

Hussam Almokayed Assignment 1 image classification

(Brain Tumor MRI)

** Brain Tumor MRI Dataset — 4 class : glioma · meningioma · notumor · pituitary

download dataset from Kagel

dataset path <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset>

```
In [1]: import os
import shutil
import numpy as np
import pandas as pd

import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms as T, models

import cv2
from PIL import Image

from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix, accuracy_score, precision_score, rec

import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm

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

الخطوة 1 — أساسيات الصورة (المحاضرة 1)

ونطبع أبعادها، عشان نتأكد إن الداتا موجودة Grayscale و RGB نقرأ صورة عينة واحدة ونعرضها بصيغة وشكلها سليم.

```
In [10]: # مجلد المصدر (عدّل المسار لو اسم المجلد مختلف عندك)
SOURCE_DIR = "Brain Tumor MRI Dataset/Training"
# كعينة glioma اختر أول صورة من كلاس
sample_class = os.path.join(SOURCE_DIR, "glioma")
sample_img = os.path.join(sample_class, os.listdir(sample_class)[0])

# BGR يقرأ بترتيب (OpenCV قراءة الصورة)
```

```

image = cv2.imread(sample_img)

# للعرض الصحيح إلى RGB تحويل BGR
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

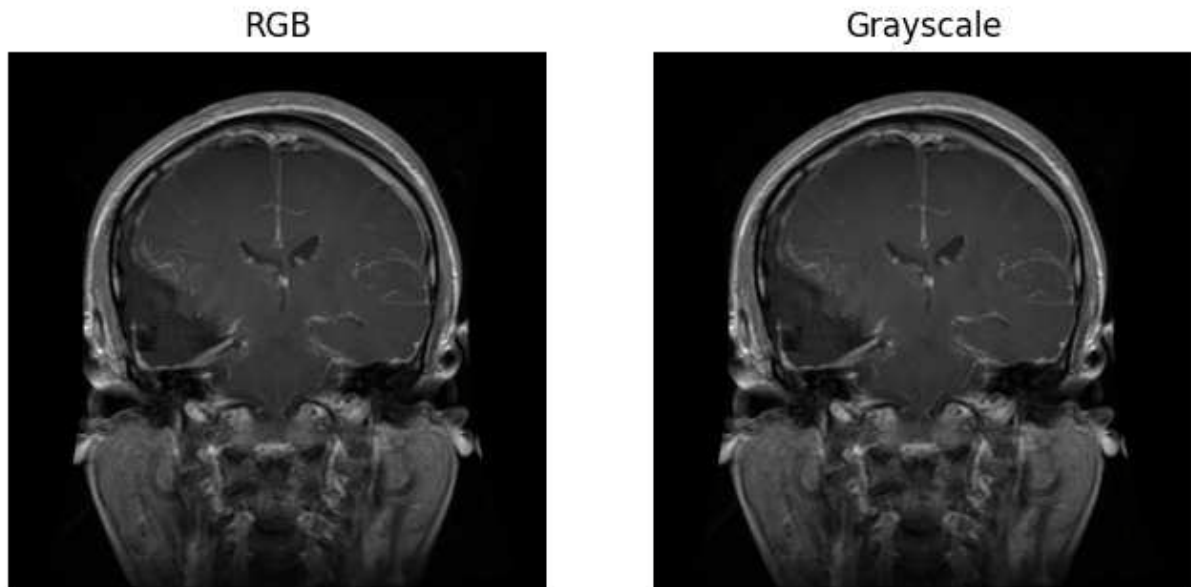
# Grayscale تحويل إلى
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# طباعة الأبعاد
h, w = gray_image.shape
print(f"Width: {w}, Height: {h}")

# العرض
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1); plt.imshow(image_rgb); plt.title("RGB"); plt
plt.subplot(1, 2, 2); plt.imshow(gray_image, cmap="gray"); plt.title("Grayscale");
plt.show()

```

Width: 512, Height: 512



الخطوة 2 — تصغير حجم الصور (المحاضرة 4)

هذا يقلل حجم الداتا ويسرع التدريب . `output` نصغر كل الصور إلى حجم موحد ونحفظها في مجلد.

```

In [11]: input_root = SOURCE_DIR
output_root = "output"
resize_size = (256, 256) # الحجم الموحد

os.makedirs(output_root, exist_ok=True)

for class_name in os.listdir(input_root):
    class_path = os.path.join(input_root, class_name)
    if not os.path.isdir(class_path):
        continue

    output_class_path = os.path.join(output_root, class_name)

```

```

os.makedirs(output_class_path, exist_ok=True)

for image_name in os.listdir(class_path):
    if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
        in_path = os.path.join(class_path, image_name)
        out_name = os.path.splitext(image_name)[0] + '.jpg'
        out_path = os.path.join(output_class_path, out_name)
        try:
            with Image.open(in_path) as img:
                img = img.convert('RGB')
                img = img.resize(resize_size)
                img.save(out_path, 'JPEG')
        except Exception as e:
            print(f"Failed to process {in_path}: {e}")

print("output تم تصغير جميع الصور وحفظها في مجلد")

```

output تم تصغير جميع الصور وحفظها في مجلد.

train / validation / test الخطوة 3 — تقسيم الداتا إلى

نسبة التقسيم: 70% تدريب · 10% تحقق · 20% اختبار. كل مجلد يحتوي على مجلدات فرعية بأسماء الكلاسات.

```

In [12]: original_dataset_dir = "output"
base_dir = "new dataset"
train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")
test_dir = os.path.join(base_dir, "test")

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

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir)
                  if os.path.isfile(os.path.join(class_dir, f))]

    train_images, temp_images = train_test_split(
        all_images, test_size=(1 - train_ratio), random_state=42)

    validation_images, test_images = train_test_split(
        temp_images, test_size=test_ratio / (validation_ratio + test_ratio), random

    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.copy(file, dest_dir)

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.

Sample Image for each class from training

```

In [13]: # عرض صورة عيّنة من كل كلاس
import matplotlib.pyplot as plt
from PIL import Image

# "output" أو "Brain Tumor MRI Dataset/Training" تقدر تغيّره (مصدر الصور)
sample_dir = "new dataset/train"

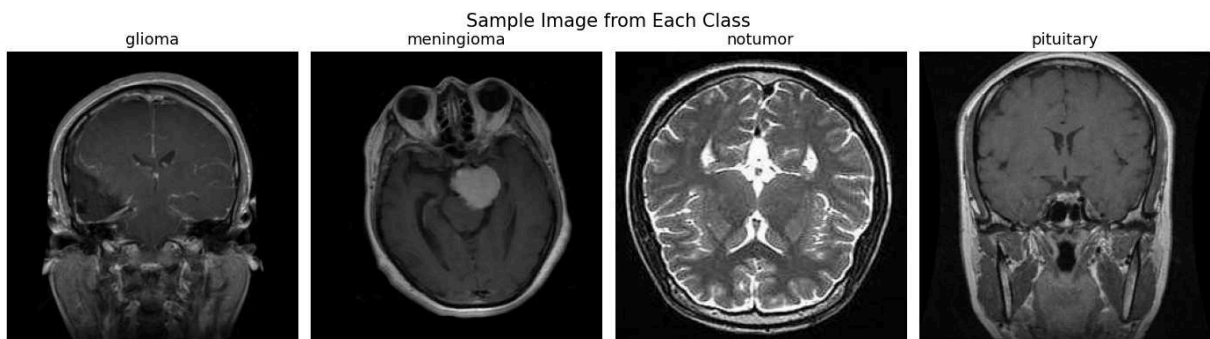
class_names = sorted([d for d in os.listdir(sample_dir)
                       if os.path.isdir(os.path.join(sample_dir, d))])

plt.figure(figsize=(14, 4))
for i, cls in enumerate(class_names):
    cls_path = os.path.join(sample_dir, cls)
    first_img = os.listdir(cls_path)[0] # أول صورة في الكلاس
    img = Image.open(os.path.join(cls_path, first_img))

    plt.subplot(1, len(class_names), i + 1)
    plt.imshow(img)
    plt.title(cls, fontsize=13)
    plt.axis("off")

plt.suptitle("Sample Image from Each Class", fontsize=15)
plt.tight_layout()
plt.show()

```



Augmentation و DataLoaders الخطوة 4 — المعالجة و

- (قلب أفقي/رأسي، تدوير) Augmentation على بيانات التدريب تطبق.
- فقط Normalize و Resize على التحقق والاختبار تطبق.
- (منها Inception-V3) كل الصور تُحوّل إلى حجم 299×299 لتناسب جميع الموديلات.

على ويندوز أحيانًا يعلّق مع قيم أعلى. لو Jupyter لأن `num_workers=0` ملاحظة ويندوز: خلّيت جهازك ما يعطي مشاكل تقدر ترفعها إلى 2 أو 4 لتسريع التحميل.

```
In [14]: def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

def get_data_loaders(data_dir, batch_size, train=False):
    if train:
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),
            T.Resize((299, 299)),
            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=0)
        return train_loader, len(train_data)
    else:
        transform = T.Compose([
            T.Resize((299, 299)),
            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=0)
        test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=0)
        return val_loader, test_loader, len(val_data), len(test_data)

dataset_path = "new dataset"
batch_size = 32

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)

classes = get_classes(os.path.join(dataset_path, "train"))
num_classes = len(classes)
print("Classes:", classes, "| Count:", num_classes)

dataloaders = {"train": train_loader, "val": val_loader}
dataset_sizes = {"train": train_data_len, "val": valid_data_len}

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

Classes: ['glioma', 'meningioma', 'notumor', 'pituitary'] | Count: 4
Device: cuda

الخطوة 5 — دالة التدريب

الخاصة بموديل auxiliary logits نفس دالة التدريب المستخدمة في المحاضرات، وتتعامل مع ال Inception-V3.

```
In [16]: def train_model(model, criterion, optimizer, scheduler, num_epochs=5):
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

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

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)

                    # Inception-V3 التعامل مع مخرجات (aux Logits)
                    if isinstance(outputs, tuple) or hasattr(outputs, "logits"):
                        main_outputs = outputs.logits if hasattr(outputs, "logits")
                        aux_outputs = outputs.aux_logits if hasattr(outputs, "aux_
                        loss = criterion(main_outputs, labels)
                        if aux_outputs is not None:
                            loss += 0.4 * criterion(aux_outputs, labels)
                    else:
                        main_outputs = outputs
                        loss = criterion(main_outputs, labels)

                _, preds = torch.max(main_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} Acc: {epoch_acc:.4f}")

    return model
```

الخطوة 6 — دوال التنبؤ والتقييم

- `get_predictions` : تمزّ على بيانات الاختبار وتُرجع القيم الحقيقية والمتوقعة والاحتمالات
- `evaluate_model` : تطبع المقاييس + تقرير التصنيف + مصفوفة الالتباس، وتخزّن النتيجة للمقارنة.

```
In [17]: results = {} # لتجميع نتائج كل الموديلات للمقارنة لاحقًا

def get_predictions(model, loader):
    model.eval()
    y_true, y_pred, y_proba = [], [], []
    with torch.no_grad():
        for x, y in tqdm(loader):
            output = model(x.to(device))
            probs = torch.softmax(output, dim=1)
            preds = torch.argmax(probs, dim=1).cpu().numpy()
            y_true.extend(y.numpy())
            y_proba.extend(probs.cpu().numpy())
            y_pred.extend(preds)
    return np.array(y_true), np.array(y_pred), np.array(y_proba)

def evaluate_model(model_name, y_true, y_pred, classes):
    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("Model:", 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=classes))

    # تخزين النتيجة للمقارنة
    results[model_name] = {"Accuracy": accuracy, "Precision": precision,
                          "Recall": recall, "F1-score": f1}

    # مصفوفة الالتباس
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=classes, yticklabels=classes)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.show()
```

الخطوة 7 — إعدادات التدريب

- `PRETRAINED = True` يعطي دقة — (Transfer Learning) يعني نستخدم أوزانًا مدربة مسبقًا أعلى بكثير.
- وطلعت نتائجها (`pretrained=False`) في المحاضرة بعض الموديلات دُرِّبَت من الصفر ضعيفة.
- فقط `False` لو الدكتور تبي التدريب من الصفر، غيّر القيمة إلى
- لنتائج أفضل (يبدأ بطيء ثم يتحسن) `EPOCHS` تقدر تزيد عدد.

```
In [19]: PRETRAINED = True
          EPOCHS      = 5
```

1 الموديل — EfficientNet-B0

```
In [20]: model = models.efficientnet_b0(pretrained=PRETRAINED)
          model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
          model = model.to(device)

          criterion = nn.CrossEntropyLoss().to(device)
          optimizer = optim.AdamW(model.parameters(), lr=0.001)
          exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

          model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=EPOCHS)

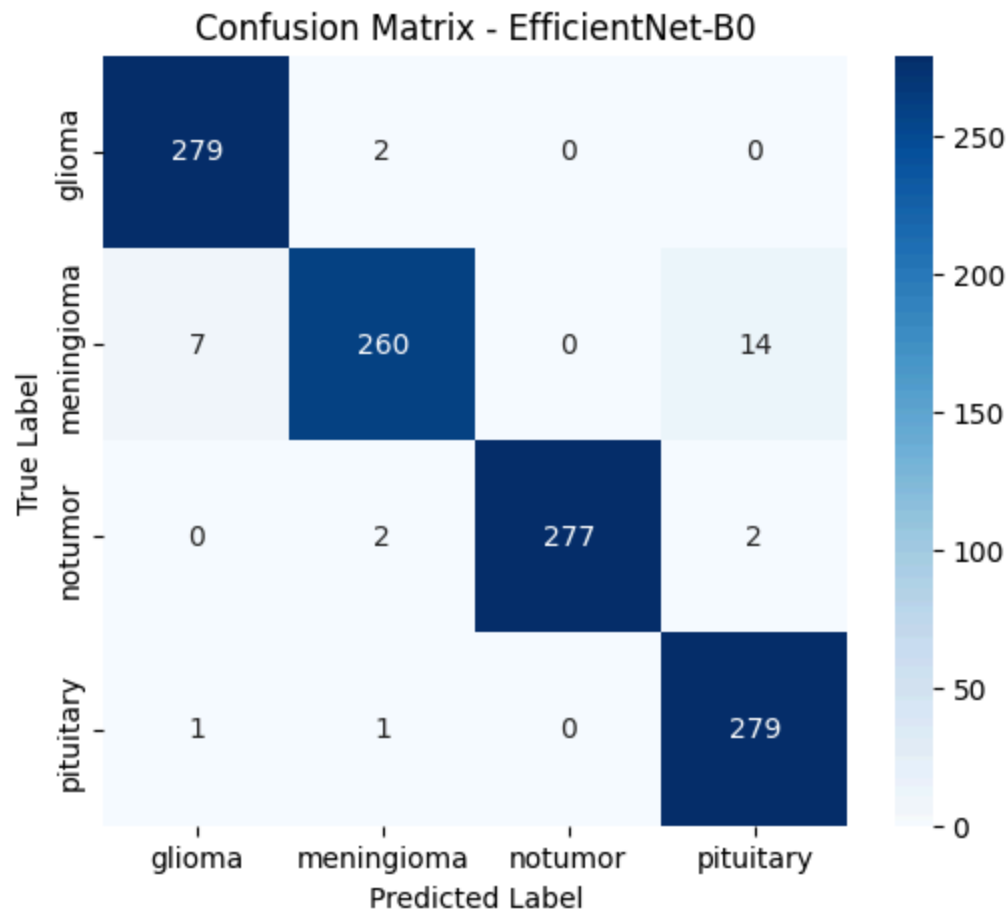
          y_true, y_pred, y_proba = get_predictions(model, test_loader)
          evaluate_model("EfficientNet-B0", y_true, y_pred, classes)
```

```
Epoch 1/5
-----
train Loss: 0.3498 Acc: 0.8790
val Loss: 0.1558 Acc: 0.9500
Epoch 2/5
-----
train Loss: 0.1862 Acc: 0.9328
val Loss: 0.0817 Acc: 0.9732
Epoch 3/5
-----
train Loss: 0.1425 Acc: 0.9553
val Loss: 0.0766 Acc: 0.9732
Epoch 4/5
-----
train Loss: 0.0956 Acc: 0.9665
val Loss: 0.2372 Acc: 0.9268
Epoch 5/5
-----
train Loss: 0.0950 Acc: 0.9655
val Loss: 0.1242 Acc: 0.9625
100%|██████████| 36/36 [00:01<00:00, 21.31it/s]
```

Model: EfficientNet-B0
 Accuracy : 0.9741992882562278
 Precision: 0.9747550557190438
 Recall : 0.9741992882562278
 F1-score : 0.9740892149491932

Classification Report:

	precision	recall	f1-score	support
glioma	0.97	0.99	0.98	281
meningioma	0.98	0.93	0.95	281
notumor	1.00	0.99	0.99	281
pituitary	0.95	0.99	0.97	281
accuracy			0.97	1124
macro avg	0.97	0.97	0.97	1124
weighted avg	0.97	0.97	0.97	1124



الموديل — ResNet-18

```
In [21]: model = models.resnet18(pretrained=PRETRAINED)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

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

```
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=EPOCH

y_true, y_pred, y_proba = get_predictions(model, test_loader)
evaluate_model("ResNet-18", y_true, y_pred, classes)
```

Epoch 1/5

train Loss: 0.4354 Acc: 0.8399

val Loss: 1.2550 Acc: 0.7268

Epoch 2/5

train Loss: 0.2759 Acc: 0.9017

val Loss: 0.7510 Acc: 0.7589

Epoch 3/5

train Loss: 0.2372 Acc: 0.9198

val Loss: 0.2350 Acc: 0.9179

Epoch 4/5

train Loss: 0.1953 Acc: 0.9331

val Loss: 0.2010 Acc: 0.9375

Epoch 5/5

train Loss: 0.1748 Acc: 0.9367

val Loss: 0.2352 Acc: 0.9232

100%|██████████| 36/36 [00:01<00:00, 22.21it/s]

Model: ResNet-18

Accuracy : 0.9350533807829181

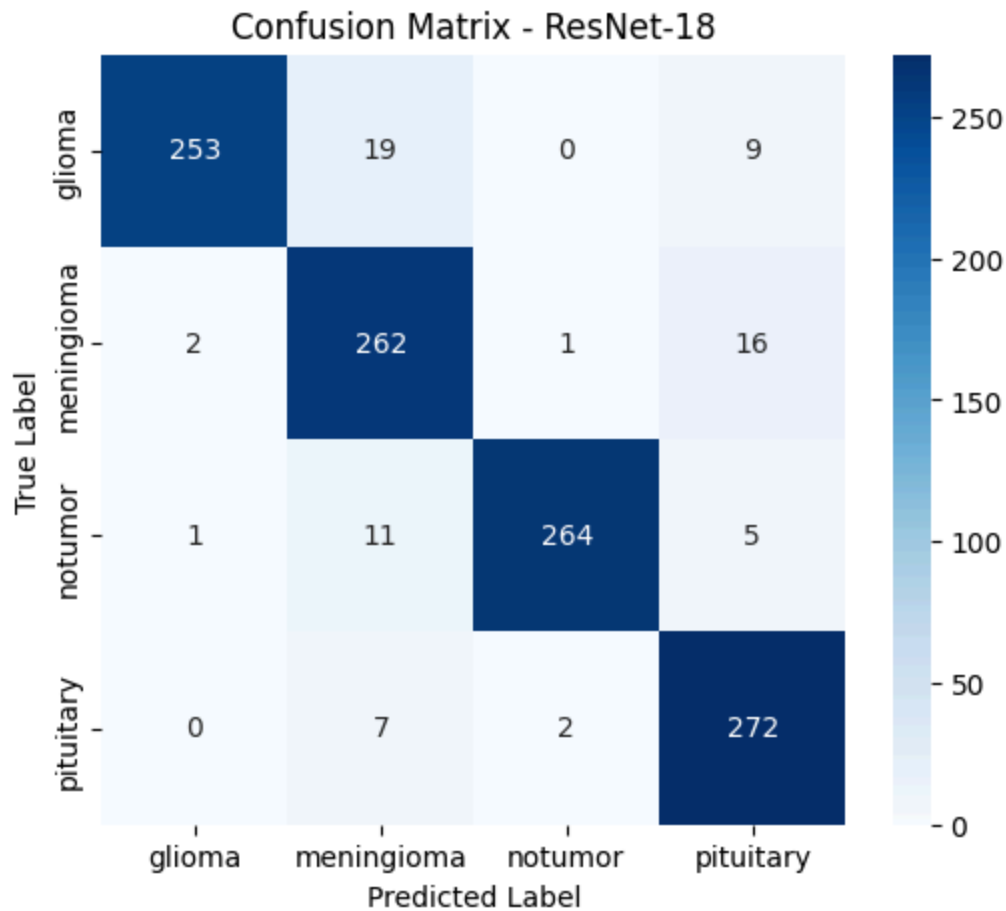
Precision: 0.9384904318003641

Recall : 0.9350533807829181

F1-score : 0.9355821093835778

Classification Report:

	precision	recall	f1-score	support
glioma	0.99	0.90	0.94	281
meningioma	0.88	0.93	0.90	281
notumor	0.99	0.94	0.96	281
pituitary	0.90	0.97	0.93	281
accuracy			0.94	1124
macro avg	0.94	0.94	0.94	1124
weighted avg	0.94	0.94	0.94	1124



3 الموديل — DenseNet-121

```
In [22]: model = models.densenet121(pretrained=PRETRAINED)
model.classifier = nn.Linear(model.classifier.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=EPOCH)

y_true, y_pred, y_proba = get_predictions(model, test_loader)
evaluate_model("DenseNet-121", y_true, y_pred, classes)
```

Epoch 1/5

train Loss: 0.3817 Acc: 0.8588

val Loss: 0.3522 Acc: 0.8946

Epoch 2/5

train Loss: 0.2554 Acc: 0.9083

val Loss: 1.1829 Acc: 0.7625

Epoch 3/5

train Loss: 0.2288 Acc: 0.9188

val Loss: 0.2555 Acc: 0.9179

Epoch 4/5

train Loss: 0.1806 Acc: 0.9354

val Loss: 0.3470 Acc: 0.8679

Epoch 5/5

train Loss: 0.1745 Acc: 0.9374

val Loss: 0.1262 Acc: 0.9554

100%|██████████| 36/36 [00:02<00:00, 17.93it/s]

Model: DenseNet-121

Accuracy : 0.9653024911032029

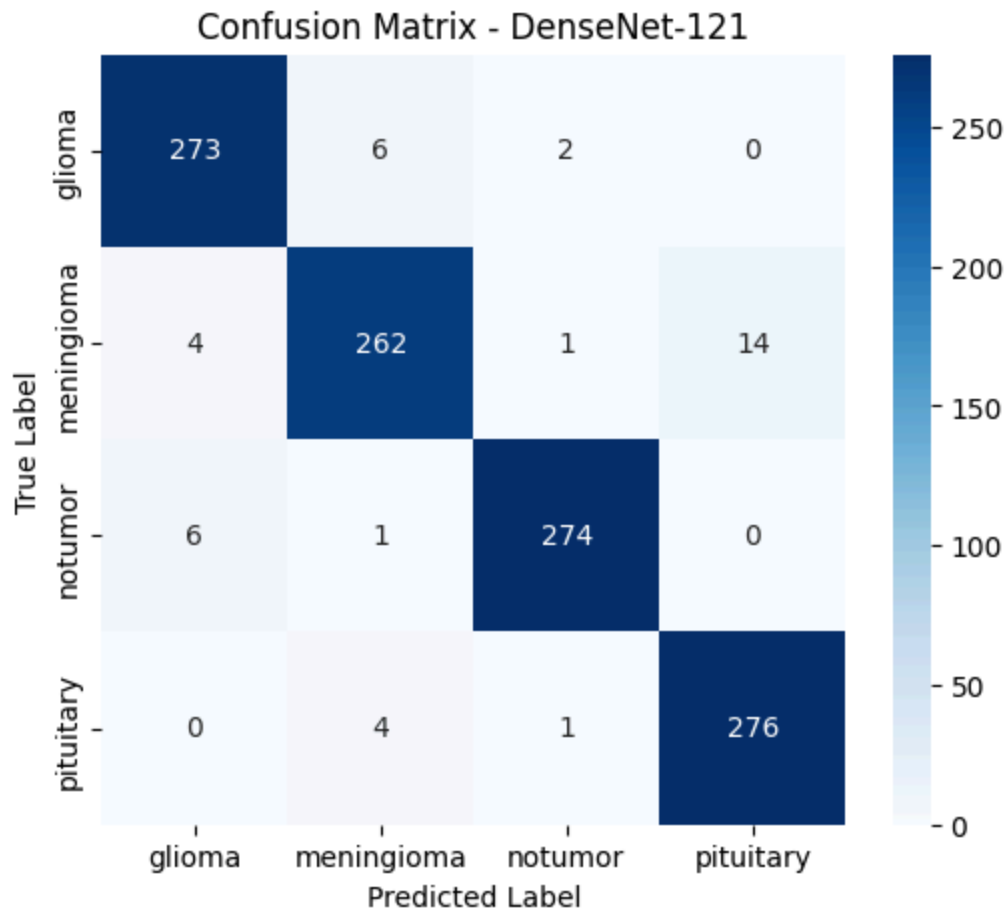
Precision: 0.9654267298458563

Recall : 0.9653024911032029

F1-score : 0.9652451322987242

Classification Report:

	precision	recall	f1-score	support
glioma	0.96	0.97	0.97	281
meningioma	0.96	0.93	0.95	281
notumor	0.99	0.98	0.98	281
pituitary	0.95	0.98	0.97	281
accuracy			0.97	1124
macro avg	0.97	0.97	0.97	1124
weighted avg	0.97	0.97	0.97	1124



4 الموديل — VGG-16

```
In [23]: model = models.vgg16(pretrained=PRETRAINED)
model.classifier[6] = nn.Linear(model.classifier[6].in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=EPOCH)

y_true, y_pred, y_proba = get_predictions(model, test_loader)
evaluate_model("VGG-16", y_true, y_pred, classes)
```

Epoch 1/5

train Loss: 1.4753 Acc: 0.3751

val Loss: 0.9926 Acc: 0.5286

Epoch 2/5

train Loss: 1.0164 Acc: 0.5401

val Loss: 0.9388 Acc: 0.5679

Epoch 3/5

train Loss: 0.8968 Acc: 0.5840

val Loss: 0.8103 Acc: 0.6161

Epoch 4/5

train Loss: 0.8413 Acc: 0.6279

val Loss: 0.8099 Acc: 0.6196

Epoch 5/5

train Loss: 0.7678 Acc: 0.6494

val Loss: 0.7094 Acc: 0.6839

100%|██████████| 36/36 [00:02<00:00, 15.24it/s]

Model: VGG-16

Accuracy : 0.7090747330960854

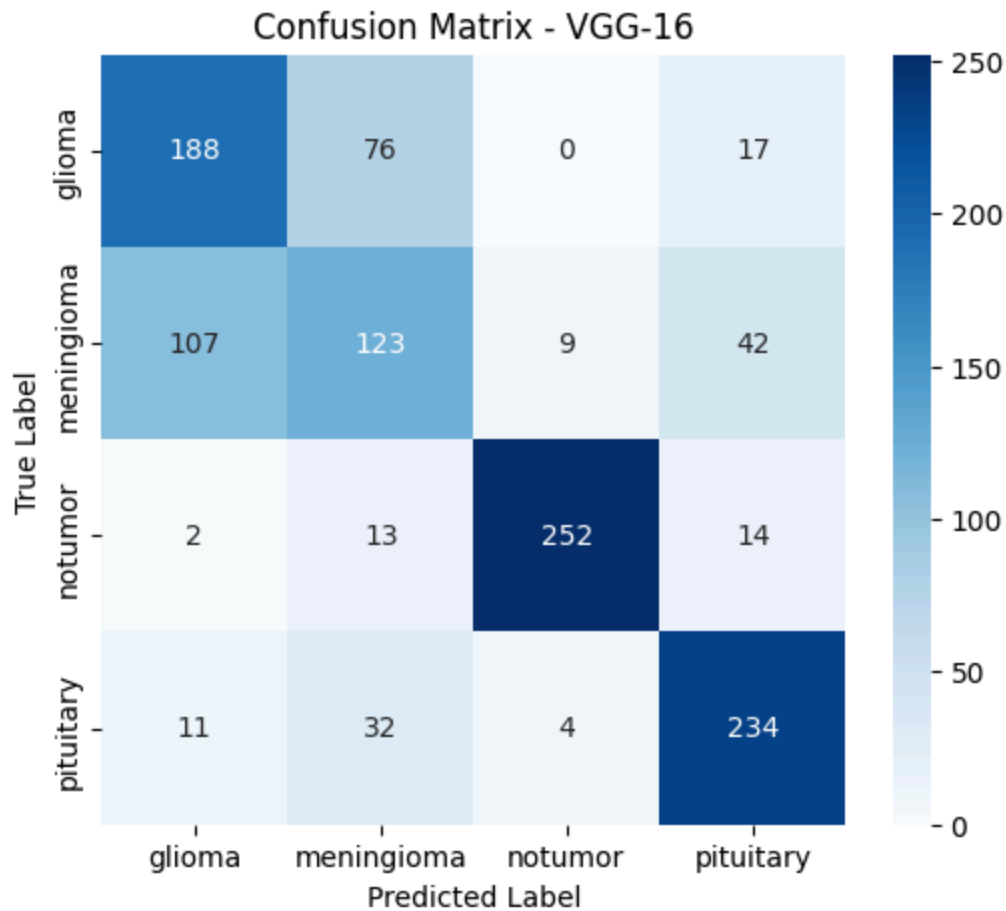
Precision: 0.7069115877462795

Recall : 0.7090747330960854

F1-score : 0.7064842094601979

Classification Report:

	precision	recall	f1-score	support
glioma	0.61	0.67	0.64	281
meningioma	0.50	0.44	0.47	281
notumor	0.95	0.90	0.92	281
pituitary	0.76	0.83	0.80	281
accuracy			0.71	1124
macro avg	0.71	0.71	0.71	1124
weighted avg	0.71	0.71	0.71	1124



5 الموديل — Inception-V3

هذا الموديل يحتاج مدخلات 299×299 (وهو السبب اللي خلّانا نستخدم هذا الحجم من البداية)، ونستبدل auxiliary classifier. كلّاً من الطبقة الأخيرة وطبقة الـ

```
In [24]: model = models.inception_v3(pretrained=PRETRAINED, aux_logits=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)
if model.aux_logits:
    model.AuxLogits.fc = nn.Linear(model.AuxLogits.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=EPOCH)

y_true, y_pred, y_proba = get_predictions(model, test_loader)
evaluate_model("Inception-V3", y_true, y_pred, classes)
```

Epoch 1/5

train Loss: 0.5630 Acc: 0.8631

val Loss: 0.4837 Acc: 0.8286

Epoch 2/5

train Loss: 0.3377 Acc: 0.9170

val Loss: 0.2323 Acc: 0.9179

Epoch 3/5

train Loss: 0.2409 Acc: 0.9392

val Loss: 0.4121 Acc: 0.8625

Epoch 4/5

train Loss: 0.2423 Acc: 0.9428

val Loss: 0.2622 Acc: 0.9161

Epoch 5/5

train Loss: 0.2223 Acc: 0.9425

val Loss: 0.3626 Acc: 0.8821

100%|██████████| 36/36 [00:01<00:00, 20.89it/s]

Model: Inception-V3

Accuracy : 0.8825622775800712

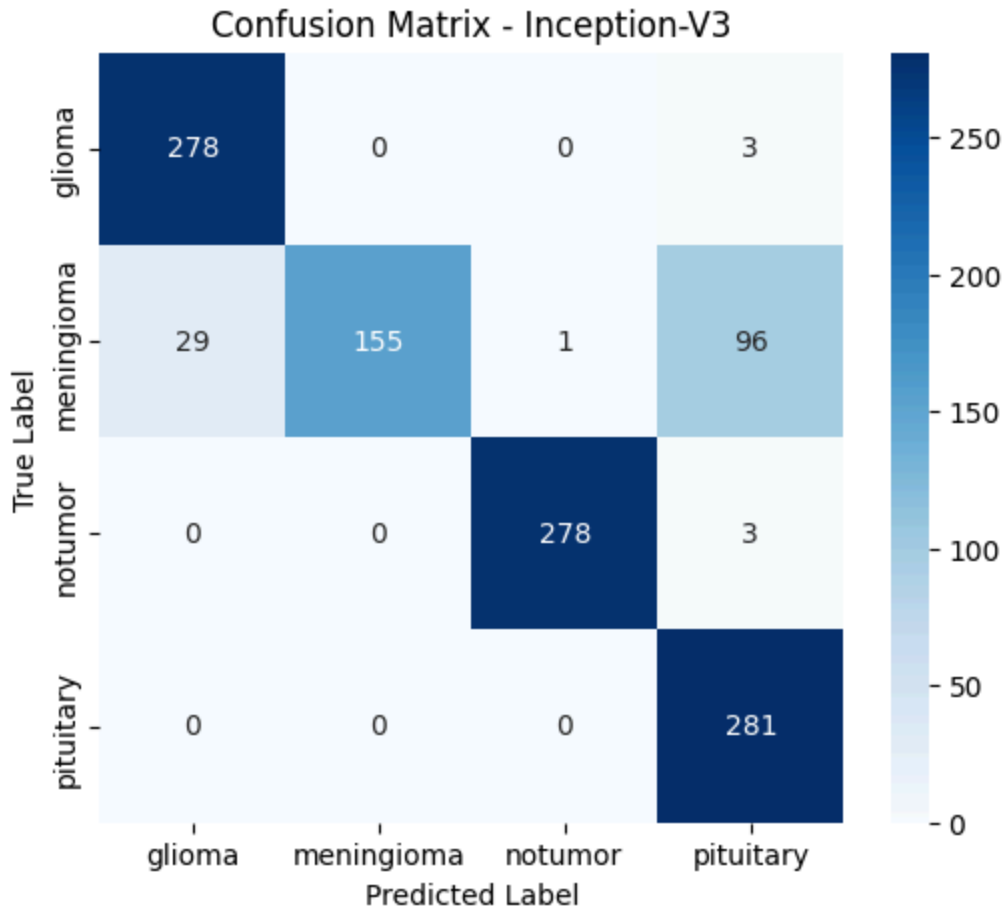
Precision: 0.9089086730084246

Recall : 0.8825622775800712

F1-score : 0.8739575226575653

Classification Report:

	precision	recall	f1-score	support
glioma	0.91	0.99	0.95	281
meningioma	1.00	0.55	0.71	281
notumor	1.00	0.99	0.99	281
pituitary	0.73	1.00	0.85	281
accuracy			0.88	1124
macro avg	0.91	0.88	0.87	1124
weighted avg	0.91	0.88	0.87	1124



الخطوة 8 — مقارنة أداء الموديلات

نجمع نتائج كل الموديلات في جدول واحد ونرسمها للمقارنة.

```
In [27]: df = pd.DataFrame(results).T
df = df.sort_values("F1-score", ascending=False)
print(df.round(4))
```

	Accuracy	Precision	Recall	F1-score
EfficientNet-B0	0.9742	0.9748	0.9742	0.9741
DenseNet-121	0.9653	0.9654	0.9653	0.9652
ResNet-18	0.9351	0.9385	0.9351	0.9356
Inception-V3	0.8826	0.9089	0.8826	0.8740
VGG-16	0.7091	0.7069	0.7091	0.7065

```
In [28]: # جدول المقارنة بصيغة نسبة مئوية مع تنسيق أفضل
df = pd.DataFrame(results).T
df = df.sort_values("F1-score", ascending=False)

# تحويل القيم إلى نسبة مئوية
df_pct = df * 100

styled = (df_pct.style
          .format("{:.2f}%")
          .background_gradient(cmap="Greens")
          .highlight_max(color="#38a169", axis=0))

# نسبة مئوية برقمين عشريين
# تدرج لوني حسب القيمة
# تمييز الأفضل في كل عمود
```

```
.set_caption("📊 Models Comparison (%)")
.set_properties(**{"text-align": "center", "font-size": "14px"})
.set_table_styles([{"selector": "caption",
                    "props": [("font-size", "16px"), ("font-weight", "bold")]}],
                    styled
```

Out[28]:

	Accuracy	Precision	Recall	F1-score
EfficientNet-B0	97.42%	97.48%	97.42%	97.41%
DenseNet-121	96.53%	96.54%	96.53%	96.52%
ResNet-18	93.51%	93.85%	93.51%	93.56%
Inception-V3	88.26%	90.89%	88.26%	87.40%
VGG-16	70.91%	70.69%	70.91%	70.65%

In []: