

Brain Tumor MRI Classification

Using PyTorch — ResNet18, EfficientNet-B0, MobileNet-V2

Pretrained (Transfer Learning) & From Scratch

Dataset: Brain Tumor MRI Dataset — 4 Classes: glioma, meningioma, pituitary, no_tumor

Step 1: Install & Import Libraries

In [6]: `!pip install kagglehub`

```
import kagglehub
```

```
path = kagglehub.dataset_download("masoudnickparvar/brain-tumor-mri-dataset")  
print("Path to dataset files:", path)
```

Requirement already satisfied: kagglehub in c:\users\f\anaconda3\lib\site-packages (1.0.2)
Requirement already satisfied: kagglesdk<1.0,>=0.1.22 in c:\users\f\anaconda3\lib\site-packages (from kagglehub) (0.1.33)
Requirement already satisfied: packaging in c:\users\f\anaconda3\lib\site-packages (from kagglehub) (24.2)
Requirement already satisfied: pyyaml in c:\users\f\anaconda3\lib\site-packages (from kagglehub) (6.0.3)
Requirement already satisfied: requests in c:\users\f\anaconda3\lib\site-packages (from kagglehub) (2.32.3)
Requirement already satisfied: tqdm in c:\users\f\anaconda3\lib\site-packages (from kagglehub) (4.67.1)
Requirement already satisfied: protobuf in c:\users\f\anaconda3\lib\site-packages (from kagglesdk<1.0,>=0.1.22->kagglehub) (5.29.3)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\f\anaconda3\lib\site-packages (from requests->kagglehub) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\f\anaconda3\lib\site-packages (from requests->kagglehub) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\f\anaconda3\lib\site-packages (from requests->kagglehub) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\f\anaconda3\lib\site-packages (from requests->kagglehub) (2025.10.5)
Requirement already satisfied: colorama in c:\users\f\anaconda3\lib\site-packages (from tqdm->kagglehub) (0.4.6)
Path to dataset files: C:\Users\F\cache\kagglehub\datasets\masoudnickparvar\brain-tumor-mri-dataset\versions\2

In [7]: `!pip install torch torchvision torchaudio`

Requirement already satisfied: torch in c:\users\f\anaconda3\lib\site-packages (2.12.1)
 Requirement already satisfied: torchvision in c:\users\f\anaconda3\lib\site-packages (0.27.1)
 Requirement already satisfied: torchaudio in c:\users\f\anaconda3\lib\site-packages (2.11.0)
 Requirement already satisfied: filelock in c:\users\f\anaconda3\lib\site-packages (from torch) (3.17.0)
 Requirement already satisfied: typing-extensions>=4.10.0 in c:\users\f\anaconda3\lib\site-packages (from torch) (4.12.2)
 Requirement already satisfied: setuptools<82 in c:\users\f\anaconda3\lib\site-packages (from torch) (72.1.0)
 Requirement already satisfied: sympy>=1.13.3 in c:\users\f\anaconda3\lib\site-packages (from torch) (1.13.3)
 Requirement already satisfied: networkx>=2.5.1 in c:\users\f\anaconda3\lib\site-packages (from torch) (3.4.2)
 Requirement already satisfied: jinja2 in c:\users\f\anaconda3\lib\site-packages (from torch) (3.1.6)
 Requirement already satisfied: fsspec>=0.8.5 in c:\users\f\anaconda3\lib\site-packages (from torch) (2025.3.2)
 Requirement already satisfied: numpy in c:\users\f\anaconda3\lib\site-packages (from torchvision) (2.1.3)
 Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in c:\users\f\anaconda3\lib\site-packages (from torchvision) (11.1.0)
 Requirement already satisfied: mpmath<1.4,>=1.1.0 in c:\users\f\anaconda3\lib\site-packages (from sympy>=1.13.3->torch) (1.3.0)
 Requirement already satisfied: MarkupSafe>=2.0 in c:\users\f\anaconda3\lib\site-packages (from jinja2->torch) (3.0.2)

```
In [8]: import os
import shutil
import numpy as np
import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix, accuracy_score,
                             precision_score, recall_score,
                             f1_score, classification_report)

import matplotlib.pyplot as plt
import seaborn as sns

print('All libraries imported successfully!')
print(f'PyTorch version: {torch.__version__}')
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Device: {device}')
```

All libraries imported successfully!
 PyTorch version: 2.12.0+cpu
 Device: cpu

Step 2: Download Dataset from Kaggle

Go to: <https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset> The dataset contains 4 classes:

- glioma
- meningioma
- pituitary
- notumor

Step 3: Split Dataset (70% Train / 10% Validation / 20% Test)

```
In [9]: # ---- CHANGE THIS PATH to your downloaded dataset folder ----
original_dataset_dir = path + '/Training'# -----

base_dir = '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_state=42
    )
    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)

# Apply to all class subfolders
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('Dataset split completed!')
for split in ['train', 'validation', 'test']:
    split_path = os.path.join(base_dir, split)
    total = sum(
        len(files)
```

```

        for _, _, files in os.walk(split_path)
    )
    print(f' {split}: {total} images')

```

Dataset split completed!

```

train: 3916 images
validation: 560 images
test: 1124 images

```

Step 4: Get Class Names

```

In [10]: def get_classes(data_dir):
        all_data = datasets.ImageFolder(data_dir)
        return all_data.classes

classes = get_classes('dataset/train')
num_classes = len(classes)
print(f'Classes: {classes}')
print(f'Number of classes: {num_classes}')

```

```

Classes: ['glioma', 'meningioma', 'notumor', 'pituitary']
Number of classes: 4

```

Step 5: Image Augmentation & Preprocessing (Data Loaders)

```

In [11]: def get_data_loaders(data_dir, batch_size, train=False):
        """Prepare data loaders for training, validation, and testing."""
        if train:
            # Data augmentation for training
            transform = T.Compose([
                T.RandomHorizontalFlip(),
                T.RandomVerticalFlip(),
                T.RandomRotation(10),
                T.Resize((224, 224)),
                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:
            # Validation and test transformations (no augmentation)
            transform = T.Compose([
                T.Resize((224, 224)),
                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)

dataset_path = '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
)

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

print(f'Train size: {train_data_len}')
print(f'Validation size: {valid_data_len}')
print(f'Test size: {test_data_len}')

```

Train size: 3916
 Validation size: 560
 Test size: 1124

Step 6: Training Function

```

In [12]: def train_model(model, criterion, optimizer, scheduler, num_epochs=5):
    """Train and validate the model."""
    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)
                    if isinstance(outputs, tuple) or hasattr(outputs, 'logits'):
                        main_outputs = outputs.logits if hasattr(outputs, 'logit')
                        aux_outputs = outputs.aux_logits if hasattr(outputs, 'au
                        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


def evaluate_model(model_name, y_true, y_pred):
    """Print metrics and plot confusion matrix."""
    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)
    cm = confusion_matrix(y_true, y_pred)

    print(f'\n===== {model_name} Results =====')
    print(f'Accuracy : {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall : {recall:.4f}')
    print(f'F1-score : {f1:.4f}')
    print('\nClassification Report:')
    print(classification_report(y_true, y_pred, target_names=classes, zero_division=0))

    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.tight_layout()
    plt.show()

    return accuracy, precision, recall, f1


def get_predictions(model):
    """Run inference on test set and return true/predicted labels."""
    y_true, y_pred = [], []
    model.eval()
    with torch.no_grad():
        for x, y in tqdm(test_loader):
            output = model(x.to(device))

```

```

        if hasattr(output, 'logits'):
            output = output.logits
        probabilities = torch.softmax(output, dim=1)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_true.extend(y.numpy())
        y_pred.extend(pred)
    return np.array(y_true), np.array(y_pred)

print('Training and evaluation functions defined!')

```

Training and evaluation functions defined!

Model 1: EfficientNet-B0 — Pretrained (Transfer Learning)

```

In [13]: print('Training EfficientNet-B0 (Pretrained)...')

model_eff_pretrained = models.efficientnet_b0(pretrained=True)
model_eff_pretrained.classifier[1] = nn.Linear(
    model_eff_pretrained.classifier[1].in_features, num_classes
)
model_eff_pretrained = model_eff_pretrained.to(device)

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

model_eff_pretrained = train_model(
    model_eff_pretrained, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_eff_pretrained)
eff_pre_results = evaluate_model('EfficientNet-B0 (Pretrained)', y_true, y_pred)

```

Training EfficientNet-B0 (Pretrained)...

Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to C:\Users\F/.cache\torch\hub\checkpoints\efficientnet_b0_rwightman-7f5810bc.pth

100%|██████████| 20.5M/20.5M [00:02<00:00, 9.18MB/s]

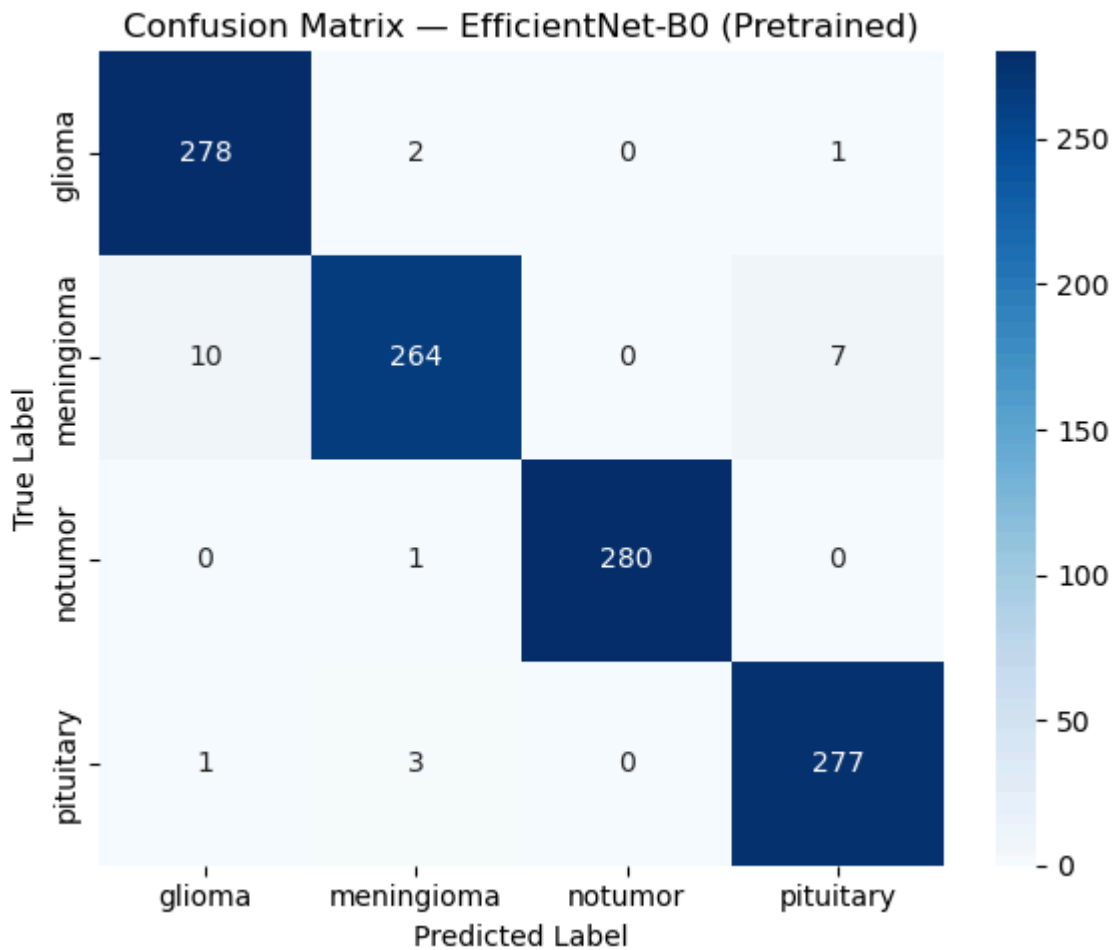
```

Epoch 1/5
-----
train Loss: 0.3467  Acc: 0.8797
val Loss: 0.2360  Acc: 0.9125
Epoch 2/5
-----
train Loss: 0.1741  Acc: 0.9400
val Loss: 0.1130  Acc: 0.9625
Epoch 3/5
-----
train Loss: 0.1462  Acc: 0.9510
val Loss: 0.1426  Acc: 0.9375
Epoch 4/5
-----
train Loss: 0.1073  Acc: 0.9663
val Loss: 0.0855  Acc: 0.9714
Epoch 5/5
-----
train Loss: 0.0826  Acc: 0.9729
val Loss: 0.0722  Acc: 0.9732
100%|██████████| 36/36 [00:54<00:00, 1.53s/it]
===== EfficientNet-B0 (Pretrained) Results =====
Accuracy : 0.9778
Precision: 0.9779
Recall   : 0.9778
F1-score : 0.9777

Classification Report:

```

	precision	recall	f1-score	support
glioma	0.96	0.99	0.98	281
meningioma	0.98	0.94	0.96	281
notumor	1.00	1.00	1.00	281
pituitary	0.97	0.99	0.98	281
accuracy			0.98	1124
macro avg	0.98	0.98	0.98	1124
weighted avg	0.98	0.98	0.98	1124



Model 2: EfficientNet-B0 — From Scratch

```
In [14]: print('Training EfficientNet-B0 (From Scratch)...')

model_eff_scratch = models.efficientnet_b0(pretrained=False)
model_eff_scratch.classifier[1] = nn.Linear(
    model_eff_scratch.classifier[1].in_features, num_classes
)
model_eff_scratch = model_eff_scratch.to(device)

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

model_eff_scratch = train_model(
    model_eff_scratch, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_eff_scratch)
eff_scr_results = evaluate_model('EfficientNet-B0 (From Scratch)', y_true, y_pre
```

Training EfficientNet-B0 (From Scratch)...

Epoch 1/5

train Loss: 0.8975 Acc: 0.6479

val Loss: 2.0422 Acc: 0.5018

Epoch 2/5

train Loss: 0.6238 Acc: 0.7697

val Loss: 1.1411 Acc: 0.5911

Epoch 3/5

train Loss: 0.5035 Acc: 0.8172

val Loss: 0.7618 Acc: 0.7411

Epoch 4/5

train Loss: 0.4487 Acc: 0.8391

val Loss: 2.4030 Acc: 0.4268

Epoch 5/5

train Loss: 0.3992 Acc: 0.8562

val Loss: 0.6499 Acc: 0.7643

100%|██████████| 36/36 [00:35<00:00, 1.01it/s]

==== EfficientNet-B0 (From Scratch) Results ====

Accuracy : 0.7349

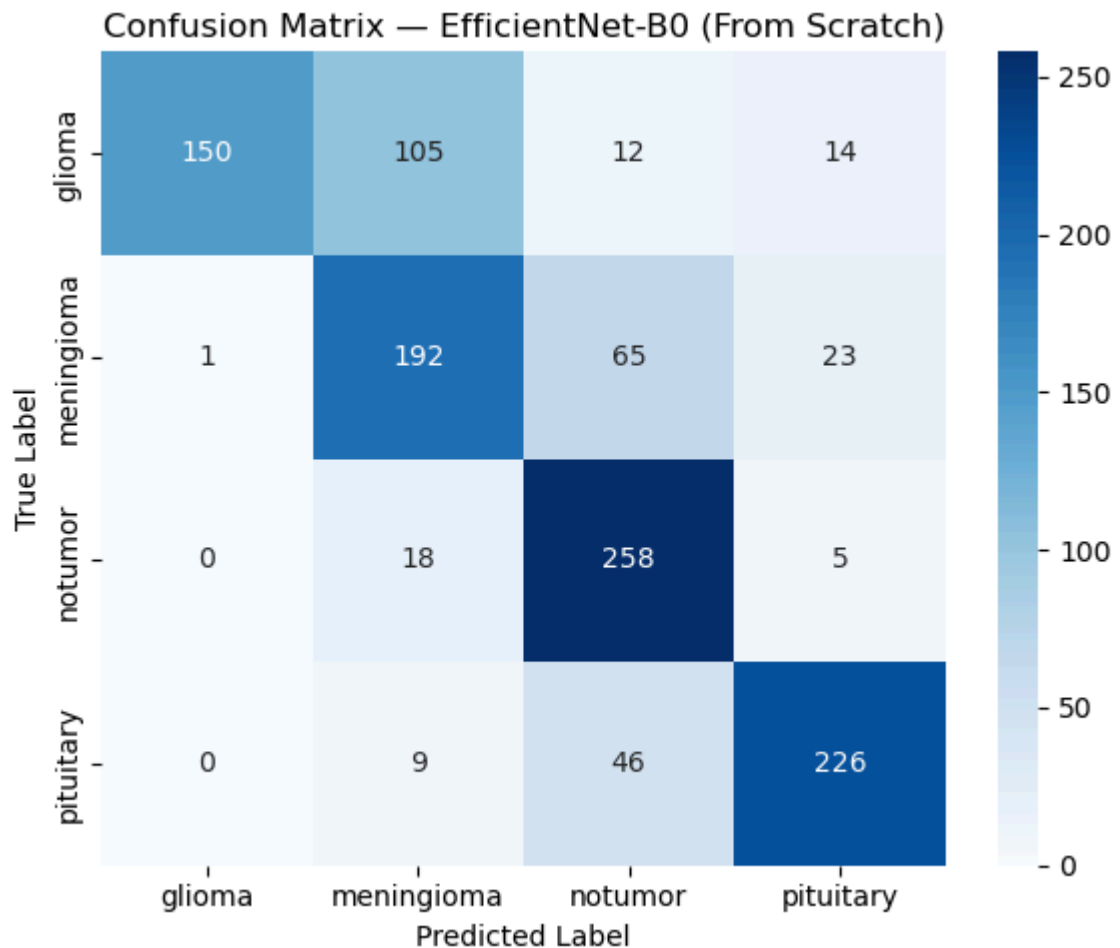
Precision: 0.7766

Recall : 0.7349

F1-score : 0.7330

Classification Report:

	precision	recall	f1-score	support
glioma	0.99	0.53	0.69	281
meningioma	0.59	0.68	0.63	281
notumor	0.68	0.92	0.78	281
pituitary	0.84	0.80	0.82	281
accuracy			0.73	1124
macro avg	0.78	0.73	0.73	1124
weighted avg	0.78	0.73	0.73	1124



Model 3: ResNet18 — Pretrained (Transfer Learning)

```
In [15]: print('Training ResNet18 (Pretrained)...')

model_res_pretrained = models.resnet18(pretrained=True)
model_res_pretrained.fc = nn.Linear(
    model_res_pretrained.fc.in_features, num_classes
)
model_res_pretrained = model_res_pretrained.to(device)

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

model_res_pretrained = train_model(
    model_res_pretrained, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_res_pretrained)
res_pre_results = evaluate_model('ResNet18 (Pretrained)', y_true, y_pred)
```

Training ResNet18 (Pretrained)...

Downloading: "<https://download.pytorch.org/models/resnet18-f37072fd.pth>" to C:\Users\F/.cache\torch\hub\checkpoints\resnet18-f37072fd.pth

100%|██████████| 44.7M/44.7M [00:03<00:00, 12.0MB/s]

Epoch 1/5

train Loss: 0.4510 Acc: 0.8361

val Loss: 0.5023 Acc: 0.8357

Epoch 2/5

train Loss: 0.2600 Acc: 0.9060

val Loss: 0.3018 Acc: 0.8946

Epoch 3/5

train Loss: 0.2154 Acc: 0.9280

val Loss: 0.5407 Acc: 0.8232

Epoch 4/5

train Loss: 0.1913 Acc: 0.9356

val Loss: 0.6314 Acc: 0.7464

Epoch 5/5

train Loss: 0.1886 Acc: 0.9344

val Loss: 0.3727 Acc: 0.8929

100%|██████████| 36/36 [00:40<00:00, 1.11s/it]

==== ResNet18 (Pretrained) Results =====

Accuracy : 0.8968

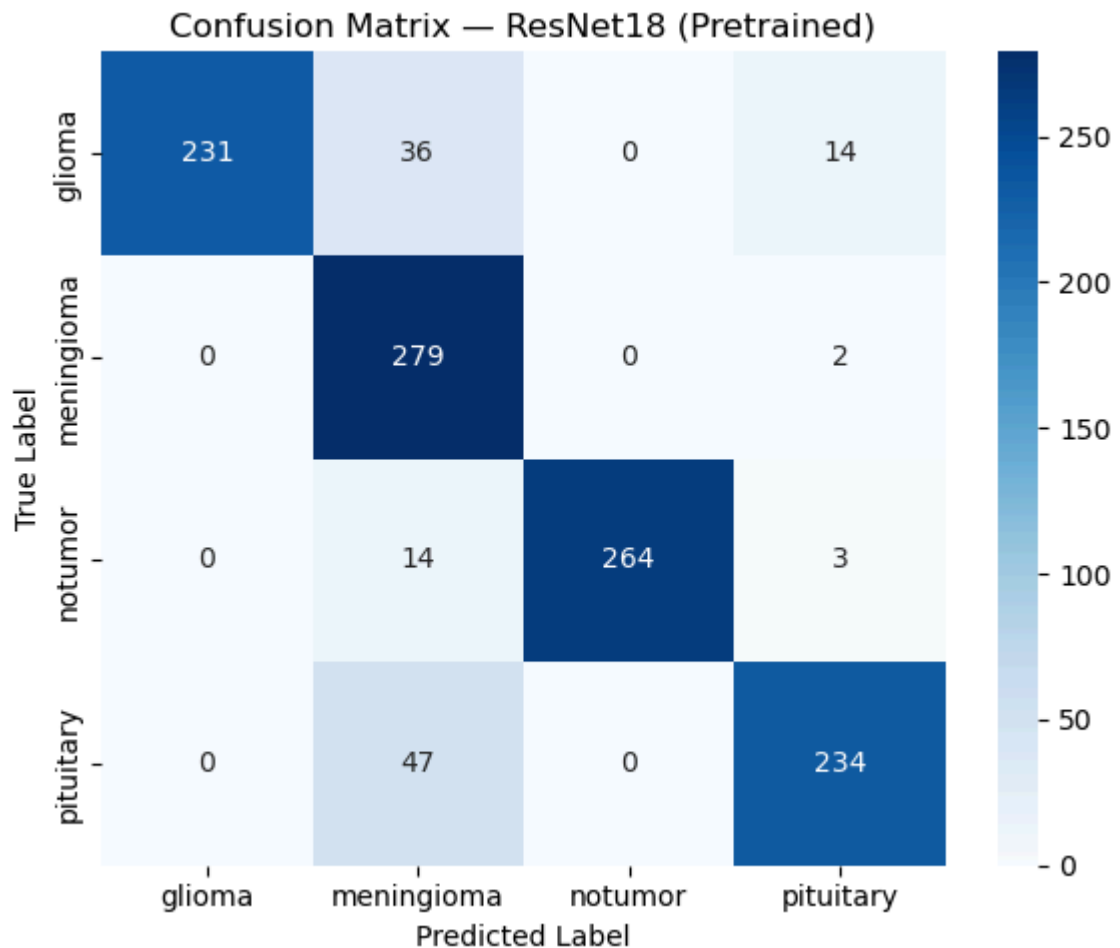
Precision: 0.9167

Recall : 0.8968

F1-score : 0.8992

Classification Report:

	precision	recall	f1-score	support
glioma	1.00	0.82	0.90	281
meningioma	0.74	0.99	0.85	281
notumor	1.00	0.94	0.97	281
pituitary	0.92	0.83	0.88	281
accuracy			0.90	1124
macro avg	0.92	0.90	0.90	1124
weighted avg	0.92	0.90	0.90	1124



Model 4: ResNet18 — From Scratch

```
In [16]: print('Training ResNet18 (From Scratch)...')

model_res_scratch = models.resnet18(pretrained=False)
model_res_scratch.fc = nn.Linear(
    model_res_scratch.fc.in_features, num_classes
)
model_res_scratch = model_res_scratch.to(device)

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

model_res_scratch = train_model(
    model_res_scratch, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_res_scratch)
res_scr_results = evaluate_model('ResNet18 (From Scratch)', y_true, y_pred)
```

Training ResNet18 (From Scratch)...

Epoch 1/5

train Loss: 0.8444 Acc: 0.6708

val Loss: 0.7555 Acc: 0.7125

Epoch 2/5

train Loss: 0.6297 Acc: 0.7561

val Loss: 0.6486 Acc: 0.7589

Epoch 3/5

train Loss: 0.5458 Acc: 0.7962

val Loss: 0.6498 Acc: 0.7375

Epoch 4/5

train Loss: 0.4481 Acc: 0.8327

val Loss: 0.5656 Acc: 0.8036

Epoch 5/5

train Loss: 0.4152 Acc: 0.8445

val Loss: 1.4288 Acc: 0.5375

100%|██████████| 36/36 [00:47<00:00, 1.33s/it]

==== ResNet18 (From Scratch) Results =====

Accuracy : 0.5418

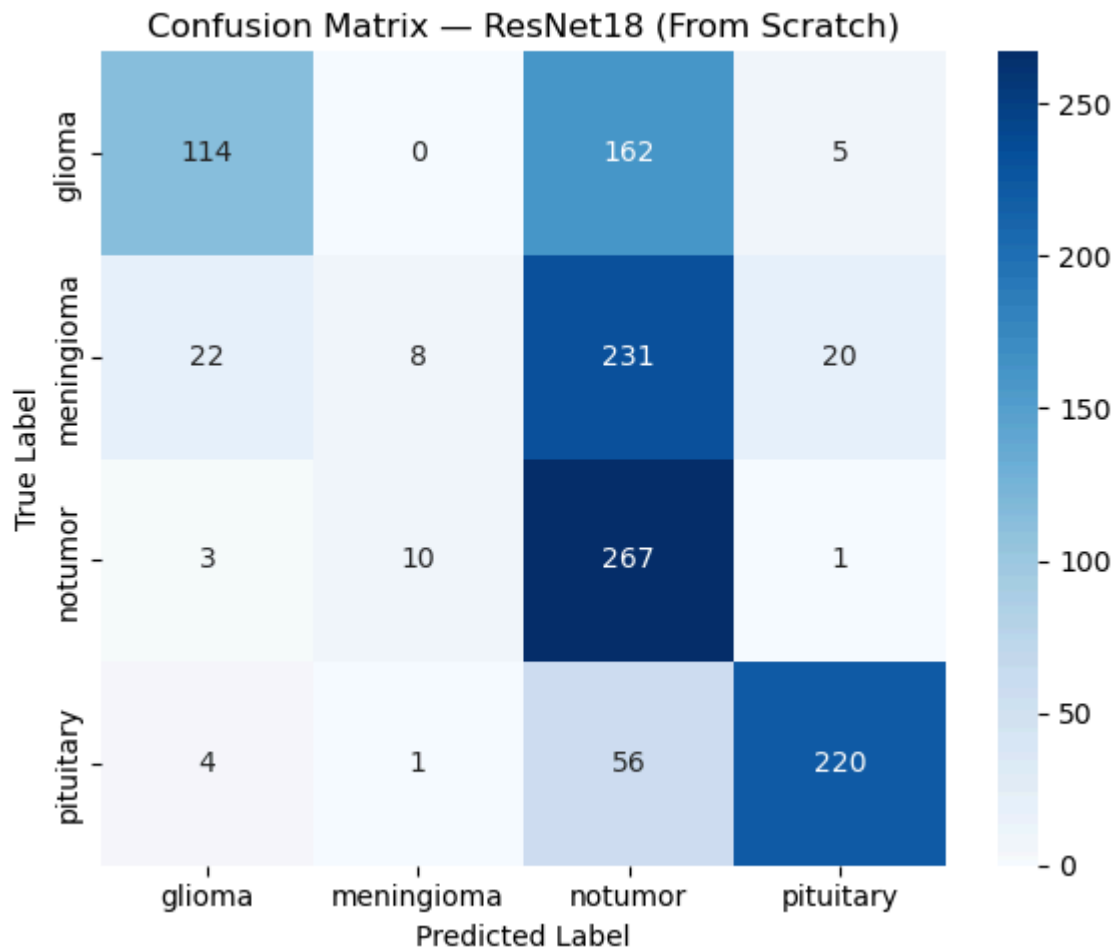
Precision: 0.6214

Recall : 0.5418

F1-score : 0.4904

Classification Report:

	precision	recall	f1-score	support
glioma	0.80	0.41	0.54	281
meningioma	0.42	0.03	0.05	281
notumor	0.37	0.95	0.54	281
pituitary	0.89	0.78	0.83	281
accuracy			0.54	1124
macro avg	0.62	0.54	0.49	1124
weighted avg	0.62	0.54	0.49	1124



Model 5: MobileNet-V2 — Pretrained (Transfer Learning)

```
In [17]: print('Training MobileNet-V2 (Pretrained)...')

model_mob_pretrained = models.mobilenet_v2(pretrained=True)
model_mob_pretrained.classifier[1] = nn.Linear(
    model_mob_pretrained.classifier[1].in_features, num_classes
)
model_mob_pretrained = model_mob_pretrained.to(device)

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

model_mob_pretrained = train_model(
    model_mob_pretrained, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_mob_pretrained)
mob_pre_results = evaluate_model('MobileNet-V2 (Pretrained)', y_true, y_pred)
```

Training MobileNet-V2 (Pretrained)...

Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to
C:\Users\F\.cache\torch\hub\checkpoints\mobilenet_v2-b0353104.pth

100%|██████████| 13.6M/13.6M [00:01<00:00, 9.35MB/s]

Epoch 1/5

train Loss: 0.3882 Acc: 0.8621

val Loss: 0.2517 Acc: 0.9250

Epoch 2/5

train Loss: 0.2612 Acc: 0.9060

val Loss: 0.3144 Acc: 0.9143

Epoch 3/5

train Loss: 0.2144 Acc: 0.9236

val Loss: 0.2134 Acc: 0.9286

Epoch 4/5

train Loss: 0.1822 Acc: 0.9379

val Loss: 0.1084 Acc: 0.9714

Epoch 5/5

train Loss: 0.1801 Acc: 0.9387

val Loss: 0.1554 Acc: 0.9589

100%|██████████| 36/36 [00:37<00:00, 1.04s/it]

==== MobileNet-V2 (Pretrained) Results =====

Accuracy : 0.9520

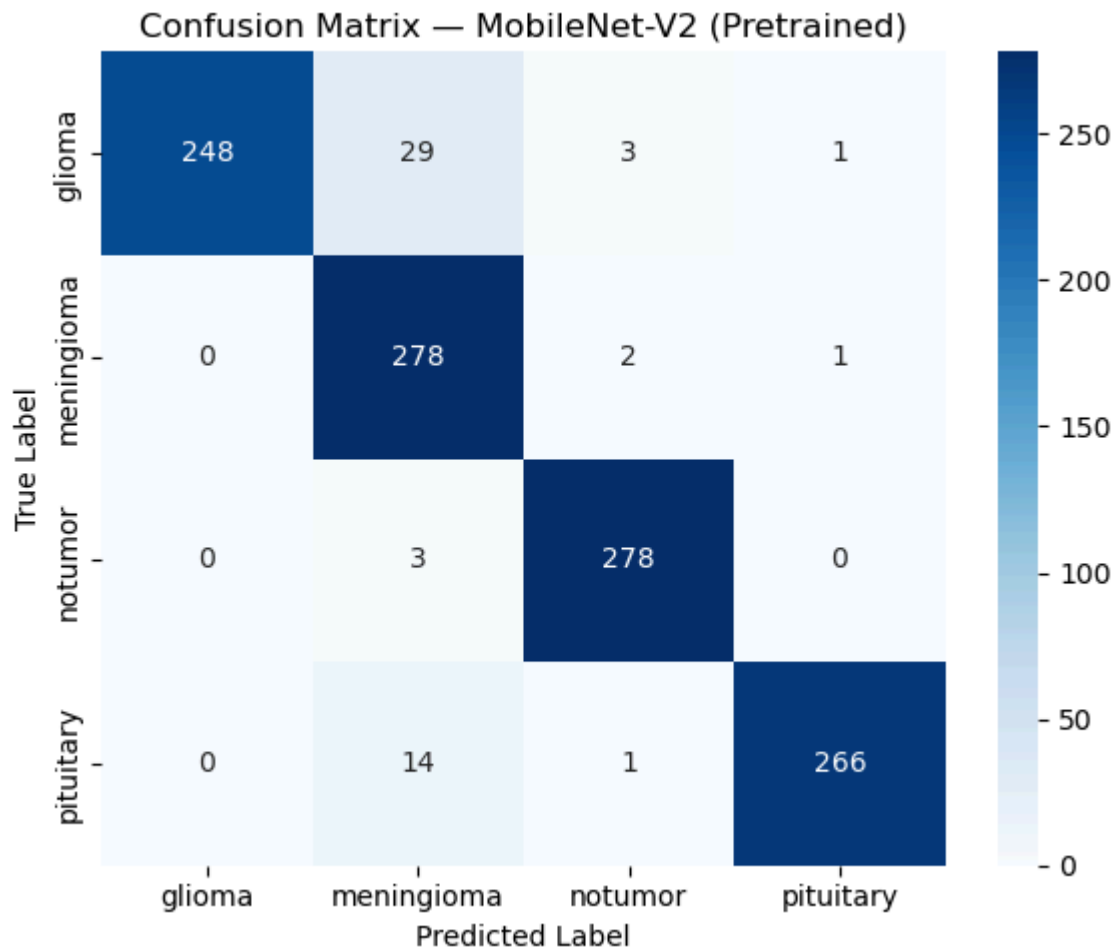
Precision: 0.9574

Recall : 0.9520

F1-score : 0.9524

Classification Report:

	precision	recall	f1-score	support
glioma	1.00	0.88	0.94	281
meningioma	0.86	0.99	0.92	281
notumor	0.98	0.99	0.98	281
pituitary	0.99	0.95	0.97	281
accuracy			0.95	1124
macro avg	0.96	0.95	0.95	1124
weighted avg	0.96	0.95	0.95	1124



Model 6: MobileNet-V2 — From Scratch

```
In [18]: print('Training MobileNet-V2 (From Scratch)...')

model_mob_scratch = models.mobilenet_v2(pretrained=False)
model_mob_scratch.classifier[1] = nn.Linear(
    model_mob_scratch.classifier[1].in_features, num_classes
)
model_mob_scratch = model_mob_scratch.to(device)

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

model_mob_scratch = train_model(
    model_mob_scratch, criterion, optimizer, scheduler, num_epochs=5
)

y_true, y_pred = get_predictions(model_mob_scratch)
mob_scr_results = evaluate_model('MobileNet-V2 (From Scratch)', y_true, y_pred)
```

Training MobileNet-V2 (From Scratch)...

Epoch 1/5

train Loss: 0.8921 Acc: 0.6320

val Loss: 1.1558 Acc: 0.6036

Epoch 2/5

train Loss: 0.6125 Acc: 0.7697

val Loss: 0.7634 Acc: 0.6768

Epoch 3/5

train Loss: 0.5159 Acc: 0.8049

val Loss: 0.7069 Acc: 0.7089

Epoch 4/5

train Loss: 0.4360 Acc: 0.8353

val Loss: 0.3781 Acc: 0.8500

Epoch 5/5

train Loss: 0.3650 Acc: 0.8636

val Loss: 0.9178 Acc: 0.6750

100%|██████████| 36/36 [00:30<00:00, 1.19it/s]

==== MobileNet-V2 (From Scratch) Results =====

Accuracy : 0.6904

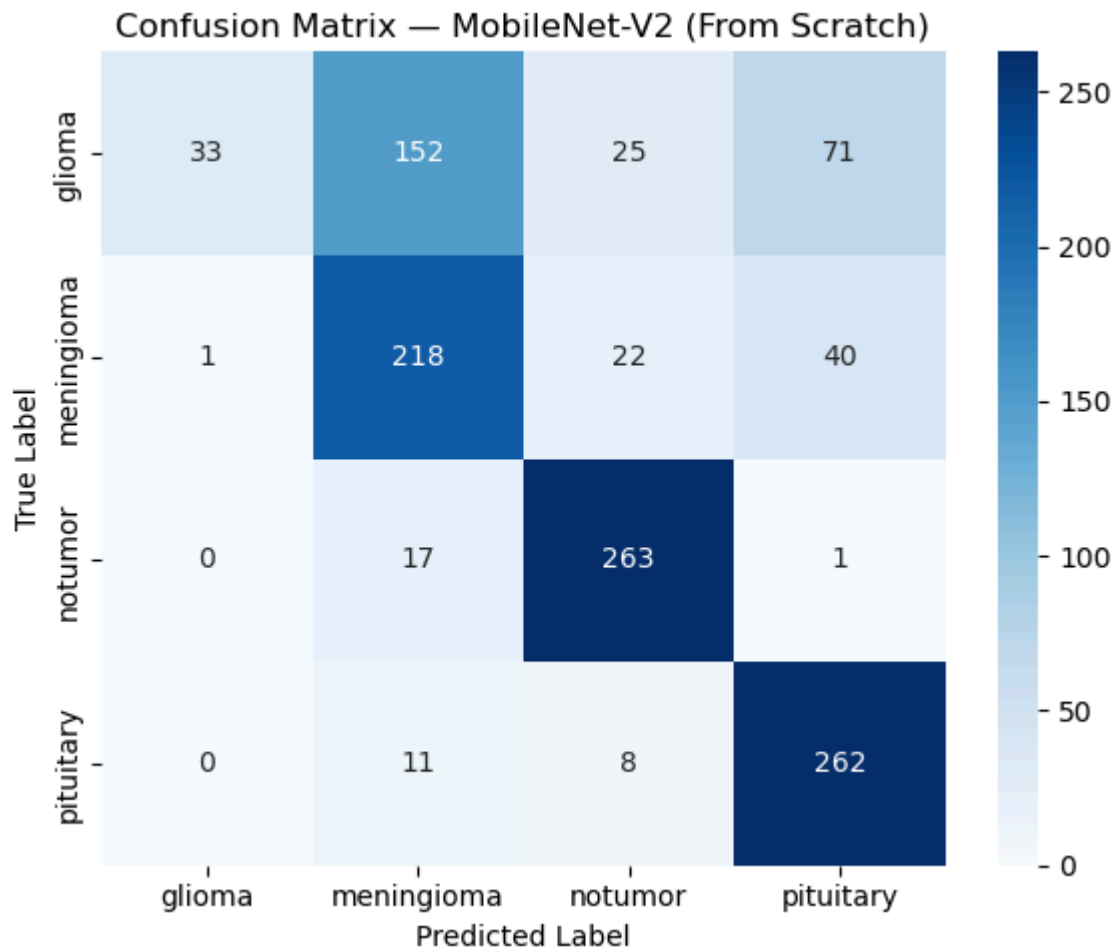
Precision: 0.7615

Recall : 0.6904

F1-score : 0.6324

Classification Report:

	precision	recall	f1-score	support
glioma	0.97	0.12	0.21	281
meningioma	0.55	0.78	0.64	281
notumor	0.83	0.94	0.88	281
pituitary	0.70	0.93	0.80	281
accuracy			0.69	1124
macro avg	0.76	0.69	0.63	1124
weighted avg	0.76	0.69	0.63	1124



Final Comparison: All Models

```
In [19]: model_names = [
    'EfficientNet-B0\n(Pretrained)',
    'EfficientNet-B0\n(Scratch)',
    'ResNet18\n(Pretrained)',
    'ResNet18\n(Scratch)',
    'MobileNet-V2\n(Pretrained)',
    'MobileNet-V2\n(Scratch)'
]

all_results = [
    eff_pre_results,
    eff_scr_results,
    res_pre_results,
    res_scr_results,
    mob_pre_results,
    mob_scr_results
]

accuracies = [r[0] for r in all_results]
precisions = [r[1] for r in all_results]
recalls = [r[2] for r in all_results]
f1_scores = [r[3] for r in all_results]

# ---- Summary Table ----
```

```

print('\n' + '='*75)
print(f'{"Model":<35} {"Accuracy":>9} {"Precision":>10} {"Recall":>8} {"F1":>8}')
print('='*75)
for name, acc, prec, rec, f1 in zip(
    model_names, accuracies, precisions, recalls, f1_scores
):
    name_clean = name.replace('\n', ' ')
    print(f'{"name_clean":<35} {"acc":>9.4f} {"prec":>10.4f} {"rec":>8.4f} {"f1":>8.4f}')
print('='*75)

# ---- Bar Chart ----
x = np.arange(len(model_names))
width = 0.2

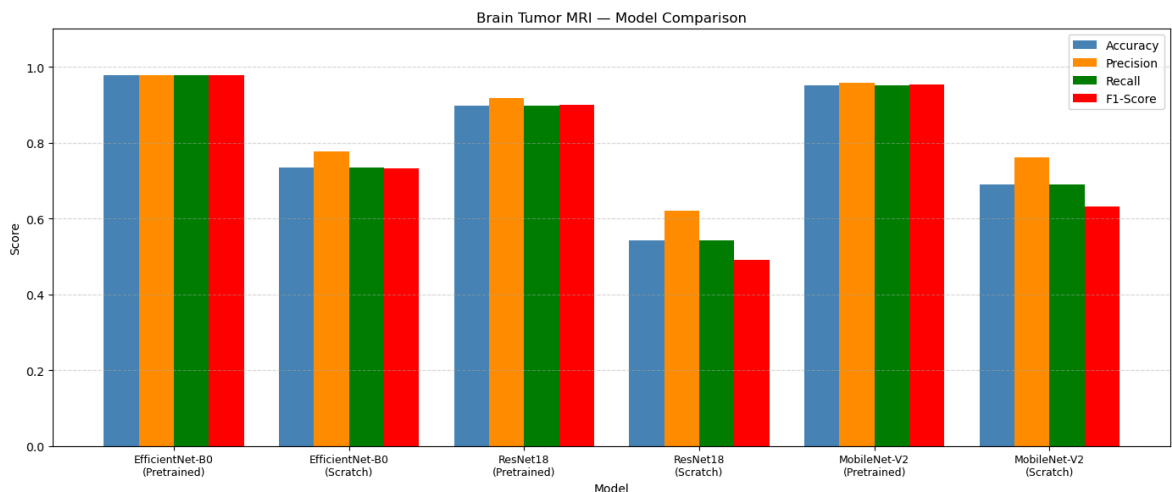
fig, ax = plt.subplots(figsize=(14, 6))
ax.bar(x - 1.5*width, accuracies, width, label='Accuracy', color='steelblue')
ax.bar(x - 0.5*width, precisions, width, label='Precision', color='darkorange')
ax.bar(x + 0.5*width, recalls, width, label='Recall', color='green')
ax.bar(x + 1.5*width, f1_scores, width, label='F1-Score', color='red')

ax.set_xlabel('Model')
ax.set_ylabel('Score')
ax.set_title('Brain Tumor MRI — Model Comparison')
ax.set_xticks(x)
ax.set_xticklabels(model_names, fontsize=9)
ax.set_ylim(0, 1.1)
ax.legend()
ax.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

best_idx = np.argmax(accuracies)
print(f'\nBest Model: {model_names[best_idx].replace(chr(10), " ")} '
      f'with Accuracy = {accuracies[best_idx]:.4f}')

```

Model	Accuracy	Precision	Recall	F1
EfficientNet-B0 (Pretrained)	0.9778	0.9779	0.9778	0.9777
EfficientNet-B0 (Scratch)	0.7349	0.7766	0.7349	0.7330
ResNet18 (Pretrained)	0.8968	0.9167	0.8968	0.8992
ResNet18 (Scratch)	0.5418	0.6214	0.5418	0.4904
MobileNet-V2 (Pretrained)	0.9520	0.9574	0.9520	0.9524
MobileNet-V2 (Scratch)	0.6904	0.7615	0.6904	0.6324



Best Model: EfficientNet-B0 (Pretrained) with Accuracy = 0.9778



Fatimah Aldighethair



253000292