brain-tumor-detection-using-cnn

July 18, 2024

[22]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

[23]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[24]:
```
tf.__version__
```

[24]:
```
'2.15.0'
```

[25]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('/content/Brain MRI Images/
↪brain_tumor_dataset',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 253 images belonging to 3 classes.
```

[26]:
```
print(training_set.class_indices)
```

```
{'.ipynb_checkpoints': 0, 'no': 1, 'yes': 2}
```

[27]:
```
# Initialising the CNN
cnn = tf.keras.models.Sequential()

# Convolution
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',␣
 ↪input_shape=[64, 64, 3]))

# Pooling
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

#Adding a second convolutional layer
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

[28]:
```
# Flattening
cnn.add(tf.keras.layers.Flatten())

# Full Connection
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))

# Output Layer
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

#Compiling the CNN
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])

#Training the CNN on the Training set
cnn.fit(x = training_set, epochs = 25)
```

```
Epoch 1/25
8/8 [==============================] - 9s 600ms/step - loss: -10.3076 -
accuracy: 0.3360
Epoch 2/25
8/8 [==============================] - 4s 530ms/step - loss: -96.3953 -
accuracy: 0.3874
Epoch 3/25
8/8 [==============================] - 5s 646ms/step - loss: -509.0490 -
accuracy: 0.3874
Epoch 4/25
8/8 [==============================] - 2s 181ms/step - loss: -1699.8770 -
accuracy: 0.3874
Epoch 5/25
8/8 [==============================] - 2s 185ms/step - loss: -4807.7915 -
accuracy: 0.3874
Epoch 6/25
```

```
8/8 [==============================] - 2s 183ms/step - loss: -11633.4385 -
accuracy: 0.3874
Epoch 7/25
8/8 [==============================] - 2s 299ms/step - loss: -24540.4395 -
accuracy: 0.3874
Epoch 8/25
8/8 [==============================] - 2s 189ms/step - loss: -47227.3359 -
accuracy: 0.3874
Epoch 9/25
8/8 [==============================] - 2s 181ms/step - loss: -86518.8281 -
accuracy: 0.3874
Epoch 10/25
8/8 [==============================] - 2s 182ms/step - loss: -149408.7656 -
accuracy: 0.3874
Epoch 11/25
8/8 [==============================] - 2s 173ms/step - loss: -242831.6875 -
accuracy: 0.3874
Epoch 12/25
8/8 [==============================] - 2s 178ms/step - loss: -375284.3125 -
accuracy: 0.3874
Epoch 13/25
8/8 [==============================] - 2s 293ms/step - loss: -564481.6250 -
accuracy: 0.3874
Epoch 14/25
8/8 [==============================] - 2s 195ms/step - loss: -812956.4375 -
accuracy: 0.3874
Epoch 15/25
8/8 [==============================] - 2s 183ms/step - loss: -1165115.6250 -
accuracy: 0.3874
Epoch 16/25
8/8 [==============================] - 2s 181ms/step - loss: -1609687.8750 -
accuracy: 0.3874
Epoch 17/25
8/8 [==============================] - 2s 177ms/step - loss: -2190454.2500 -
accuracy: 0.3874
Epoch 18/25
8/8 [==============================] - 1s 175ms/step - loss: -2931506.7500 -
accuracy: 0.3874
Epoch 19/25
8/8 [==============================] - 2s 184ms/step - loss: -3783400.5000 -
accuracy: 0.3874
Epoch 20/25
8/8 [==============================] - 2s 202ms/step - loss: -4913412.0000 -
accuracy: 0.3874
Epoch 21/25
8/8 [==============================] - 2s 281ms/step - loss: -6294026.0000 -
accuracy: 0.3874
Epoch 22/25
```

```
8/8 [==============================] - 1s 177ms/step - loss: -7876591.5000 -
accuracy: 0.3874
Epoch 23/25
8/8 [==============================] - 1s 175ms/step - loss: -9804295.0000 -
accuracy: 0.3874
Epoch 24/25
8/8 [==============================] - 2s 180ms/step - loss: -12039269.0000 -
accuracy: 0.3874
Epoch 25/25
8/8 [==============================] - 1s 182ms/step - loss: -14719471.0000 -
accuracy: 0.3874
```

[28]:
```
<keras.src.callbacks.History at 0x7a0269a765f0>
```

[29]:
```python
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('/content/Brain MRI Images/brain_tumor_dataset/no/
 ↪34 no.jpg', target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'Yes'
else:
  prediction = 'No'
```

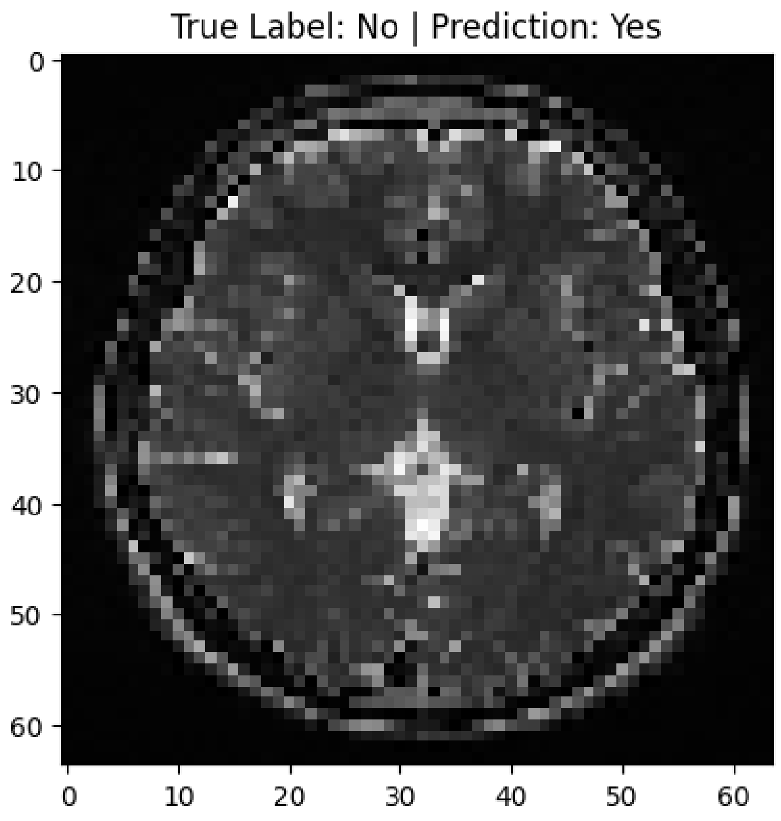
```
1/1 [==============================] - 0s 77ms/step
```

[30]:
```python
print(prediction)
```

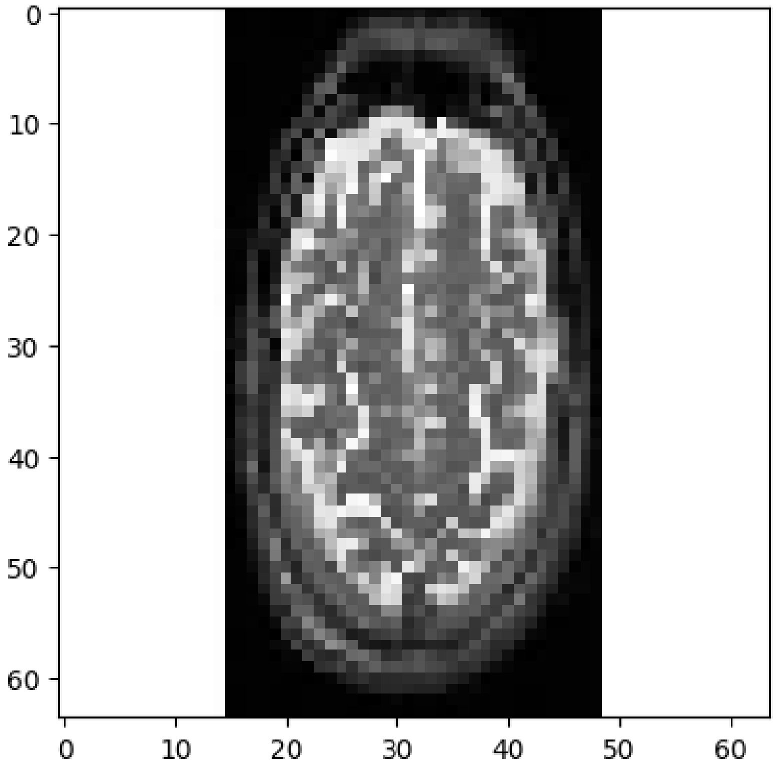
```
Yes
```

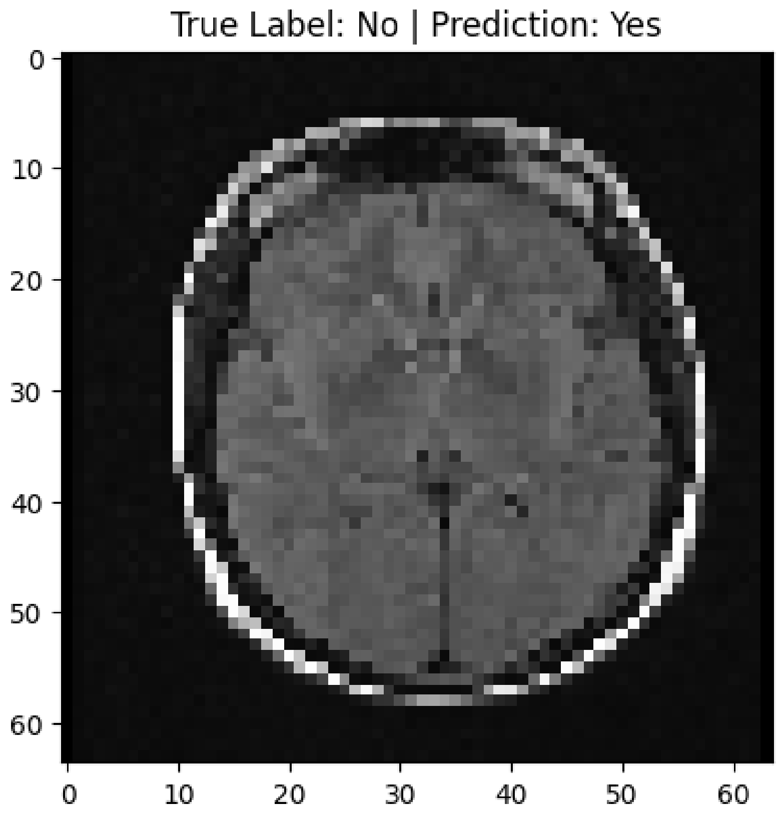
[31]:
```python
import random
from keras.preprocessing import image
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

# Function to predict an image without visualization
def predict_image(model, file_path):
    img = image.load_img(file_path, target_size=(64, 64))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    result = model.predict(img_array, verbose=0)

    return 'Yes' if result[0][0] == 1 else 'No'

# Function to visualize predictions for a subset of images with true labels
```

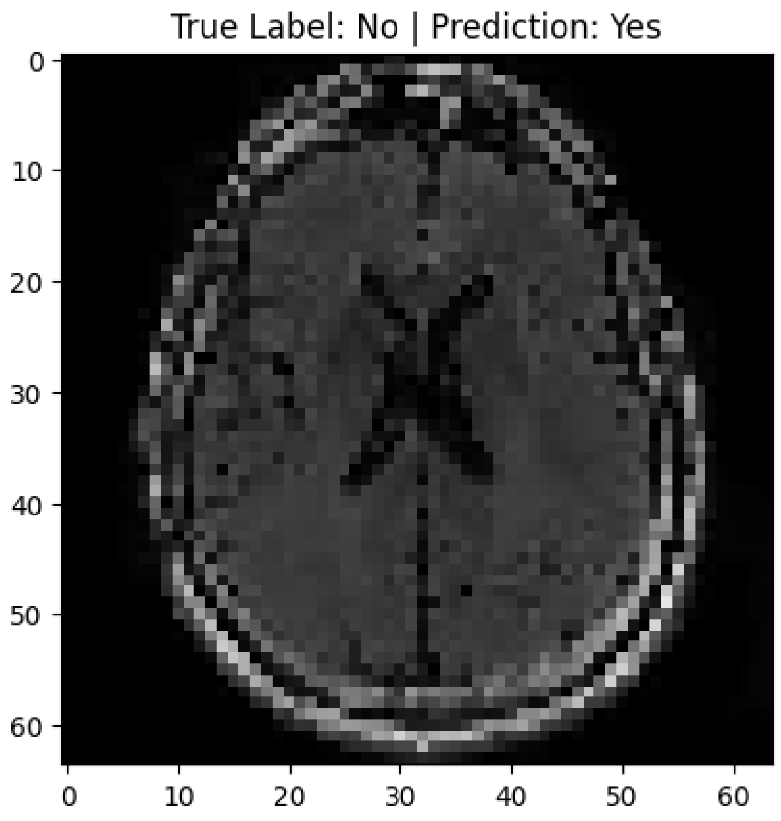
```python
def visualize_predictions_with_labels(model, no_dir, yes_dir):
    no_files = random.sample(os.listdir(no_dir), 5)
    yes_files = random.sample(os.listdir(yes_dir), 5)

    for file_name in no_files:
        file_path = os.path.join(no_dir, file_name)
        prediction = predict_image(model, file_path)

        img = image.load_img(file_path, target_size=(64, 64))
        plt.imshow(img)
        plt.title(f'True Label: No | Prediction: {prediction}')
        plt.show()

    for file_name in yes_files:
        file_path = os.path.join(yes_dir, file_name)
        prediction = predict_image(model, file_path)

        img = image.load_img(file_path, target_size=(64, 64))
        plt.imshow(img)
        plt.title(f'True Label: Yes | Prediction: {prediction}')
        plt.show()

# Function to generate classification report using all files
def generate_full_classification_report(model, no_dir, yes_dir):
    predictions = []
    true_labels = []

    for class_label, directory in [('No', no_dir), ('Yes', yes_dir)]:
        for file_name in os.listdir(directory):
            file_path = os.path.join(directory, file_name)
            prediction = predict_image(model, file_path)
            predictions.append(prediction)
            true_labels.append(class_label)

    print(classification_report(true_labels, predictions))

# Example usage:
no_dir = '/content/Brain MRI Images/brain_tumor_dataset/no'
yes_dir = '/content/Brain MRI Images/brain_tumor_dataset/yes'

# Visualize predictions for 10 random images (5 from each folder) with true␣
 ↪labels
visualize_predictions_with_labels(cnn, no_dir, yes_dir)

# Generate classification report using all files from both folders
generate_full_classification_report(cnn, no_dir, yes_dir)
```

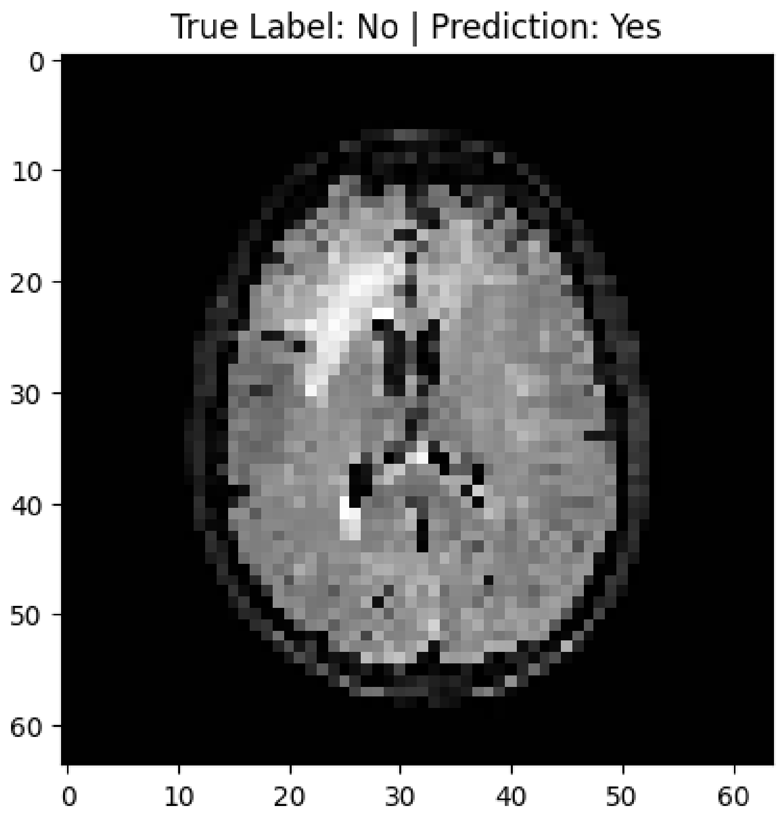

True Label: No | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60

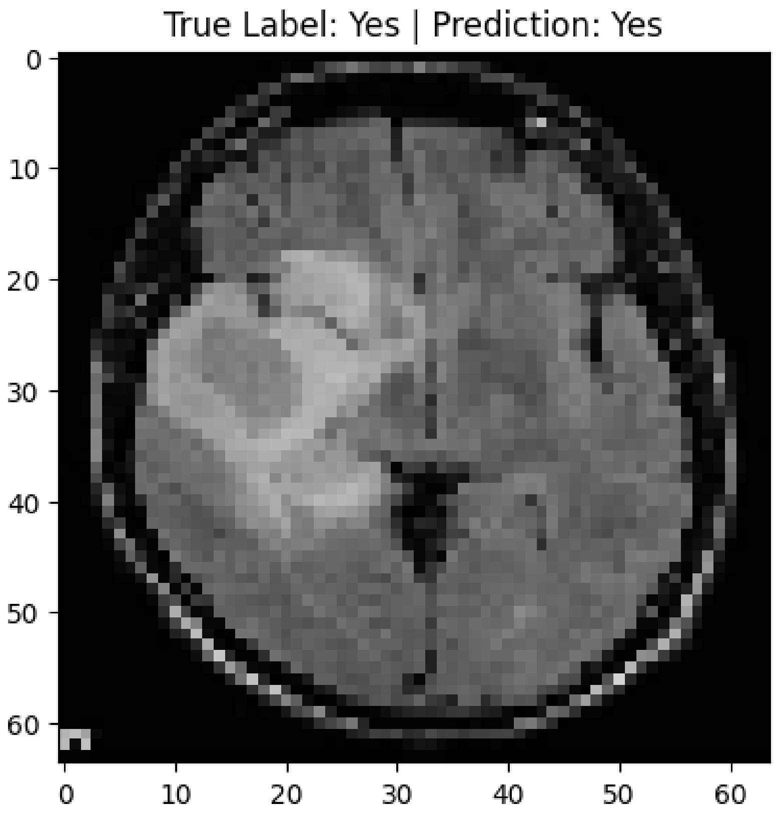
True Label: No | Prediction: Yes

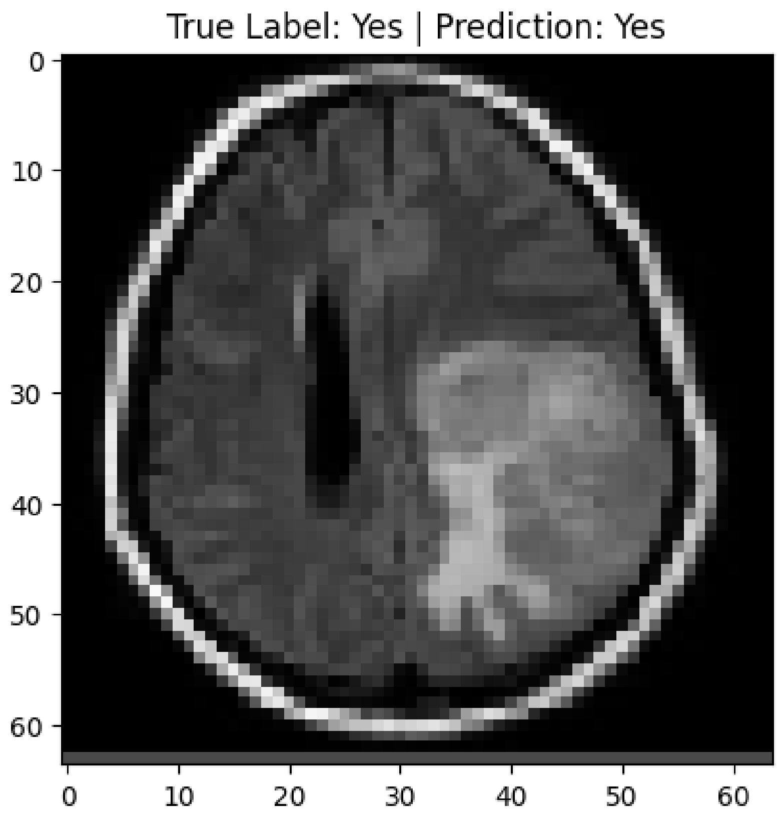


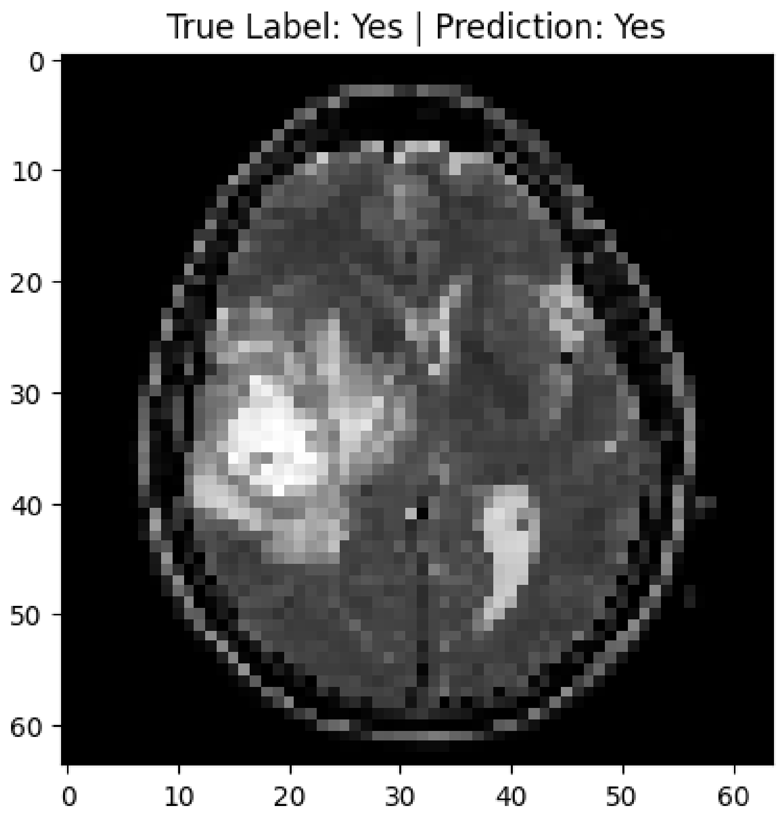

True Label: No | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60

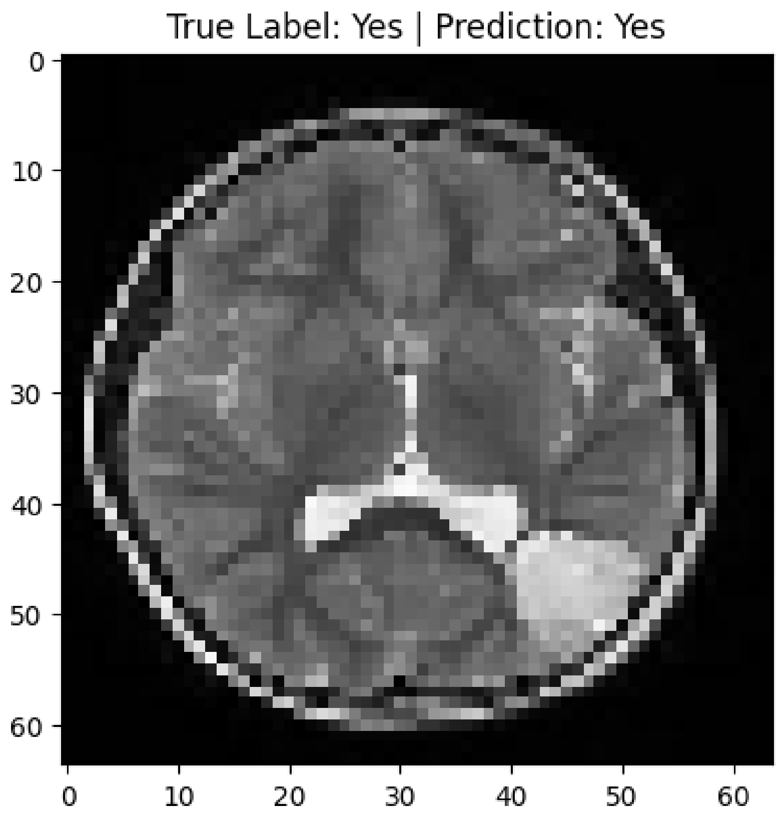

True Label: No | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60

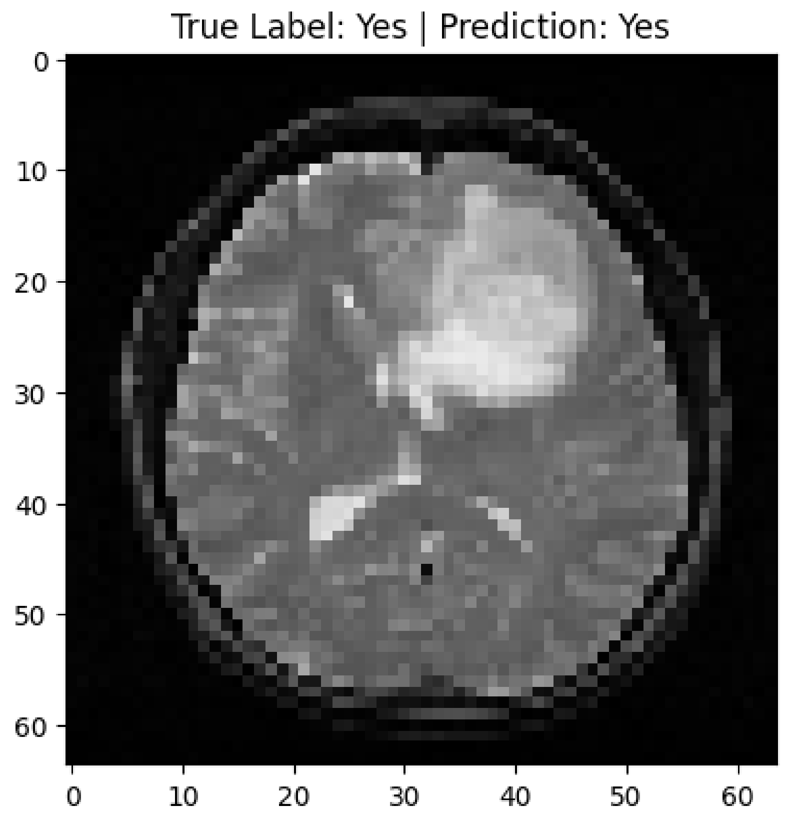

True Label: No | Prediction: Yes
0
10
20
30
40
50
60


True Label: Yes | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60


True Label: Yes | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60


True Label: Yes | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60


True Label: Yes | Prediction: Yes
0
10
20
30
40
50
60
0
10
20
30
40
50
60



```
              precision    recall  f1-score   support

          No       0.00      0.00      0.00        98
         Yes       0.61      1.00      0.76       155

    accuracy                           0.61       253
   macro avg       0.31      0.50      0.38       253
weighted avg       0.38      0.61      0.47       253

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
```

```
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[32]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import DataLoader, Dataset,Subset
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
from sklearn.metrics import confusion_matrix,␣
 ↪classification_report,precision_recall_fscore_support
from sklearn.model_selection import KFold
import numpy as np
import matplotlib.pyplot as plt
import os
import seaborn as sns
from PIL import Image
```

[33]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

[34]:
```
class BrainTumorDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = ['glioma', 'meningioma', 'notumor', 'pituitary']
        self.image_paths = []
        self.labels = []

        for label, class_name in enumerate(self.classes):
            class_dir = os.path.join(root_dir, class_name)
            for img_name in os.listdir(class_dir):
                img_path = os.path.join(class_dir, img_name)
                self.image_paths.append(img_path)
                self.labels.append(label)
    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert('RGB')
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label
```

[35]:
```
train_dataset = BrainTumorDataset(root_dir='/content/archive/Training',␣
 ↪transform=train_transform)
test_dataset = BrainTumorDataset(root_dir='/content/archive/Testing',␣
 ↪transform=test_transform)
```

[36]:
```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)


class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(64 * 28 * 28, 512)
        self.fc2 = nn.Linear(512, 4)  # 4 classes

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = self.pool(nn.functional.relu(self.conv3(x)))
        x = x.view(-1, 64 * 28 * 28)  # Flatten
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

[37]:
```
model = CNNModel().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

[38]:
```
save_path = '/kaggle/working/trained_model.pth'

# Ensure the directory exists before saving
os.makedirs(os.path.dirname(save_path), exist_ok=True)
```

[39]:
```
def train_model(model, criterion, optimizer, num_epochs=200,␣
 ↪save_path='trained_model.pth'):
    model.train()
    best_accuracy = 0.0  # Track best accuracy to save the best model

    for epoch in range(num_epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item() * inputs.size(0)

            # Calculate accuracy
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        epoch_loss = running_loss / len(train_dataset)
        epoch_accuracy = correct / total

        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f},␣
 ↪Accuracy: {epoch_accuracy:.4f}")

        # Save the model if it has the best accuracy
        if epoch_accuracy > best_accuracy:
            best_accuracy = epoch_accuracy
            torch.save(model.state_dict(), save_path)

    print(f"Training complete. Best accuracy: {best_accuracy:.4f}. Model saved␣
 ↪to {save_path}")
```

[40]:
```
def test_model(model):
    model.eval()
    correct = 0
    total = 0
    predictions = []
    true_labels = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            predictions.extend(predicted.cpu().numpy())
            true_labels.extend(labels.cpu().numpy())

    accuracy = correct / total
    print(f"Accuracy on test set: {accuracy:.4f}")

    return true_labels, predictions
```

[41]:
```
# Train the model and save it to /kaggle/working/
train_model(model=model, criterion=criterion, optimizer=optimizer,␣
 ↪num_epochs=10, save_path=save_path)
```

```
Epoch [1/10], Loss: 0.5810, Accuracy: 0.7654
Epoch [2/10], Loss: 0.3756, Accuracy: 0.8380
Epoch [3/10], Loss: 0.2943, Accuracy: 0.8627
Epoch [4/10], Loss: 0.2572, Accuracy: 0.8720
Epoch [5/10], Loss: 0.2458, Accuracy: 0.8778
Epoch [6/10], Loss: 0.2392, Accuracy: 0.8833
Epoch [7/10], Loss: 0.1995, Accuracy: 0.8967
Epoch [8/10], Loss: 0.1738, Accuracy: 0.9072
Epoch [9/10], Loss: 0.1708, Accuracy: 0.9059
Epoch [10/10], Loss: 0.1641, Accuracy: 0.9085
Training complete. Best accuracy: 0.9085. Model saved to
/kaggle/working/trained_model.pth
```

[42]:
```
true_labels, predictions = test_model(model)
```

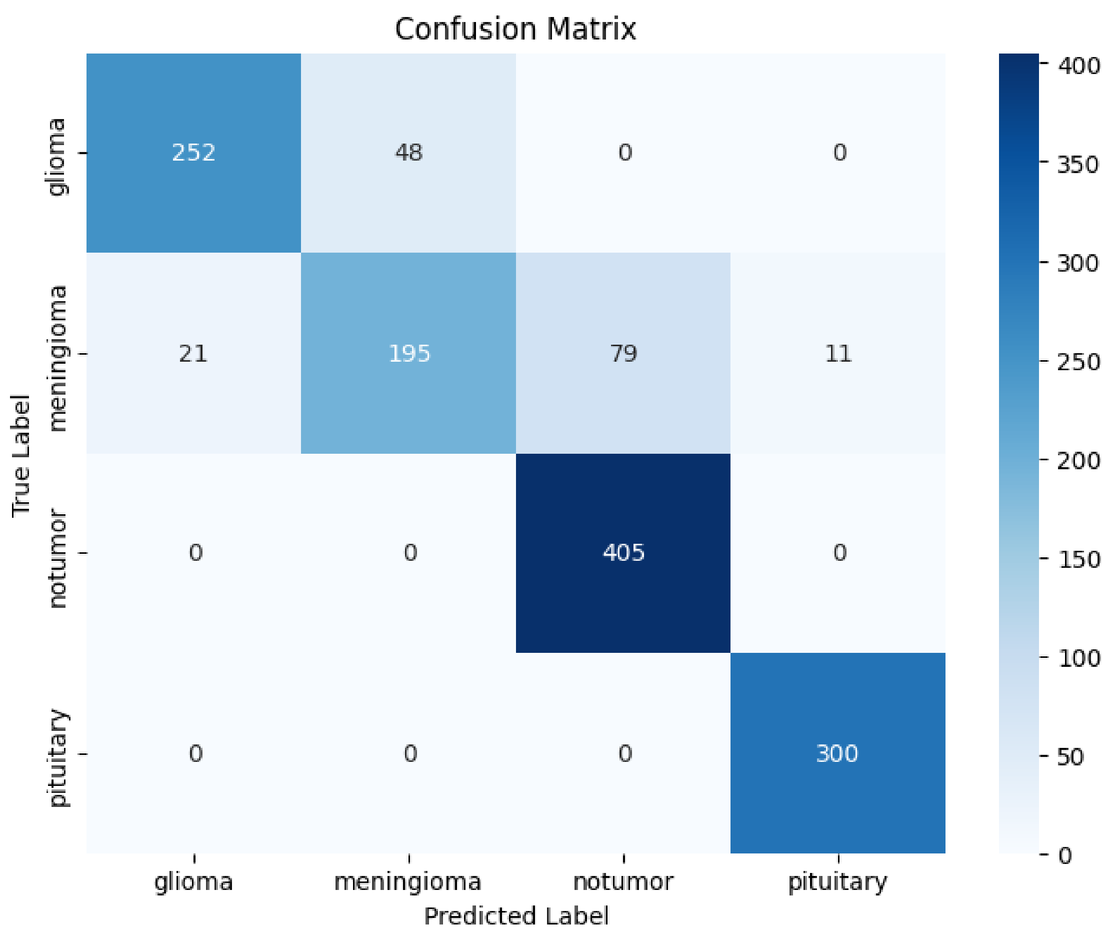
```
Accuracy on test set: 0.8787
```

[43]:
```
print("Classification Report:")
print(classification_report(true_labels, predictions, target_names=['glioma',␣
 ↪'meningioma', 'notumor', 'pituitary']))
```

```
Classification Report:
```

```
              precision    recall  f1-score   support

      glioma       0.92      0.84      0.88       300
  meningioma       0.80      0.64      0.71       306
     notumor       0.84      1.00      0.91       405
   pituitary       0.96      1.00      0.98       300

    accuracy                           0.88      1311
   macro avg       0.88      0.87      0.87      1311
weighted avg       0.88      0.88      0.87      1311
```

[44]:
```python
# Generate the confusion matrix
cm = confusion_matrix(true_labels, predictions)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=['glioma',␣
 ↪'meningioma', 'notumor', 'pituitary'], yticklabels=['glioma', 'meningioma',␣
 ↪'notumor', 'pituitary'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



[44]:
```
from sklearn.metrics import precision_recall_fscore_support

precision, recall, f1_score, _ = precision_recall_fscore_support(true_labels,␣
 ↪predictions, average='weighted')
print(f"Weighted Precision: {precision:.4f}, Recall: {recall:.4f}, F1-score:␣
 ↪{f1_score:.4f}")

precision, recall, f1_score, _ = precision_recall_fscore_support(true_labels,␣
 ↪predictions, average='macro')
print(f"Macro-averaged Precision: {precision:.4f}, Recall: {recall:.4f},␣
 ↪F1-score: {f1_score:.4f}")
```

[44]:
```
import torch
import torch.nn as nn
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset
from PIL import Image
```

```python
import numpy as np
import os

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define transforms for the testing dataset
test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
# Custom Dataset class for loading brain tumor data
class BrainTumorDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = ['glioma', 'meningioma', 'notumor', 'pituitary']
        self.image_paths = []
        self.labels = []

        for label, class_name in enumerate(self.classes):
            class_dir = os.path.join(root_dir, class_name)
            for img_name in os.listdir(class_dir):
                img_path = os.path.join(class_dir, img_name)
                self.image_paths.append(img_path)
                self.labels.append(label)

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        image = Image.open(img_path).convert('RGB')
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

# Load the model state dictionary
model_path = '/kaggle/working/trained_model.pth'
state_dict = torch.load(model_path, map_location=device)

# Define your model architecture
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.conv3 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,␣
 ↪stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(64 * 28 * 28, 512)
        self.fc2 = nn.Linear(512, 4)  # 4 classes

    def forward(self, x):
      x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = self.pool(nn.functional.relu(self.conv3(x)))
        x = x.view(-1, 64 * 28 * 28)  # Flatten
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Instantiate your model
model = CNNModel().to(device)

# Load state_dict into your model
model.load_state_dict(state_dict)
model.eval()  # Set the model to evaluation mode

# Create dataset instance for testing
test_dataset = BrainTumorDataset(root_dir='/content/archive/Testing',␣
 ↪transform=test_transform)

# Randomly select 10 images from the test set
num_images = 20
random_indices = np.random.choice(len(test_dataset), num_images, replace=False)

print("Actual Label | Predicted Label")
print("------------------------------")

# Perform inference on each selected image
for idx in random_indices:
    image, label = test_dataset[idx]
    image = image.unsqueeze(0).to(device)  # Add batch dimension and move to␣
 ↪device
    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output.data, 1)
    actual_label = test_dataset.classes[label]
    predicted_label = test_dataset.classes[predicted.item()]

    print(f"{actual_label:12} | {predicted_label:14}")
```

```
[44]: def cross_validate(model_class, dataset, criterion, optimizer_class,␣
       ↪num_epochs=10, k=5):
          kf = KFold(n_splits=k, shuffle=True)
          fold_accuracies = []

          for fold, (train_idx, test_idx) in enumerate(kf.split(dataset)):
              print(f"Fold {fold+1}/{k}")

              train_subset = Subset(dataset, train_idx)
              test_subset = Subset(dataset, test_idx)

              train_loader = DataLoader(train_subset, batch_size=32, shuffle=True)
              test_loader = DataLoader(test_subset, batch_size=32, shuffle=False)

              model = model_class().to(device)
              optimizer = optimizer_class(model.parameters(), lr=0.001)

              model.train()
              for epoch in range(num_epochs):
                  running_loss = 0.0
                  correct = 0
                  total = 0
                  for inputs, labels in train_loader:
                      inputs, labels = inputs.to(device), labels.to(device)
                      optimizer.zero_grad()
                      outputs = model(inputs)
                      loss = criterion(outputs, labels)
                      loss.backward()
                      optimizer.step()
                      running_loss += loss.item() * inputs.size(0)
                      _, predicted = torch.max(outputs.data, 1)
                      total += labels.size(0)
                      correct += (predicted == labels).sum().item()
                  epoch_loss = running_loss / len(train_subset)
                  epoch_accuracy = correct / total
                  print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f},␣
       ↪Accuracy: {epoch_accuracy:.4f}")

              model.eval()
              correct = 0
              total = 0
              with torch.no_grad():
```

```
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        accuracy = correct / total
        fold_accuracies.append(accuracy)
        print(f"Fold {fold+1} Accuracy: {accuracy:.4f}")

    return fold_accuracies

# Example usage
dataset = BrainTumorDataset(root_dir='/content/archive/Training',␣
 ↪transform=train_transform)
criterion = nn.CrossEntropyLoss()

fold_accuracies = cross_validate(CNNModel, dataset, criterion, optim.Adam,␣
 ↪num_epochs=1, k=5)
print(f"Cross-Validation Accuracies: {fold_accuracies}")
print(f"Mean Accuracy: {np.mean(fold_accuracies):.4f}")
```

[45]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
 ↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
 ↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪all files under the input directory

import os
for dirname, _, filenames in os.walk('content'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

[46]:
```
import cv2
import os
import numpy as np
import pandas as pd
import seaborn as sns
import random
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.utils import shuffle
import glob
from PIL import Image
```

[47]:
```
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

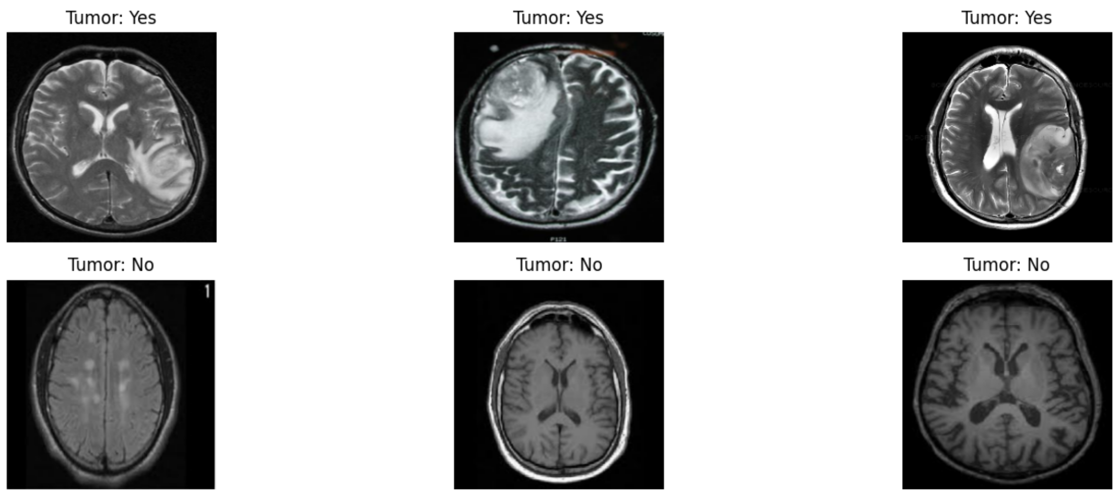
[48]:
```
def load_images(folders, label_map):
    # creating two lists to store the images and labels
    images = []
    labels = []

    # loading the images from each folder in the dataset
    for folder in folders:
        for category in os.listdir(folder):
            category_path = os.path.join(folder, category)
            if os.path.isdir(category_path):
                if category in label_map:  # Check if the category is present␣
 ↪in the label_map
                    label = label_map[category]
                    file_list = os.listdir(category_path)
                    for filename in file_list:
                        img_path = os.path.join(category_path, filename)
                        image = cv2.imread(img_path)
                        # resizing the images to create a standard and so that␣
 ↪it can be suitable for the model input
                        image = cv2.resize(image, (224, 224))
                        # cv2 reads the image as BGR so we need to convert it␣
 ↪back to RGB
                        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                        images.append(image)
                        labels.append(label)

    return np.array(images), np.array(labels)
```

[49]:
```
data_folders = [
    '/content/Brain MRI Images/brain_tumor_dataset',
]

# encoding the labels
label_map = {'no': 0, 'yes': 1}  # Map negative to 0 (no) and positive to 1␣
 ↪(yes)

images, labels = load_images(data_folders, label_map)
```

```
# Shape: (number, height, length, channel RGB)

print("Shape of images:", images.shape)
print("Shape of labels:", labels.shape)
```

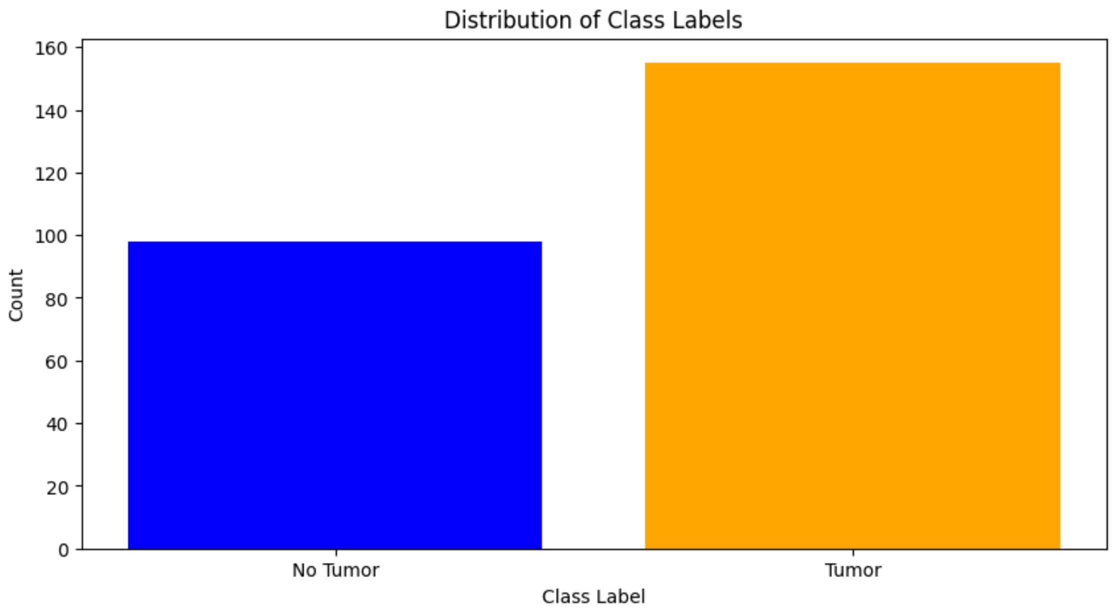
```
Shape of images: (253, 224, 224, 3)
Shape of labels: (253,)
```

[50]:
```
plt.figure(figsize=(15, 5))

# Display tumor images with label 'yes'
for i in range(3):
    plt.subplot(2, 3, i+1)
    plt.imshow(images[labels == 1][i])  # Filter images with label 'yes'
    plt.title("Tumor: Yes")
    plt.axis('off')

# Display no_tumor images with label 'no'
for i in range(3):
    plt.subplot(2, 3, i+4)
    plt.imshow(images[labels == 0][i])  # Filter images with label 'no'
    plt.title("Tumor: No")
    plt.axis('off')

plt.tight_layout()
plt.show()
```



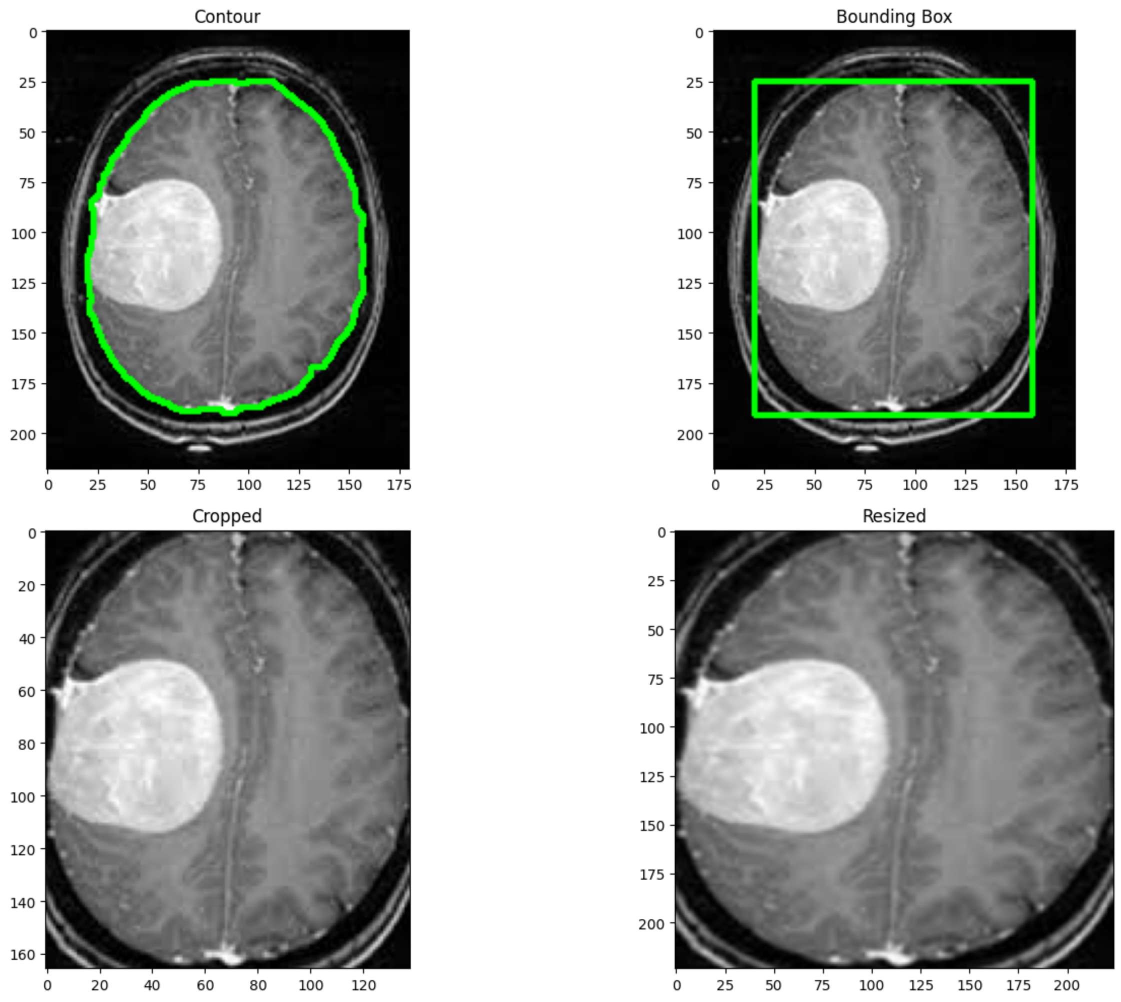
[51]:
```
# Counting the occurrences of each class label
unique_labels, label_counts = np.unique(labels, return_counts=True)
```

```
plt.figure(figsize=(10,5))
plt.bar(unique_labels, label_counts, color=['blue', 'orange'])
plt.xticks(unique_labels, ['No Tumor', 'Tumor'])
plt.xlabel('Class Label')
plt.ylabel('Count')
plt.title('Distribution of Class Labels')
plt.show()
```



[52]:
```
import keras.preprocessing.image

def crop_brain_region(image, size):

    # Converting the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

    # Applying Gaussian blur to smooth the image and reduce noise
    gray = cv2.GaussianBlur(gray, (5, 5), 0)

    # Thresholding the image to create a binary mask
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]

    # Performing morphological operations to remove noise
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    # Finding contours in the binary mask
    contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.
 ↪CHAIN_APPROX_SIMPLE)
    # Assuming the brain part of the image has the largest contour
    c = max(contours, key=cv2.contourArea)

    # Getting the bounding rectangle of the brain part
    x, y, w, h = cv2.boundingRect(c)

    # Drawing contours on the original image
    contour_image = cv2.drawContours(image.copy(), [c], -1, (0, 255, 0), 2)

    # Drawing bounding box on the original image
    bounding_box_image = cv2.rectangle(image.copy(), (x, y), (x + w, y + h),␣
 ↪(0, 255, 0), 2)

    # Cropping the image around the bounding rectangle
    cropped_image = image[y:y+h, x:x+w]

    # Resizing cropped image to the needed size
    resized_image = cv2.resize(cropped_image, size)

    return contour_image, bounding_box_image, cropped_image, resized_image
```

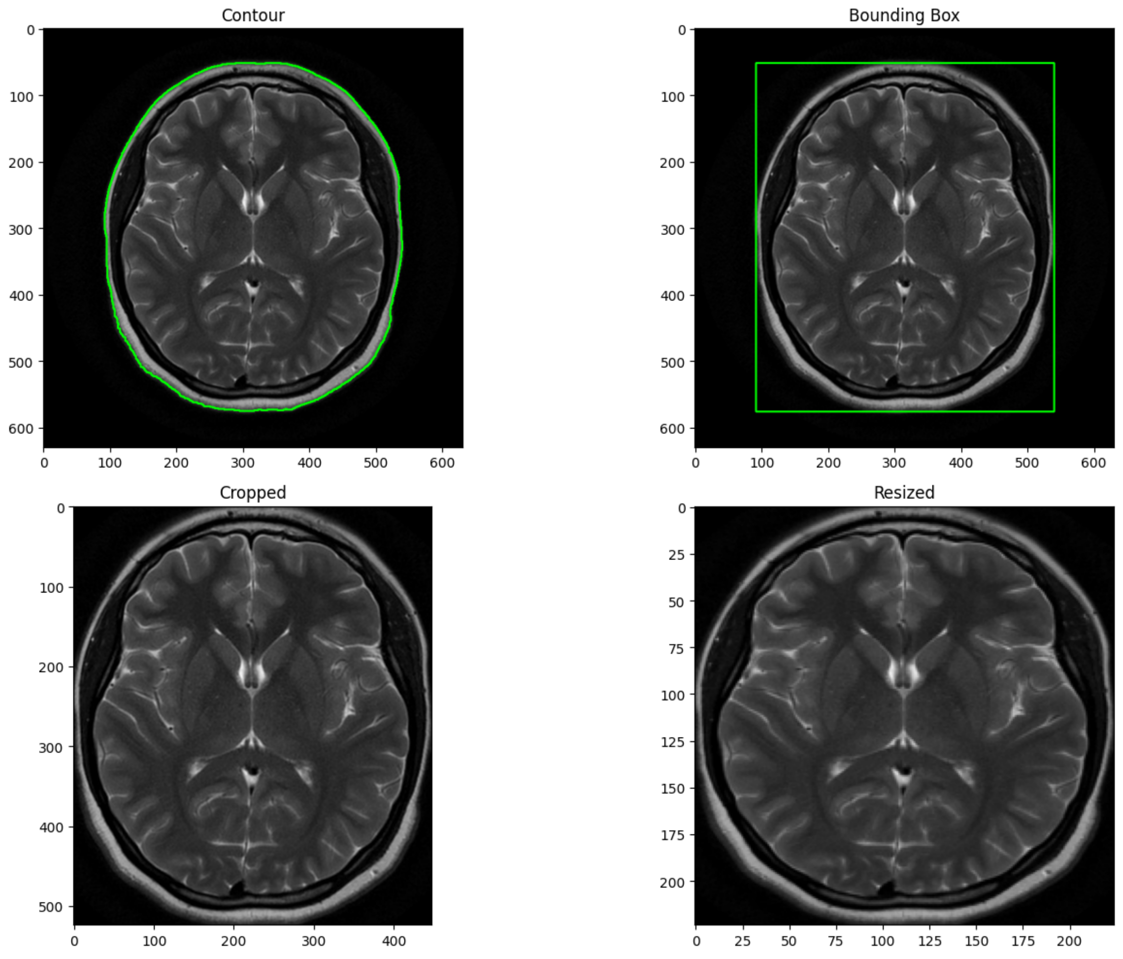
[54]:
```
output_size = (224, 224)
example_image = cv2.imread('/content/Brain MRI Images/brain_tumor_dataset/yes/
 ↪Y1.jpg')
example_image = cv2.cvtColor(example_image, cv2.COLOR_BGR2RGB)

contour_image, bounding_box_image, cropped_image, resized_image =␣
 ↪crop_brain_region(example_image, output_size)


plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.imshow(contour_image)
plt.title("Contour")

plt.subplot(2, 2, 2)
plt.imshow(bounding_box_image)
plt.title("Bounding Box")

plt.subplot(2, 2, 3)
plt.imshow(cropped_image)
plt.title("Cropped")
plt.subplot(2, 2, 4)
plt.imshow(resized_image)
plt.title("Resized")

plt.tight_layout()
plt.show()

all_cropped = []

# Applying the crop function to each one of our images
for image in images:
    _, _, _, resized_image = crop_brain_region(image, output_size)
    all_cropped.append(resized_image)
```



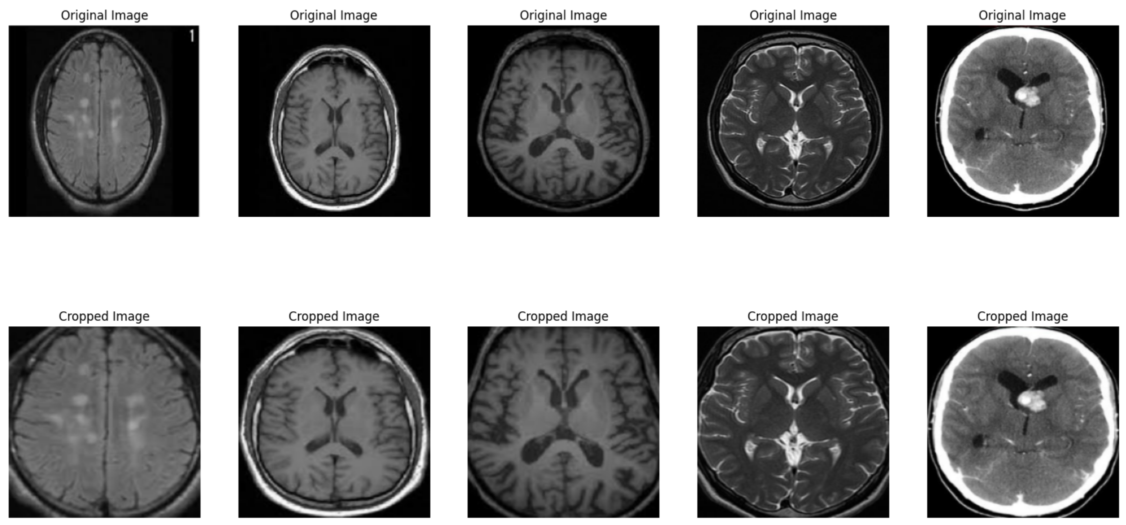
```
[ ]:
```

```
[55]: output_size = (224, 224)
```

```python
example_image = cv2.imread('/content/Brain MRI Images/brain_tumor_dataset/no/1␣
↪no.jpeg')
example_image = cv2.cvtColor(example_image, cv2.COLOR_BGR2RGB)

contour_image, bounding_box_image, cropped_image, resized_image =␣
↪crop_brain_region(example_image, output_size)


plt.figure(figsize=(15, 10))

plt.subplot(2, 2, 1)
plt.imshow(contour_image)
plt.title("Contour")

plt.subplot(2, 2, 2)
plt.imshow(bounding_box_image)
plt.title("Bounding Box")

plt.subplot(2, 2, 3)
plt.imshow(cropped_image)
plt.title("Cropped")
plt.subplot(2, 2, 4)
plt.imshow(resized_image)
plt.title("Resized")

plt.tight_layout()
plt.show()

all_cropped = []

# Applying the crop function to each one of our images
for image in images:
    _, _, _, resized_image = crop_brain_region(image, output_size)
    all_cropped.append(resized_image)
```



[56]:
```python
num_images_per_class = 5

class_0_counter = 0
class_1_counter = 0

plt.figure(figsize=(20, 10))

for i in range(num_images_per_class):
    plt.subplot(2, num_images_per_class, i + 1)
    plt.imshow(images[i])
    plt.title("Original Image")
    plt.axis("off")

for i in range(num_images_per_class):
    plt.subplot(2, num_images_per_class, num_images_per_class + i + 1)
    plt.imshow(all_cropped[i])
    plt.title("Cropped Image")
    plt.axis("off")

plt.show()
```



[57]:
```python
from sklearn.model_selection import train_test_split

all_cropped=np.array(all_cropped)
X_train, X_test, y_train, y_test = train_test_split(all_cropped, labels,␣
 ↪test_size=0.2,shuffle=True, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=42)
```

[58]:
```python
print("X_train shape:", X_train.shape)
print("X_val shape:", X_val.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_val shape:", y_val.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (202, 224, 224, 3)
X_val shape: (25, 224, 224, 3)
X_test shape: (26, 224, 224, 3)
y_train shape: (202,)
y_val shape: (25,)
y_test shape: (26,)
```

[59]:
```python
train_folder = 'Train'
val_folder = 'Validation'
test_folder = 'Test'
```

```
os.makedirs(train_folder, exist_ok=True)
os.makedirs(val_folder, exist_ok=True)
os.makedirs(test_folder, exist_ok=True)

label_map_decoded = {1: 'yes', 0: 'no'}

def copy_images_to_folder(images, labels, folder):
    for i, (image, label) in enumerate(zip(images, labels)):
        class_name = label_map_decoded[label]
        class_folder = os.path.join(folder, class_name)
        os.makedirs(class_folder, exist_ok=True)
        img_filename = f'{class_name}_{i}.jpg'  # Assuming images are in JPG␣
 ↪format
        img_path = os.path.join(class_folder, img_filename)
        cv2.imwrite(img_path, cv2.cvtColor(image, cv2.COLOR_RGB2BGR))  # Save␣
 ↪image directly without converting to PIL format
```

[60]:
```
copy_images_to_folder(X_train, y_train, train_folder)
copy_images_to_folder(X_val, y_val, val_folder)
copy_images_to_folder(X_test, y_test, test_folder)
```

[61]:
```
print(np.max(X_train))
print(np.min(X_train))
```

```
255
0
```

[62]:
```
X_train_scaled=X_train/255
X_test_scaled=X_test/255
X_val_scaled=X_val/255
```

[64]:
```
print(np.max(X_train_scaled))
print(np.min(X_train_scaled))
```

```
1.0
0.0
```

[65]:
```
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.layers import Input,Conv2D, Dense, Flatten ,Dropout␣
 ↪,MaxPooling2D,BatchNormalization,GlobalAveragePooling2D
from tensorflow.keras.callbacks import␣
 ↪EarlyStopping,LearningRateScheduler,ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from keras.optimizers import Adam,RMSprop

model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(224,␣
 ↪224, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', metrics=['accuracy'],␣
 ↪optimizer=Adam(learning_rate=1e-4))
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_4 (MaxPoolin  (None, 111, 111, 32)      0
 g2D)

 conv2d_5 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_5 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_6 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten_2 (Flatten)         (None, 86528)             0

 dropout (Dropout)           (None, 86528)             0

 dense_4 (Dense)             (None, 128)               11075712

 dropout_1 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 11169089 (42.61 MB)
Trainable params: 11169089 (42.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

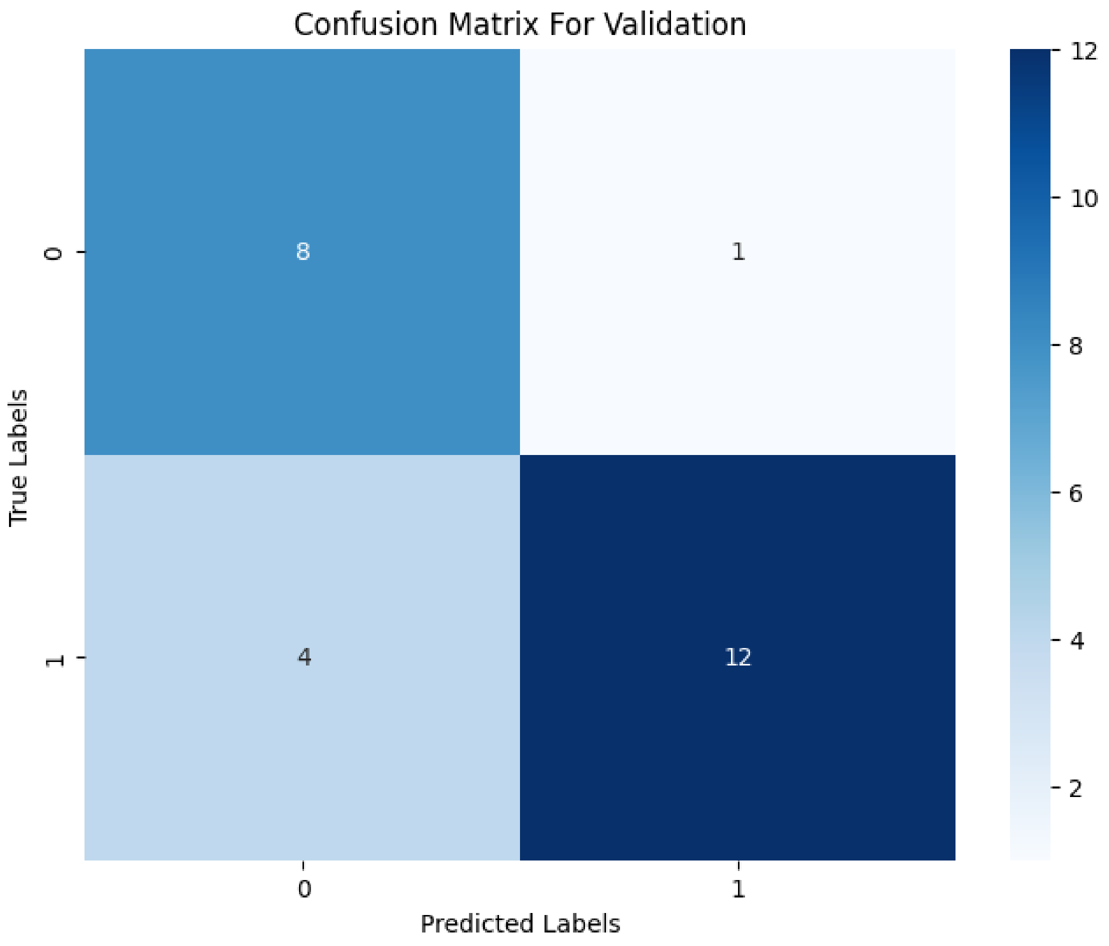
[68]:
```
from tensorflow.keras.callbacks import␣
  ↪EarlyStopping,LearningRateScheduler,ReduceLROnPlateau
from sklearn.metrics import accuracy_score, confusion_matrix

epochs = 50
batch_size = 32

# Defining early stopping to stop the model from overfitting
early_stopping = EarlyStopping(patience=5, monitor='val_loss')

# Train the model
#history = model.
  ↪fit(X_train_scaled,y_train,batch_size=batch_size,epochs=epochs,validation_data=(X_val_scale

history = model.fit(X_train_scaled,
                    y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(X_val_scaled,y_val),
                   callbacks=[early_stopping])
```

```
Epoch 1/50
7/7 [==============================] - 26s 4s/step - loss: 0.1168 - accuracy:
0.9752 - val_loss: 0.4948 - val_accuracy: 0.8000
Epoch 2/50
7/7 [==============================] - 24s 3s/step - loss: 0.1178 - accuracy:
0.9752 - val_loss: 0.4757 - val_accuracy: 0.7600
Epoch 3/50
7/7 [==============================] - 25s 3s/step - loss: 0.0965 - accuracy:
0.9851 - val_loss: 0.5968 - val_accuracy: 0.7600
Epoch 4/50
7/7 [==============================] - 32s 4s/step - loss: 0.1137 - accuracy:
0.9752 - val_loss: 0.4743 - val_accuracy: 0.7200
Epoch 5/50
7/7 [==============================] - 25s 3s/step - loss: 0.0825 - accuracy:
0.9802 - val_loss: 0.6126 - val_accuracy: 0.6800
Epoch 6/50
7/7 [==============================] - 27s 4s/step - loss: 0.0766 - accuracy:
0.9851 - val_loss: 0.6195 - val_accuracy: 0.7200
Epoch 7/50
7/7 [==============================] - 24s 3s/step - loss: 0.0667 - accuracy:
0.9752 - val_loss: 0.5839 - val_accuracy: 0.7200
```

```
Epoch 8/50
7/7 [==============================] - 24s 3s/step - loss: 0.0859 - accuracy:
0.9752 - val_loss: 0.5409 - val_accuracy: 0.7600
Epoch 9/50
7/7 [==============================] - 27s 4s/step - loss: 0.0626 - accuracy:
0.9950 - val_loss: 0.6942 - val_accuracy: 0.6800
```

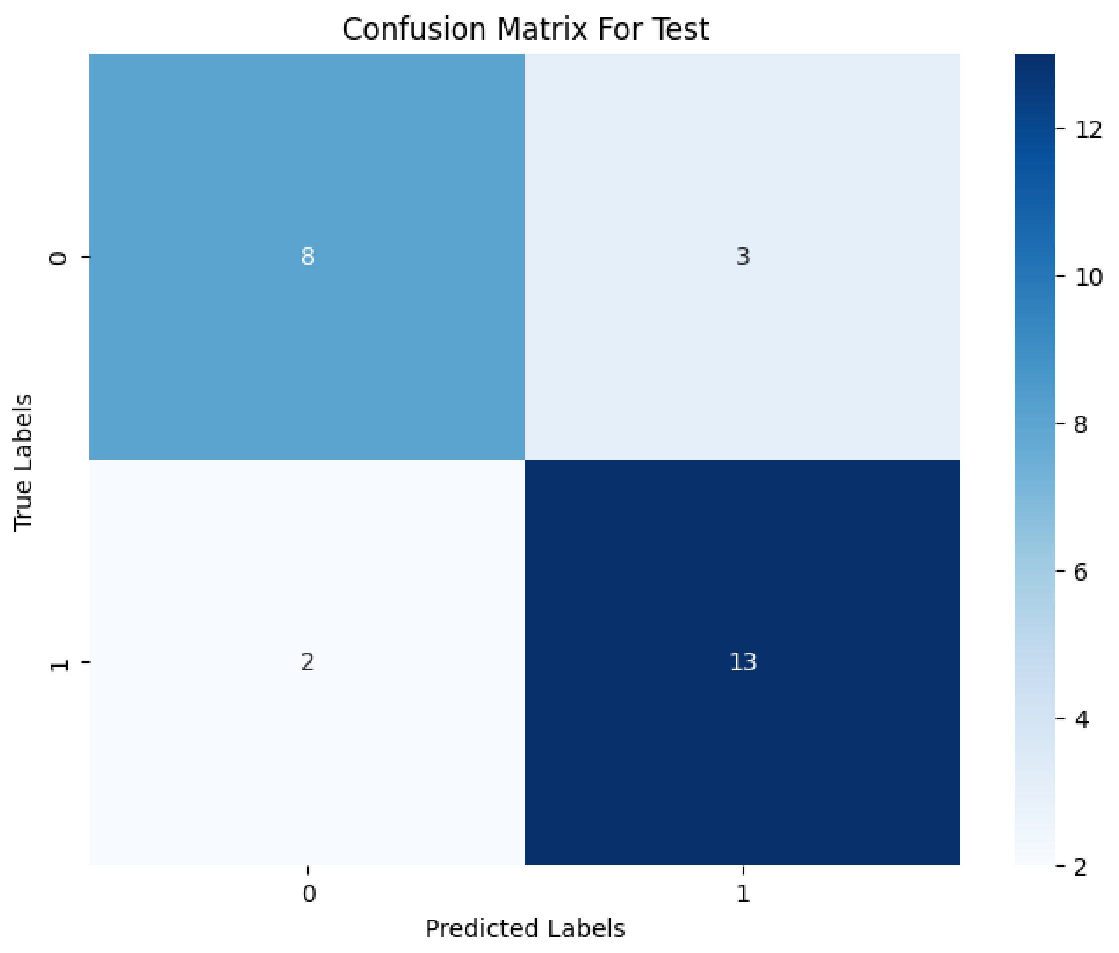
[67]:
```
predictions = model.predict(X_val_scaled)
threshold = 0.5
binary_predictions = (predictions > threshold).astype(int)

conf_matrix = confusion_matrix(y_val, binary_predictions)

accuracy = accuracy_score(y_val, binary_predictions)
print("Accuracy on Validation Set: {:.3f} %".format(accuracy))

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix For Validation')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```
1/1 [==============================] - 1s 939ms/step
Accuracy on Validation Set: 0.800 %
```

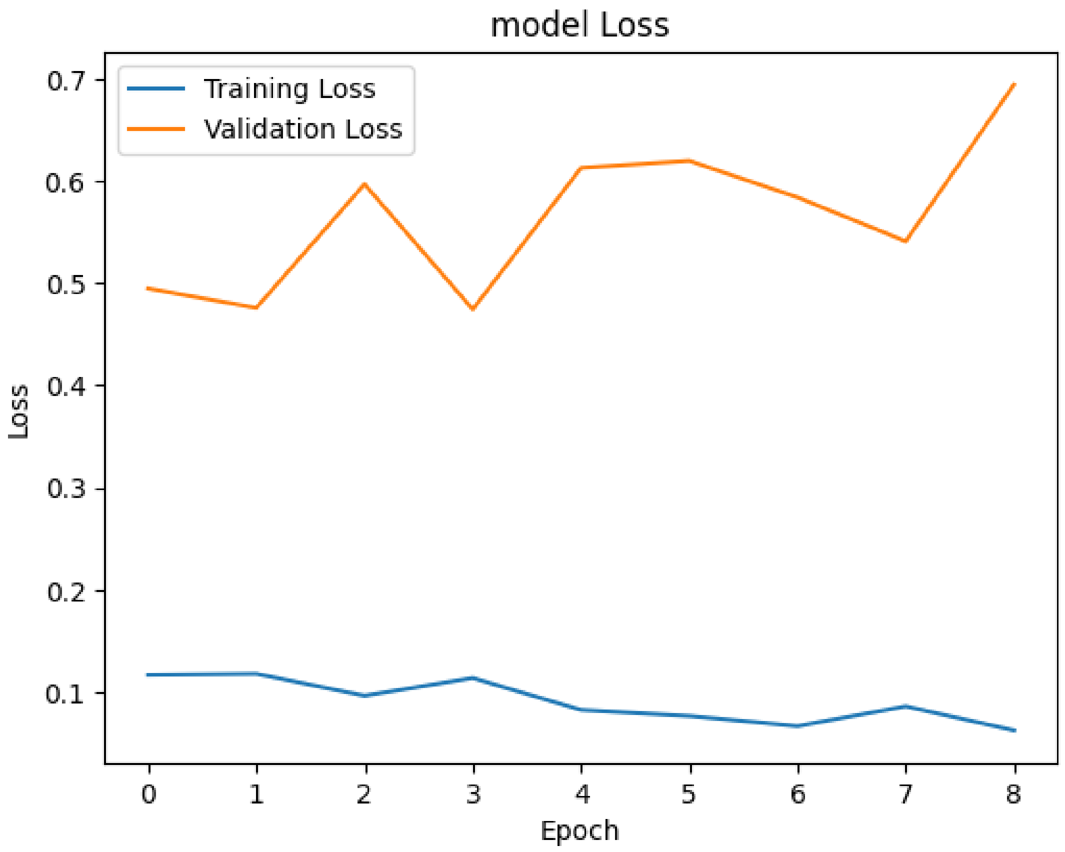


```
[69]: predictions = model.predict(X_test_scaled)
threshold = 0.5
binary_predictions = (predictions > threshold).astype(int)

conf_matrix = confusion_matrix(y_test, binary_predictions)

accuracy = accuracy_score(y_test, binary_predictions)
print("Accuracy on Test Set: {:.3f} %".format(accuracy))

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix For Test')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

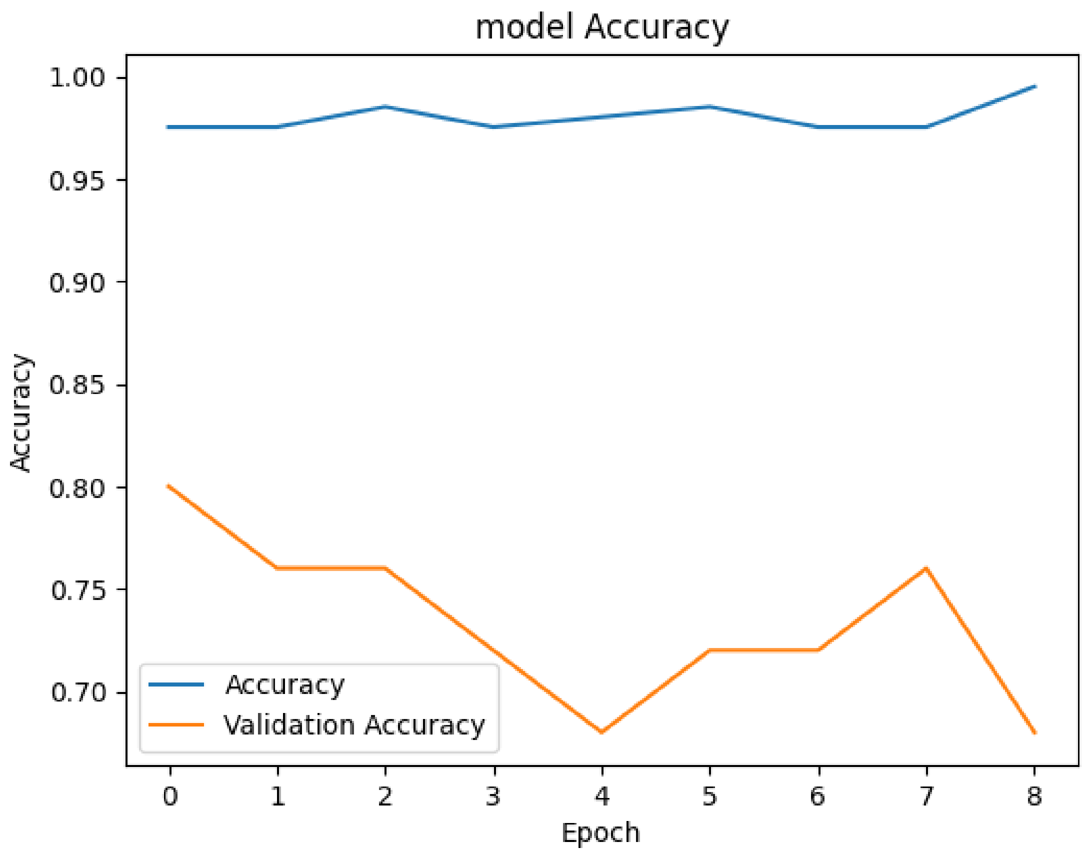
```
1/1 [==============================] - 1s 810ms/step
Accuracy on Test Set: 0.808 %
```



[70]:
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



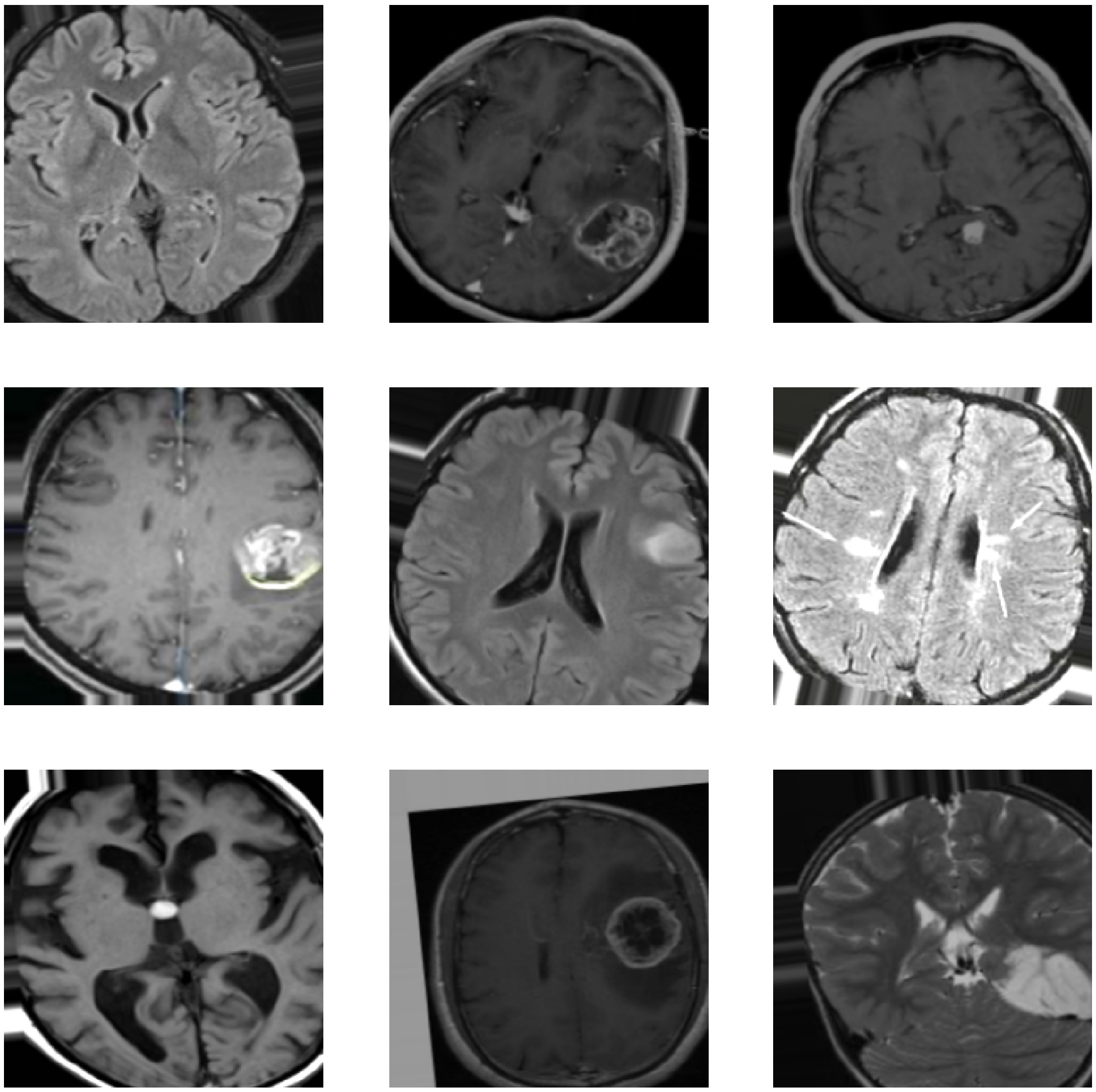
[71]:
```
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[72]:
```
# from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    shear_range=0.1,
    brightness_range=[0.5, 1.5],
    rescale=1./255
)

val_datagen = ImageDataGenerator(rescale=1./255)

image_size=(224,224)

train_generator = datagen.flow_from_directory(
    train_folder,
    color_mode='rgb',
    target_size=image_size,
    batch_size=32,
    class_mode='binary'
)
val_generator = val_datagen.flow_from_directory(
    val_folder,
    color_mode='rgb',
    target_size=image_size,
    batch_size=32,
    class_mode='binary'
)
```

```
Found 202 images belonging to 2 classes.
Found 25 images belonging to 2 classes.
```

[73]:
```
augmented_images, _ = next(datagen.flow(X_train, y_train, batch_size=32))

plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[i])
    plt.axis('off')
plt.show()
```



```
[ ]:
```