

```
1 !rm -rf /content/drive
2
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

 Mounted at /content/drive

```
1 import os
2
3 base_path = "/content/drive/MyDrive/image_processing"
4 for root, dirs, files in os.walk(base_path):
5     print(root)
6     # توقف بعد أول 10 نتائج فقط
7     if len(root.split('/')) > 8:
8         break
9
```

 /content/drive/MyDrive/image_processing
/content/drive/MyDrive/image_processing/lung_colon_image_set
/content/drive/MyDrive/image_processing/lung_colon_image_set/lung_scc
/content/drive/MyDrive/image_processing/lung_colon_image_set/colon_image_sets
/content/drive/MyDrive/image_processing/lung_colon_image_set/colon_image_sets/colon_n
/content/drive/MyDrive/image_processing/lung_colon_image_set/colon_image_sets/colon_aca
/content/drive/MyDrive/image_processing/lung_colon_image_set/dataset
/content/drive/MyDrive/image_processing/lung_colon_image_set/dataset/train
/content/drive/MyDrive/image_processing/lung_colon_image_set/dataset/train/lung_scc

```
1 folder_path = '/content/drive/MyDrive/image processing /lung_colon_image_set'
2
```

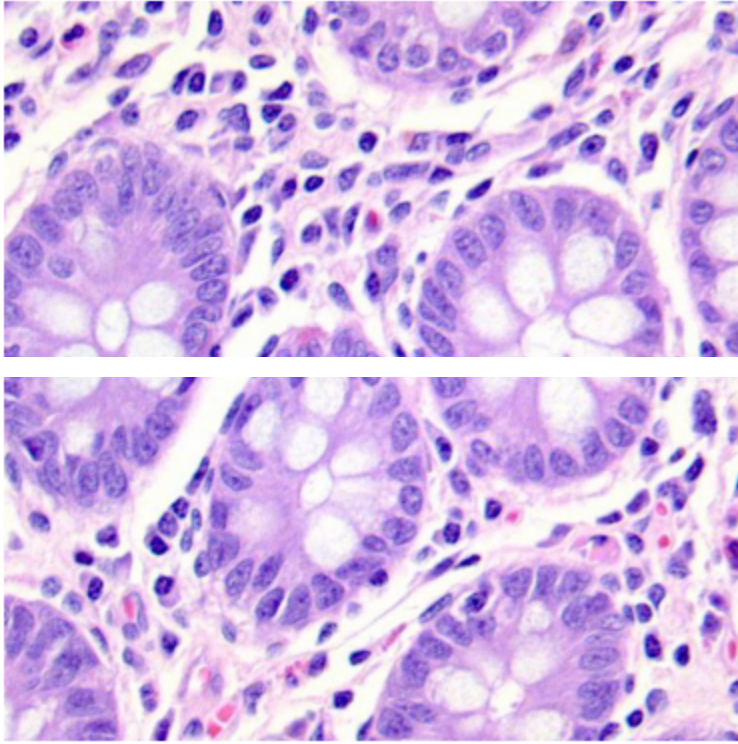
```
1 folder_path = '/content/drive/MyDrive/image_processing/lung_colon_image_set/'
2
```

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 img_path = '/content/drive/MyDrive/image_processing/lung_colon_image_set/col
5 image = cv2.imread(img_path)
6 image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
7
8 plt.imshow(image_rgb)
9 plt.title("Loaded Image")
10 plt.axis('off')
```

```
11 plt.show()  
12
```



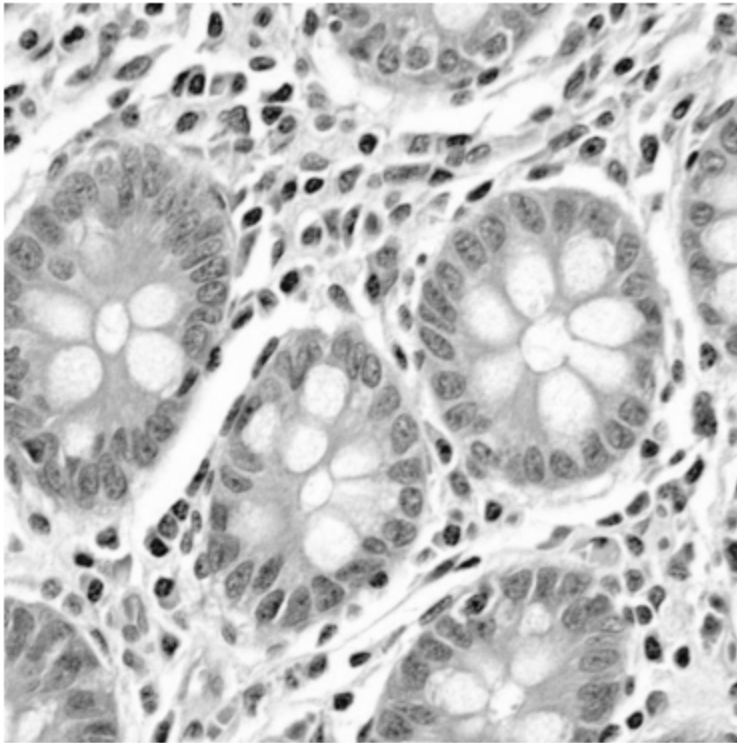
Loaded Image



```
1 gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  
2  
3 plt.imshow(gray_image, cmap="gray")  
4 plt.axis("off")  
5 plt.title("Grayscale Image")  
6 plt.show()  
7
```



Grayscale Image



```
1 height, width, channels = image.shape
2 print(f"Width: {width}, Height: {height}")
3
```



Width: 768, Height: 768

```
1 from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5
6 def display_augmented_images(img_path, datagen, num_augmented=3):
7     if not os.path.exists(img_path):
8         raise FileNotFoundError(f"Image not found: {img_path}")
9
10    img = load_img(img_path)
11    img_array = img_to_array(img)
12    img_array = np.expand_dims(img_array, axis=0)
13
14    augmented_images = datagen.flow(img_array, batch_size=1)
15
16    plt.figure(figsize=(4 * (num_augmented + 1), 5))
17    plt.subplot(1, num_augmented + 1, 1)
18    plt.title('Original')
19    plt.imshow(img)
```

```

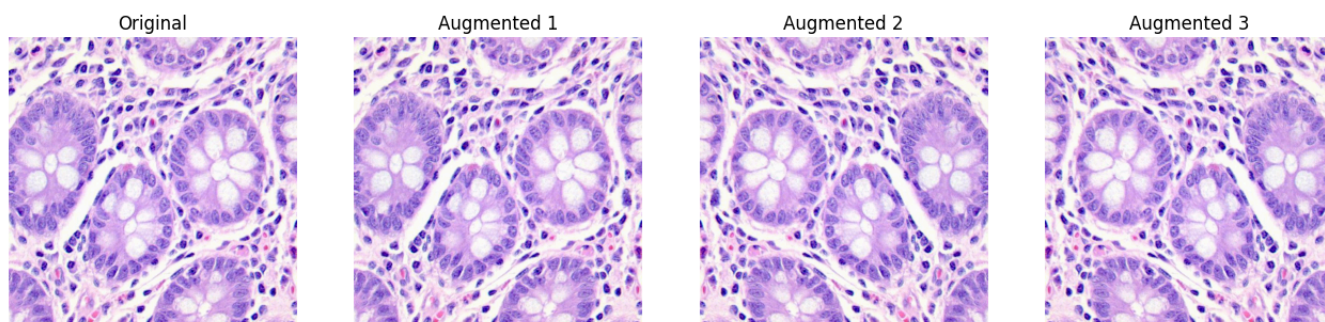
20 plt.axis('off')
21
22 for i in range(num_augmented):
23     batch = next(augmented_images)
24     aug_img = batch[0].astype('uint8')
25     plt.subplot(1, num_augmented + 1, i + 2)
26     plt.title(f'Augmented {i+1}')
27     plt.imshow(aug_img)
28     plt.axis('off')
29 plt.show()
30

```

```

1 datagen = ImageDataGenerator(horizontal_flip=True)
2 display_augmented_images(img_path, datagen)
3

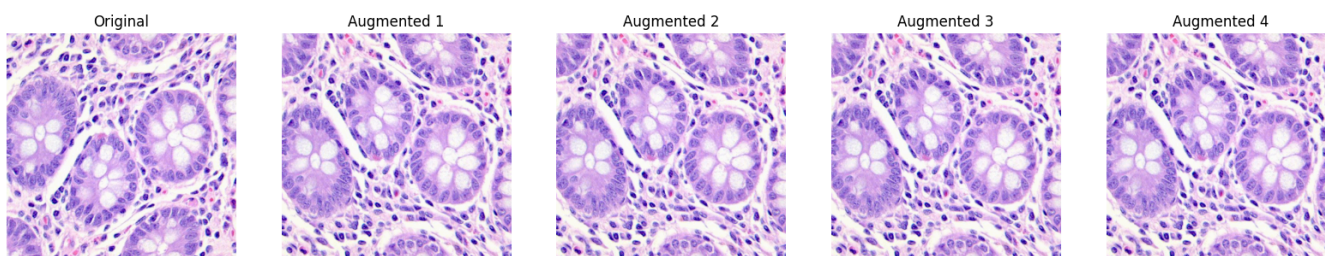
```



```

1 datagen = ImageDataGenerator(vertical_flip=True)
2 display_augmented_images(img_path, datagen, num_augmented=4)
3

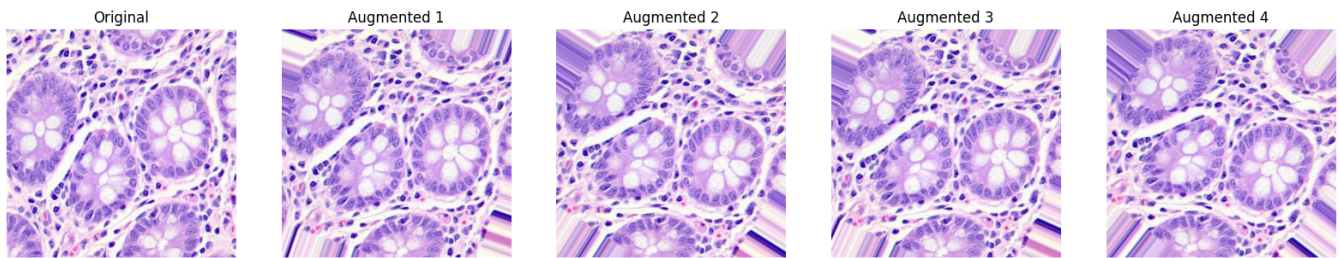
```



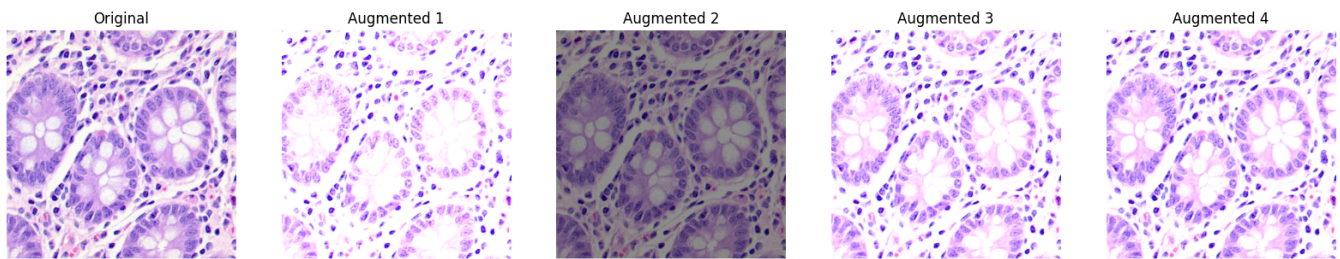
```

1 datagen = ImageDataGenerator(rotation_range=40)
2 display_augmented_images(img_path, datagen, num_augmented=4)
3

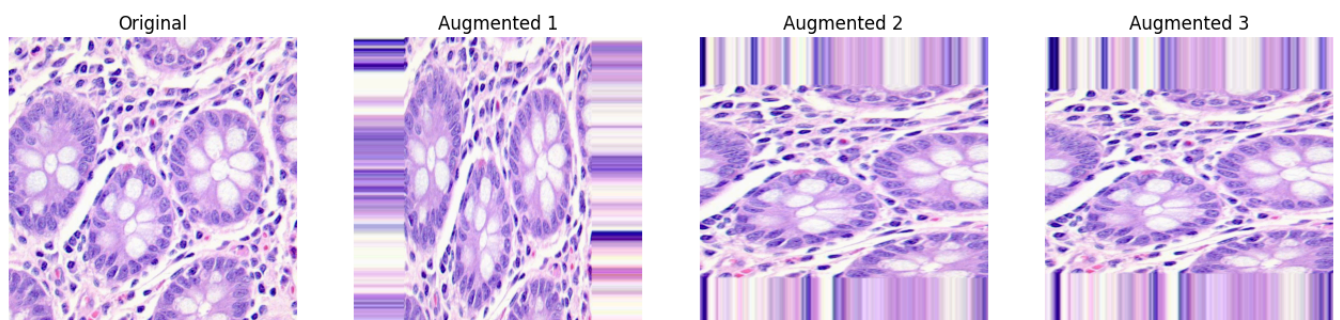
```



```
1 datagen = ImageDataGenerator(brightness_range=[0.5, 1.5])
2 display_augmented_images(img_path, datagen, num_augmented=4)
3
```



```
1 datagen = ImageDataGenerator(zoom_range=0.6)
2 display_augmented_images(img_path, datagen)
3
```



```
1 import os, shutil
2 from sklearn.model_selection import train_test_split
3
4 original_dataset_dir = '/content/drive/MyDrive/image_processing/lung_colon_i
5 base_dir = '/content/lung_dataset_split'
6 train_dir = os.path.join(base_dir, 'train')
7 validation_dir = os.path.join(base_dir, 'validation')
8 test_dir = os.path.join(base_dir, 'test')
9
10 for dir in [train_dir, validation_dir, test_dir]:
11     os.makedirs(dir, exist_ok=True)
12
13 train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2
14
```

```

15 def split_and_copy_files(class_dir, dest_base_dir):
16     all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir)
17                     if os.path.isfile(os.path.join(class_dir, f))]
18
19     train_images, temp_images = train_test_split(all_images, test_size=(1 -
20 validation_images, test_images = train_test_split(temp_images, test_size=
21
22     for images, folder_name in zip([train_images, validation_images, test_im
23         dest_dir = os.path.join(dest_base_dir, folder_name, os.path.basename
24         os.makedirs(dest_dir, exist_ok=True)
25         for file in images:
26             shutil.copy(file, dest_dir)
27
28 # (لو موجودة) lung_scc تطبيق على كل مجلد تصنيف داخل
29 for class_dir in os.listdir(original_dataset_dir):
30     full_class_dir = os.path.join(original_dataset_dir, class_dir)
31     if os.path.isdir(full_class_dir):
32         split_and_copy_files(full_class_dir, base_dir)
33

```

```

1 import tensorflow as tf
2 from tensorflow.keras.preprocessing.image import ImageDataGenerator
3 from tensorflow.keras import layers, models
4
5 # إعداد مولد البيانات
6 train_path = '/content/lung_dataset_split/train'
7 val_path = '/content/lung_dataset_split/validation'
8
9 # تعيين حجم الصور
10 img_height, img_width = 150, 150
11 batch_size = 16
12
13 train_datagen = ImageDataGenerator(rescale=1./255)
14 val_datagen = ImageDataGenerator(rescale=1./255)
15
16 train_generator = train_datagen.flow_from_directory(
17     train_path,
18     target_size=(img_height, img_width),
19     batch_size=batch_size,
20     class_mode='categorical'
21 )
22
23 val_generator = val_datagen.flow_from_directory(
24     val_path,
25     target_size=(img_height, img_width),
26     batch_size=batch_size,

```

```
27     class_mode='categorical'
28 )
29
30 # بناء نموذج CNN
31 model = models.Sequential([
32     layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img.
33     layers.MaxPooling2D(2,2),
34
35     layers.Conv2D(64, (3,3), activation='relu'),
36     layers.MaxPooling2D(2,2),
37
38     layers.Conv2D(128, (3,3), activation='relu'),
39     layers.MaxPooling2D(2,2),
40
41     layers.Flatten(),
42     layers.Dense(128, activation='relu'),
43     layers.Dense(train_generator.num_classes, activation='softmax')
44 ])
45
46 # تجميع النموذج
47 model.compile(optimizer='adam',
48               loss='categorical_crossentropy',
49               metrics=['accuracy'])
50
51 model.summary()
52
```



Found 0 images belonging to 0 classes.

Found 0 images belonging to 0 classes.

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 128)	4,735,104
dense_1 (Dense)	(None, 0)	0

Total params: 4,828,352 (18.42 MB)

Trainable params: 4,828,352 (18.42 MB)



```
1 import os
2
3 train_root = '/content/lung_dataset_split/train'
4 for root, dirs, files in os.walk(train_root):
5     print(f"📁 {root} → 📄 {len(files)} ملفات")
6
```



📁 /content/lung_dataset_split/train → 📄 0 ملفات

```
1 print("تصنيفات داخل train:")
2 print(os.listdir('/content/lung_dataset_split/train'))
3
```



تصنيفات داخل train:

```
[]
```

```
1 import os, shutil
2 from sklearn.model_selection import train_test_split
3
4 original_dataset_dir = '/content/drive/MyDrive/image_processing/lung_colon_i
5 base_dir = '/content/lung_dataset_split'
```

```

6 train_dir = os.path.join(base_dir, 'train')
7 validation_dir = os.path.join(base_dir, 'validation')
8 test_dir = os.path.join(base_dir, 'test')
9
10 for dir in [train_dir, validation_dir, test_dir]:
11     os.makedirs(dir, exist_ok=True)
12
13 train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2
14
15 def split_and_copy_files(class_dir, dest_base_dir):
16     all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir)
17                    if os.path.isfile(os.path.join(class_dir, f))]
18
19     train_images, temp_images = train_test_split(all_images, test_size=(1 -
20 validation_ratio), random_state=42)
21     validation_images, test_images = train_test_split(temp_images, test_size=
22 test_ratio, random_state=42)
23
24     for images, folder_name in zip([train_images, validation_images, test_im
25 ages], ['train', 'validation', 'test']):
26         dest_dir = os.path.join(dest_base_dir, folder_name, os.path.basename
27 class_dir)
28         os.makedirs(dest_dir, exist_ok=True)
29         for file in images:
30             shutil.copy(file, dest_dir)
31
32 # تطبيق على كل مجلد تصنيف داخل lung_scc
33 for class_dir in os.listdir(original_dataset_dir):
34     full_class_dir = os.path.join(original_dataset_dir, class_dir)
35     if os.path.isdir(full_class_dir):
36         split_and_copy_files(full_class_dir, base_dir)
37

```

```

1 from sklearn.model_selection import train_test_split
2 import os, shutil
3 # يتم تقسيم البيانات إلى: تدريب - تحقق - اختبار
4

```

```

1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
2

```

```

1 train_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/tr
2 val_dir    = '/content/drive/MyDrive/image_processing/lung_colon_image_set/va
3 test_dir   = '/content/drive/MyDrive/image_processing/lung_colon_image_set/te
4

```

```

1 import os
2 import numpy as np

```

```

3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 from tensorflow.keras.preprocessing.image import ImageDataGenerator
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
8 from sklearn.metrics import classification_report, confusion_matrix
9
10
11 import os
12 import shutil
13 import random
14 from glob import glob
15
16 # المسار الأصلي للبيانات
17 source_root = '/content/drive/MyDrive/image_processing/lung_colon_image_set'
18 destination_root = os.path.join(source_root, 'dataset')
19
20 # النسب
21 train_split = 0.7
22 val_split = 0.15
23 test_split = 0.15
24
25 # التأكد من إنشاء مجلدات train/val/test
26 splits = ['train', 'val', 'test']
27 for split in splits:
28     os.makedirs(os.path.join(destination_root, split), exist_ok=True)
29
30 # lung_scc و colon_image_sets البحث في
31 class_dirs = glob(os.path.join(source_root, '*'))
32 class_dirs = [d for d in class_dirs if os.path.isdir(d) and 'dataset' not in d]
33
34 # لكل فئة
35 for class_dir in class_dirs:
36     class_name = os.path.basename(class_dir)
37     print(f'📁 المعالجة: {class_name}')
38
39     # جمع كل الصور
40     images = glob(os.path.join(class_dir, '*.jpeg')) + glob(os.path.join(class_dir, '*.png'))
41     random.shuffle(images)
42
43     # تحديد عدد الصور
44     total = len(images)
45     train_end = int(train_split * total)
46     val_end = train_end + int(val_split * total)
47
48     # توزيع الصور

```

```

39     split_map = {
40         'train': images[:train_end],
41         'val': images[train_end:val_end],
42         'test': images[val_end:]
43     }
44
45     for split, files in split_map.items():
46         split_dir = os.path.join(destination_root, split, class_name)
47         os.makedirs(split_dir, exist_ok=True)
48         for img in files:
49             shutil.copy(img, split_dir)
50
51 print("\n✅ تم التقسيم بنجاح.")
52

```

 المعالجة: lung_scc
 المعالجة: colon_image_sets

✅ تم التقسيم بنجاح.

```

1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import tensorflow as tf
5 from tensorflow.keras.preprocessing.image import ImageDataGenerator
6 from tensorflow.keras.models import Sequential
7 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
8 from sklearn.metrics import classification_report, confusion_matrix

1 base_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/data'
2 train_dir = os.path.join(base_dir, 'train')
3 val_dir = os.path.join(base_dir, 'val')
4 test_dir = os.path.join(base_dir, 'test')
5
6
1 train_datagen = ImageDataGenerator(
2     rescale=1./255,
3     rotation_range=20,
4     zoom_range=0.2,
5     horizontal_flip=True
6 )
7
8 val_test_datagen = ImageDataGenerator(rescale=1./255)
9

```

```

1 train_generator = train_datagen.flow_from_directory(
2     train_dir, target_size=(150, 150), batch_size=32, class_mode='categorical'
3
4 val_generator = val_test_datagen.flow_from_directory(
5     val_dir, target_size=(150, 150), batch_size=32, class_mode='categorical'
6
7 test_generator = val_test_datagen.flow_from_directory(
8     test_dir, target_size=(150, 150), batch_size=32, class_mode='categorical'
9

```

 Found 5000 images belonging to 2 classes.
 Found 4474 images belonging to 2 classes.
 Found 4370 images belonging to 2 classes.

```

1 model = Sequential([
2     Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
3     MaxPooling2D(2, 2),
4     Conv2D(64, (3, 3), activation='relu'),
5     MaxPooling2D(2, 2),
6     Conv2D(128, (3, 3), activation='relu'),
7     MaxPooling2D(2, 2),
8     Flatten(),
9     Dense(128, activation='relu'),
10    Dropout(0.5),
11    Dense(train_generator.num_classes, activation='softmax')
12 ])
13

```

```

1 model.compile(
2     loss='categorical_crossentropy',
3     optimizer='adam',
4     metrics=['accuracy']
5 )
6

```

```

1 # تصميم النموذج
2 model = Sequential([
3     Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
4     MaxPooling2D(2, 2),
5     Conv2D(64, (3, 3), activation='relu'),
6     MaxPooling2D(2, 2),
7     Conv2D(128, (3, 3), activation='relu'),
8     MaxPooling2D(2, 2),
9     Flatten(),
10    Dense(128, activation='relu'),

```

```

11 Dropout(0.5),
12 Dense(train_generator.num_classes, activation='softmax')
13 ])
14
15 # ⚠ لا تنسَ هذا السطر 📌
16 model.compile(
17     loss='categorical_crossentropy',
18     optimizer='adam',
19     metrics=['accuracy']
20 )
21
22 # ثم درّب النموذج
23 history = model.fit(
24     train_generator,
25     validation_data=val_generator,
26     epochs=10
27 )
28

```

```

➡ /usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:100:
self._warn_if_super_not_called()
Epoch 1/10
157/157 ————— 1778s 11s/step - accuracy: 0.9966 - loss: 0.0226 - val_accuracy: 0.9966
Epoch 2/10
157/157 ————— 336s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/10
157/157 ————— 337s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/10
157/157 ————— 337s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/10
157/157 ————— 391s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/10
157/157 ————— 376s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/10
157/157 ————— 384s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/10
157/157 ————— 337s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/10
157/157 ————— 390s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/10
157/157 ————— 346s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000

```

```

1 from PIL import Image
2 import os
3
4 # Correct test directory path
5 test_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/data'
6
7 def remove_corrupted_images(directory):

```

```

8     corrupted_count = 0
9     for root, _, files in os.walk(directory):
10         for fname in files:
11             if fname.lower().endswith(('jpg', 'jpeg', 'png')):
12                 fpath = os.path.join(root, fname)
13                 try:
14                     img = Image.open(fpath)
15                     img.verify() # Check without loading the image data into memory
16                 except Exception as e:
17                     print(f"❌ Deleting corrupted image: {fpath}")
18                     os.remove(fpath)
19                     corrupted_count += 1
20     print(f"\n✅ Removed {corrupted_count} corrupted images.")
21
22 remove_corrupted_images(test_dir)

```



✅ Removed 0 corrupted images.

```

1 from torchvision import datasets, transforms
2 from torch.utils.data import DataLoader
3
4 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/data'
5
6 transform = {
7     'train': transforms.Compose([
8         transforms.Resize((224, 224)),
9         transforms.RandomHorizontalFlip(),
10        transforms.ToTensor(),
11        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
12    ]),
13    'val': transforms.Compose([
14        transforms.Resize((224, 224)),
15        transforms.ToTensor(),
16        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
17    ]),
18    'test': transforms.Compose([
19        transforms.Resize((224, 224)),
20        transforms.ToTensor(),
21        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
22    ])
23 }
24
25 batch_size = 32
26
27 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x), transform)

```

```

28 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size, shuffl
29
30 # الآن نعرف class_names
31 class_names = dataloaders['train'].dataset.classes
32 print("عدد الفئات:", len(class_names))
33 print("أسماء الفئات:", class_names)
34

```

→ عدد الفئات: 2
أسماء الفئات: ['colon_image_sets', 'lung_scc']

```

1 import torch
2
3 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
4 print("الجهاز المستخدم:", device)
5

```

→ الجهاز المستخدم: cpu

```

1 model_resnet = model_resnet.to(device)

```

1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

```

1 المكتبات 1.
2 import torch
3 import torchvision
4 from torchvision import datasets, transforms, models
5 from torch.utils.data import DataLoader
6 import torch.nn as nn
7 import torch.optim as optim
8 import os
9 import matplotlib.pyplot as plt
10 import numpy as np
11 from sklearn.metrics import classification_report, confusion_matrix,
12 from sklearn.preprocessing import label_binarize

```

```

13
14 2. تحديد device
15 device = torch.device("cuda" if torch.cuda.is_available() else "cpu"
16 print("الجهاز المستخدم:", device)
17
18 3. تحميل البيانات
19 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_image
20
21 transform = {
22     'train': transforms.Compose([
23         transforms.Resize((224,224)),
24         transforms.RandomHorizontalFlip(),
25         transforms.ToTensor(),
26         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0
27     ]),
28     'val': transforms.Compose([
29         transforms.Resize((224,224)),
30         transforms.ToTensor(),
31         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0
32     ]),
33     'test': transforms.Compose([
34         transforms.Resize((224,224)),
35         transforms.ToTensor(),
36         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0
37     ])
38
39
40 batch_size = 32
41 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x),
42 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size
43 class_names = dataloaders['train'].dataset.classes
44 print("عدد الفئات:", len(class_names))
45 print("أسماء الفئات:", class_names)
46
47 4. بناء نموذج ResNet18
48 model_resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAU
49 model_resnet.fc = nn.Linear(model_resnet.fc.in_features, len(class_n
50 model_resnet = model_resnet.to(device)
51
52 5. Optimizer و Loss
53 criterion = nn.CrossEntropyLoss()
54 optimizer = optim.Adam(model_resnet.fc.parameters(), lr=0.001)
55
56 6. التدريب
57 epochs = 5
58 for epoch in range(epochs):
59     model_resnet.train()

```

```

60     running_loss, correct, total = 0.0, 0, 0
61
62     for inputs, labels in dataloaders['train']:
63         inputs, labels = inputs.to(device), labels.to(device)
64         optimizer.zero_grad()
65         outputs = model_resnet(inputs)
66         loss = criterion(outputs, labels)
67         loss.backward()
68         optimizer.step()
69
70         running_loss += loss.item()
71         _, preds = torch.max(outputs, 1)
72         correct += (preds == labels).sum().item()
73         total += labels.size(0)
74
75     print(f"Epoch {epoch+1}/{epochs} - Loss: {running_loss/len(datalo
76
77 7. التقييم على test set
78 model_resnet.eval()
79 y_true, y_pred, y_probs = [], [], []
80
81 with torch.no_grad():
82     for inputs, labels in dataloaders['test']:
83         inputs = inputs.to(device)
84         outputs = model_resnet(inputs)
85         probs = nn.functional.softmax(outputs, dim=1)
86         _, preds = torch.max(probs, 1)
87
88         y_true.extend(labels.numpy())
89         y_pred.extend(preds.cpu().numpy())
90         y_probs.extend(probs.cpu().numpy())
91
92 y_true = np.array(y_true)
93 y_pred = np.array(y_pred)
94 y_probs = np.array(y_probs)
95
96 8. التقارير
97 print("📊 Classification Report:")
98 print(classification_report(y_true, y_pred, target_names=class_names
99
100 print("🔍 Confusion Matrix:")
101 print(confusion_matrix(y_true, y_pred))
102
103 9. ROC Curve و AUC
104 y_true_bin = label_binarize(y_true, classes=range(len(class_names)))
105
106 for i, class_name in enumerate(class_names):

```

```

107 fpr, tpr, _ = roc_curve(y_true_bin[:, i], y_probs[:, i])
108 auc_score = roc_auc_score(y_true_bin[:, i], y_probs[:, i])
109 plt.plot(fpr, tpr, label=f'{class_name} (AUC = {auc_score:.2f})'
110
111 lt.plot([0,1], [0,1], 'k--')
112 lt.title('ROC Curve - ResNet18')
113 lt.xlabel('False Positive Rate')
114 lt.ylabel('True Positive Rate')
115 lt.legend()
116 lt.show()
117

```



الجهاز المستخدم: cpu

عدد الفئات: 1

أسماء الفئات: ['lung_scc']

Epoch 1/5 - Loss: 0.0000 - Accuracy: 1.0000

Epoch 2/5 - Loss: 0.0000 - Accuracy: 1.0000

Epoch 3/5 - Loss: 0.0000 - Accuracy: 1.0000

Epoch 4/5 - Loss: 0.0000 - Accuracy: 1.0000

Epoch 5/5 - Loss: 0.0000 - Accuracy: 1.0000

UnidentifiedImageError Traceback (most recent call last)
 /tmp/ipython-input-55-2688606112.py in <cell line: 0>()

```

80
81 with torch.no_grad():
---> 82     for inputs, labels in dataloaders['test']:
83         inputs = inputs.to(device)
84         outputs = model_resnet(inputs)

```

7 frames

```

/usr/local/lib/python3.11/dist-packages/PIL/Image.py in open(fp, mode, formats)
3570     warnings.warn(message)
3571     msg = "cannot identify image file %r" % (filename if filename else fp)
-> 3572     raise UnidentifiedImageError(msg)
3573
3574

```

UnidentifiedImageError: cannot identify image file <_io.BufferedReader
 name='/content/drive/MyDrive/image_processing/lung_colon_image_set/dataset_fixed/test/lu

Next steps: [Explain error](#)

```

1 print("فئات موجودة في test set:", test_generator.class_indices)
2 print("القيم الحقيقية في y_true:", np.unique(y_true))
3

```



test set: {'colon_image_sets': 0, 'lung_scc': 1}
 y_true: [0]

```
1 from PIL import Image
2 import os
3
4 def remove_corrupted_images(directory):
5     corrupted = 0
6     for root, _, files in os.walk(directory):
7         for file in files:
8             if file.lower().endswith(('jpg', 'jpeg', 'png')):
9                 file_path = os.path.join(root, file)
10                try:
11                    img = Image.open(file_path)
12                    img.verify()
13                except Exception as e:
14                    print(f"❌ حذف الصورة التالفة: {file_path}")
15                    os.remove(file_path)
16                    corrupted += 1
17            print(f"\n✅ تم حذف {corrupted} صورة تالفة من {directory}")
18
19 remove_corrupted_images('/content/drive/MyDrive/image_processing/lung_colon_
20
```



✅ تم حذف 0 صورة تالفة من /content/drive/MyDrive/image_processing/lung_colon_image_set/data

```

1 from torchvision import transforms
2
3 transform = {
4     'train': transforms.Compose([
5         transforms.Resize((224,224)),
6         transforms.RandomHorizontalFlip(),
7         transforms.ToTensor(),
8         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
9     ]),
10    'val': transforms.Compose([
11        transforms.Resize((224,224)),
12        transforms.ToTensor(),
13        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
14    ]),
15    'test': transforms.Compose([
16        transforms.Resize((224,224)),
17        transforms.ToTensor(),
18        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
19    ])
20 }
21

```

```

1 import os
2 import shutil
3 import random
4 from glob import glob
5
6 # المسار الأصلي للصور
7 source_root = '/content/drive/MyDrive/image_processing/lung_colon_ir
8 destination_root = os.path.join(source_root, 'dataset_fixed')
9
10 # نسب التقسيم
11 train_ratio = 0.7
12 val_ratio = 0.15
13 test_ratio = 0.15
14
15 # حذف المجلد إذا كان موجود لإعادة التقسيم من جديد (اختياري)
16 if os.path.exists(destination_root):
17     shutil.rmtree(destination_root)
18     print("✅ تم حذف المجلد القديم dataset_fixed")
19
20 # إنشاء مجلدات train / val / test
21 splits = ['train', 'val', 'test']
22 for split in splits:
23     os.makedirs(os.path.join(destination_root, split), exist_ok=True)
24

```

```

25 # استكشف الفئات في المجلد الأصلي
26 class_dirs = glob(os.path.join(source_root, '*'))
27 class_dirs = [d for d in class_dirs if os.path.isdir(d) and 'dataset'
28
29 for class_dir in class_dirs:
30     dest_class_name = os.path.basename(class_dir)
31     print(f"🔍 معالجة الفئة: {dest_class_name}")
32
33     images = glob(os.path.join(class_dir, '*.jpeg')) + glob(os.path.
34     random.shuffle(images)
35
36     total = len(images)
37     train_end = int(train_ratio * total)
38     val_end = train_end + int(val_ratio * total)
39
40     splits_images = {
41         'train': images[:train_end],
42         'val': images[train_end:val_end],
43         'test': images[val_end:]
44     }
45
46     for split, split_images in splits_images.items():
47         dest_dir = os.path.join(destination_root, split, dest_class_
48         os.makedirs(dest_dir, exist_ok=True)
49         for img in split_images:
50             shutil.copy(img, dest_dir)
51
52 print("\n✅ تم التقسيم بنجاح إلى dataset_fixed")
53

```



✅ تم حذف المجلد القديم dataset_fixed

🔍 معالجة الفئة: lung_scc

🔍 معالجة الفئة: colon_image_sets

🔍 معالجة الفئة: dataset

✅ تم التقسيم بنجاح إلى dataset_fixed

```

1 base_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/dat
2 train_dir = os.path.join(base_dir, 'train')
3 val_dir = os.path.join(base_dir, 'val')
4 test_dir = os.path.join(base_dir, 'test')
5

```

```

1 from tensorflow.keras.preprocessing.image import ImageDataGenerator
2
3 train_datagen = ImageDataGenerator(
4     rescale=1./255,

```

```

5     rotation_range=20,
6     zoom_range=0.2,
7     horizontal_flip=True
8 )
9
10 val_test_datagen = ImageDataGenerator(rescale=1./255)
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

 Found 3500 images belonging to 3 classes.
 Found 750 images belonging to 3 classes.
 Found 750 images belonging to 3 classes.

```

1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
3
4 model = Sequential([
5     Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
6     MaxPooling2D(2,2),
7     Conv2D(64, (3,3), activation='relu'),
8     MaxPooling2D(2,2),
9     Conv2D(128, (3,3), activation='relu'),
10    MaxPooling2D(2,2),
11    Flatten(),
12    Dense(128, activation='relu'),
13    Dropout(0.5),
14    Dense(train_generator.num_classes, activation='softmax')
15 ])
16
17 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['a
18

```

 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

```

1 history = model.fit(
2     train_generator,
3     validation_data=val_generator,
4     epochs=10
5 )
6

```

```

➔ /usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:104:
  self._warn_if_super_not_called()
Epoch 1/10
110/110 ————— 215s 2s/step - accuracy: 0.9853 - loss: 0.0541 - val_accuracy: 0.9853
Epoch 2/10
110/110 ————— 200s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/10
110/110 ————— 209s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/10
110/110 ————— 207s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/10
110/110 ————— 199s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/10
110/110 ————— 205s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/10
110/110 ————— 201s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 8/10
110/110 ————— 198s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/10
110/110 ————— 199s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/10
110/110 ————— 206s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000

```

```

1 loss, acc = model.evaluate(test_generator)
2 print(f"✅ Test Accuracy: {acc*100:.2f}%")
3

```

```

➔ 24/24 ————— 12s 491ms/step - accuracy: 1.0000 - loss: 0.0000e+00
✅ Test Accuracy: 100.00%

```

```

1 import os
2 from torchvision import datasets, transforms
3 from torch.utils.data import DataLoader
4
5 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/data'
6 import os
7
8 train_path = os.path.join(data_dir, 'train')
9
10 for folder in os.listdir(train_path):
11     folder_path = os.path.join(train_path, folder)
12     if os.path.isdir(folder_path):

```

```

13     images = [img for img in os.listdir(folder_path) if img.lower().ends
14     print(f'{folder}: {len(images)} صور')
15
16 import shutil
17
18 wrong_folders = ['colon_image_sets', 'dataset']
19 for folder in wrong_folders:
20     for split in ['train', 'val', 'test']:
21         path = os.path.join(data_dir, split, folder)
22         if os.path.exists(path):
23             shutil.rmtree(path)
24             print(f'✗ تم حذف المجلد الغير صحيح {path}')
25
26 transform = {
27     'train': transforms.Compose([
28         transforms.Resize((224,224)),
29         transforms.RandomHorizontalFlip(),
30         transforms.ToTensor(),
31         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
32     ]),
33     'val': transforms.Compose([
34         transforms.Resize((224,224)),
35         transforms.ToTensor(),
36         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
37     ]),
38     'test': transforms.Compose([
39         transforms.Resize((224,224)),
40         transforms.ToTensor(),
41         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
42     ])
43 }
44
45 batch_size = 32
46
47 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x), transform)
48 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size, shuffle=
49
50 class_names = dataloaders['train'].dataset.classes
51 print("عدد الفئات:", len(class_names))
52 print("أسماء الفئات:", class_names)
53

```



```

lung_scc: 3500 صور
colon_image_sets: 0 صور
dataset: 0 صور

```

```

✗ تم حذف المجلد الغير صحيح /content/drive/MyDrive/image_processing/lung_colon_image_set/dat
✗ تم حذف المجلد الغير صحيح /content/drive/MyDrive/image_processing/lung_colon_image_set/dat

```

✖ تم حذف المجلد الغير صحيح: /content/drive/MyDrive/image_processing/lung_colon_image_set/dat
 ✖ تم حذف المجلد الغير صحيح: /content/drive/MyDrive/image_processing/lung_colon_image_set/dat
 ✖ تم حذف المجلد الغير صحيح: /content/drive/MyDrive/image_processing/lung_colon_image_set/dat
 ✖ تم حذف المجلد الغير صحيح: /content/drive/MyDrive/image_processing/lung_colon_image_set/dat
 عدد الفئات: 1
 أسماء الفئات: ['lung_scc']

```

1 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x), transform=transform) for x in os.listdir(data_dir)}
2 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size, shuffle=True) for x in ['train', 'val']}
3 class_names = dataloaders['train'].dataset.classes
4 print("عدد الفئات:", len(class_names))
5 print("أسماء الفئات:", class_names)
6

```

➡ عدد الفئات: 1
 أسماء الفئات: ['lung_scc']

```

1 import os
2 from torchvision import datasets, transforms
3 from torch.utils.data import DataLoader
4
5 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/data'
6
7 transform = {
8     'train': transforms.Compose([
9         transforms.Resize((224,224)),
10        transforms.RandomHorizontalFlip(),
11        transforms.ToTensor(),
12        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
13    ]),
14    'val': transforms.Compose([
15        transforms.Resize((224,224)),
16        transforms.ToTensor(),
17        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
18    ]),
19    'test': transforms.Compose([
20        transforms.Resize((224,224)),
21        transforms.ToTensor(),
22        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
23    ])
24 }
25
26 batch_size = 32
27
28 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x), transform=transform) for x in os.listdir(data_dir)}
29 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size, shuffle=True) for x in ['train', 'val']}
30

```

```

30
31 class_names = dataloaders['train'].dataset.classes
32 print("عدد الفئات:", len(class_names))
33 print("أسماء الفئات:", class_names)
34

```

→ عدد الفئات: 1
أسماء الفئات: ['lung_scc']

```

1 from sklearn.metrics import classification_report, confusion_matrix
2
3 print("📊 Classification Report:")
4 print(classification_report(y_true, y_pred, labels=np.unique(y_true)))
5
6 print("🔍 Confusion Matrix:")
7 print(confusion_matrix(y_true, y_pred))
8

```

→ 📊 Classification Report:

	precision	recall	f1-score	support
2	1.00	1.00	1.00	750
accuracy			1.00	750
macro avg	1.00	1.00	1.00	750
weighted avg	1.00	1.00	1.00	750

🔍 Confusion Matrix:

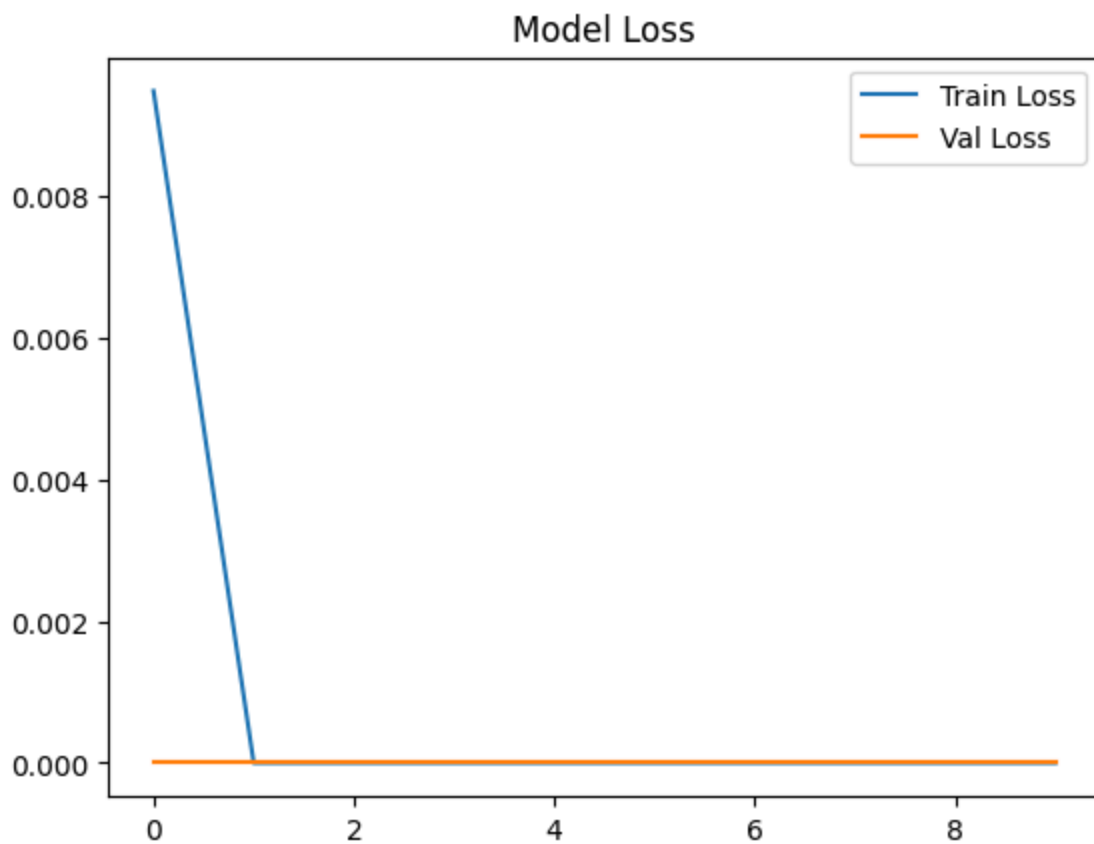
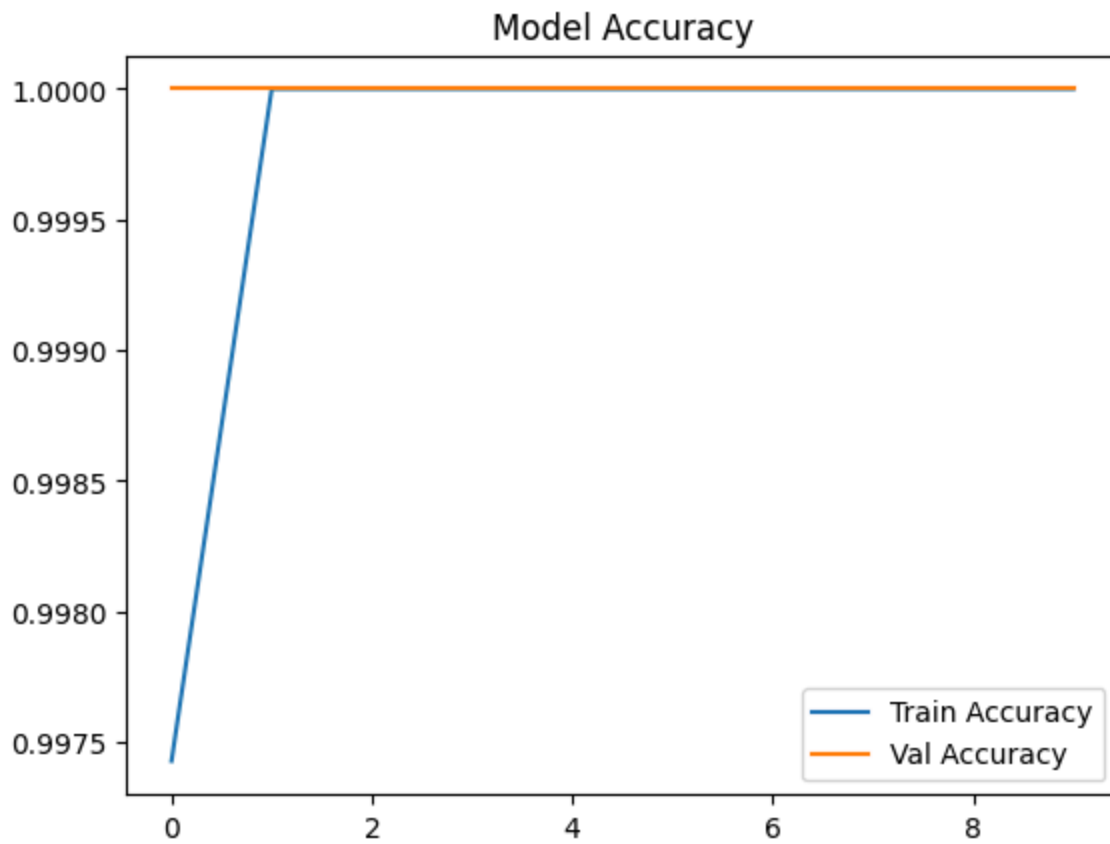
[[750]]

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning: warnings.warn(

```

1 import matplotlib.pyplot as plt
2
3 plt.plot(history.history['accuracy'], label='Train Accuracy')
4 plt.plot(history.history['val_accuracy'], label='Val Accuracy')
5 plt.title('Model Accuracy')
6 plt.legend()
7 plt.show()
8
9 plt.plot(history.history['loss'], label='Train Loss')
10 plt.plot(history.history['val_loss'], label='Val Loss')
11 plt.title('Model Loss')
12 plt.legend()
13 plt.show()
14

```



```
1 model.save('/content/drive/MyDrive/image_processing/lung_model_cnn.h5')  
2
```

 WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savi`

```

1 from PIL import Image
2 import os
3
4 def remove_corrupted_images(directory):
5     corrupted = 0
6     for root, _, files in os.walk(directory):
7         for file in files:
8             if file.lower().endswith(('jpg', 'jpeg', 'png')):
9                 filepath = os.path.join(root, file)
10                try:
11                    img = Image.open(filepath)
12                    img.verify()
13                except Exception:
14                    print(f"❌ حذف الصورة التالفة: {filepath}")
15                    os.remove(filepath)
16                    corrupted += 1
17            print(f"\n✅ تم حذف {corrupted} صورة تالفة من {directory}")
18
19 base_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/dat
20
21 for split in ['train', 'val', 'test']:
22     remove_corrupted_images(os.path.join(base_dir, split))
23
24 # Recreate the test_generator after removing corrupted images
25 test_dir = os.path.join(base_dir, 'test')
26 test_generator = val_test_datagen.flow_from_directory(
27     test_dir, target_size=(150,150), batch_size=32, class_mode='categorical'

```



✅ تم حذف 0 صورة تالفة من /content/drive/MyDrive/image_processing/lung_colon_image_set/data

✅ تم حذف 0 صورة تالفة من /content/drive/MyDrive/image_processing/lung_colon_image_set/data

✅ تم حذف 0 صورة تالفة من /content/drive/MyDrive/image_processing/lung_colon_image_set/data

Found 750 images belonging to 1 classes.

```

1 loss, acc = model.evaluate(test_generator)
2 print(f"✅ Test Accuracy: {acc*100:.2f}%")

```



/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:100: self._warn_if_super_not_called()

24/24 ————— 13s 494ms/step - accuracy: 0.0000e+00 - loss: 692.6285

✅ Test Accuracy: 0.00%

```
1 import torch
2 import torchvision
3 from torchvision import datasets, transforms, models
4 from torch.utils.data import DataLoader
5 import torch.nn as nn
6 import torch.optim as optim
7 import matplotlib.pyplot as plt
8 import numpy as np
9 from sklearn.metrics import classification_report, confusion_matrix, roc_auc
10 import os
11
```

```
1 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_image_set/dat
2
3 # Augmentation + Normalization
4 transform = {
5     'train': transforms.Compose([
6         transforms.Resize((224, 224)),
7         transforms.RandomHorizontalFlip(),
8         transforms.ToTensor(),
9         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
10     ]),
11     'val': transforms.Compose([
12         transforms.Resize((224, 224)),
13         transforms.ToTensor(),
14         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
15     ]),
16     'test': transforms.Compose([
17         transforms.Resize((224, 224)),
18         transforms.ToTensor(),
19         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
20     ]),
21 }
22
23 batch_size = 32
24
```

```
1 import os
2
3 os.listdir('/content/drive/MyDrive/image_processing/lung_colon_image_set/dat
4
5
6 import os
7
```

```

8 train_path = '/content/drive/MyDrive/image_processing/lung_colon_image_set/d
9
10 for folder in os.listdir(train_path):
11     folder_path = os.path.join(train_path, folder)
12     if os.path.isdir(folder_path):
13         print(f'📁 {folder} يحتوي: {len(os.listdir(folder_path))} صور')
14
15

```

📁 lung_scc يحتوي: 3500 صور

```

1 import torch
2 import torchvision
3 from torchvision import datasets, transforms, models
4 from torch.utils.data import DataLoader
5 import torch.nn as nn
6 import torch.optim as optim
7 import os
8 import matplotlib.pyplot as plt
9 import numpy as np
10 from sklearn.metrics import classification_report, confusion_matrix
11 from sklearn.preprocessing import label_binarize
12
13 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
14
15 data_dir = '/content/drive/MyDrive/image_processing/lung_colon_imag
16
17 transform = {
18     'train': transforms.Compose([
19         transforms.Resize((224,224)),
20         transforms.RandomHorizontalFlip(),
21         transforms.ToTensor(),
22         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
23     ]),
24     'val': transforms.Compose([
25         transforms.Resize((224,224)),
26         transforms.ToTensor(),
27         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
28     ]),
29     'test': transforms.Compose([
30         transforms.Resize((224,224)),
31         transforms.ToTensor(),
32         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224,
33     ])
34 }
35
36 batch_size = 32

```

```

36 batch_size = 32
37
38 datasets_full = {x: datasets.ImageFolder(os.path.join(data_dir, x),
39 dataloaders = {x: DataLoader(datasets_full[x], batch_size=batch_size)
40 class_names = dataloaders['train'].dataset.classes
41
42 from torchvision.models import efficientnet_b0
43 model = efficientnet_b0(weights='DEFAULT')
44 model.classifier[1] = nn.Linear(model.classifier[1].in_features, 1000)
45 model = model.to(device)
46
47 criterion = nn.CrossEntropyLoss()
48 optimizer = optim.Adam(model.parameters(), lr=0.001)
49
50 epochs = 5
51 for epoch in range(epochs):
52     model.train()
53     running_loss, correct, total = 0.0, 0, 0
54     for inputs, labels in dataloaders['train']:
55         inputs, labels = inputs.to(device), labels.to(device)
56         optimizer.zero_grad()
57         outputs = model(inputs)
58         loss = criterion(outputs, labels)
59         loss.backward()
60         optimizer.step()
61
62         running_loss += loss.item()
63         _, preds = torch.max(outputs, 1)
64         correct += (preds == labels).sum().item()
65         total += labels.size(0)
66
67     print(f"Epoch {epoch+1}/{epochs} - Loss: {running_loss/len(data_loader['train'])}")
68
69 model.eval()
70 y_true, y_pred, y_probs = [], [], []
71
72 with torch.no_grad():
73     for inputs, labels in dataloaders['test']:
74         inputs = inputs.to(device)
75         outputs = model(inputs)
76         probs = nn.functional.softmax(outputs, dim=1)
77         _, preds = torch.max(probs, 1)
78
79         y_true.extend(labels.cpu().numpy())
80         y_pred.extend(preds.cpu().numpy())
81         y_probs.extend(probs.cpu().numpy())
82

```

```
83 y_true = np.array(y_true)
84 y_pred = np.array(y_pred)
85 y_probs = np.array(y_probs)
86
87 print("📊 Classification Report:")
88 print(classification_report(y_true, y_pred, target_names=class_name)
89
90 print("🔍 Confusion Matrix:")
91 print(confusion_matrix(y_true, y_pred))
92
93 y_true_bin = label_binarize(y_true, classes=range(len(class_names)))
94
95 for i, class_name in enumerate(class_names):
96     fpr, tpr, _ = roc_curve(y_true_bin[:, i], y_probs[:, i])
97     auc_score = roc_auc_score(y_true_bin[:, i], y_probs[:, i])
98     plt.plot(fpr, tpr, label=f'{class_name} (AUC = {auc_score:.2f})')
99
100 plt.plot([0,1], [0,1], 'k--')
101 plt.title('ROC Curve - EfficientNetB0')
102 plt.xlabel('False Positive Rate')
103 plt.ylabel('True Positive Rate')
104 plt.legend()
105 plt.show()
106
```

```

Epoch 1/5 - Loss: 0.0000 - Accuracy: 1.0000
Epoch 2/5 - Loss: 0.0000 - Accuracy: 1.0000
Epoch 3/5 - Loss: 0.0000 - Accuracy: 1.0000
Epoch 4/5 - Loss: 0.0000 - Accuracy: 1.0000
Epoch 5/5 - Loss: 0.0000 - Accuracy: 1.0000

```

🇧🇷 Classification Report:

	precision	recall	f1-score	support
lung_scc	1.00	1.00	1.00	750
accuracy			1.00	750
macro avg	1.00	1.00	1.00	750
weighted avg	1.00	1.00	1.00	750

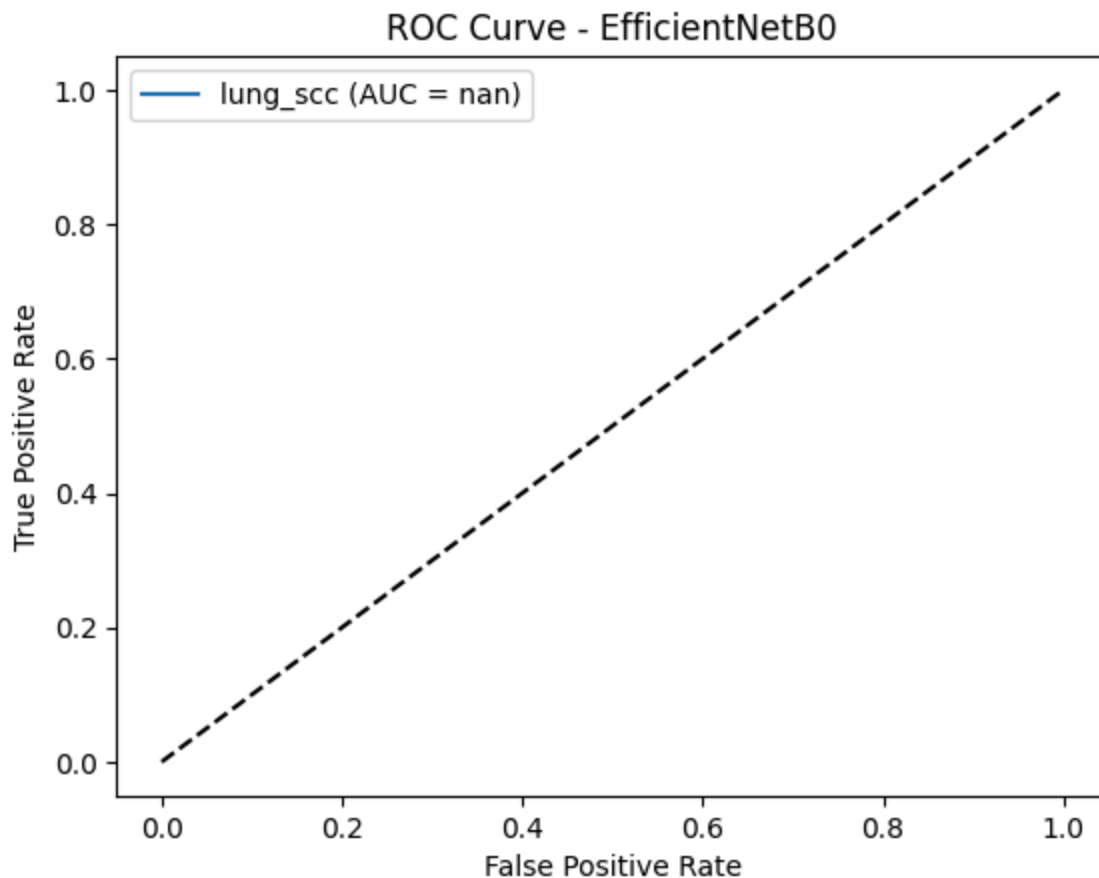
✖ Confusion Matrix:

[[750]]

```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetricWarning:
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:379: UndefinedMetricWarning:
  warnings.warn(

```



```

1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.Adam(model.classifier[1].parameters(), lr=0.001)
3

```

```

1 from torchvision.models import resnet18
2
3 # تحميل ResNet18 مع weights pretrained
4 model_resnet = resnet18(pretrained=True)
5
6 # تجميد الطبقات
7 for param in model_resnet.parameters():
8     param.requires_grad = False
9
10 # تعديل الطبقة الأخيرة حسب عدد الفئات
11 num_ftrs = model_resnet.fc.in_features
12 model_resnet.fc = nn.Linear(num_ftrs, len(class_names))
13 model_resnet = model_resnet.to(device)
14

```

 /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: 1
 warnings.warn(
 /usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: 1
 warnings.warn(msg)

```

1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.Adam(model_resnet.fc.parameters(), lr=0.001)
3

```

```

1 from torchvision.models import efficientnet_b0
2 import torch.nn as nn
3
4 model = efficientnet_b0(weights='DEFAULT')
5
6 # تعديل الطبقة النهائية حسب عدد الفئات
7 model.classifier[1] = nn.Linear(model.classifier[1].in_features, len(class_n
8
9 # Freeze أولاً كل الطبقات
10 for param in model.parameters():
11     param.requires_grad = False
12
13 # Unfreeze آخر block + التصنيف
14 for name, param in model.named_parameters():
15     if 'features.6' in name or 'classifier' in name:
16         param.requires_grad = True
17
18 model = model.to(device)
19

```

```

1 torch.save(model.state_dict(), 'fine_tuned_model.pth')
2

1 import torch.optim as optim
2
3 optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()))
4

1 epochs = 5
2 for epoch in range(epochs):
3     model.train()
4     running_loss, correct, total = 0.0, 0, 0
5
6     for inputs, labels in dataloaders['train']:
7         inputs, labels = inputs.to(device), labels.to(device)
8
9         optimizer.zero_grad()
10        outputs = model(inputs)
11        loss = criterion(outputs, labels)
12        loss.backward()
13        optimizer.step()
14
15        running_loss += loss.item()
16        _, preds = torch.max(outputs, 1)
17        correct += (preds == labels).sum().item()
18        total += labels.size(0)
19
20    print(f'Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(dataloaders["t
21

```

```

→ Epoch 1/5, Loss: 0.0000, Accuracy: 1.0000
Epoch 2/5, Loss: 0.0000, Accuracy: 1.0000
Epoch 3/5, Loss: 0.0000, Accuracy: 1.0000
Epoch 4/5, Loss: 0.0000, Accuracy: 1.0000
Epoch 5/5, Loss: 0.0000, Accuracy: 1.0000

```

```

1 model.eval()
2 y_true, y_pred, y_probs = [], [], []
3
4 with torch.no_grad():
5     for inputs, labels in dataloaders['test']:
6         inputs = inputs.to(device)
7         outputs = model(inputs)
8         probs = nn.functional.softmax(outputs, dim=1)
9         _, preds = torch.max(probs, 1)
10

```

```

11     y_true.extend(labels.cpu().numpy())
12     y_pred.extend(preds.cpu().numpy())
13     y_probs.extend(probs.cpu().numpy())
14

```

```

1 torch.save(model.state_dict(), 'fine_tuned_model.pth')
2

```

```

1 from sklearn.metrics import classification_report, confusion_matrix
2
3 print("📊 Classification Report:")
4 print(classification_report(y_true, y_pred, target_names=class_names))
5
6 print("🔍 Confusion Matrix:")
7 print(confusion_matrix(y_true, y_pred))
8

```

📊 Classification Report:

	precision	recall	f1-score	support
lung_scc	1.00	1.00	1.00	750
accuracy			1.00	750
macro avg	1.00	1.00	1.00	750
weighted avg	1.00	1.00	1.00	750

🔍 Confusion Matrix:

```
[[750]]
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:407: UserWarning:
warnings.warn(
```

```

1 from sklearn.preprocessing import label_binarize
2 from sklearn.metrics import roc_auc_score, roc_curve
3
4 y_true_bin = label_binarize(y_true, classes=range(len(class_names)))
5 y_probs = np.array(y_probs)
6
7 for i, class_name in enumerate(class_names):
8     fpr, tpr, _ = roc_curve(y_true_bin[:, i], y_probs[:, i])
9     auc_score = roc_auc_score(y_true_bin[:, i], y_probs[:, i])
10    plt.plot(fpr, tpr, label=f'{class_name} (AUC = {auc_score:.2f})'
11
12 plt.plot([0,1], [0,1], 'k--')
13 plt.title('ROC Curve')
14 plt.xlabel('False Positive Rate')
15 plt.ylabel('True Positive Rate')

```

```
→ /usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:1188: UndefinedMetric
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_ranking.py:379: UndefinedMetric
    warnings.warn(
```