STEP 1: Dataset Download & Paths

```python
import kagglehub
import os

path = kagglehub.dataset_download(

    "masoudnickparvar/brain-tumor-mri-dataset"
)

TRAIN_DIR = os.path.join(path, "Training")
TEST_DIR  = os.path.join(path, "Testing")

print("Training classes:", os.listdir(TRAIN_DIR))
print("Testing classes :", os.listdir(TEST_DIR))
```

```
Using Colab cache for faster access to the 'brain-tumor-mri-dataset'
dataset.
Training classes: ['pituitary', 'notumor', 'meningioma', 'glioma']
Testing classes : ['pituitary', 'notumor', 'meningioma', 'glioma']
```

STEP 2: Imports & Device

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np

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

```
Using device: cuda
```

STEP 3: Preprocessing & Augmentation

```python
train_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.Grayscale(num_output_channels=1),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])

test_transforms = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.Grayscale(num_output_channels=1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
```

STEP 4: Dataset & Loaders

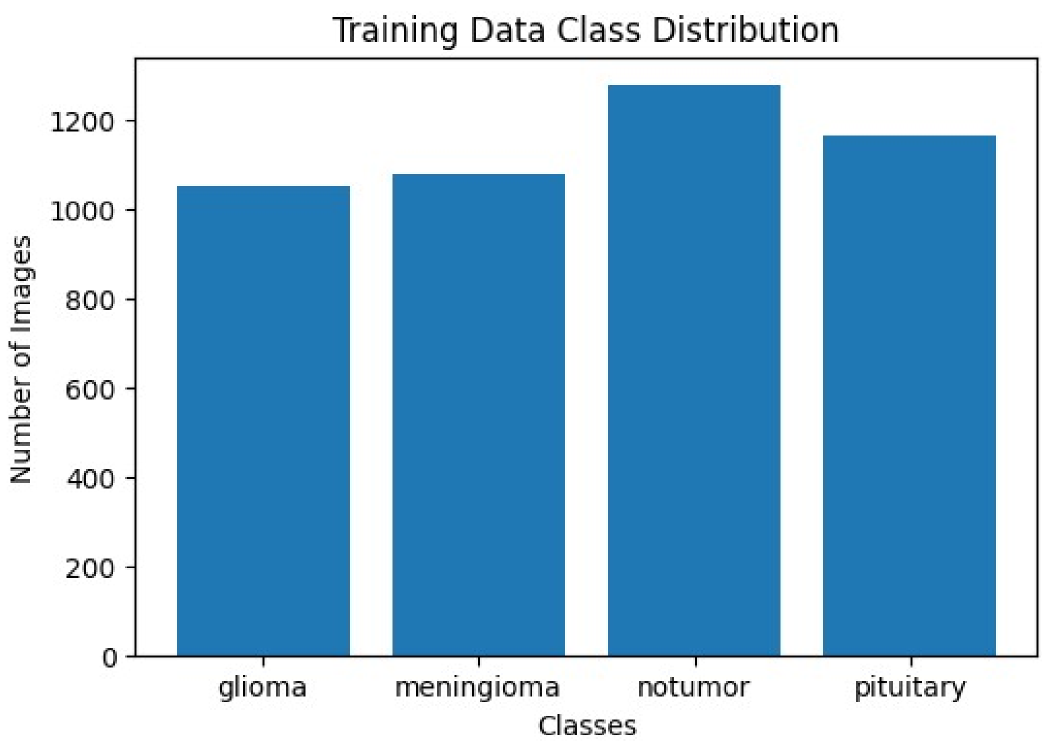
```
full_train_dataset = datasets.ImageFolder(
    TRAIN_DIR, transform=train_transforms
)

test_dataset = datasets.ImageFolder(
    TEST_DIR, transform=test_transforms
)

class_names = full_train_dataset.classes
num_classes = len(class_names)
print("Classes:", class_names)
train_size = int(0.8 * len(full_train_dataset))
val_size   = len(full_train_dataset) - train_size

generator = torch.Generator().manual_seed(42)
train_dataset, val_dataset = random_split(
    full_train_dataset, [train_size, val_size], generator=generator
)


train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader   = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
Classes: ['glioma', 'meningioma', 'notumor', 'pituitary']
```

STEP 5: Data Visualization

(a) Class Distribution

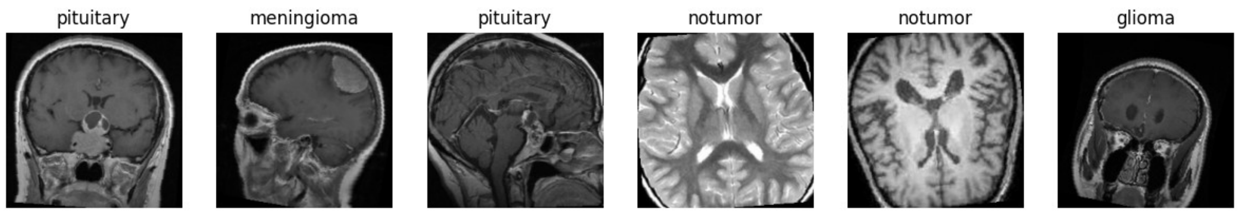
```
# This plot shows the number of samples per tumor category.import
matplotlib.pyplot as plt
from collections import Counter

labels = [label for _, label in train_dataset]
label_count = Counter(labels)

plt.figure(figsize=(6,4))
plt.bar(class_names, [label_count[i] for i in
range(len(class_names))])
plt.title("Training Data Class Distribution")
plt.xlabel("Classes")
plt.ylabel("Number of Images")
plt.show()
```



(b) Sample Images

```
def show_images(loader):
    images, labels = next(iter(loader))
    images, labels = images[:6], labels[:6]

    fig, axes = plt.subplots(1, 6, figsize=(15,3))
```

```
    for i in range(6):
        img = images[i].squeeze(0) * 0.5 + 0.5
        axes[i].imshow(img, cmap='gray')
        axes[i].set_title(class_names[labels[i]])
        axes[i].axis("off")
    plt.show()

show_images(train_loader)
```



STEP 6: CNN Model

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.AdaptiveAvgPool2d((7,7))
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 7 * 7, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
```

```
        x = self.classifier(x)
        return x
```

STEP 7: Loss & Optimizer

```
model = SimpleCNN(num_classes).to(device)
print(model)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
SimpleCNN(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 
1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 
1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), 
padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, 
ceil_mode=False)
    (9): AdaptiveAvgPool2d(output_size=(7, 7))
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=6272, out_features=256, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=4, bias=True)
  )
)
```

STEP 8: TRAIN

```
def train_model(epochs=25):
    train_accs, val_accs, test_accs = [], [], []
    train_losses, val_losses, test_losses = [], [], []

    for epoch in range(epochs):
        # -------- TRAINING --------
```

```
        model.train()
        correct, total = 0, 0
        running_loss = 0.0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        train_acc = 100 * correct / total
        train_loss = running_loss / len(train_loader)
        train_accs.append(train_acc)
        train_losses.append(train_loss)

        # -------- VALIDATION --------
        model.eval()
        correct, total = 0, 0
        running_loss = 0.0

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)

                running_loss += loss.item()
                _, preds = torch.max(outputs, 1)
                correct += (preds == labels).sum().item()
                total += labels.size(0)

        val_acc = 100 * correct / total
        val_loss = running_loss / len(val_loader)
        val_accs.append(val_acc)
        val_losses.append(val_loss)

        # -------- TEST --------
        correct, total = 0, 0
        running_loss = 0.0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
```

```
                outputs = model(images)
                loss = criterion(outputs, labels)

                running_loss += loss.item()
                _, preds = torch.max(outputs, 1)
                correct += (preds == labels).sum().item()
                total += labels.size(0)

        test_acc = 100 * correct / total
        test_loss = running_loss / len(test_loader)
        test_accs.append(test_acc)
        test_losses.append(test_loss)

        print(f"Epoch [{epoch+1}/{epochs}] "
              f"Train Acc: {train_acc:.2f}% | Val Acc: {val_acc:.2f}%
| Test Acc: {test_acc:.2f}% "
              f"Train Loss: {train_loss:.4f} | Val Loss:
{val_loss:.4f} | Test Loss: {test_loss:.4f}")

    return train_accs, val_accs, test_accs, train_losses, val_losses,
test_losses

EPOCHS = 25
train_accs, val_accs, test_accs, train_losses, val_losses, test_losses
= train_model(EPOCHS)
```

```
Epoch [1/25] Train Acc: 72.01% | Val Acc: 79.18% | Test Acc: 72.08%
Train Loss: 0.7303 | Val Loss: 0.5602 | Test Loss: 0.6716
Epoch [2/25] Train Acc: 78.14% | Val Acc: 83.64% | Test Acc: 78.18%
Train Loss: 0.5553 | Val Loss: 0.4395 | Test Loss: 0.5550
Epoch [3/25] Train Acc: 82.03% | Val Acc: 80.40% | Test Acc: 79.25%
Train Loss: 0.4852 | Val Loss: 0.4594 | Test Loss: 0.5118
Epoch [4/25] Train Acc: 83.41% | Val Acc: 85.48% | Test Acc: 80.32%
Train Loss: 0.4335 | Val Loss: 0.3568 | Test Loss: 0.4412
Epoch [5/25] Train Acc: 84.85% | Val Acc: 87.84% | Test Acc: 83.75%
Train Loss: 0.3952 | Val Loss: 0.3153 | Test Loss: 0.3675
Epoch [6/25] Train Acc: 86.06% | Val Acc: 89.76% | Test Acc: 87.03%
Train Loss: 0.3606 | Val Loss: 0.2714 | Test Loss: 0.3277
Epoch [7/25] Train Acc: 87.90% | Val Acc: 87.23% | Test Acc: 86.35%
Train Loss: 0.3253 | Val Loss: 0.3312 | Test Loss: 0.3581
Epoch [8/25] Train Acc: 88.29% | Val Acc: 90.38% | Test Acc: 88.94%
Train Loss: 0.3039 | Val Loss: 0.2499 | Test Loss: 0.2847
Epoch [9/25] Train Acc: 89.43% | Val Acc: 92.48% | Test Acc: 91.08%
Train Loss: 0.2872 | Val Loss: 0.2154 | Test Loss: 0.2376
Epoch [10/25] Train Acc: 90.22% | Val Acc: 91.43% | Test Acc: 91.38%
Train Loss: 0.2657 | Val Loss: 0.2184 | Test Loss: 0.2212
Epoch [11/25] Train Acc: 90.79% | Val Acc: 92.48% | Test Acc: 90.54%
Train Loss: 0.2496 | Val Loss: 0.2096 | Test Loss: 0.2431
Epoch [12/25] Train Acc: 91.31% | Val Acc: 93.35% | Test Acc: 91.30%
Train Loss: 0.2277 | Val Loss: 0.1853 | Test Loss: 0.2216
```

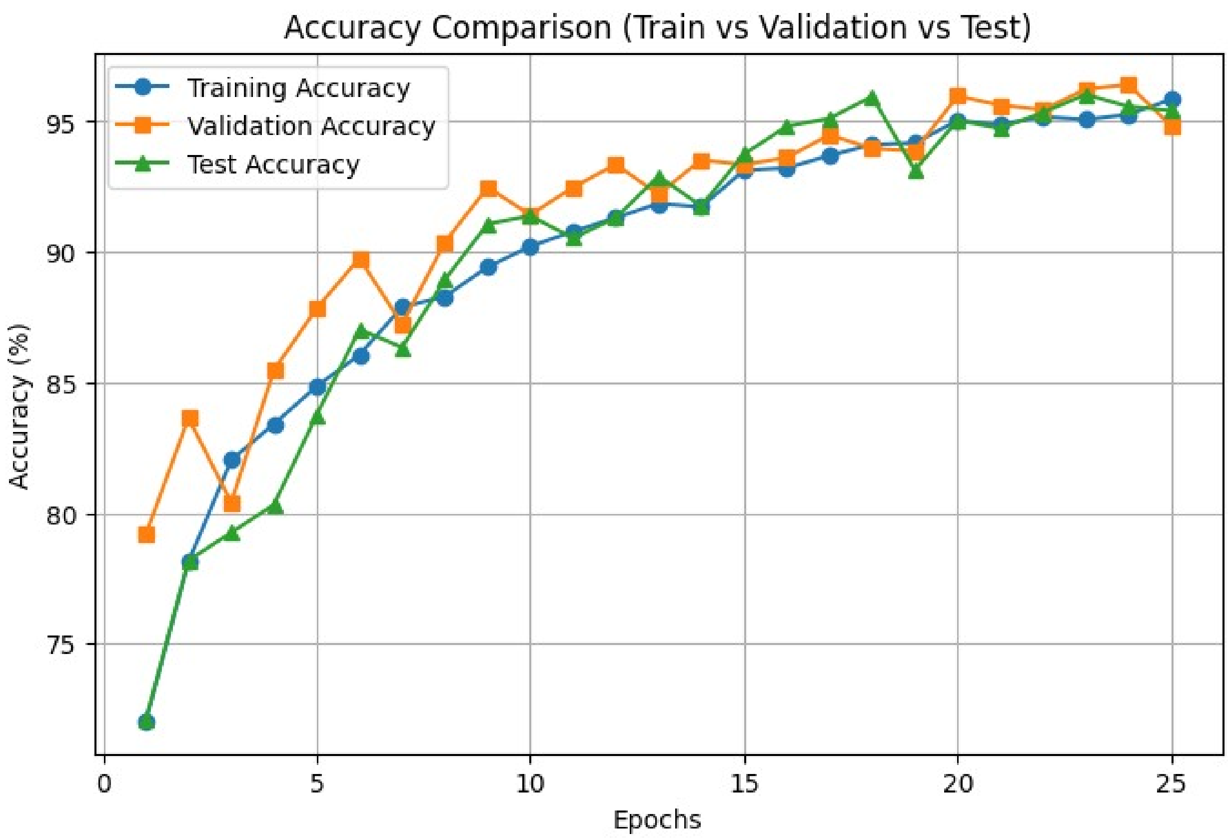
```
Epoch [13/25] Train Acc: 91.86% | Val Acc: 92.21% | Test Acc: 92.91%
Train Loss: 0.2257 | Val Loss: 0.1852 | Test Loss: 0.1969
Epoch [14/25] Train Acc: 91.73% | Val Acc: 93.53% | Test Acc: 91.76%
Train Loss: 0.2127 | Val Loss: 0.1875 | Test Loss: 0.2071
Epoch [15/25] Train Acc: 93.13% | Val Acc: 93.35% | Test Acc: 93.75%
Train Loss: 0.1913 | Val Loss: 0.1752 | Test Loss: 0.1712
Epoch [16/25] Train Acc: 93.22% | Val Acc: 93.61% | Test Acc: 94.81%
Train Loss: 0.1843 | Val Loss: 0.1685 | Test Loss: 0.1452
Epoch [17/25] Train Acc: 93.70% | Val Acc: 94.49% | Test Acc: 95.12%
Train Loss: 0.1723 | Val Loss: 0.1486 | Test Loss: 0.1463
Epoch [18/25] Train Acc: 94.11% | Val Acc: 93.96% | Test Acc: 95.96%
Train Loss: 0.1513 | Val Loss: 0.1716 | Test Loss: 0.1347
Epoch [19/25] Train Acc: 94.18% | Val Acc: 93.88% | Test Acc: 93.14%
Train Loss: 0.1666 | Val Loss: 0.1653 | Test Loss: 0.1559
Epoch [20/25] Train Acc: 95.03% | Val Acc: 95.98% | Test Acc: 95.04%
Train Loss: 0.1482 | Val Loss: 0.1302 | Test Loss: 0.1381
Epoch [21/25] Train Acc: 94.88% | Val Acc: 95.63% | Test Acc: 94.74%
Train Loss: 0.1423 | Val Loss: 0.1331 | Test Loss: 0.1594
Epoch [22/25] Train Acc: 95.18% | Val Acc: 95.45% | Test Acc: 95.35%
Train Loss: 0.1322 | Val Loss: 0.1463 | Test Loss: 0.1214
Epoch [23/25] Train Acc: 95.08% | Val Acc: 96.24% | Test Acc: 96.03%
Train Loss: 0.1328 | Val Loss: 0.1138 | Test Loss: 0.1136
Epoch [24/25] Train Acc: 95.27% | Val Acc: 96.41% | Test Acc: 95.58%
Train Loss: 0.1244 | Val Loss: 0.1328 | Test Loss: 0.1289
Epoch [25/25] Train Acc: 95.84% | Val Acc: 94.84% | Test Acc: 95.42%
Train Loss: 0.1284 | Val Loss: 0.1484 | Test Loss: 0.1196
```

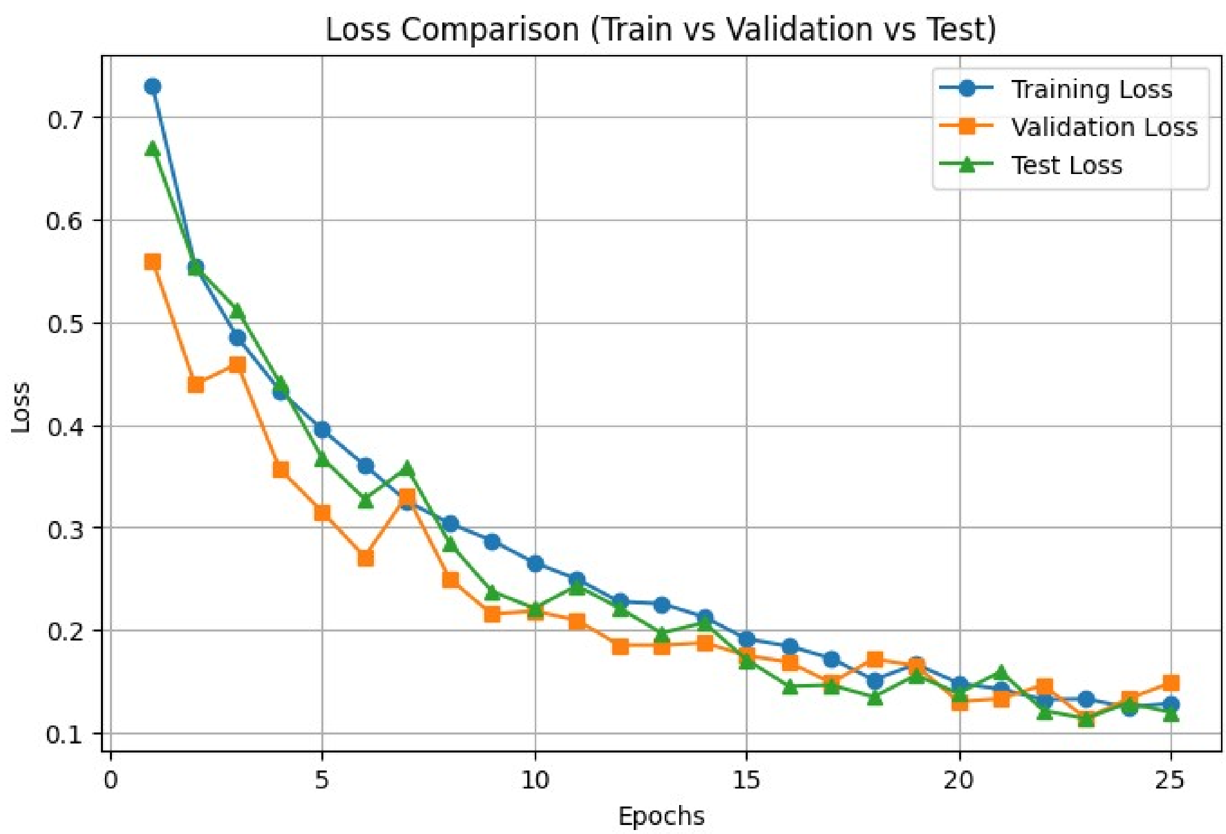
```python
test_loss = 0.0

with torch.no_grad():
    correct, total = 0, 0
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()  # add batch loss

        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

test_acc = 100 * correct / total
test_loss = test_loss / len(test_loader)  # average loss per batch

print(f"Test Accuracy: {test_acc:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```

```
Test Accuracy: 95.42%
Test Loss: 0.1196
```

STEP 9: Loss & Accuracy Curves

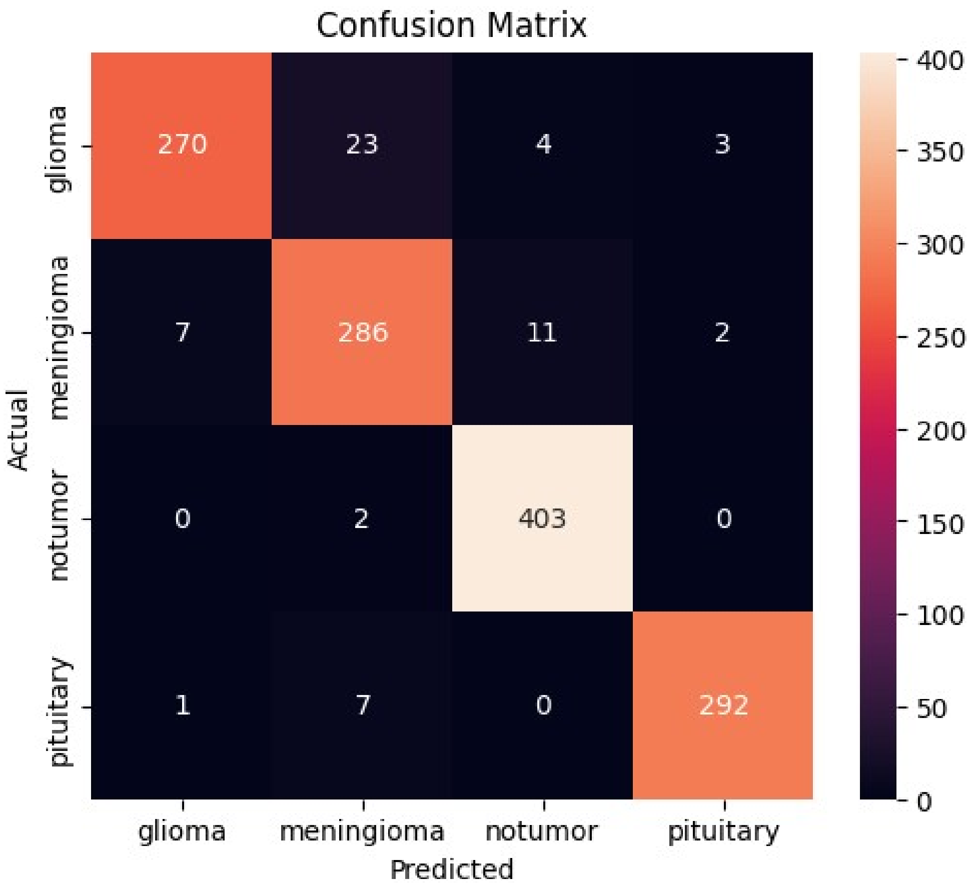
```python
import matplotlib.pyplot as plt

epochs = range(1, EPOCHS + 1)

# ----- Accuracy Curve -----
plt.figure(figsize=(8,5))
plt.plot(epochs, train_accs, marker='o', label='Training Accuracy')
plt.plot(epochs, val_accs, marker='s', label='Validation Accuracy')

# Plot Test Accuracy if available
if 'test_accs' in globals():
    plt.plot(epochs, test_accs, marker='^', label='Test Accuracy')

plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.title("Accuracy Comparison (Train vs Validation vs Test)")
plt.legend()
plt.grid(True)
plt.show()

# ----- Loss Curve -----
plt.figure(figsize=(8,5))
plt.plot(epochs, train_losses, marker='o', label='Training Loss')
plt.plot(epochs, val_losses, marker='s', label='Validation Loss')

# Plot Test Loss if available
if 'test_losses' in globals():
    plt.plot(epochs, test_losses, marker='^', label='Test Loss')

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss Comparison (Train vs Validation vs Test)")
plt.legend()
plt.grid(True)
plt.show()
```


Accuracy Comparison (Train vs Validation vs Test)
Training Accuracy
Validation Accuracy
Test Accuracy
Accuracy (%)
Epochs
95
90
85
80
75
0
5
10
15
20
25


Loss Comparison (Train vs Validation vs Test)
Training Loss
Validation Loss
Test Loss
Loss
Epochs
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25

STEP 11: Confusion Matrix & Classification Report

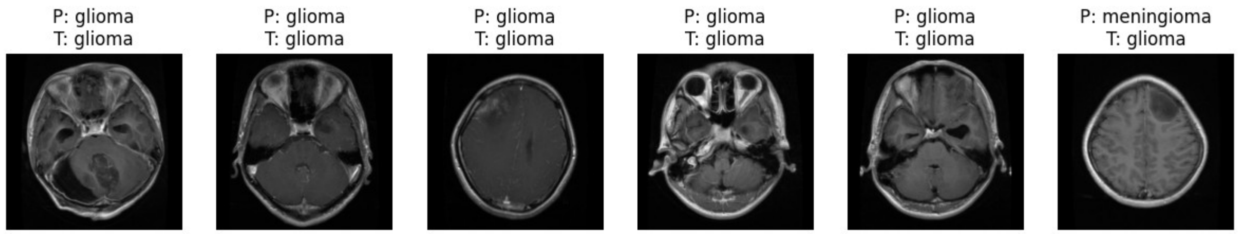
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

all_preds, all_labels = [], []

model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        outputs = model(images)
        _, preds = torch.max(outputs, 1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.numpy())
cm = confusion_matrix(all_labels, all_preds)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=class_names,
            yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
print(classification_report(all_labels, all_preds,
target_names=class_names))
```




```
              precision    recall  f1-score   support

      glioma       0.97      0.90      0.93       300
  meningioma       0.90      0.93      0.92       306
     notumor       0.96      1.00      0.98       405
   pituitary       0.98      0.97      0.98       300

    accuracy                           0.95      1311
   macro avg       0.95      0.95      0.95      1311
weighted avg       0.95      0.95      0.95      1311
```

```
def visualize_predictions():
    images, labels = next(iter(test_loader))
    images, labels = images[:6].to(device), labels[:6].to(device)

    model.eval()
    with torch.no_grad():
        outputs = model(images)
        _, preds = torch.max(outputs, 1)

    fig, axes = plt.subplots(1, 6, figsize=(15,3))
    for i in range(6):
```

```
        img = images[i].cpu().squeeze(0)
        img = img * 0.5 + 0.5   # unnormalize
        axes[i].imshow(img, cmap='gray')
        axes[i].set_title(
            f"P: {class_names[preds[i]]}\nT: {class_names[labels[i]]}"
        )
        axes[i].axis("off")

    plt.show()

visualize_predictions()
```



STEP 10: Save & Load Model (.pth)

```
torch.save(model.state_dict(), "brain_tumor_cnn.pth")
print("Model saved successfully!")

Model saved successfully!
```

Load later:

```
model.load_state_dict(torch.load("brain_tumor_cnn.pth"))
model.eval()

SimpleCNN(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (7): ReLU()
```

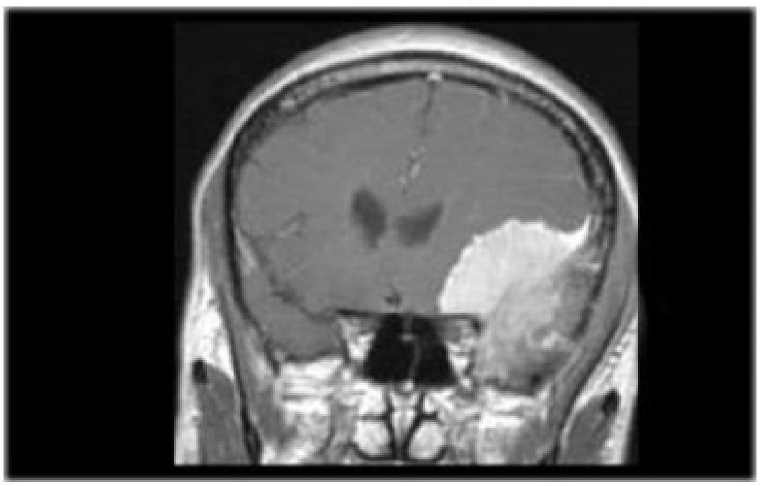
```
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (9): AdaptiveAvgPool2d(output_size=(7, 7))
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=6272, out_features=256, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=4, bias=True)
  )
)
```

```
from google.colab import files
from PIL import Image
import torch
from torchvision import transforms
import matplotlib.pyplot as plt

# Upload image
uploaded = files.upload()  # opens a file chooser
for img_name in uploaded.keys():
    print(f"Uploaded file: {img_name}")

    # Predict function
    def predict_image(img_path):
        img = Image.open(img_path).convert('L')  # grayscale

        # Transformation
        transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5], std=[0.5])
        ])
        img_tensor = transform(img).unsqueeze(0).to(device)  # add
batch dim

        # Prediction
        model.eval()
        with torch.no_grad():
            output = model(img_tensor)
            _, pred = torch.max(output, 1)
            predicted_class = class_names[pred.item()]
            print(f"Predicted Class: {predicted_class}")

        # Display image with label
        plt.figure(figsize=(5,5))
        plt.imshow(img, cmap='gray')
        plt.title(f"Predicted: {predicted_class}")
```

```
        plt.axis('off')
        plt.show()

    # Call prediction
    predict_image(img_name)
```

```
<IPython.core.display.HTML object>

Saving menin.jpg to menin.jpg
Uploaded file: menin.jpg
Predicted Class: meningioma
```

Predicted: meningioma



```

```