Brain Tumor Classification Converlutional Nural Network

Group Number :35

Members:

Bandara U.G.R.S	EG/2020/3847
Virajani M.Y	EG/2020/4254
Wahalathanthri W.A.S	EG/2020/4257
Welagedara B.M.K.C	EG/2020/4276

Introduction

The classification of brain tumors is a critical task in medical sector. It gives significantly impacting when doing treatment. With advancements in artificial intelligence and deep learning, automated systems can now assist in accurately classifying brain tumors. Through out this project we are trying to build a CNN model to clasify the brain tumer based on four four types: glioma, meningioma, pituitary tumor, and no tumor.

Literature Survey

Brain tumors are the diseases that are facing by both children and adults. When concider the anuval helth report around 12000 are faced in this Brain tumer diseases. when concider the survival rates approximately 34% for men and 36% for women are survival. Acording the medical sector these tumors are categorized into types including malignant, pituitary, and benign tumors.

Accurate diagnosis and treatment planning are crucial to improving patient outcomes. Magnetic Resonance Imaging (MRI) is the preferred method for detecting brain tumors in the modern world.It provides a large amount of image data. However, manual examination can be error-prone due to the complexity of brain tumors.

To over come these challenges, automated classification techniques utilizing Machine Learning (ML) and Artificial Intelligence (AI) have been introduced. These methods have demonstrated higher accuracy and reliability compared to manual methods. Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), Artificial Neural Networks (ANNs), and Transfer Learning (TL), have significantly enhanced medical image analysis.

Techniques Used in Medical Image Classification

1. CNN (Convolutional Neural Networks): CNNs are essential for analyzing medical images because they can learn features directly from raw image data. Well-known CNN models like AlexNet, VGGNet, ResNet, and Inception have achieved notable success in medical imaging. For instance, Pereira et al. [1] developed a CNN that effectively identified brain tumors in MRI scans, demonstrating the potential of CNNs in this domain.

2. Transfer Learning (TL): Transfer Learning is a technique that adapts pre-trained models for specific tasks, which is particularly useful when there is limited labeled data available. For example, Mathivanan et al. [2] applied deep learning and transfer learning to improve brain tumor detection accuracy significantly. They adapted pre-trained models to brain tumor datasets, achieving enhanced classification accuracy.

3. ANN techniques are used to diagnose and classify brain tumors, distinguishing between primary and secondary tumors such as gliomas, meningiomas, and medulloblastomas. Diagnosis typically involves neurological exams and imaging scans like MRI, sometimes followed by a biopsy. Treatment options include surgery, radiation, and chemotherapy, tailored to the type of tumor and patient's health. Prognosis varies widely, and ongoing research is focused on genetic and molecular causes, as well as developing new treatments and diagnostics. [3]

Current Challenges and Future Directions of AI Techniques in Brain Tumor Detection

Detecting brain tumors with AI faces many challenges due to variations in tumor location, shape, and size. Unlike standard image recognition tools that identify everyday objects, AI for brain tumors must handle the complexities of MRI brain images. These challenges are compounded by the brain's complexity and the serious nature of tumors, which make early detection difficult. [4] Researchers are using AI to improve brain tumor detection. They combine different techniques to make MRI images clearer and to find groups of tumor cells. They also use methods to classify tumors based on how they look, which are both accurate and fast. Another approach, called radiomics, extracts a lot of information from medical images to understand tumors better and predict how well treatments will work. However, there are still problems with the quality of data, checking the results, and understanding them. Also, the lack of large, standardized datasets makes it hard to create AI that can be used in different hospitals. One big issue is that AI often works like a black box, making accurate predictions without clear reasons. This confuses doctors and regulators who need to understand AI decisions.

Dataset Description

The dataset used in our project is designed for classifying brain MRI images into four categories:

1. Glioma Tumor
2. Meningioma Tumor
3. Pituitary Tumor
4. No Tumor

> This dataset is publicly available on Kaggle[5], specifically under the title "Brain Tumor Classification (MRI)".It includes a balanced distribution of images across these categories, ensuring that each class has a significant number of samples for training and testing machine learning models.

In [1]:
```
!pip install opendatasets -q
!pip install opencv-python
# !pip install keras
# !pip install tensorflow
# !pip uninstall scikit-learn imbalanced-learn
# !pip install scikit-learn imbalanced-learn
```

```
Requirement already satisfied: opencv-python in c:\users\dell\anaconda3\lib\site-pack
ages (4.10.0.82)
Requirement already satisfied: numpy>=1.21.2 in c:\users\dell\anaconda3\lib\site-pack
ages (from opencv-python) (1.24.3)
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import multilabel_confusion_matrix
import opendatasets as od
import os
```

In [3]:
```
import tensorflow as tf
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.metrics import accuracy_score
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from imblearn.over_sampling import RandomOverSampler
```

```
WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\losses.p
y:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.
compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [4]:
```
# Download the dataset
od.download("https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-
```

```
Skipping, found downloaded files in ".\brain-tumor-classification-mri" (use force=Tru
e to force download)
```

In [5]:
```
x_data = []
y_data = []
image_size = 255
```

```
labels = sorted(os.listdir('./brain-tumor-classification-mri/Training'))
```

In [6]:
```
# Load training images and labels
for i in labels:
    folderPath = os.path.join('./brain-tumor-classification-mri/Training', i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath, j))
        img = cv2.resize(img, (image_size, image_size))
        x_data.append(img)
        y_data.append(i)
```

In [7]:
```
# Load testing images and labels
for i in labels:
    folderPath = os.path.join('./brain-tumor-classification-mri/Testing', i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath, j))
        img = cv2.resize(img, (image_size, image_size))
        x_data.append(img)
        y_data.append(i)
```

In [8]:
```
x_data = np.array(x_data)
y_data = np.array(y_data)
```

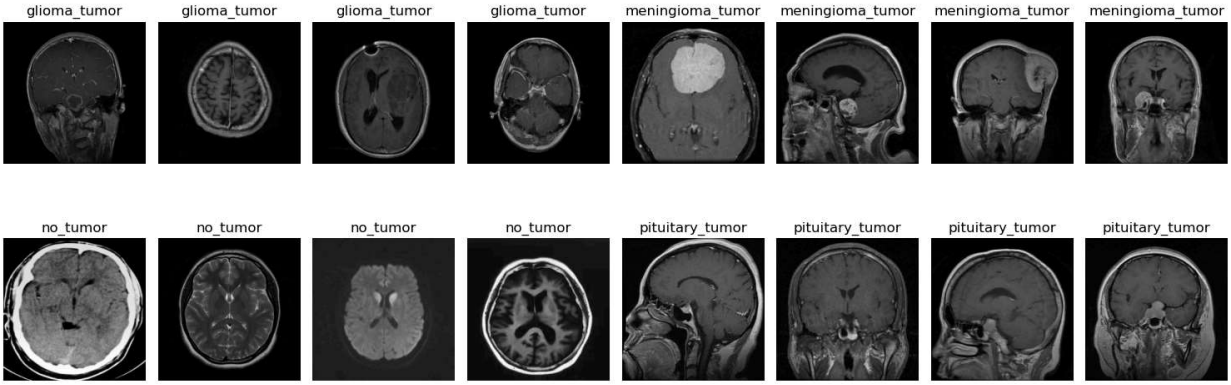
In [9]:
```
# Display the total number of samples
print("Total number of image samples:", len(x_data))
print("Total number of label samples:", len(y_data))
```

```
Total number of image samples: 3264
Total number of label samples: 3264
```

In [10]:
```
# Define class names based on your labels
class_names = labels
print(class_names)
```

```
['glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor']
```

In [11]:
```
# Plot data sample from different categories
plt.figure(figsize=(15, 6))
num_classes = len(class_names)
samples_per_class = 4
total_samples = num_classes * samples_per_class
rows = 2
cols = (total_samples + rows - 1) // rows

for class_idx, class_name in enumerate(class_names):
    class_indices = np.where(y_data == class_name)[0]
    sampled_indices = np.random.choice(class_indices, samples_per_class, replace=False
    for i, idx in enumerate(sampled_indices):
        plt.subplot(rows, cols, class_idx * samples_per_class + i + 1)
        plt.imshow(x_data[idx])
        plt.title(class_name)
        plt.axis("off")

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



Exploratory Data Analysis

The analysis includes visualizations such as histograms and box plots to depict the distribution of pixel intensities and the number of images per category

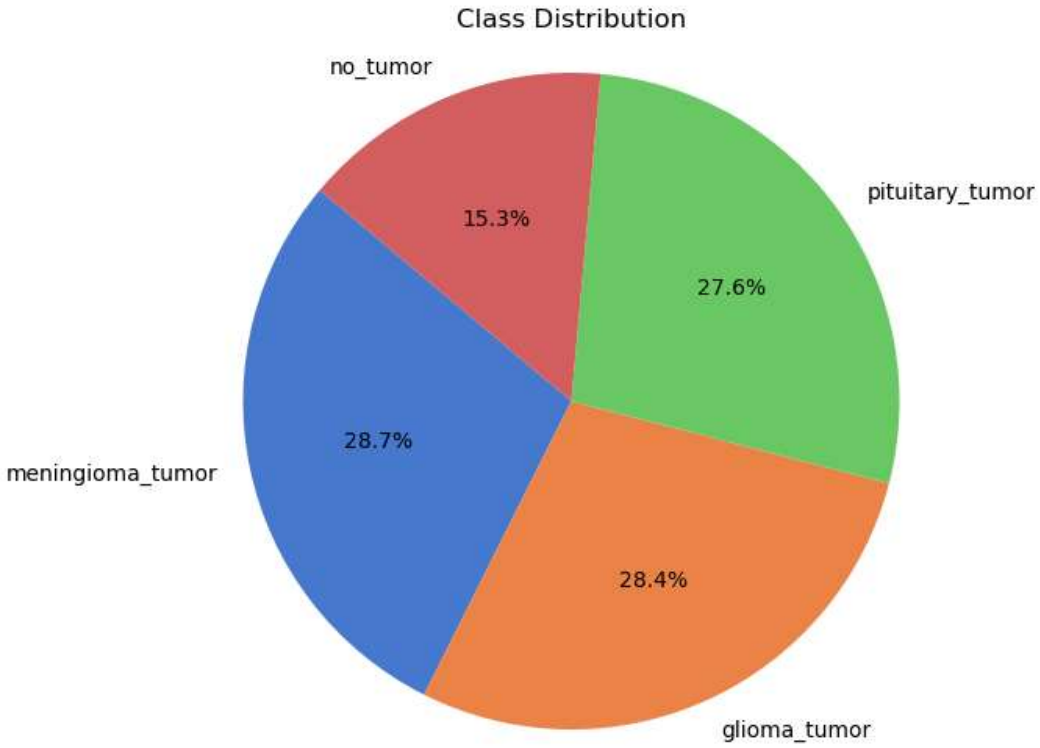
Summary of the Data Set Class distribution visualization

In [12]:
```
class_counts = pd.Series(y_data).value_counts()
print("Number of samples in each category:")
print(class_counts)
```

```
Number of samples in each category:
meningioma_tumor    937
glioma_tumor        926
pituitary_tumor     901
no_tumor            500
Name: count, dtype: int64
```

Pie chart visualization of the data

In [13]:
```
plt.figure(figsize=(10, 6))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=140, cc
plt.title("Class Distribution")
plt.axis('equal')
plt.show()
```



Data Preprocessing

Steps Are Taken

1. Normalize the pixel values
2. Handling Categorical Data:
3. Apply Oversampling to Minority Classes
4. Augmentation Data
5. Crop Image Data

Normalize the pixel values

By doing this We try to enhace

- Standardization: Converts pixel values from 0-255 to 0-1, making data easier for the neural network to process.
- Improved Convergence: Speeds up training as optimization algorithms work more efficiently with normalized data.
- Numerical Stability: Reduces the risk of issues like overflow and excessively large gradients, leading to more stable training.
- Consistent Scaling: Ensures all input features are proportionate, enhancing neural network performance.

In [14]:
```
x_data = x_data / 255.0
```

In [15]:
```
x_data, y_data = shuffle(x_data, y_data, random_state=101)
```

Handling Categorical Data:

- One hot encoding

  Neural networks require numerical input. One-hot encoding transforms categorical labels . Here different types of brain tumors into a numerical format

---

In [16]:
```
y_data_new = [labels.index(i) for i in y_data]
y_data = tf.keras.utils.to_categorical(y_data_new, num_classes=len(labels))
```

Reshape y_train for oversampling

In [17]:
```
y_data_labels = np.argmax(y_data, axis=1)
```

Apply Oversampling to Minority Classes

Why Apply Oversampling to Minority Classes:

1. Imbalanced Dataset Issue:
   - In many real-world datasets, especially medical ones like brain tumor classification, certain classes (e.g., types of tumors) may have significantly fewer samples compared to others.
   - This imbalance can lead to biased models that perform well on majority classes but poorly on minority classes.

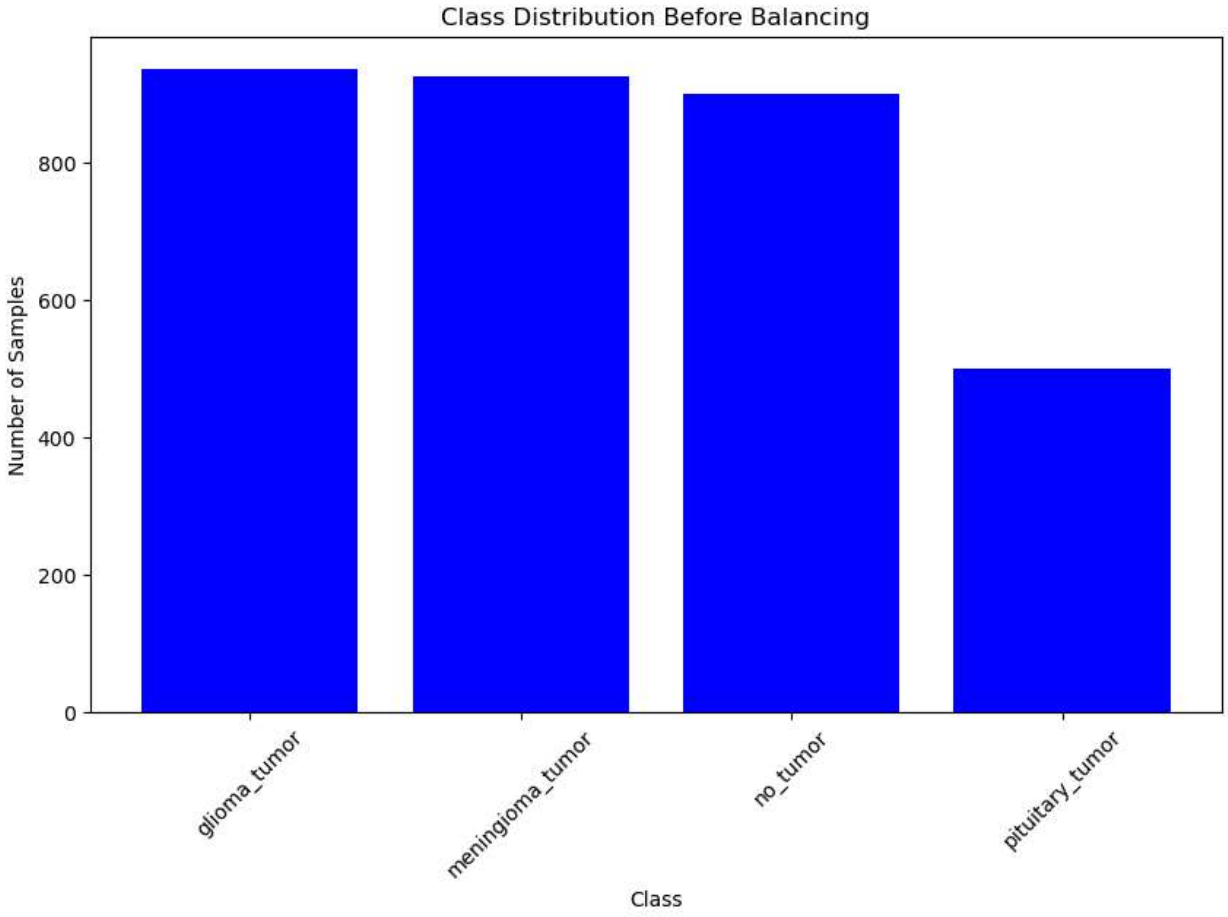
1. Ensuring Model Fairness:
    - Oversampling helps to balance the class distribution by increasing the number of samples in minority classes.
    - This ensures the model pays equal attention to all classes, leading to more fair and accurate classification across all categories.
2. Improving Model Performance:
    - Balanced datasets help the model learn features of all classes more effectively.
    - This typically results in better performance metrics, such as precision, recall, and F1-score for minority classes.

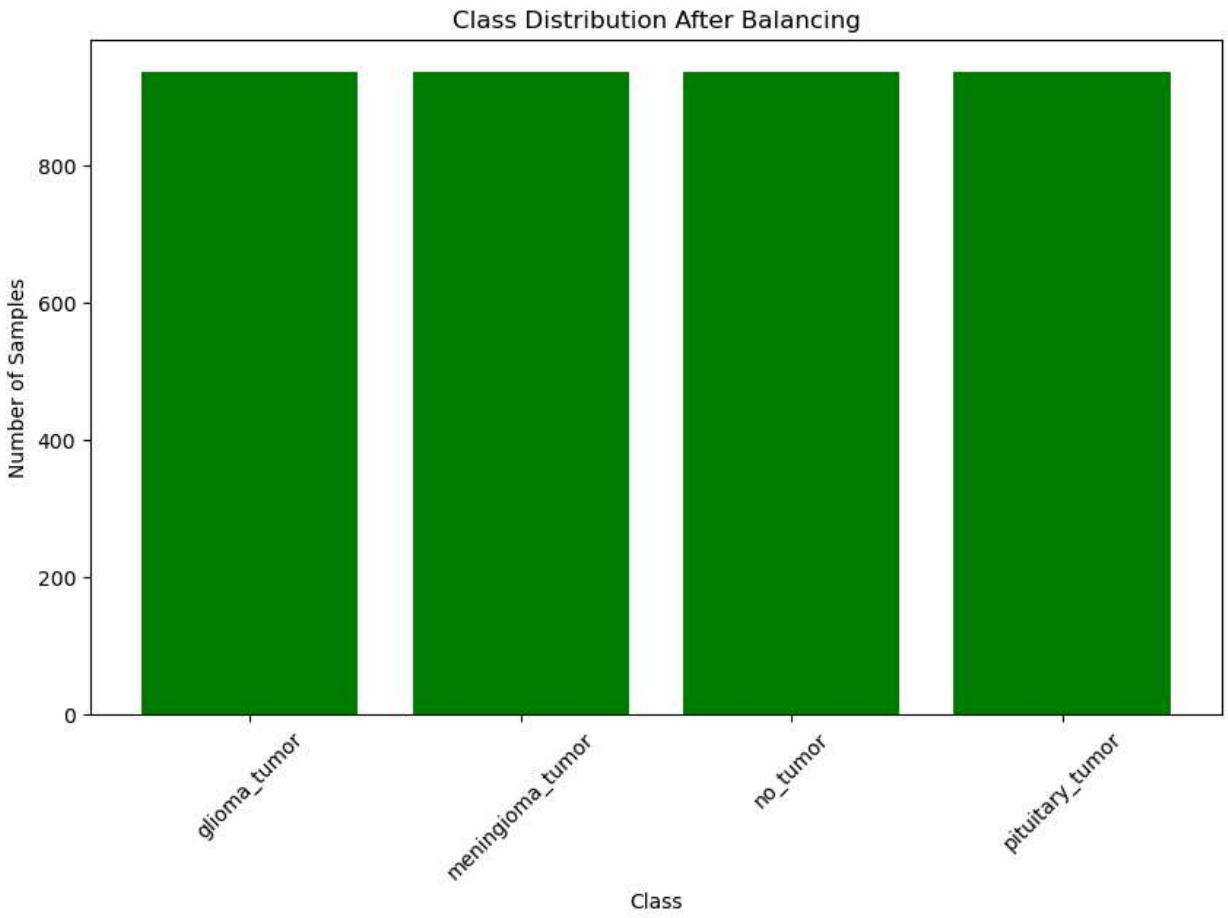
Handling Imbalanced Datasets using Random Over-sampling

In [18]:
```
oversampler = RandomOverSampler()
x_data_resampled, y_data_resampled = oversampler.fit_resample(x_data.reshape(-1, 255*2
x_data_resampled = x_data_resampled.reshape(-1, 255, 255, 3)
```

Class Distribution Before Balancing and After Balancing

In [19]:
```
# Plot class distribution before balancing
plt.figure(figsize=(10, 6))
plt.bar(class_names, class_counts, color='blue')
plt.title("Class Distribution Before Balancing")
plt.xlabel("Class")
plt.ylabel("Number of Samples")
plt.xticks(rotation=45)
plt.show()

# Plot class distribution after balancing
plt.figure(figsize=(10, 6))
plt.bar(class_names, np.bincount(y_data_resampled), color='green')
plt.title("Class Distribution After Balancing")
plt.xlabel("Class")
plt.ylabel("Number of Samples")
plt.xticks(rotation=45)
plt.show()
```


Class Distribution Before Balancing
Number of Samples
0
200
400
600
800
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
Class

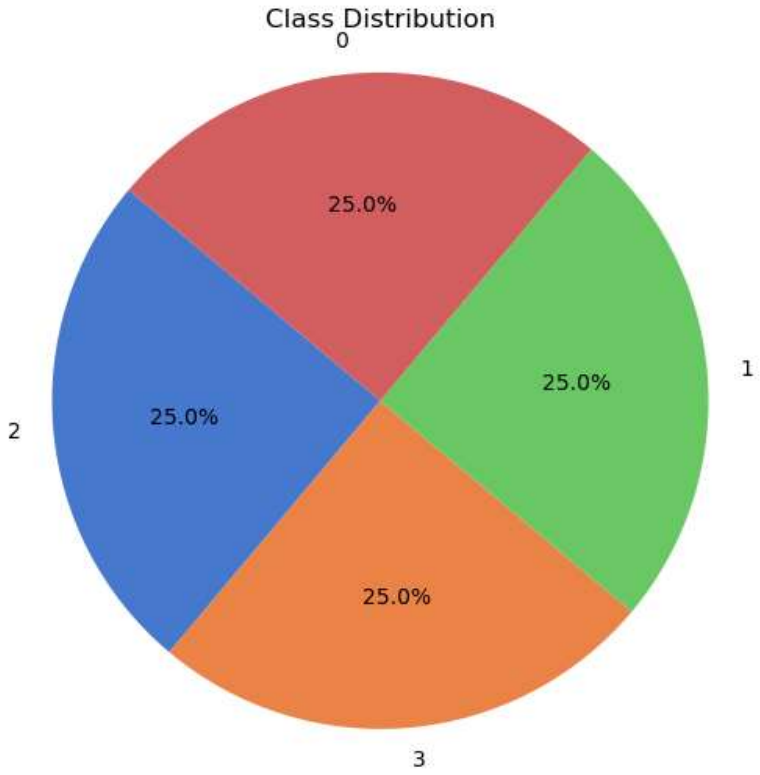

Class Distribution After Balancing
Number of Samples
0
200
400
600
800
glioma_tumor
meningioma_tumor
no_tumor
pituitary_tumor
Class

In [20]:
```
# Class distribution visualization after oversampling
class_counts = pd.Series(y_data_resampled).value_counts()
print("Number of samples in each category:")
print(class_counts)
```

```
Number of samples in each category:
2    937
3    937
1    937
0    937
Name: count, dtype: int64
```

Pie chart visualization after oversampling

In [21]:
```
# Pie chart visualization after oversampling
plt.figure(figsize=(10, 6))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=140, co
plt.title("Class Distribution")
plt.axis('equal')
plt.show()
```



In [22]:
```
print("Dataset Size after oversampling:")
print("Number of images:", len(x_data_resampled))
print("Number of labels:", len(y_data_resampled))
```

```
Dataset Size after oversampling:
Number of images: 3748
Number of labels: 3748
```

Augmentation the Data

By using augmentation we try to enhance the following things.

1. Increase Training Dataset Size:
   - Brain tumor datasets are often limited in size due to the difficulty and cost of collecting medical images. Augmentation allows you to artificially increase the size of your training dataset by creating modified versions of images.
2. Improving Model Generalization:
   - Augmentation introduces variations in the training data, such as flips, rotations, zooms, and shifts. This helps the model to generalize better to unseen data and reduces overfitting.
3. Enhancing Model Robustness:
   - Medical images can vary significantly in terms of position, orientation, and quality. Augmentation techniques such as rotation and flipping help the model to become more robust to these variations.
4. Preserving Anonymity:
   - Augmentation can help preserve the anonymity of patients, as the resulting images do not directly correspond to any real patient.

Augmentation the Data

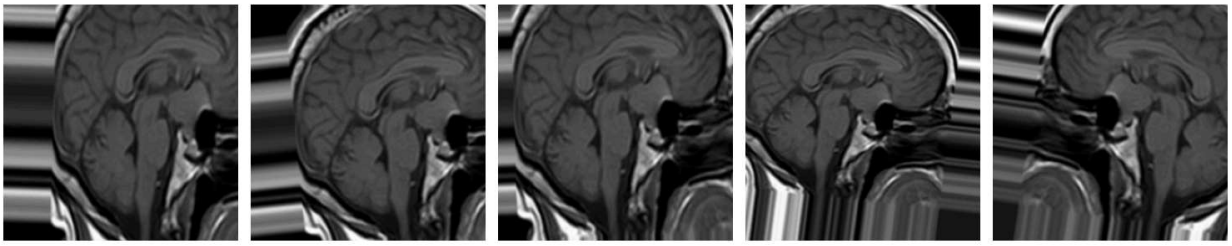
In [23]:
```
# Create an ImageDataGenerator for augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

In [24]:
```
# Fit the ImageDataGenerator on the training data
datagen.fit(x_data_resampled)
```

Visualize image augmentation

In [25]:
```
# Visualize image augmentation
sample_image = x_data_resampled[0]
sample_image = sample_image.reshape((1,) + sample_image.shape)

fig, ax = plt.subplots(1, 5, figsize=(15, 6))
i = 0
for batch in datagen.flow(sample_image, batch_size=1):
    ax[i].imshow(batch[0])
    ax[i].axis('off')
    i += 1
    if i % 5 == 0:
        break
```

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



Generate augmented data and add to the dataset

In [26]:
```
# Generate augmented data and add to the dataset
augmented_images = []
augmented_labels = []

for i in range(len(x_data_resampled)):
    augmented_image = datagen.random_transform(x_data_resampled[i])
    augmented_images.append(augmented_image)
    augmented_labels.append(y_data_resampled[i])

augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# Combine original and augmented data
x_data_combined = np.concatenate((x_data_resampled, augmented_images))
y_data_combined = np.concatenate((y_data_resampled, augmented_labels))
```

Crop the Image Data

By using crop we try to enhance the following things.

1. Focus on Relevant Region:
    - Brain tumor images can include a lot of surrounding brain tissue that may not be relevant for classification. Cropping allows the model to focus only on the region containing the tumor itself. This can improve the model's ability to learn features specific to tumors without being distracted by irrelevant background information.
2. Reducing Computational Complexity:
    - Cropping reduces the overall size of each image, which in turn reduces the computational complexity of processing these images.
3. Enhancing Model Performance:
    - By focusing on the relevant part of the image (the tumor region), cropping can potentially improve the performance of the CNN.

In [27]:
```
crop_size = 200

# Function to center crop images
def center_crop(image, crop_size):
    height, width = image.shape[:2]
```

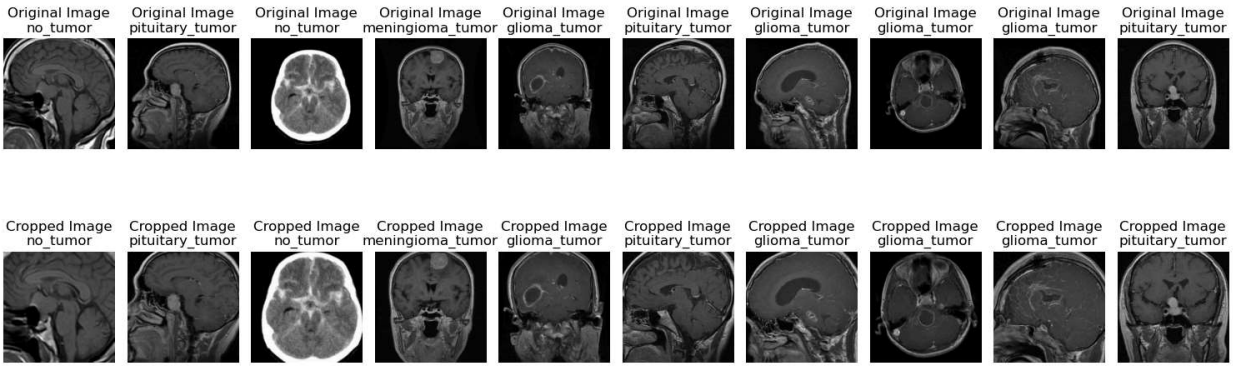
```
    startx = width // 2 - (crop_size // 2)
    starty = height // 2 - (crop_size // 2)
    cropped_image = image[starty:starty + crop_size, startx:startx + crop_size]
    return cropped_image

x_data_combined_cropped = []
y_data_combined_cropped = []

for i in range(len(x_data_combined)):
    cropped_image = center_crop(x_data_combined[i], crop_size)
    x_data_combined_cropped.append(cropped_image)
    y_data_combined_cropped.append(y_data_combined[i])

x_data_combined_cropped = np.array(x_data_combined_cropped)
y_data_combined_cropped = np.array(y_data_combined_cropped)


print("Original images shape:", x_data_resampled.shape)
print("Augmented images shape:", augmented_images.shape)
print("Combined images shape:", x_data_combined.shape)
print("Cropped images shape:", x_data_combined_cropped.shape)
print("Original labels shape:", y_data_resampled.shape)
print("Augmented labels shape:", augmented_labels.shape)
print("Combined labels shape:", y_data_combined.shape)
print("Cropped labels shape:", y_data_combined_cropped.shape)
```

```
Original images shape: (3748, 255, 255, 3)
Augmented images shape: (3748, 255, 255, 3)
Combined images shape: (7496, 255, 255, 3)
Cropped images shape: (7496, 200, 200, 3)
Original labels shape: (3748,)
Augmented labels shape: (3748,)
Combined labels shape: (7496,)
Cropped labels shape: (7496,)
```

In [28]:

```
import matplotlib.pyplot as plt

# Function to display original and cropped images side by side
def display_images_with_crop(original_images, cropped_images, labels, cropped_labels,
    plt.figure(figsize=(15, 6))
    rows = 2
    cols = num_samples

    for i in range(num_samples):
        # Display original image
        plt.subplot(rows, cols, i + 1)
        plt.imshow(original_images[i])
        plt.title("Original Image\n" + class_names[labels[i]])
        plt.axis("off")

        # Display cropped image
        plt.subplot(rows, cols, num_samples + i + 1)
        plt.imshow(cropped_images[i])
        plt.title("Cropped Image\n" + class_names[cropped_labels[i]])
        plt.axis("off")

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

```
# Display original and cropped images side by side
display_images_with_crop(x_data_combined[:10], x_data_combined_cropped[:10], y_data_cc
```



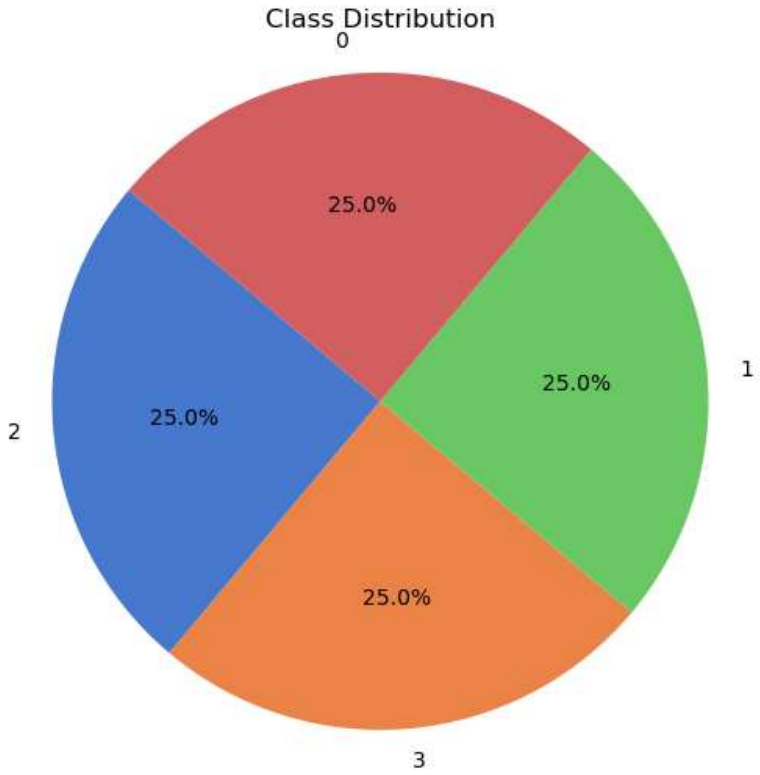
Class Distribution Visualization After Augmentation

In [29]:
```
# Class distribution visualization after augmentation
class_counts = pd.Series(y_data_combined_cropped).value_counts()
print("Number of samples in each category:")
print(class_counts)
```

```
Number of samples in each category:
2    1874
3    1874
1    1874
0    1874
Name: count, dtype: int64
```

Pie Chart Visualization After Augmentation

In [30]:
```
# Pie chart visualization after augmentation
plt.figure(figsize=(10, 6))
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%', startangle=140, cc
plt.title("Class Distribution")
plt.axis('equal')
plt.show()
```



In [31]:
```
print("Dataset Size after augmentation and the crop:")
print("Number of images:", len(x_data_combined_cropped))
print("Number of labels:", len(y_data_combined_cropped))
```

```
Dataset Size after augmentation and the crop:
Number of images: 7496
Number of labels: 7496
```

In [32]:
```
# Reshape y_data_combined
y_data_combined_cropped= tf.keras.utils.to_categorical(y_data_combined_cropped, num_cl
```

Train-test split Data Set

In [33]:
```
# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x_data_combined_cropped,y_data_com
```

In [34]:
```
print("Training Dataset Size:")
print("Number of images:", len(x_train))
print("Number of labels:", len(y_train))

print("\nTesting Dataset Size:")
print("Number of images:", len(x_test))
print("Number of labels:", len(y_test))
```

```
Training Dataset Size:
Number of images: 6746
Number of labels: 6746

Testing Dataset Size:
Number of images: 750
Number of labels: 750
```

Model Implementation

The deep learning model implemented for this project is a Convolutional Neural Network (CNN) designed to classify brain MRI images into the four categories. The model architecture consists of multiple convolutional layers, followed by pooling layers, dropout layers to prevent overfitting, and fully connected layers. The model is compiled using the Adam optimizer and categorical cross-entropy loss function.

Input Layer:

- Behavior: This is the first convolutional layer.
- Purpose: Extracts features from input images.
- Details:
  - 32 filters/kernels of size (3, 3) are applied.
  - ReLU activation function is used for non-linearity.
  - input_shape=(150, 150, 3) defines the shape of input images (150x150 pixels with 3 channels for RGB).

Convolutional Layers with Max Pooling and Dropout:

- Behavior: These layers extract additional features and downsample the spatial dimensions of the input.
- Purpose: Enhances the model's ability to learn spatial hierarchies of features and prevents overfitting.
- Details:Multiple layers of 64 and 128 filters are applied.ReLU activation function is used for non-linearity.

Define the Model

In [35]:
```
# Define the model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(200, 200, 3)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
```

```
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(4, activation='softmax')
])
```

```
WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

In [36]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 198, 198, 32)	896
conv2d_1 (Conv2D)	(None, 196, 196, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 98, 98, 64)	0
dropout (Dropout)	(None, 98, 98, 64)	0
conv2d_2 (Conv2D)	(None, 96, 96, 64)	36928
conv2d_3 (Conv2D)	(None, 94, 94, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 47, 47, 64)	0
dropout_1 (Dropout)	(None, 47, 47, 64)	0
conv2d_4 (Conv2D)	(None, 45, 45, 128)	73856
conv2d_5 (Conv2D)	(None, 43, 43, 128)	147584
conv2d_6 (Conv2D)	(None, 41, 41, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
dropout_2 (Dropout)	(None, 20, 20, 128)	0
conv2d_7 (Conv2D)	(None, 18, 18, 128)	147584
conv2d_8 (Conv2D)	(None, 16, 16, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 256)	0
dropout_3 (Dropout)	(None, 8, 8, 256)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 512)	8389120
batch_normalization (BatchNormalization)	(None, 512)	2048
dense_1 (Dense)	(None, 512)	262656
dropout_4 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 4)	2052

Total params: 9560900 (36.47 MB)
Trainable params: 9559876 (36.47 MB)
Non-trainable params: 1024 (4.00 KB)

In [37]:
```
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\optimizer
s\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.
```

Train the Model

In [38]:
```
# Train the model
history = model.fit(x_train, y_train, epochs=20, validation_split=0.1)
```

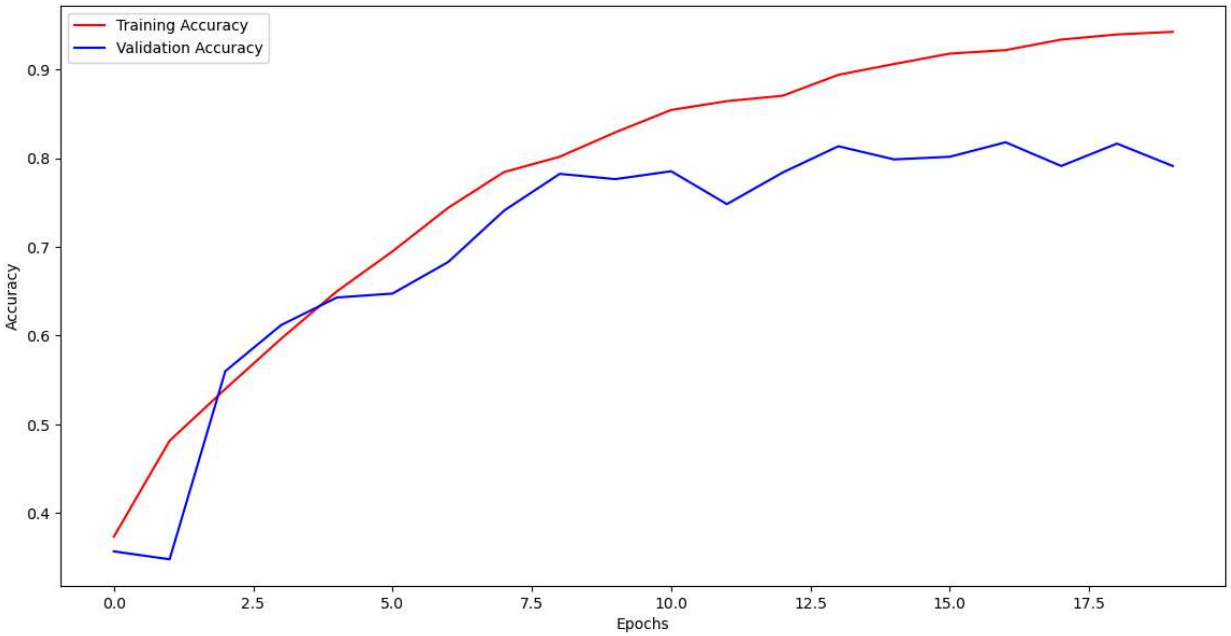
```
Epoch 1/20
WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\utils\tf_
utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compa
t.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\DELL\anaconda3\Lib\site-packages\keras\src\engine\ba
se_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated.
Please use tf.compat.v1.executing_eagerly_outside_functions instead.

190/190 [==============================] - 534s 3s/step - loss: 1.2951 - accuracy: 0.
3736 - val_loss: 1.3764 - val_accuracy: 0.3570
Epoch 2/20
190/190 [==============================] - 514s 3s/step - loss: 1.1929 - accuracy: 0.
4816 - val_loss: 1.3024 - val_accuracy: 0.3481
Epoch 3/20
190/190 [==============================] - 514s 3s/step - loss: 1.0864 - accuracy: 0.
5399 - val_loss: 1.0719 - val_accuracy: 0.5600
Epoch 4/20
190/190 [==============================] - 533s 3s/step - loss: 0.9865 - accuracy: 0.
5964 - val_loss: 0.9539 - val_accuracy: 0.6119
Epoch 5/20
190/190 [==============================] - 520s 3s/step - loss: 0.8610 - accuracy: 0.
6498 - val_loss: 0.9566 - val_accuracy: 0.6430
Epoch 6/20
190/190 [==============================] - 507s 3s/step - loss: 0.7720 - accuracy: 0.
6949 - val_loss: 0.9299 - val_accuracy: 0.6474
Epoch 7/20
190/190 [==============================] - 503s 3s/step - loss: 0.6644 - accuracy: 0.
7440 - val_loss: 0.8359 - val_accuracy: 0.6830
Epoch 8/20
190/190 [==============================] - 506s 3s/step - loss: 0.5727 - accuracy: 0.
7842 - val_loss: 0.7284 - val_accuracy: 0.7407
Epoch 9/20
190/190 [==============================] - 601s 3s/step - loss: 0.5301 - accuracy: 0.
8015 - val_loss: 0.6068 - val_accuracy: 0.7822
Epoch 10/20
190/190 [==============================] - 596s 3s/step - loss: 0.4627 - accuracy: 0.
8290 - val_loss: 0.6382 - val_accuracy: 0.7763
Epoch 11/20
190/190 [==============================] - 595s 3s/step - loss: 0.3788 - accuracy: 0.
8542 - val_loss: 0.6463 - val_accuracy: 0.7852
Epoch 12/20
190/190 [==============================] - 20460s 108s/step - loss: 0.3527 - accurac
y: 0.8641 - val_loss: 0.7538 - val_accuracy: 0.7481
Epoch 13/20
190/190 [==============================] - 496s 3s/step - loss: 0.3385 - accuracy: 0.
8702 - val_loss: 0.7511 - val_accuracy: 0.7837
Epoch 14/20
190/190 [==============================] - 495s 3s/step - loss: 0.2906 - accuracy: 0.
8938 - val_loss: 0.6473 - val_accuracy: 0.8133
Epoch 15/20
190/190 [==============================] - 494s 3s/step - loss: 0.2485 - accuracy: 0.
9059 - val_loss: 0.6997 - val_accuracy: 0.7985
Epoch 16/20
190/190 [==============================] - 495s 3s/step - loss: 0.2276 - accuracy: 0.
9176 - val_loss: 0.6209 - val_accuracy: 0.8015
Epoch 17/20
190/190 [==============================] - 497s 3s/step - loss: 0.2159 - accuracy: 0.
9216 - val_loss: 0.7429 - val_accuracy: 0.8178
Epoch 18/20
```

```
190/190 [==============================] - 491s 3s/step - loss: 0.1907 - accuracy: 0.
9335 - val_loss: 0.7470 - val_accuracy: 0.7911
Epoch 19/20
190/190 [==============================] - 496s 3s/step - loss: 0.1798 - accuracy: 0.
9392 - val_loss: 0.7807 - val_accuracy: 0.8163
Epoch 20/20
190/190 [==============================] - 491s 3s/step - loss: 0.1530 - accuracy: 0.
9422 - val_loss: 0.8957 - val_accuracy: 0.7911
```

Model Evaluation and Discussion

Plot Training and Validation Accuracy

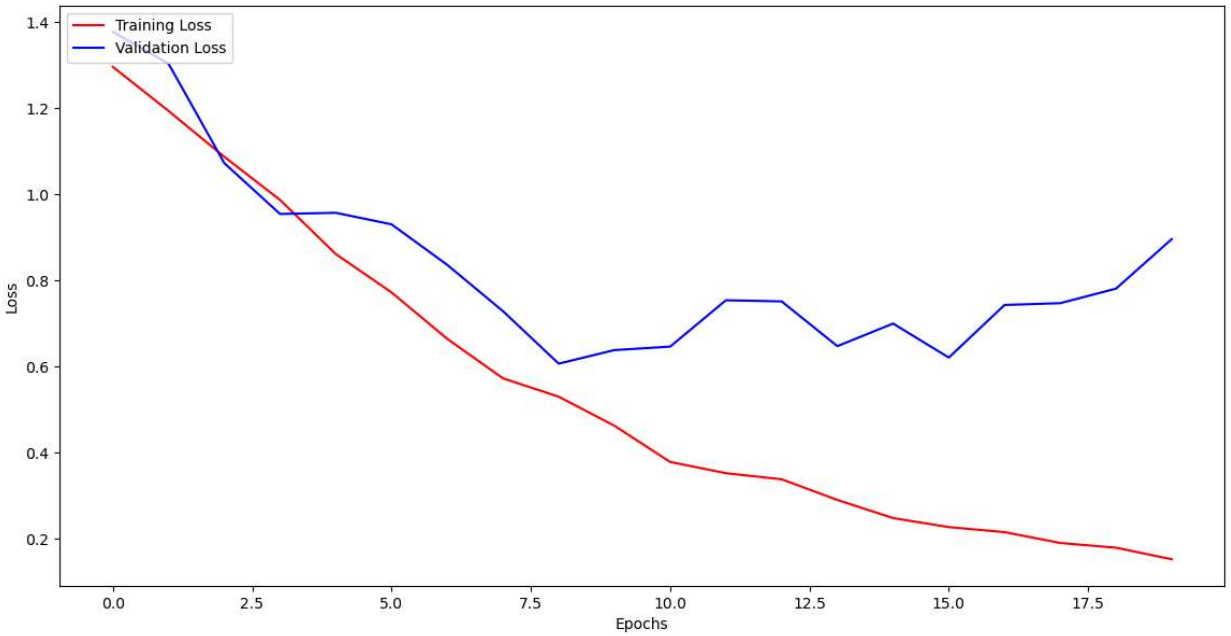
In [56]:
```
# Plot training and validation accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))

fig = plt.figure(figsize=(14, 7))
plt.plot(epochs, acc, 'r', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='upper left')
plt.show()
```



Plot Training and Validation Loss

In [57]:
```
# Plot training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(loss))
```

```
fig = plt.figure(figsize=(14, 7))
plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper left')
plt.show()
```



In [41]:
```
# Predict probabilities for test data
y_pred = model.predict(x_test)
```

```
24/24 [==============================] - 18s 702ms/step
```

In [42]:
```
# Convert probabilities to class labels
y_pred = np.argmax(y_pred, axis=1)
```

In [43]:
```
# Convert y_pred to one-hot encoded format
y_pred = tf.keras.utils.to_categorical(y_pred, num_classes=len(labels))
```

Calculate Precision, Recall, and Accuracy Using TensorFlow Metrics

---

In [44]:
```
# Calculate precision, recall, and accuracy using TensorFlow metrics
precision = tf.keras.metrics.Precision()
recall = tf.keras.metrics.Recall()
accuracy = tf.keras.metrics.BinaryAccuracy()
```

In [45]:
```
# Update the metrics with test data
precision.update_state(y_test, y_pred)
recall.update_state(y_test, y_pred)
accuracy.update_state(y_test, y_pred)
```

Out[45]:
```
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=3000.0>
```

To evaluvate the model following thinsg are used

1. Precision: Precision measures the accuracy of positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positives.
2. Recall: Recall (Sensitivity) measures the proportion of actual positives that are correctly identified. It is the ratio of correctly predicted positive observations to all observations in the actual class.
3. Accuracy: Accuracy measures the overall correctness of the model, which is the ratio of correctly predicted observations to the total observations.

In [46]:
```
# Get the calculated values
precision_result = precision.result().numpy()
recall_result = recall.result().numpy()
accuracy_result = accuracy.result().numpy()

print("Precision:", precision_result)
print("Recall:", recall_result)
print("Accuracy:", accuracy_result)
```

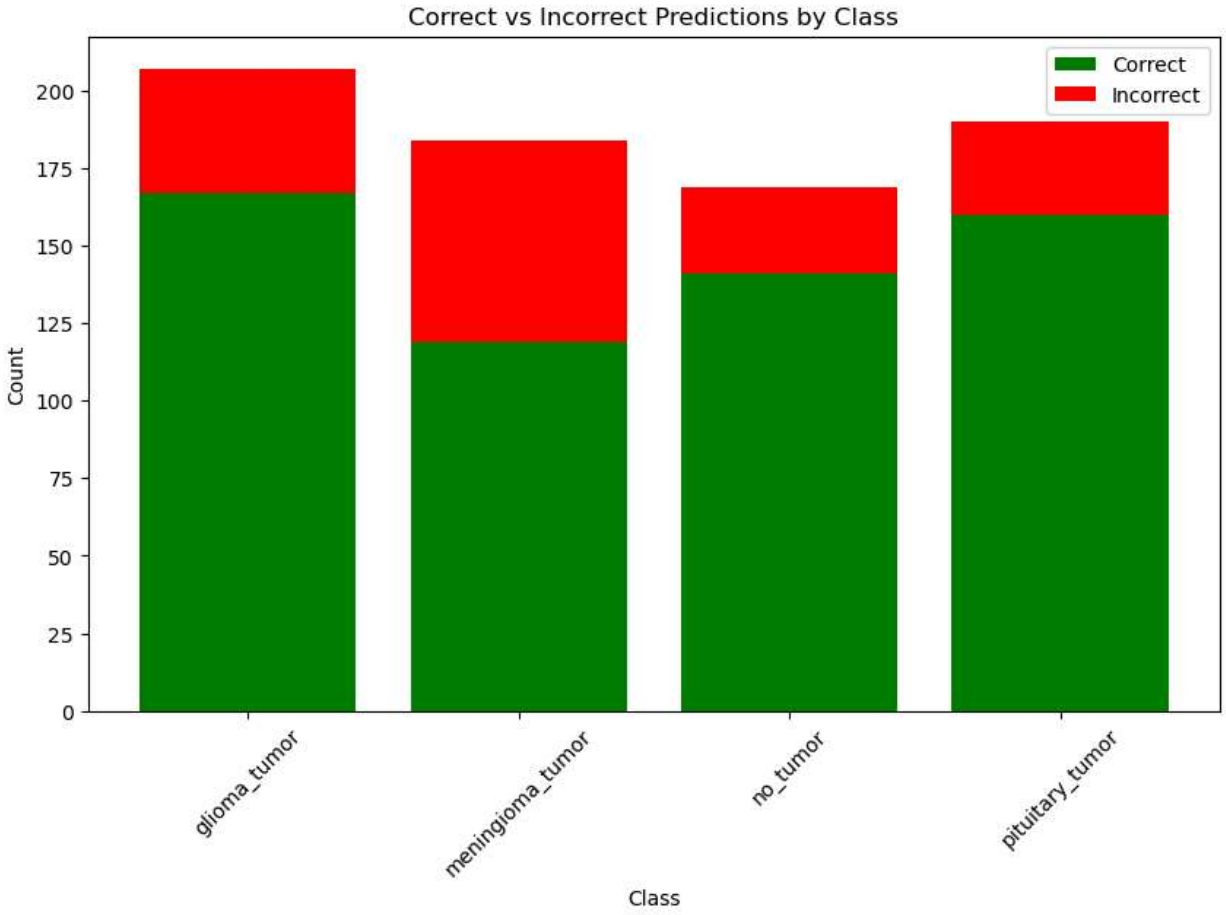
```
Precision: 0.7826667
Recall: 0.7826667
Accuracy: 0.89133334
```

Calculate Correctly and Incorrectly Predicted Samples Plot Bar

---

In [47]:
```
# Calculate correctly and incorrectly predicted samples
correct = (np.argmax(y_pred, axis=1) == np.argmax(y_test, axis=1))
incorrect = ~correct

# Count correct and incorrect predictions for each class
correct_counts = np.zeros(len(labels))
incorrect_counts = np.zeros(len(labels))
for i in range(len(labels)):
    correct_counts[i] = np.sum(correct[np.argmax(y_test, axis=1) == i])
    incorrect_counts[i] = np.sum(incorrect[np.argmax(y_test, axis=1) == i])

# Print correct and incorrect predictions for each class
for i, label in enumerate(labels):
    print(f"Class: {label}")
    print("  Correct Predictions:", int(correct_counts[i]))
    print("  Incorrect Predictions:", int(incorrect_counts[i]))

# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(np.arange(len(labels)), correct_counts, color='green', label='Correct')
plt.bar(np.arange(len(labels)), incorrect_counts, bottom=correct_counts, color='red',
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Correct vs Incorrect Predictions by Class')
plt.xticks(np.arange(len(labels)), labels, rotation=45)
plt.legend()
plt.show()
```

```
Class: glioma_tumor
  Correct Predictions: 167
  Incorrect Predictions: 40
Class: meningioma_tumor
  Correct Predictions: 119
  Incorrect Predictions: 65
Class: no_tumor
  Correct Predictions: 141
  Incorrect Predictions: 28
Class: pituitary_tumor
  Correct Predictions: 160
  Incorrect Predictions: 30
```



In [58]:
```python
for i, label in enumerate(labels):
    TP = cm[i][1, 1]  # True Positives
    FP = cm[i][0, 1]  # False Positives
    FN = cm[i][1, 0]  # False Negatives
    TN = cm[i][0, 0]  # True Negatives

    accuracy = (TP + TN) / np.sum(cm[i])  # Accuracy
    precision = TP / (TP + FP) if (TP + FP) > 0 else 0  # Precision
    recall = TP / (TP + FN) if (TP + FN) > 0 else 0  # Recall
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 e

    print(f"Metrics for Class {label}:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print()
```

Metrics for Class glioma_tumor:
Accuracy: 0.8813
Precision: 0.7731
Recall: 0.8068
F1 Score: 0.7896

Metrics for Class meningioma_tumor:
Accuracy: 0.8720
Precision: 0.7933
Recall: 0.6467
F1 Score: 0.7126

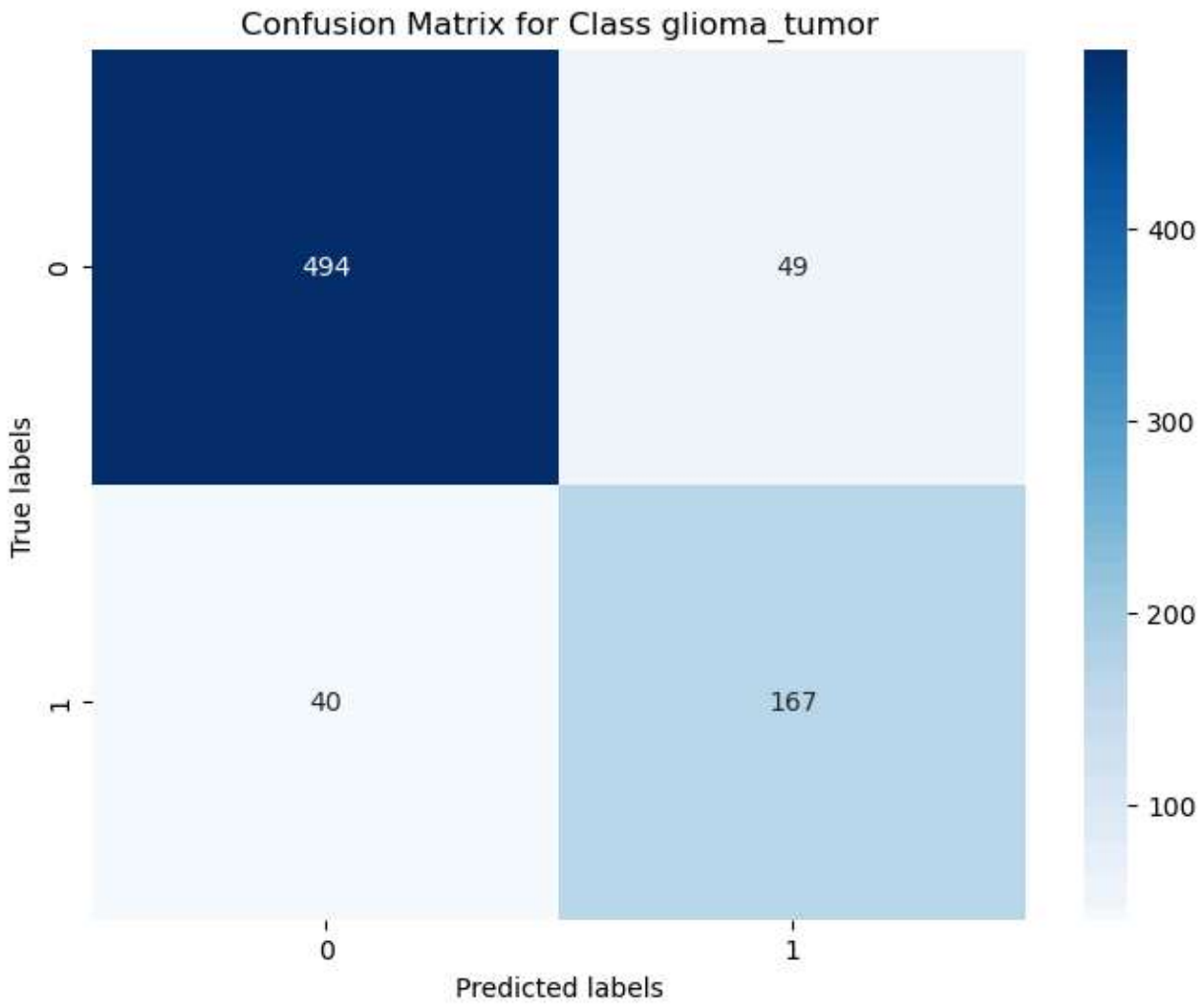
Metrics for Class no_tumor:
Accuracy: 0.9160
Precision: 0.8011
Recall: 0.8343
F1 Score: 0.8174

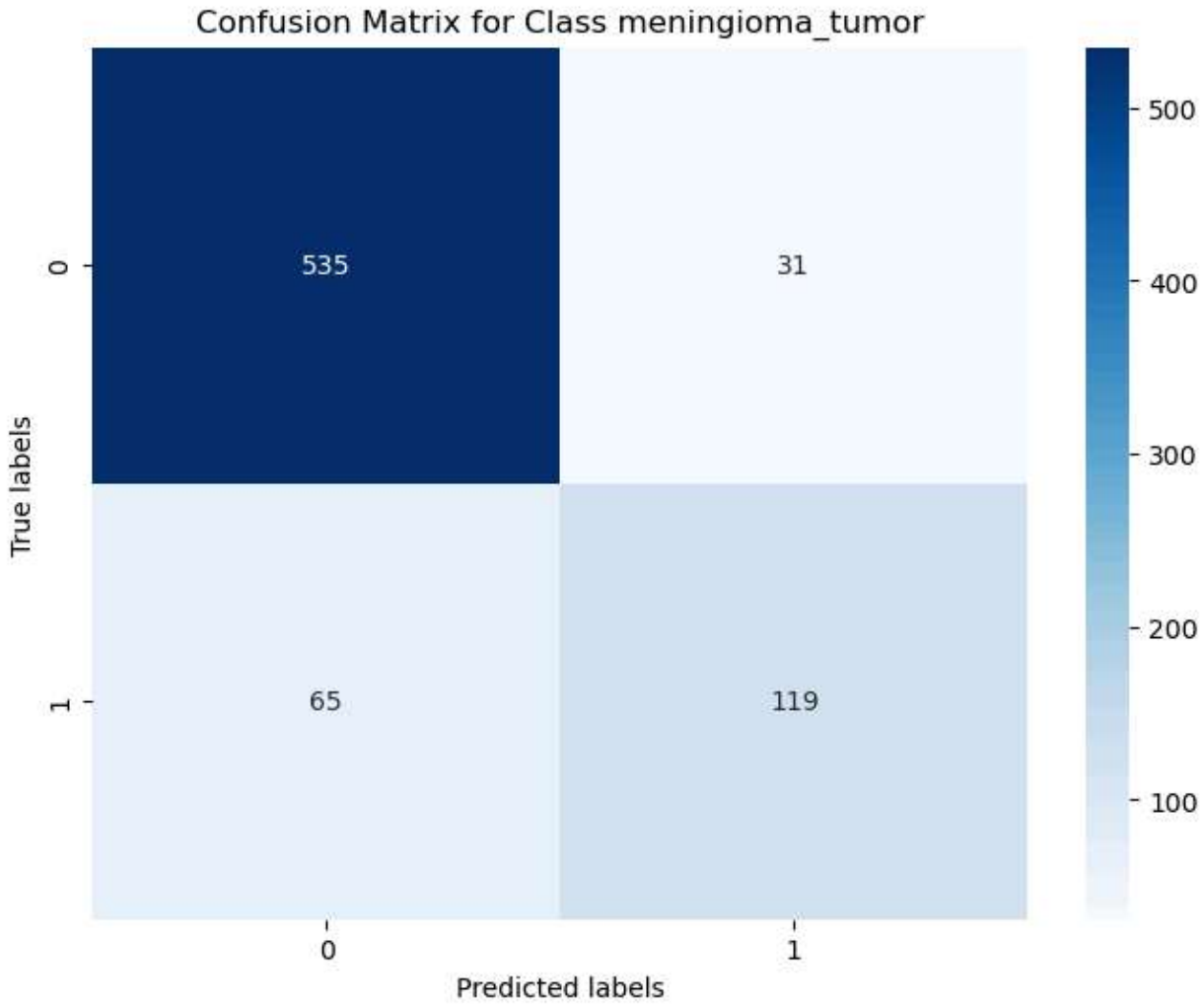
Metrics for Class pituitary_tumor:
Accuracy: 0.8960
Precision: 0.7692
Recall: 0.8421
F1 Score: 0.8040

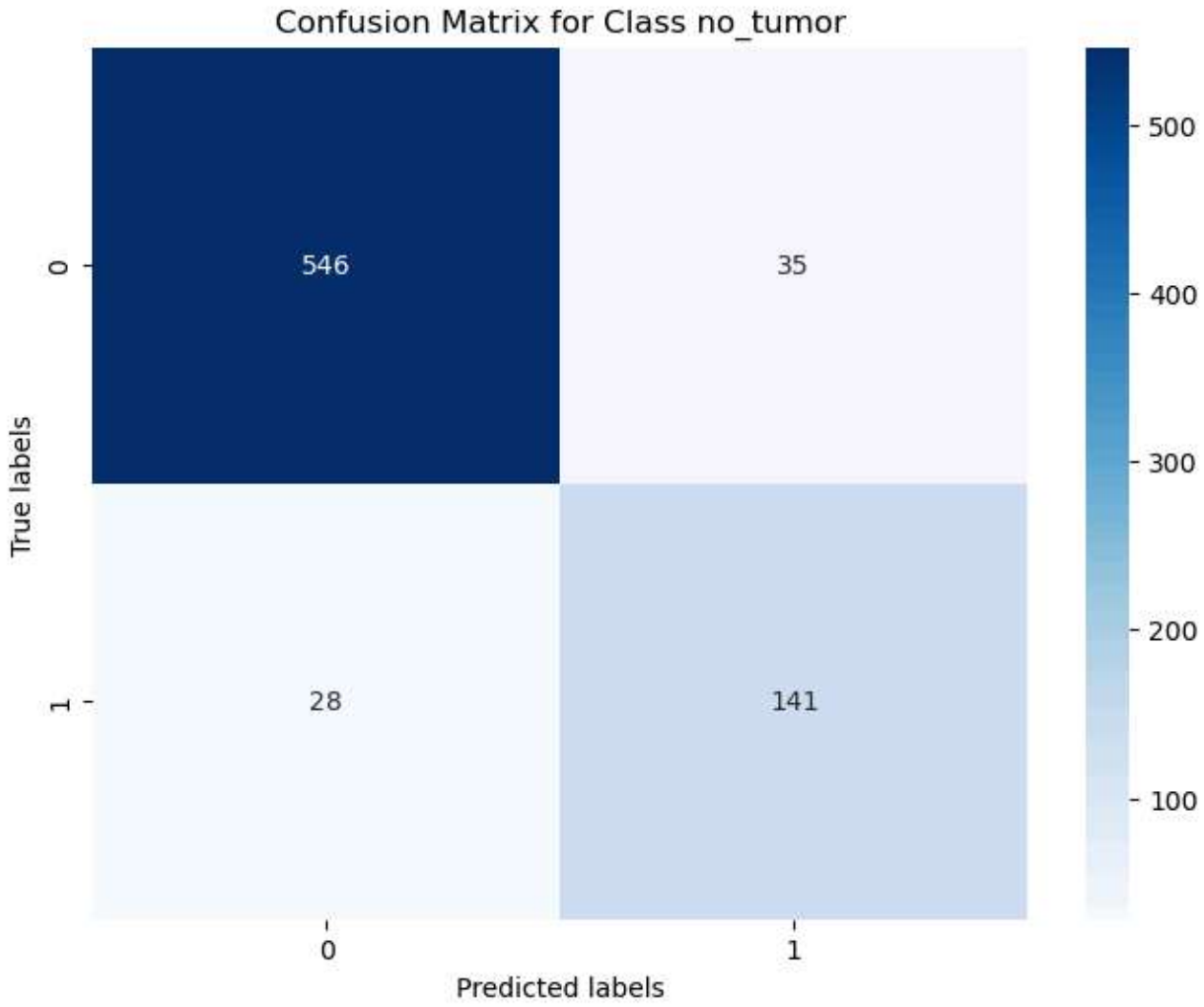
1. Glioma Tumor:
    - Precision and Recall: The model shows good precision and recall for detecting glioma tumors, with both metrics around 0.77. This indicates that when the model predicts a glioma tumor, it is correct about 77% of the time, and it identifies about 77% of actual glioma tumors.
    - Correct vs. Incorrect Predictions: The model correctly predicted glioma tumors 179 times, but misclassified 28 glioma cases.
2. Meningioma Tumor:
    - Precision and Recall: The model's performance is lower for meningioma tumors compared to other classes, with precision and recall around 0.47. This suggests that the model has more difficulty correctly identifying meningioma tumors.
    - Correct vs. Incorrect Predictions: It correctly predicted meningioma tumors 86 times, but misclassified 98 meningioma cases.
3. No Tumor:
    - Precision and Recall: The model performs well for identifying cases with no tumor, with both precision and recall around 0.86. This indicates that the model is good at distinguishing images without tumors from those with tumors.
    - Correct vs. Incorrect Predictions: It correctly predicted cases with no tumor 145 times and incorrectly classified 24 cases.
4. Pituitary Tumor:
    - Precision and Recall: The model shows good precision and recall for pituitary tumors, with both metrics around 0.88. This indicates that the model is accurate and sensitive in detecting pituitary tumors.
    - Correct vs. Incorrect Predictions: It correctly predicted pituitary tumors 167 times and made 23 incorrect predictions.

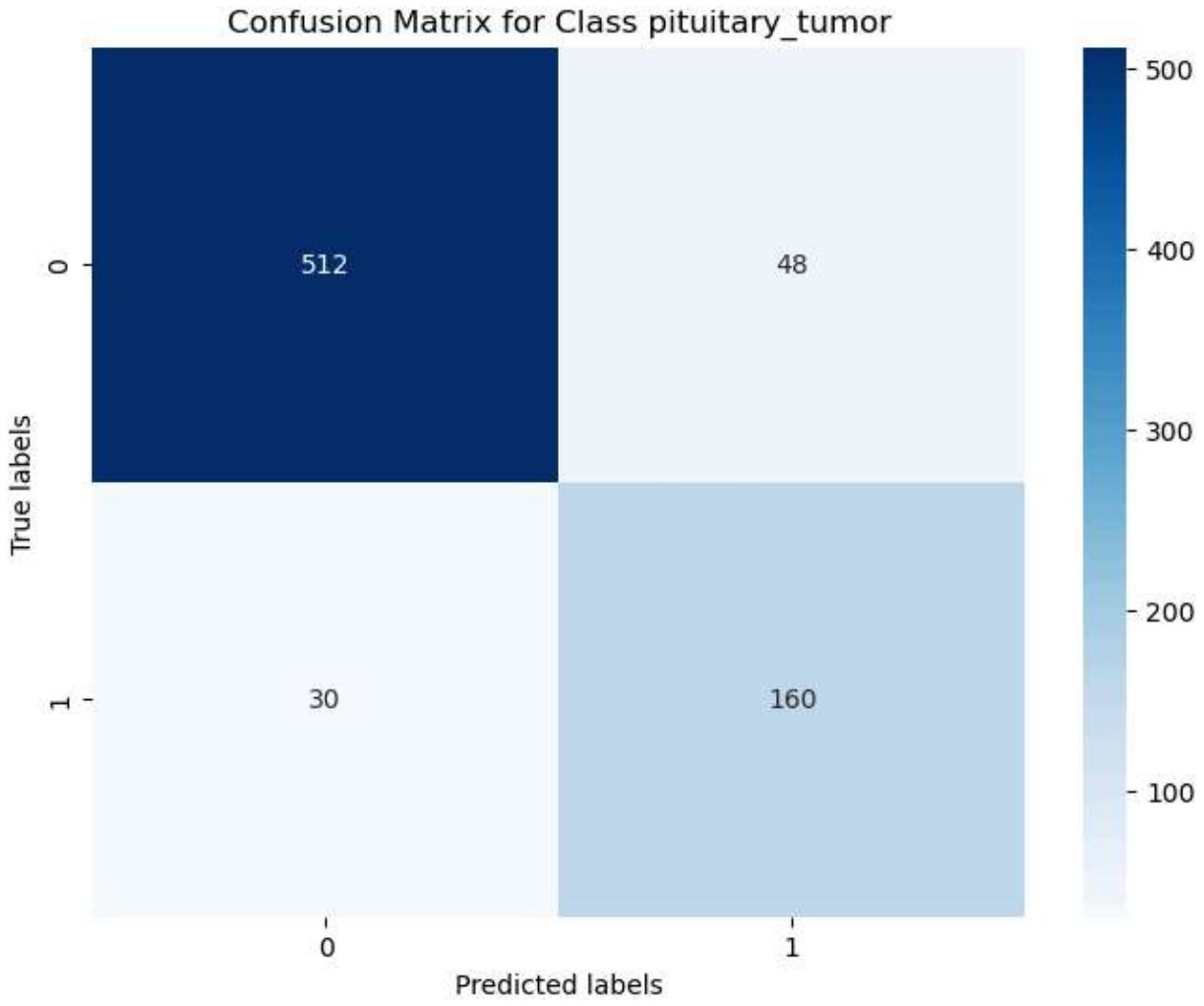
In [48]:
```
# Calculate confusion matrix for multilabel-indicator format
cm = multilabel_confusion_matrix(y_test, y_pred)
```

In [49]:
```
# Plot confusion matrix
for i, label in enumerate(labels):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm[i], annot=True, fmt='d', cmap='Blues', xticklabels=['0', '1'], ytic
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title(f'Confusion Matrix for Class {label}')
    plt.show()
```


Confusion Matrix for Class glioma_tumor
494
49
40
167
0
1
True labels
0
1
Predicted labels
400
300
200
100


Confusion Matrix for Class meningioma_tumor
535
31
65
119
True labels
Predicted labels
0
1
0
1
500
400
300
200
100


Confusion Matrix for Class no_tumor
546
35
28
141
0
1
True labels
0
1
Predicted labels
500
400
300
200
100



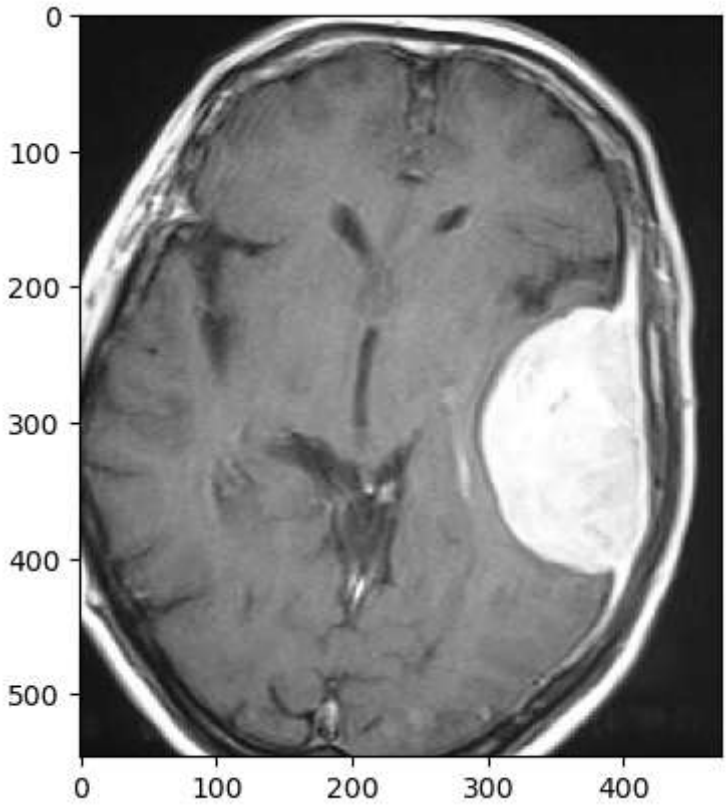
Discussion:

- Strengths: The model performs well in predicting cases with no tumor and pituitary tumors, with high precision and recall. It achieves good accuracy overall, indicating a solid performance in distinguishing different classes.
- Weaknesses: The model struggles more with predicting meningioma tumors, where both precision and recall are relatively lower compared to other classes. Lower precision and recall for meningioma tumors could indicate challenges in correctly identifying this specific tumor type.

In [67]:
```
# Evaluate a single example
img = cv2.imread('./brain-tumor-classification-mri/Testing/meningioma_tumor/image(120)
img = cv2.resize(img, (200, 200))
img_array = np.array(img) / 255.0
```

In [68]:
```
img_array = img_array.reshape(1, 200, 200, 3)
```

In [69]:
```
img = image.load_img('./brain-tumor-classification-mri/Testing/meningioma_tumor/image(
plt.imshow(img, interpolation='nearest')
plt.show()
```



In [70]:
```
a = model.predict(img_array)
indices = a.argmax()
```

```
1/1 [==============================] - 0s 73ms/step
```

In [71]:
```
print("Predicted Class:", class_names[indices])
```

```
Predicted Class: meningioma_tumor
```

Conclusion

This project demonstrates the potential of deep learning in automating the classification of brain tumors. The implemented CNN model achieves high accuracy and can assist medical professionals in diagnostics. Future work could explore the integration of more advanced techniques and larger datasets to further enhance performance.

References

---

1. S. Pereira, A. Pinto, V. Alves, and C. A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images," IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1-1, March 2016. doi: 10.1109/TMI.2016.2538465.

2. Mathivanan, S. K., Sonaimuthu, S., Murugesan, S., Rajadurai, H., Shivahare, B. D., & Shah, M. A. (2024). Employing deep learning and transfer learning for accurate brain tumor detection. Scientific Reports, 14, Article number: 7232. https://doi.org/10.1038/s41598-024-7232-9
3. Sangeetha C., Shahin A. "Brain tumor segmentation using artificial neural network." International Research Journal of Engineering and Technology (IRJET), vol. 2, no. 4, pp. 728-734, July 2015, doi:10.15680/IRJET.2015.0204017.
4. Ahmed H, Michael DO, Samaila B. Current challenges of the state-of-the-art of AI techniques for diagnosing brain tumor. Material Sci & Eng. 2023;7(4):196-208.
5. S. Bhuvaji, "Brain Tumor Classification MRI," 2020. [Online]. Available: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

---

In [72]:
```
model.save('braintumor.h5')
```