```
# import kagglehub

# # Download latest version
# path = kagglehub.dataset_download("mateuszbuda/lgg-mri-segmentation")

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

```
Using Colab cache for faster access to the 'lgg-mri-segmentation' dataset.
Path to dataset files: /kaggle/input/lgg-mri-segmentation
```

```
!pip install kagglehub -q

import kagglehub
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import models, transforms
from PIL import Image
import numpy as np
import os
import glob
from collections import namedtuple
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

print("PyTorch Version:", torch.__version__)
print("CUDA Available:", torch.cuda.is_available())
if torch.cuda.is_available():
    print("CUDA Device:", torch.cuda.get_device_name(0))
```

```
PyTorch Version: 2.8.0+cu126
CUDA Available: True
CUDA Device: Tesla T4
```

Downloading the data set

```
path = kagglehub.dataset_download("mateuszbuda/lgg-mri-segmentation")
print("Path to dataset files:", path)

brain_mri_path = os.path.join(path, "kaggle_3m")
print(f"Brain MRI Path: {brain_mri_path}")
```

```
Using Colab cache for faster access to the 'lgg-mri-segmentation' dataset.
Path to dataset files: /kaggle/input/lgg-mri-segmentation
Brain MRI Path: /kaggle/input/lgg-mri-segmentation/kaggle_3m
```

Data Preparation - Create Image and Label Lists

```
def prepare_data(base_path):
    """
    Prepare image paths and labels from the dataset
    Returns list of tuples: (image_path, label)
    Label: 1 if tumor exists (has mask), 0 if no tumor
    """
    image_label_pairs = []

    # Get all patient directories
    patient_dirs = [d for d in glob.glob(os.path.join(base_path, "*")) if os.path.isdir(d)]

    print(f"Found {len(patient_dirs)} patient directories")

    for patient_dir in patient_dirs:
        # Get all image files in this directory
        image_files = glob.glob(os.path.join(patient_dir, "*[!_mask].tif"))

        for img_path in image_files:
            # Check if corresponding mask exists
            mask_path = img_path.replace('.tif', '_mask.tif')

            # Label: 1 if mask exists (tumor present), 0 otherwise
            if os.path.exists(mask_path):
                # Check if mask is not empty (has actual tumor pixels)
                try:
```

```
                    mask = Image.open(mask_path)
                    mask_array = np.array(mask)
                    label = 1 if np.any(mask_array > 0) else 0
                except:
                    label = 0
            else:
                label = 0

            image_label_pairs.append((img_path, label))

    return image_label_pairs

# Prepare dataset
all_data = prepare_data(brain_mri_path)
print(f"\nTotal images: {len(all_data)}")

# Count labels
tumor_count = sum(1 for _, label in all_data if label == 1)
no_tumor_count = len(all_data) - tumor_count
print(f"Images with tumor: {tumor_count}")
print(f"Images without tumor: {no_tumor_count}")

# Split into train and test (85-15 split as in original)
train_data, test_data = train_test_split(all_data, test_size=0.15, random_state=42, stratify=[label for _, label in all_data])
print(f"\nTraining samples: {len(train_data)}")
print(f"Testing samples: {len(test_data)}")
```

```
Found 110 patient directories

Total images: 3929
Images with tumor: 1373
Images without tumor: 2556

Training samples: 3339
Testing samples: 590
```

Custom Dataset Class

```
class BrainMRIDataset(Dataset):
    def __init__(self, data_list, transform=None):
        """
        Args:
            data_list: List of tuples (image_path, label)
            transform: Optional transform to be applied on images
        """
        self.data_list = data_list
        self.transform = transform

    def __len__(self):
        return len(self.data_list)

    def __getitem__(self, idx):
        img_path, label = self.data_list[idx]

        # Load image
        try:
            image = Image.open(img_path)
            # Convert to RGB (grayscale to 3 channels)
            image = image.convert('RGB')
        except Exception as e:
            print(f"Error loading image {img_path}: {e}")
            # Return a blank image if error
            image = Image.new('RGB', (256, 256), (0, 0, 0))

        # Apply transforms
        if self.transform:
            image = self.transform(image)

        return image, label
```

Data Transforms and Loaders

```
# Hyperparameters
use_cuda = torch.cuda.is_available()
Params = namedtuple('Params', ['batch_size', 'test_batch_size', 'epochs', 'lr', 'momentum', 'seed', 'cuda', 'log_interval'])
args = Params(batch_size=32, test_batch_size=32, epochs=10, lr=0.0001, momentum=0.5, seed=42, cuda=use_cuda, log_interval=10)

torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)
```

```
# Training transforms with augmentation
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.3),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Test transforms (no augmentation)
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Create datasets
train_dataset = BrainMRIDataset(train_data, transform=train_transform)
test_dataset = BrainMRIDataset(test_data, transform=test_transform)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=args.test_batch_size, shuffle=False, num_workers=2, pin_memory=True)

print(f"Train batches: {len(train_loader)}")
print(f"Test batches: {len(test_loader)}")
```

```
Train batches: 105
Test batches: 19
```

Model Definition

```
class BrainTumorClassifier(nn.Module):
    def __init__(self):
        super(BrainTumorClassifier, self).__init__()
        # Use pretrained ResNet18
        self.resnet = models.resnet18(pretrained=True)

        # Get number of features from last layer
        num_ftrs = self.resnet.fc.in_features

        # Replace final fully connected layer
        self.resnet.fc = nn.Sequential(
            nn.Linear(num_ftrs, 512),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.BatchNorm1d(512),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(p=0.3),
            nn.BatchNorm1d(256),
            nn.Linear(256, 2)  # 2 classes: tumor, no tumor
        )

        # Unfreeze all layers for fine-tuning
        for param in self.resnet.parameters():
            param.requires_grad = True

    def forward(self, x):
        return self.resnet(x)

# Initialize model
model = BrainTumorClassifier()

# Move to GPU if available
if args.cuda:
    model = model.cuda()
    print("Model moved to CUDA")

# Loss function with class weights to handle imbalance
criterion = nn.CrossEntropyLoss()

# Optimizer
optimizer = optim.Adam(model.parameters(), lr=args.lr, weight_decay=1e-4)

# Learning rate scheduler
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=3)
```

```
print("\nModel Summary:")
print(f"Total parameters: {sum(p.numel() for p in model.parameters())}")
print(f"Trainable parameters: {sum(p.numel() for p in model.parameters() if p.requires_grad)}")
```

```
Model moved to CUDA

Model Summary:
Total parameters: 11572546
Trainable parameters: 11572546
```

Training Function

```
def train_epoch(epoch, args, model, data_loader, optimizer, criterion):
    model.train()
    train_loss = 0
    correct = 0
    total = 0

    for batch_idx, (data, target) in enumerate(data_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        pred = output.argmax(dim=1)
        correct += pred.eq(target).sum().item()
        total += target.size(0)

        if batch_idx % args.log_interval == 0:
            print(f'Train Epoch: {epoch} [{batch_idx * len(data)}/{len(data_loader.dataset)} '
                  f'({100. * batch_idx / len(data_loader):.0f}%)]\t'
                  f'Loss: {train_loss/(batch_idx+1):.6f}\t'
                  f'Acc: {100.*correct/total:.2f}%')

    avg_loss = train_loss / len(data_loader)
    accuracy = 100. * correct / total
    return avg_loss, accuracy
```

Testing Function

```
def test_epoch(model, data_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    total = 0

    # For confusion matrix
    all_preds = []
    all_targets = []

    with torch.no_grad():
        for data, target in data_loader:
            if args.cuda:
                data, target = data.cuda(), target.cuda()

            output = model(data)
            loss = criterion(output, target)
            test_loss += loss.item()

            pred = output.argmax(dim=1)
            correct += pred.eq(target).sum().item()
            total += target.size(0)

            all_preds.extend(pred.cpu().numpy())
            all_targets.extend(target.cpu().numpy())

    avg_loss = test_loss / len(data_loader)
    accuracy = 100. * correct / total

    print(f'\nTest set: Average loss: {avg_loss:.4f}, '
          f'Accuracy: {correct}/{total} ({accuracy:.2f}%)\n')

    return avg_loss, accuracy, all_preds, all_targets
```

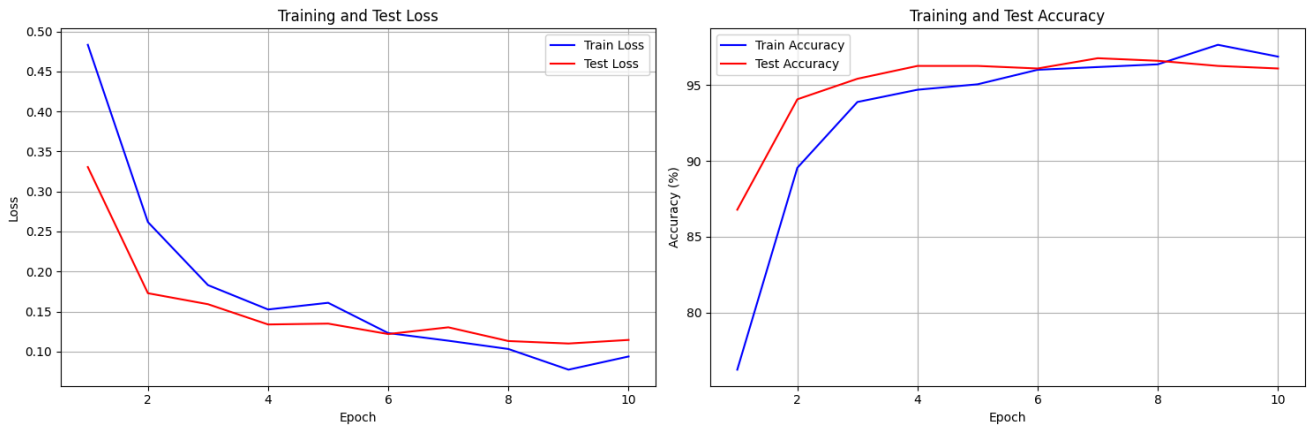
Training Loop

```
train_losses = []
train_accuracies = []
test_losses = []
test_accuracies = []

best_test_acc = 0
best_epoch = 0

print("Starting training...\n")

for epoch in range(1, args.epochs + 1):
    print(f"\n{'='*60}")
    print(f"Epoch {epoch}/{args.epochs}")
    print('='*60)

    # Train
    train_loss, train_acc = train_epoch(epoch, args, model, train_loader, optimizer, criterion)
    train_losses.append(train_loss)
    train_accuracies.append(train_acc)

    # Test
    test_loss, test_acc, preds, targets = test_epoch(model, test_loader, criterion)
    test_losses.append(test_loss)
    test_accuracies.append(test_acc)

    # Learning rate scheduling
    scheduler.step(test_loss)

    # Save best model
    if test_acc > best_test_acc:
        best_test_acc = test_acc
        best_epoch = epoch
        torch.save(model.state_dict(), 'best_brain_tumor_model.pth')
        print(f"✓ Saved best model with accuracy: {best_test_acc:.2f}%")

    print(f"\nEpoch {epoch} Summary:")
    print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%")
    print(f"Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%")

print(f"\n{'='*60}")
print(f"Training completed!")
print(f"Best test accuracy: {best_test_acc:.2f}% at epoch {best_epoch}")
print('='*60)
```

```
Train Epoch: 10 [960/3339 (29%)]        Loss: 0.074731  Acc: 97.58%
Train Epoch: 10 [1280/3339 (38%)]       Loss: 0.085198  Acc: 97.33%
Train Epoch: 10 [1600/3339 (48%)]       Loss: 0.090000  Acc: 97.18%
Train Epoch: 10 [1920/3339 (57%)]       Loss: 0.092999  Acc: 97.08%
Train Epoch: 10 [2240/3339 (67%)]       Loss: 0.089685  Acc: 97.14%
Train Epoch: 10 [2560/3339 (76%)]       Loss: 0.100943  Acc: 96.76%
Train Epoch: 10 [2880/3339 (86%)]       Loss: 0.097726  Acc: 96.81%
Train Epoch: 10 [3200/3339 (95%)]       Loss: 0.093379  Acc: 96.91%

Test set: Average loss: 0.1145, Accuracy: 567/590 (96.10%)


Epoch 10 Summary:
Train Loss: 0.0938, Train Acc: 96.89%
Test Loss: 0.1145, Test Acc: 96.10%

============================================================
Training completed!
Best test accuracy: 96.78% at epoch 7
============================================================
```

Plot Training History

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# Plot losses
ax1.plot(range(1, args.epochs + 1), train_losses, 'b-', label='Train Loss')
ax1.plot(range(1, args.epochs + 1), test_losses, 'r-', label='Test Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.set_title('Training and Test Loss')
ax1.legend()
ax1.grid(True)

# Plot accuracies
ax2.plot(range(1, args.epochs + 1), train_accuracies, 'b-', label='Train Accuracy')
ax2.plot(range(1, args.epochs + 1), test_accuracies, 'r-', label='Test Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (%)')
ax2.set_title('Training and Test Accuracy')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.savefig('training_history.png', dpi=300, bbox_inches='tight')
plt.show()
```

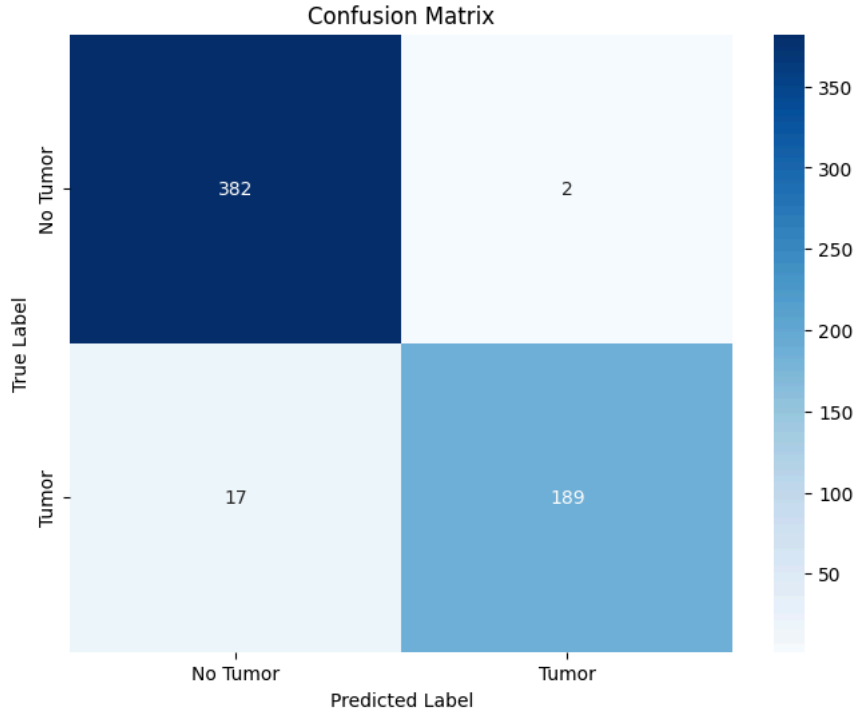



Confusion Matrix and Classification Report

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Load best model
model.load_state_dict(torch.load('best_brain_tumor_model.pth'))
_, _, final_preds, final_targets = test_epoch(model, test_loader, criterion)

# Confusion matrix
```

```
# Confusion matrix
cm = confusion_matrix(final_targets, final_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Tumor', 'Tumor'], yticklabels=['No Tumor', 'Tumor'])
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.savefig('confusion_matrix.png', dpi=300, bbox_inches='tight')
plt.show()

# Classification report
print("\nClassification Report:")
print(classification_report(final_targets, final_preds, target_names=['No Tumor', 'Tumor']))
```

```
Test set: Average loss: 0.1303, Accuracy: 571/590 (96.78%)
```

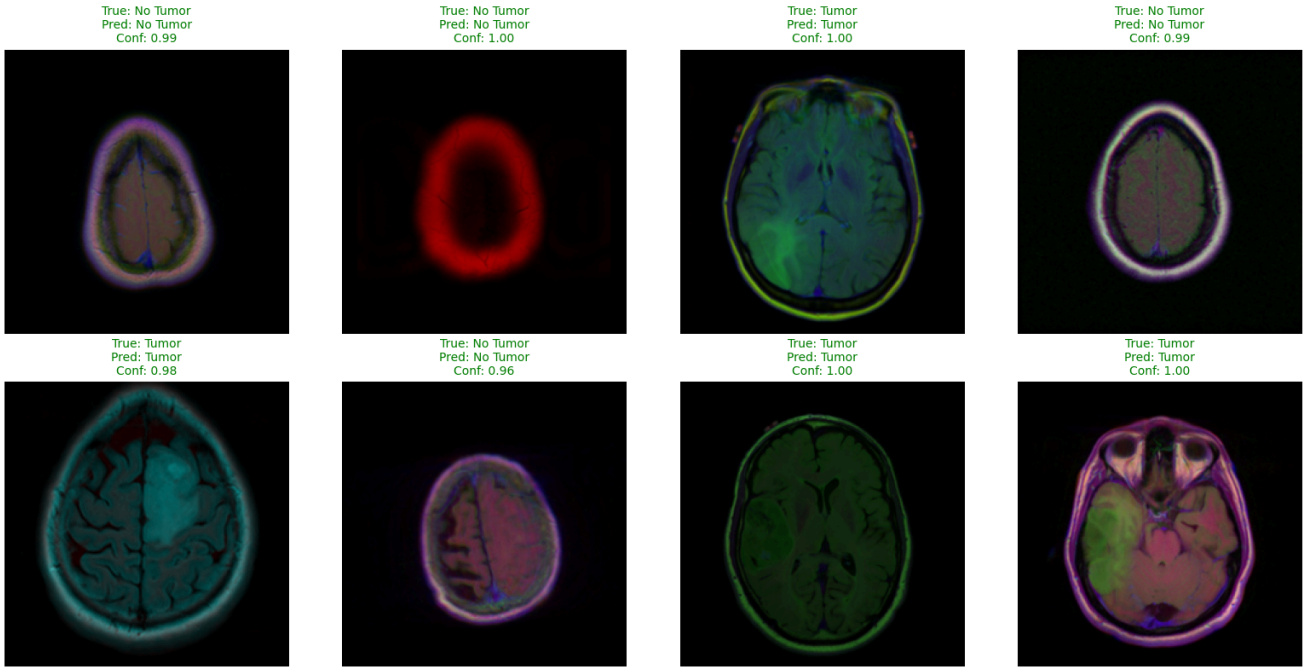



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.96      0.99      0.98       384
       Tumor       0.99      0.92      0.95       206

    accuracy                           0.97       590
   macro avg       0.97      0.96      0.96       590
weighted avg       0.97      0.97      0.97       590
```

Visualize Sample Predictions

```
def visualize_predictions(model, dataset, num_samples=8):
    model.eval()
    fig, axes = plt.subplots(2, 4, figsize=(16, 8))
    axes = axes.ravel()

    indices = np.random.choice(len(dataset), num_samples, replace=False)

    with torch.no_grad():
        for idx, ax in zip(indices, axes):
            img, true_label = dataset[idx]
            img_tensor = img.unsqueeze(0)

            if args.cuda:
                img_tensor = img_tensor.cuda()

            output = model(img_tensor)
            pred_label = output.argmax(dim=1).item()
            confidence = F.softmax(output, dim=1).max().item()

            # Denormalize image for display
            img_display = img.permute(1, 2, 0).cpu().numpy()
            img_display = img_display * np.array([0.229, 0.224, 0.225]) + np.array([0.485, 0.456, 0.406])
```

```
            img_display = np.clip(img_display, 0, 1)

            ax.imshow(img_display)
            ax.axis('off')

            true_label_text = 'Tumor' if true_label == 1 else 'No Tumor'
            pred_label_text = 'Tumor' if pred_label == 1 else 'No Tumor'
            color = 'green' if true_label == pred_label else 'red'

            ax.set_title(f'True: {true_label_text}\nPred: {pred_label_text}\nConf: {confidence:.2f}',
                         color=color, fontsize=10)

    plt.tight_layout()
    plt.savefig('sample_predictions.png', dpi=300, bbox_inches='tight')
    plt.show()

visualize_predictions(model, test_dataset, num_samples=8)

print("\n✓ Training complete! Model saved as 'best_brain_tumor_model.pth'")
```




```
✓ Training complete! Model saved as 'best_brain_tumor_model.pth'
```