0

    for inputs, labels in tqdm(dataloaders[phase]):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        with torch.set_grad_enabled(phase == "train"):
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)

            if phase == "train":
                loss.backward()
                optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)

    if phase == "train":
        scheduler.step()

    epoch_loss = running_loss / dataset_sizes[phase]
    epoch_acc = running_corrects.double() / dataset_sizes[phase]

    print(f"{phase} Loss: {epoch_loss:.4f} | {phase} Acc: {epoch_acc:.4f}")

    # حفظ أفضل مودل بناءً على val accuracy
    if phase == "val" and epoch_acc > best_val_acc:
        best_val_acc = epoch_acc
        best_model_wts = copy.deepcopy(model.state_dict())

    print(f"\nBest val Acc: {best_val_acc:.4f}")

    model.load_state_dict(best_model_wts)
    return model

```

تدريب المودل

```

model = train_model(
    model,
    criterion,
    optimizer,
    scheduler,
    num_epochs=3 # لتقليل الوقت
)

```

Epoch 1/3

```

-----
100%|██████████| 274/274 [01:36<00:00, 2.83it/s]
train Loss: 0.0930 | train Acc: 0.9702
100%|██████████| 40/40 [00:10<00:00, 3.76it/s]
val Loss: 0.0465 | val Acc: 0.9812

```

Epoch 2/3

```

-----
100%|██████████| 274/274 [01:33<00:00, 2.92it/s]
train Loss: 0.0299 | train Acc: 0.9905
100%|██████████| 40/40 [00:10<00:00, 3.82it/s]
val Loss: 0.0068 | val Acc: 0.9984

```

Epoch 3/3

```

-----
100%|██████████| 274/274 [01:36<00:00, 2.83it/s]
train Loss: 0.0264 | train Acc: 0.9913

```

100% |██████████| 40/40 [00:10<00:00, 3.73it/s] val Loss: 0.0029 | val Acc: 0.9996

Best val Acc: 0.9996

دالة تقييم النموذج على test set

```
def evaluate_model(model_name, y_true, y_pred, y_proba, labels):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n==== {model_name} =====")
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)

    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=labels))

    plt.figure(figsize=(8, 6))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=labels,
        yticklabels=labels
    )
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()
```

تنفيذ التقييم على بيانات الاختبار

```
y_true = []
y_proba = []
y_pred = []

model.eval()

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)

        output = model(x)

        # تخزين القيم الحقيقية
        y_true.extend(y.numpy())

        # استخراج الاحتمالات
        probabilities = torch.softmax(output, dim=1)
        y_proba.extend(probabilities.cpu().numpy())

        # استخراج الفئة المتوقعة
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

# تحويل numpy arrays إلى القوائم
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

evaluate_model("EfficientNet-B0 (PyTorch)", y_true, y_pred, y_proba, classes)
```

100% |██████████| 79/79 [00:21<00:00, 3.72it/s]

==== EfficientNet-B0 (PyTorch) =====

Accuracy: 0.9994005994005994

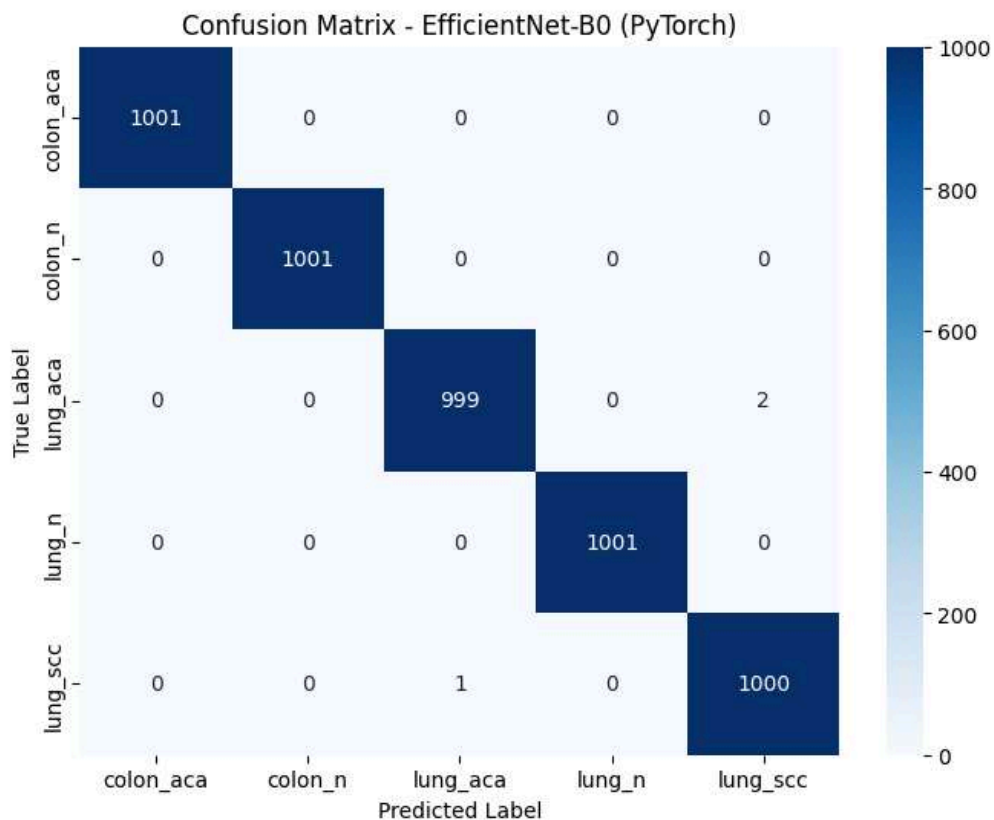
Precision: 0.9994007984031936

Recall: 0.9994005994005994

F1-score: 0.9994005992510485

Classification Report:

	precision	recall	f1-score	support
colon_aca	1.00	1.00	1.00	1001
colon_n	1.00	1.00	1.00	1001
lung_aca	1.00	1.00	1.00	1001
lung_n	1.00	1.00	1.00	1001
lung_scc	1.00	1.00	1.00	1001
accuracy			1.00	5005
macro avg	1.00	1.00	1.00	5005
weighted avg	1.00	1.00	1.00	5005



استيراد المكتبات

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
```

```
from tensorflow.keras.preprocessing import image
```

حفظ النموذج المدرب

```
import tensorflow as tf
model_path = "/kaggle/working/cancer_classifier_model.h5"

cnn_model.save(model_path)

print("Model saved to:", model_path)
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This format is deprecated. Use the Keras SavedModel format instead. Model saved to: /kaggle/working/cancer_classifier_model.h5

```
# تحميل النموذج المحفوظ
```

```
from tensorflow.keras.models import load_model
```

```
loaded_model = load_model(model_path)
```

```
print("Model loaded successfully")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be automatically created.
Model loaded successfully
```

```
# أسماء الفئات
```

```
class_names = list(train_gen.class_indices.keys())
```

```
print("Classes:", class_names)
```

```
Classes: ['colon_aca', 'colon_n', 'lung_aca', 'lung_n', 'lung_scc']
```

```
# تجهيز صورة للتنبؤ
```

```
def preprocess_image(img_path, img_size=(240,240)):
```

```
    img = image.load_img(img_path, target_size=img_size)
```

```
    img_array = image.img_to_array(img)
```

```
    img_array = img_array / 255.0
```

```
    img_array = np.expand_dims(img_array, axis=0)
```

```
    return img_array
```

```
# اختبار صورة عشوائية من
```

```
test_folder = "/kaggle/working/dataset/test"
```

```
random_class = os.listdir(test_folder)[0]
```

```
random_image = os.listdir(os.path.join(test_folder, random_class))[0]
```

```
img_path = os.path.join(test_folder, random_class, random_image)
```

```
print("Image path:", img_path)
```

```
Image path: /kaggle/working/dataset/test/lung_scc/lungsc2217.jpeg
```

```
# عرض الصورة
```

```
img = cv2.imread(img_path)
```

```
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

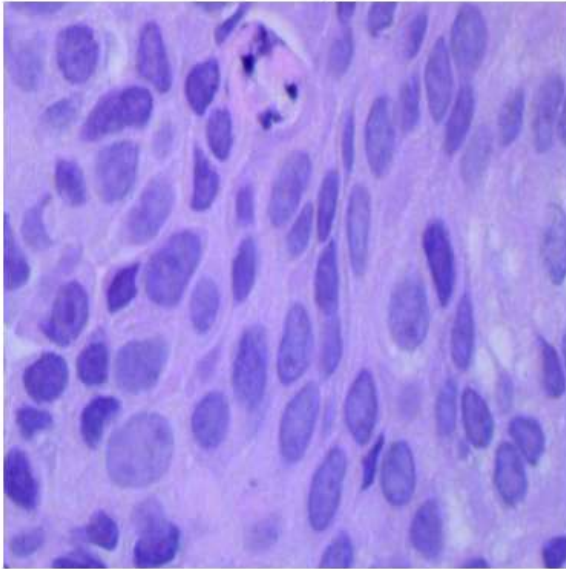
```
plt.imshow(img_rgb)
```

```
plt.title("Test Image")
```

```
plt.axis("off")
```

```
plt.show()
```

Test Image



```
# التنبؤ بالصورة

img_array = preprocess_image(img_path)

predictions = loaded_model.predict(img_array)

predicted_class = np.argmax(predictions)

confidence = np.max(predictions)

print("Predicted Class:", class_names[predicted_class])
print("Confidence:", confidence)
```

```
1/1 ————— 1s 575ms/step
Predicted Class: lung_scc
Confidence: 0.9896616
```

```
# عرض احتمالات كل الفئات

for i, prob in enumerate(predictions[0]):

    print(class_names[i], ":", round(prob,4))
```

```
colon_aca : 0.0
colon_n : 0.0
lung_aca : 0.0103
lung_n : 0.0
lung_scc : 0.9897
```

```
# رسم احتمالات التنبؤ

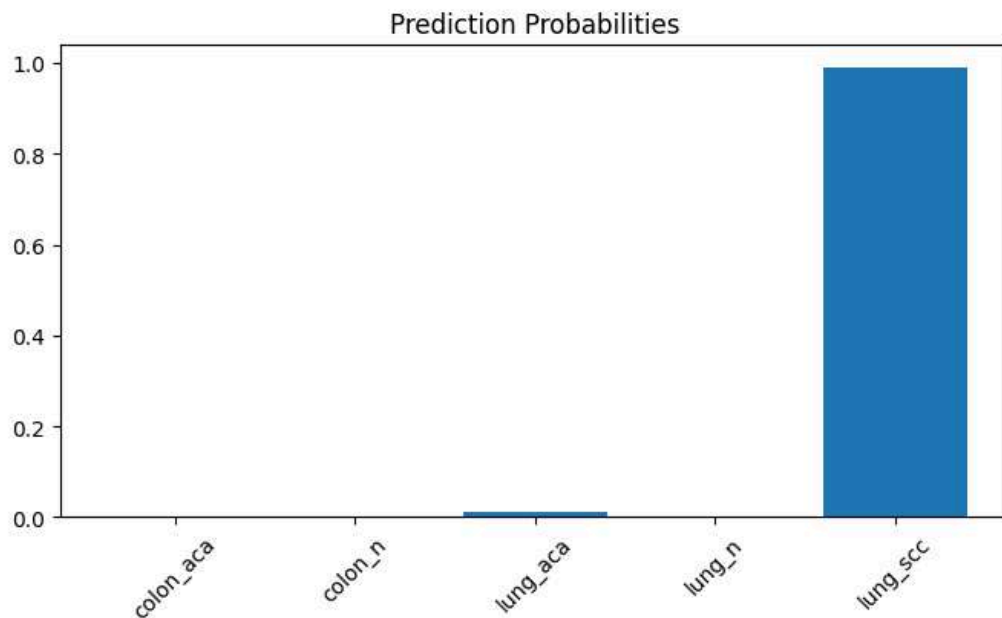
plt.figure(figsize=(8,4))

plt.bar(class_names, predictions[0])

plt.title("Prediction Probabilities")

plt.xticks(rotation=45)

plt.show()
```



```
# استيراد المكتبات

import torch
from torch import nn
import torch.optim as optim

from torchvision.models import vit_b_32

!pip -q install timm

import timm
from timm.loss import LabelSmoothingCrossEntropy
```

```
# التحقق من المتغيرات

required_vars = [
    "classes",
    "device",
    "train_model",
    "train_loader",
    "val_loader",
    "test_loader"
]

for var in required_vars:
    print(var, "=>", var in globals())
```

```
classes => True
device => True
train_model => True
train_loader => True
val_loader => True
test_loader => True
```

```
# بناء نموذج Vision Transformer (ViT)

model = vit_b_32(pretrained=True)

# عدد الخصائص الداخلة إلى التصنيف
n_inputs = model.heads.head.in_features

model.heads.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(classes))
)
```

```
# CPU أو GPU نقل المودل إلى
model = model.to(device)

print(model.heads.head)
```

```
Downloading: "https://download.pytorch.org/models/vit_b_32-d86f8d99.pth" to /root/.cache/torch/hub/checkpoint
100%|██████████| 337M/337M [00:02<00:00, 122MB/s]
Sequential(
  (0): Linear(in_features=768, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=512, out_features=5, bias=True)
)
```

```
# إعدادات التدريب الخاصة بالـ Transformer

# Loss function
criterion = LabelSmoothingCrossEntropy()

# Optimizer
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
scheduler = optim.lr_scheduler.StepLR(
    optimizer,
    step_size=3,
    gamma=0.97
)

print("Transformer settings جاهزة")
```

Transformer settings جاهزة

```
# تدريب نموذج ViT

ViT_model = train_model(
    model,
    criterion,
    optimizer,
    scheduler,
    num_epochs=3 # لتقليل الوقت
)
```

```
Epoch 1/3
-----
100%|██████████| 274/274 [01:36<00:00, 2.85it/s]
train Loss: 0.9366 | train Acc: 0.6812
100%|██████████| 40/40 [00:10<00:00, 3.81it/s]
val Loss: 0.7309 | val Acc: 0.7976

Epoch 2/3
-----
100%|██████████| 274/274 [01:35<00:00, 2.86it/s]
train Loss: 0.7125 | train Acc: 0.8269
100%|██████████| 40/40 [00:10<00:00, 3.89it/s]
val Loss: 0.6389 | val Acc: 0.8716

Epoch 3/3
-----
100%|██████████| 274/274 [01:34<00:00, 2.90it/s]
train Loss: 0.6280 | train Acc: 0.8750
100%|██████████| 40/40 [00:10<00:00, 3.83it/s] val Loss: 0.6590 | val Acc: 0.8672

Best val Acc: 0.8716
```

```
# دالة تقييم نموذج PyTorch

import numpy as np
import matplotlib.pyplot as plt
```

```

import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score

def evaluate_pytorch_model(model, test_loader, class_names, device):
    y_true = []
    y_pred = []

    model.eval()

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs = inputs.to(device)

            outputs = model(inputs)
            preds = torch.argmax(outputs, dim=1).cpu().numpy()

            y_pred.extend(preds)
            y_true.extend(labels.numpy())

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

    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average="weighted", zero_division=0)
    recall = recall_score(y_true, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_true, y_pred, average="weighted", zero_division=0)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1-score:", f1)
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=class_names, zero_division=0))

    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=class_names,
        yticklabels=class_names
    )
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix - ViT")
    plt.show()

```

```

# تقييم نموذج ViT

evaluate_pytorch_model(
    ViT_model,
    test_loader,
    classes,
    device
)

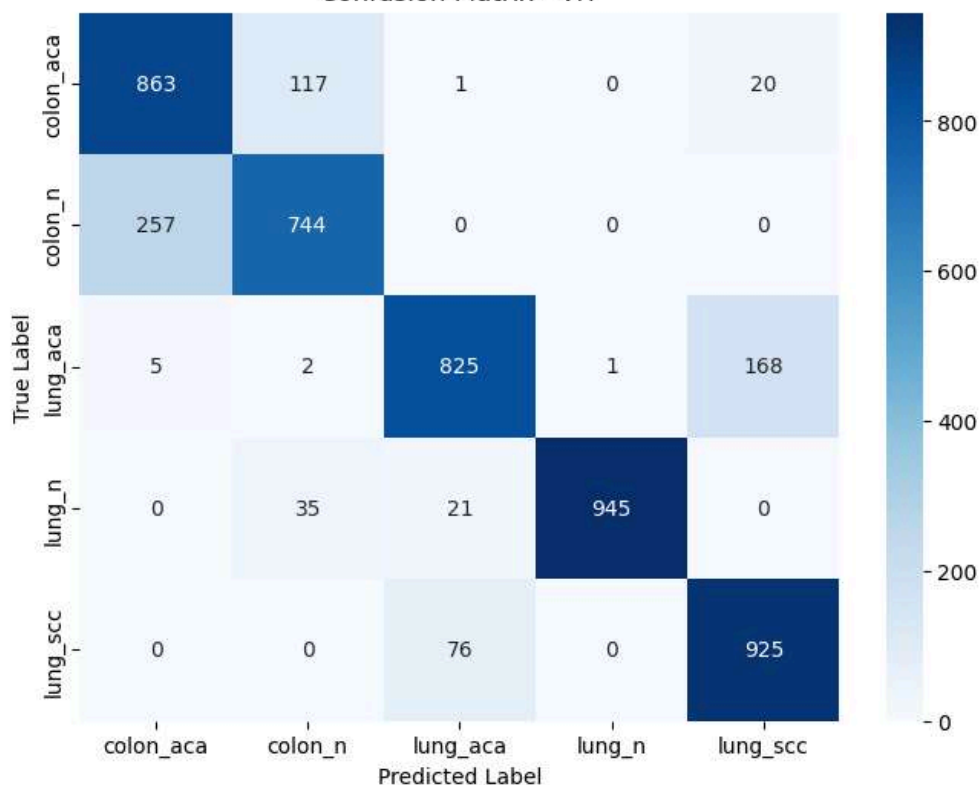
```

Accuracy: 0.8595404595404595
 Precision: 0.8638946921949106
 Recall: 0.8595404595404595
 F1-score: 0.8597708707130488

Classification Report:

	precision	recall	f1-score	support
colon_aca	0.77	0.86	0.81	1001
colon_n	0.83	0.74	0.78	1001
lung_aca	0.89	0.82	0.86	1001
lung_n	1.00	0.94	0.97	1001
lung_scc	0.83	0.92	0.88	1001
accuracy			0.86	5005
macro avg	0.86	0.86	0.86	5005
weighted avg	0.86	0.86	0.86	5005

Confusion Matrix - ViT



مقارنة أداء جميع النماذج

```
import pandas as pd
import matplotlib.pyplot as plt
```

تخزين نتائج النماذج

```
results = {
    "CNN": max(history_cnn.history["val_accuracy"]),
    "ResNet50": max(history_resnet50.history["val_accuracy"]),
    "EfficientNetB0": max(history_efficientnet.history["val_accuracy"]),
    "InceptionV3": max(history_inceptionv3.history["val_accuracy"])
}
```

تحويلها إلى DataFrame

```
results_df = pd.DataFrame(list(results.items()), columns=["Model", "Validation Accuracy"])
```

ترتيب النتائج من الأفضل للأسوأ

```
results_df = results_df.sort_values(by="Validation Accuracy", ascending=False)
```

```
print("Model Comparison")
```

```
print(results_df)
```

رسم المقارنة

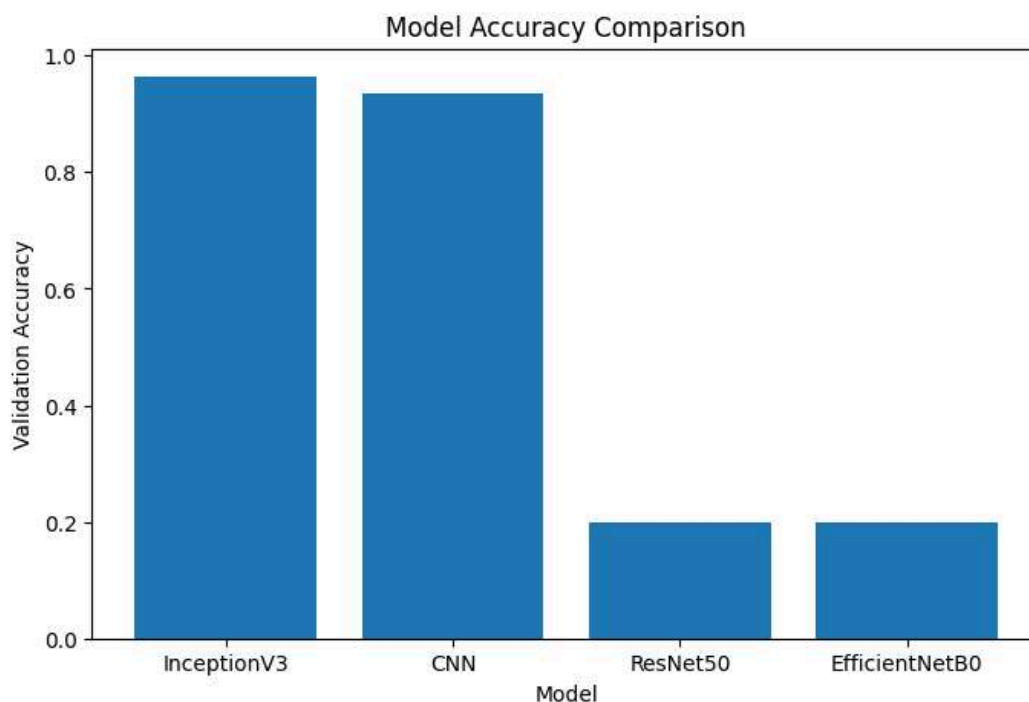
```
plt.figure(figsize=(8,5))
```

```
plt.bar(results_df["Model"], results_df["Validation Accuracy"])

plt.title("Model Accuracy Comparison")
plt.xlabel("Model")
plt.ylabel("Validation Accuracy")

plt.show()
```

```
Model Comparison
      Model  Validation Accuracy
3  InceptionV3          0.9636
0      CNN          0.9352
1  ResNet50          0.2000
2 EfficientNetB0        0.2000
```



```
best_model = results_df.iloc[0]["Model"]
print("\nأفضل نموذج هو:", best_model)
```

أفضل نموذج هو: InceptionV3

ROC Curve + AUC Score

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle
```

```
def plot_multiclass_roc_auc(model, test_gen, model_name="Model"):
```

```
    test_gen.reset()
```

```
    # التوقعات الاحتمالية من النموذج
```

```
    y_score = model.predict(test_gen, verbose=1)
```

```
    # القيم الحقيقية
```

```
    y_true = test_gen.classes
```

```
    # أسماء الفئات
```

```
    class_names = list(test_gen.class_indices.keys())
```

```
    n_classes = len(class_names)
```

```
    # تحويل y_true إلى one-hot / binarized
```

```
    y_true_bin = label_binarize(y_true, classes=np.arange(n_classes))
```

```

# إذا كانت الثنائية فقط، نخليها بصيغة مناسبة
if n_classes == 2 and y_true_bin.shape[1] == 1:
    y_true_bin = np.hstack([1 - y_true_bin, y_true_bin])

# لكل فئة ROC و AUC حساب
fpr = {}
tpr = {}
roc_auc = {}

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# حساب Micro-average ROC
fpr["micro"], tpr["micro"], _ = roc_curve(y_true_bin.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# رسم المنحنيات
plt.figure(figsize=(10, 8))

colors = cycle(["blue", "red", "green", "orange", "purple", "brown", "pink"])

for i, color in zip(range(n_classes), colors):
    plt.plot(
        fpr[i],
        tpr[i],
        color=color,
        lw=2,
        label=f"{class_names[i]} (AUC = {roc_auc[i]:.4f})"
    )

# رسم micro-average
plt.plot(
    fpr["micro"],
    tpr["micro"],
    color="black",
    linestyle="--",
    lw=2,
    label=f"micro-average (AUC = {roc_auc['micro']:.4f})"
)

# خط عشوائي
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(f"ROC Curve - {model_name}")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

# طباعة النتائج
print(f"==== ROC AUC Results for {model_name} =====")
for i in range(n_classes):
    print(f"{class_names[i]}: AUC = {roc_auc[i]:.4f}")
print(f"micro-average: AUC = {roc_auc['micro']:.4f}")

return roc_auc

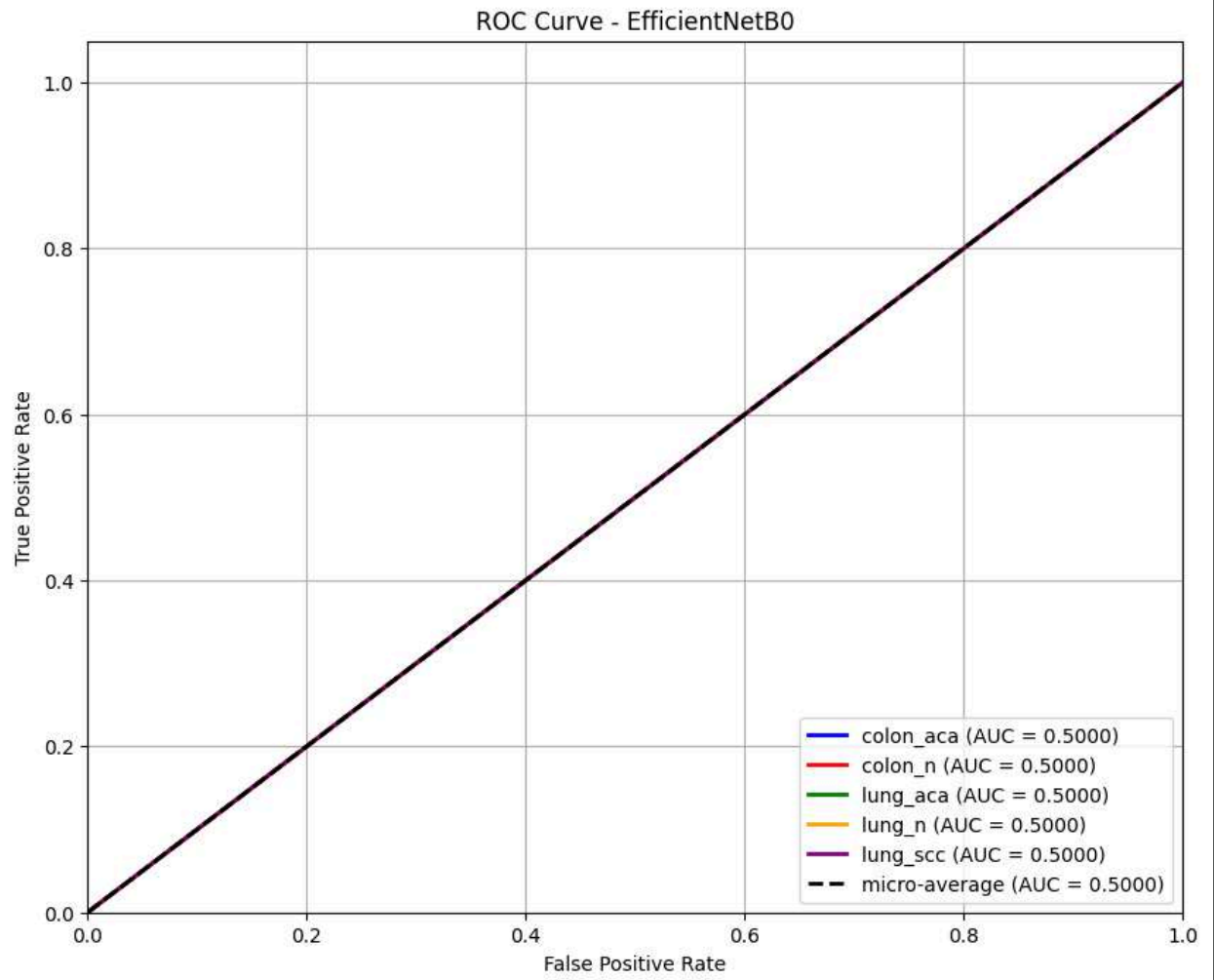
```

```

#EfficientNetB0
roc_auc_efficientnet = plot_multiclass_roc_auc(
    efficientnet_model,
    test_gen,
    model_name="EfficientNetB0"
)

```

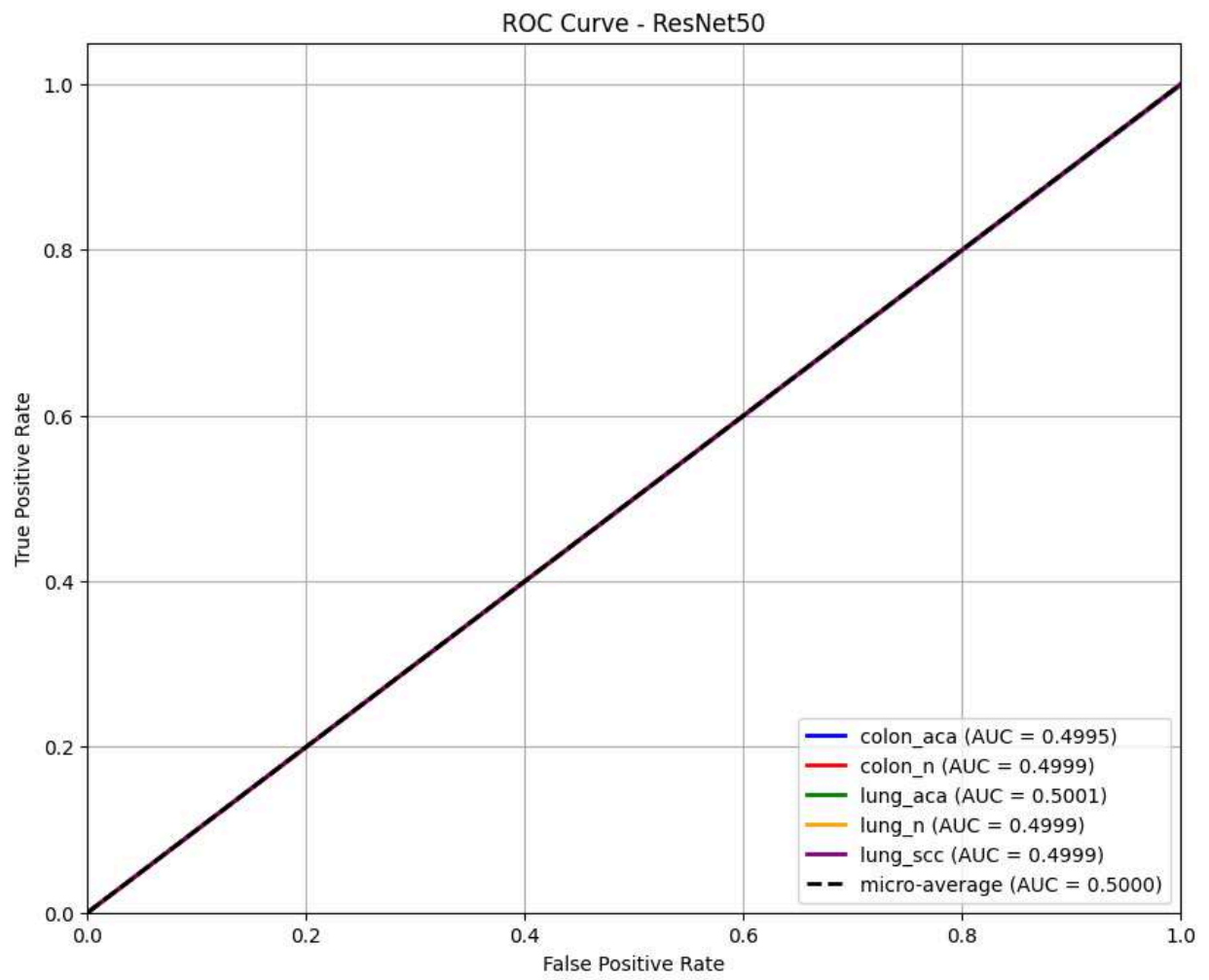
79/79 — 20s 246ms/step



```
==== ROC AUC Results for EfficientNetB0 ====  
colon_aca: AUC = 0.5000  
colon_n: AUC = 0.5000  
lung_aca: AUC = 0.5000  
lung_n: AUC = 0.5000  
lung_scc: AUC = 0.5000  
micro-average: AUC = 0.5000
```

```
#ResNet50  
roc_auc_resnet = plot_multiclass_roc_auc(  
    resnet50_model,  
    test_gen,  
    model_name="ResNet50"  
)
```

79/79 — 19s 240ms/step



```
==== ROC AUC Results for ResNet50 ====  
colon_aca: AUC = 0.4995  
colon_n: AUC = 0.4999  
lung_aca: AUC = 0.5001  
lung_n: AUC = 0.4999  
lung_scc: AUC = 0.4999  
micro-average: AUC = 0.5000
```

```
#CNN  
roc_auc_cnn = plot_multiclass_roc_auc(  
    cnn_model,  
    test_gen,  
    model_name="CNN"  
)
```

79/79 — 20s 251ms/step

