## Install kaggle library to download kaggle dataset to collab

```
!pip install kaggle

```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.7.4)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
```

Double-click (or enter) to edit

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

## Downloading Kaggle dataSet

```
!kaggle datasets download -d masoudnickparvar/brain-tumor-mri-dataset

```

```
Dataset URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset
License(s): CC0-1.0
Downloading brain-tumor-mri-dataset.zip to /content
 96% 142M/149M [00:01<00:00, 71.7MB/s]
100% 149M/149M [00:02<00:00, 74.4MB/s]
```

## Unzip the data set

```
!unzip /content/brain-tumor-mri-dataset.zip

```

```
  inflating: Training/pituitary/Tr-pi_1411.jpg
  inflating: Training/pituitary/Tr-pi_1412.jpg
  inflating: Training/pituitary/Tr-pi_1413.jpg
  inflating: Training/pituitary/Tr-pi_1414.jpg
  inflating: Training/pituitary/Tr-pi_1415.jpg
  inflating: Training/pituitary/Tr-pi_1416.jpg
  inflating: Training/pituitary/Tr-pi_1417.jpg
  inflating: Training/pituitary/Tr-pi_1418.jpg
  inflating: Training/pituitary/Tr-pi_1419.jpg
  inflating: Training/pituitary/Tr-pi_1420.jpg
  inflating: Training/pituitary/Tr-pi_1421.jpg
  inflating: Training/pituitary/Tr-pi_1422.jpg
  inflating: Training/pituitary/Tr-pi_1423.jpg
  inflating: Training/pituitary/Tr-pi_1424.jpg
  inflating: Training/pituitary/Tr-pi_1425.jpg
  inflating: Training/pituitary/Tr-pi_1426.jpg
  inflating: Training/pituitary/Tr-pi_1427.jpg
  inflating: Training/pituitary/Tr-pi_1428.jpg
  inflating: Training/pituitary/Tr-pi_1429.jpg
  inflating: Training/pituitary/Tr-pi_1430.jpg
  inflating: Training/pituitary/Tr-pi_1431.jpg
  inflating: Training/pituitary/Tr-pi_1432.jpg
  inflating: Training/pituitary/Tr-pi_1433.jpg
  inflating: Training/pituitary/Tr-pi_1434.jpg
  inflating: Training/pituitary/Tr-pi_1435.jpg
  inflating: Training/pituitary/Tr-pi_1436.jpg
  inflating: Training/pituitary/Tr-pi_1437.jpg
  inflating: Training/pituitary/Tr-pi_1438.jpg
  inflating: Training/pituitary/Tr-pi_1439.jpg
  inflating: Training/pituitary/Tr-pi_1440.jpg
  inflating: Training/pituitary/Tr-pi_1441.jpg
  inflating: Training/pituitary/Tr-pi_1442.jpg
  inflating: Training/pituitary/Tr-pi_1443.jpg
  inflating: Training/pituitary/Tr-pi_1444.jpg
  inflating: Training/pituitary/Tr-pi_1445.jpg
  inflating: Training/pituitary/Tr-pi_1446.jpg
  inflating: Training/pituitary/Tr-pi_1447.jpg
  inflating: Training/pituitary/Tr-pi_1448.jpg
  inflating: Training/pituitary/Tr-pi_1449.jpg
  inflating: Training/pituitary/Tr-pi_1450.jpg
  inflating: Training/pituitary/Tr-pi_1451.jpg
  inflating: Training/pituitary/Tr-pi_1452.jpg
  inflating: Training/pituitary/Tr-pi_1453.jpg
  inflating: Training/pituitary/Tr-pi_1454.jpg
  inflating: Training/pituitary/Tr-pi_1455.jpg
  inflating: Training/pituitary/Tr-pi_1456.jpg
```

## Getting statistical analysis from the data set

```python
import os
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# Define directories for training and testing datasets
train_directory = '/content/Training'
test_directory = '/content/Testing'

# Function to calculate the number of images in each class directory
def get_image_counts(directory):
    class_folders = os.listdir(directory)
    image_counts = {}
    total_images = 0
    for folder in class_folders:
        folder_path = os.path.join(directory, folder)
        if os.path.isdir(folder_path):
            num_images = len(os.listdir(folder_path))
            image_counts[folder] = num_images
            total_images += num_images
```

```
    return image_counts, total_images

# Get counts for training images
training_counts, training_total = get_image_counts(train_directory)

# Get counts for testing images
testing_counts, testing_total = get_image_counts(test_directory)

# Aggregate counts from both training and testing sets
combined_counts = {}
for class_name in training_counts:
    combined_counts[class_name] = training_counts[class_name] + testing_counts.get(class_name, 0)

# Generate a color map
color_map = cm.get_cmap('Set2', len(combined_counts))  # Changed color map to 'Set2'

# Plotting the results
plt.figure(figsize=(12, 7))
bars = plt.bar(combined_counts.keys(), combined_counts.values(), color=[color_map(i) for i in range(len(c

# Customize the plot appearance
plt.xlabel('Class Categories', fontsize=14)
plt.ylabel('Total Image Count', fontsize=14)
plt.title('Image Distribution Across Classes ', fontsize=16)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Add value labels inside the bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height / 2, int(height), ha='center', va='center', fontsi

# Create a custom legend with updated title
legend_labels = combined_counts.keys()
legend_colors = [color_map(i) for i in range(len(combined_counts))]
legend_elements = [plt.Line2D([0], [0], color=legend_colors[i], lw=4) for i in range(len(legend_labels))]
plt.legend(legend_elements, legend_labels, title='Classes', bbox_to_anchor=(1.05, 1), loc='upper left', f

plt.tight_layout()
plt.show()

# Output total image counts for each class
print("Combined Image Count per Class (Training + Testing):")
for class_name, count in combined_counts.items():
    print(f"{class_name}: {count}")

# Output total image counts for training and testing sets
print(f"Total Images in Training Set: {training_total}")
print(f"Total Images in Testing Set: {testing_total}")
```


```
Combined Image Count per Class (Training + Testing):
glioma: 1621
pituitary: 1757
meningioma: 1645
notumor: 2000
```

## plotting single image from the Training data set

```
import os
import numpy as np
import cv2
from tensorflow.keras.utils import to_categorical

# Define the path to the dataset folder
dataset_path = "/content/Training"


# Define the list of label folders in the dataset folder
label_folders = ['glioma','meningioma','notumor','pituitary']

# Define the size of the input images
img_height = 128
img_width = 128

# Define an empty list to store the images and their labels
data = []
labels = []

# Create a dictionary to map label folders to numerical labels
label_mapping = {label: idx for idx, label in enumerate(label_folders)}

# Loop over the label folders in the dataset folder
for label_folder in label_folders:
    # Define the path to the label folder
    label_path = os.path.join(dataset_path, label_folder)

```

```
    # Loop over the images in the label folder
    for img_name in os.listdir(label_path):
        # Define the path to the image
        img_path = os.path.join(label_path, img_name)
        # Load the image and resize it to the desired size
        img = cv2.imread(img_path)
        img = cv2.resize(img, (img_height, img_width))
        # Append the image and its numerical label to the data and labels lists
        data.append(img)
        labels.append(label_mapping[label_folder])

# Convert the data and labels lists to numpy arrays
data = np.array(data)
labels = np.array(labels)

# Convert the labels to one-hot encoded vectors
# labels = to_categorical(labels, num_classes=len(label_folders))

# Print the shape of the data and labels arrays
print("Data shape:", data.shape)
print("Labels shape:", labels.shape)
label_mapping
```

```
Data shape: (5712, 128, 128, 3)
Labels shape: (5712,)
{'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
```

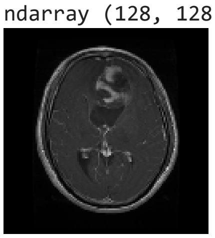
```
labels
```

```
array([0, 0, 0, ..., 3, 3, 3])
```

```
data[0]
```

```
ndarray (128, 128, 3)
```

show data



# Simple CNN model

## Load the dataset from the folder and split it into training and testing

```
import os
import numpy as np
import cv2
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the path to the dataset folders
train_dataset_path = "/content/Training"
test_dataset_path = "/content/Testing"

# Define the list of label folders in the dataset folder
label_folders = ['glioma', 'meningioma', 'notumor', 'pituitary']

# Define the size of the input images
img_height = 128
img_width = 128

# Function to load and preprocess images from a given path
def load_data(dataset_path, label_folders):
    data = []
    labels = []
    label_mapping = {label: idx for idx, label in enumerate(label_folders)}

    for label_folder in label_folders:
        label_path = os.path.join(dataset_path, label_folder)
        for img_name in os.listdir(label_path):
            img_path = os.path.join(label_path, img_name)
            img = cv2.imread(img_path)
            img = cv2.resize(img, (img_height, img_width))
            data.append(img)
            labels.append(label_mapping[label_folder])

    data = np.array(data)
    labels = np.array(labels)
    labels = to_categorical(labels, num_classes=len(label_folders))

    return data, labels

# Load and preprocess training data
train_data, train_labels = load_data(train_dataset_path, label_folders)
# Load and preprocess testing data
test_data, test_labels = load_data(test_dataset_path, label_folders)
```

## Model Creation

```python

# Define the model creation function
def create_simple_neural_network(input_shape, num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model

# Example input shape (adjust based on your image dimensions and channels)
input_shape = (128, 128, 3)

# Example number of classes (adjust based on your dataset)
num_classes = 4

```

## comppile fit and evalution of CNN model

```python

# Create the simplified neural network model
simple_neural_network = create_simple_neural_network(input_shape, num_classes)

# Display the model summary
simple_neural_network.summary()

# Compile the model
simple_neural_network.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = simple_neural_network.fit(train_data, train_labels, epochs=30, validation_split=0.1)

# Evaluate the model performance on the testing dataset
test_loss, test_accuracy = simple_neural_network.evaluate(test_data, test_labels)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")

```

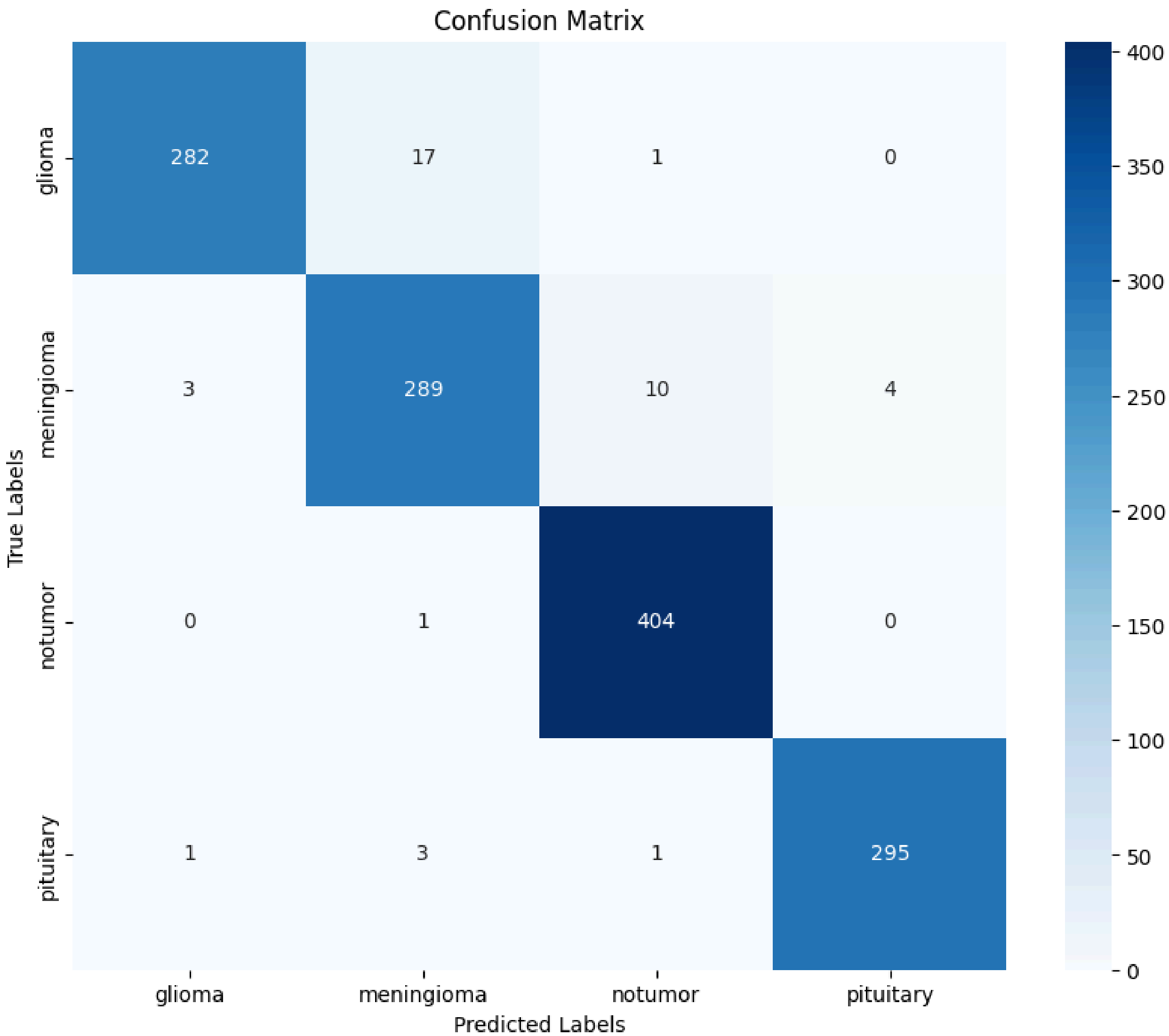
```
Epoch 12/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0668 - accuracy: 0.9796 - val_loss: 0.1579 - val_accur
Epoch 13/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0553 - accuracy: 0.9805 - val_loss: 0.4725 - val_accur
Epoch 14/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0369 - accuracy: 0.9877 - val_loss: 0.1677 - val_accur
Epoch 15/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0821 - accuracy: 0.9763 - val_loss: 0.0759 - val_accur
Epoch 16/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0350 - accuracy: 0.9874 - val_loss: 0.3689 - val_accur
Epoch 17/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0423 - accuracy: 0.9846 - val_loss: 0.0757 - val_accur
Epoch 18/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0292 - accuracy: 0.9897 - val_loss: 0.1426 - val_accur
Epoch 19/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0360 - accuracy: 0.9897 - val_loss: 0.1439 - val_accur
Epoch 20/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0250 - accuracy: 0.9912 - val_loss: 0.0807 - val_accur
Epoch 21/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0333 - accuracy: 0.9897 - val_loss: 0.0922 - val_accur
Epoch 22/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0336 - accuracy: 0.9909 - val_loss: 0.0455 - val_accur
Epoch 23/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0194 - accuracy: 0.9926 - val_loss: 0.1483 - val_accur
Epoch 24/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0287 - accuracy: 0.9914 - val_loss: 0.0588 - val_accur
Epoch 25/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0460 - accuracy: 0.9879 - val_loss: 0.3070 - val_accur
Epoch 26/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0510 - accuracy: 0.9864 - val_loss: 0.2020 - val_accur
Epoch 27/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0365 - accuracy: 0.9905 - val_loss: 0.1248 - val_accur
Epoch 28/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0314 - accuracy: 0.9928 - val_loss: 0.0587 - val_accur
Epoch 29/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0320 - accuracy: 0.9905 - val_loss: 0.0663 - val_accur
Epoch 30/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0415 - accuracy: 0.9907 - val_loss: 0.1512 - val_accur
41/41 [==============================] - 0s 9ms/step - loss: 0.1327 - accuracy: 0.9687
Test accuracy: 96.87%
```

## plot classification report confusion matrix and history graph

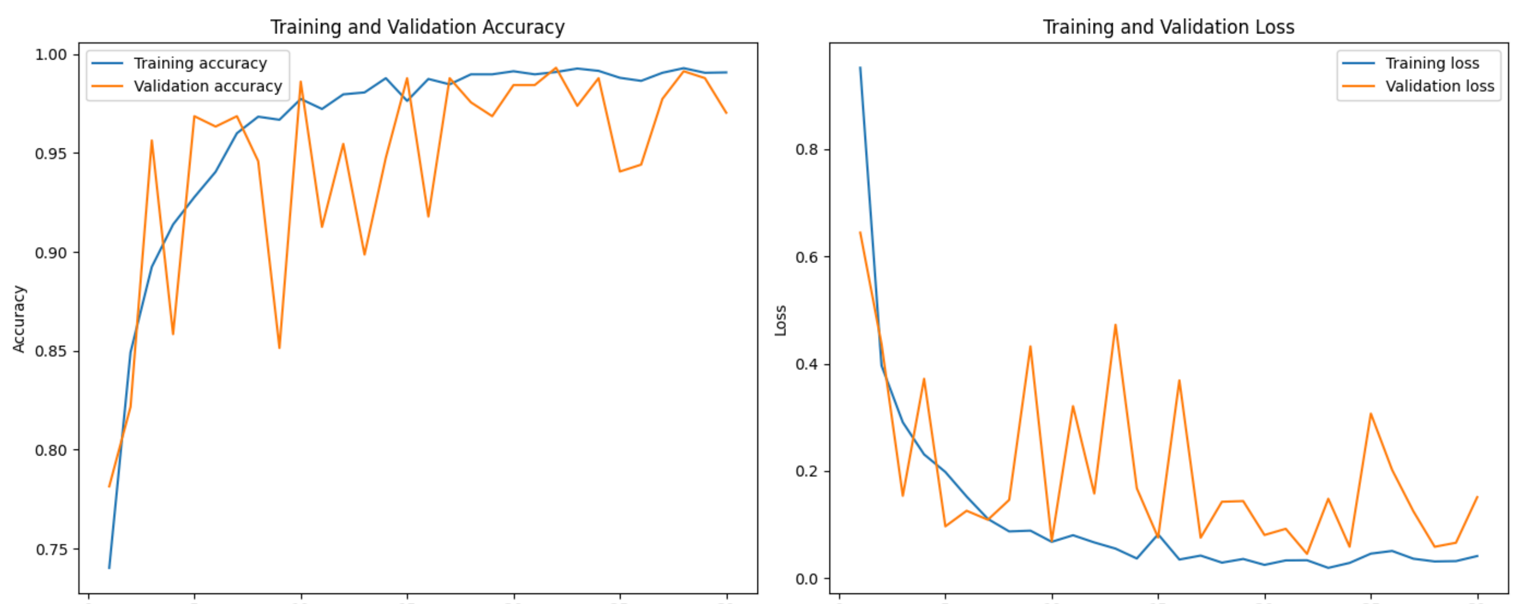
```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Predict the labels for the test data
test_predictions = simple_neural_network.predict(test_data)
test_predictions_classes = np.argmax(test_predictions, axis=1)
test_true_classes = np.argmax(test_labels, axis=1)

# Generate the classification report
report = classification_report(test_true_classes, test_predictions_classes, target_names=label_folders)
print(report)

# Compute the confusion matrix
conf_matrix = confusion_matrix(test_true_classes, test_predictions_classes)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_folders, yticklabels=label_folders)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
41/41 [==============================] - 0s 9ms/step
              precision    recall  f1-score   support

      glioma       0.99      0.94      0.96       300
  meningioma       0.93      0.94      0.94       306
     notumor       0.97      1.00      0.98       405
   pituitary       0.99      0.98      0.98       300

    accuracy                           0.97      1311
   macro avg       0.97      0.97      0.97      1311
weighted avg       0.97      0.97      0.97      1311
```



```
import matplotlib.pyplot as plt

# Extract accuracy and loss values from the training history
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

# Plot training and validation accuracy
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, accuracy, '-', label='Training accuracy')
plt.plot(epochs, val_accuracy, '-', label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, '-', label='Training loss')
plt.plot(epochs, val_loss, '-', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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




```
pip install tensorflow-addons
```

```
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.23.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (611 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 611.8/611.8 kB 6.8 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow-addons) (24.1)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow-addons)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.23.0 typeguard-2.13.3
```

# CNN Model with attention layer

## Model Creation

```

# Define the model creation function
def create_model_with_attention(input_shape, num_classes):
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(32, (3, 3), activation='relu')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(64, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D((2, 2))(x)

    # Flatten the feature maps and add attention layer
    x = layers.Flatten()(x)
    x = layers.Reshape((-1, x.shape[-1]))(x)  # Reshape for attention layer
    attention_output = layers.Attention()([x, x])
    x = layers.GlobalAveragePooling1D()(attention_output)

    # Fully connected layers
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(512, activation='relu')(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, outputs)
    return model

# Example input shape (adjust based on your image dimensions and channels)
input_shape = (128, 128, 3)

# Example number of classes (adjust based on your dataset)
num_classes = 4

# Create the model with attention layer
model_with_attention = create_model_with_attention(input_shape, num_classes)

```

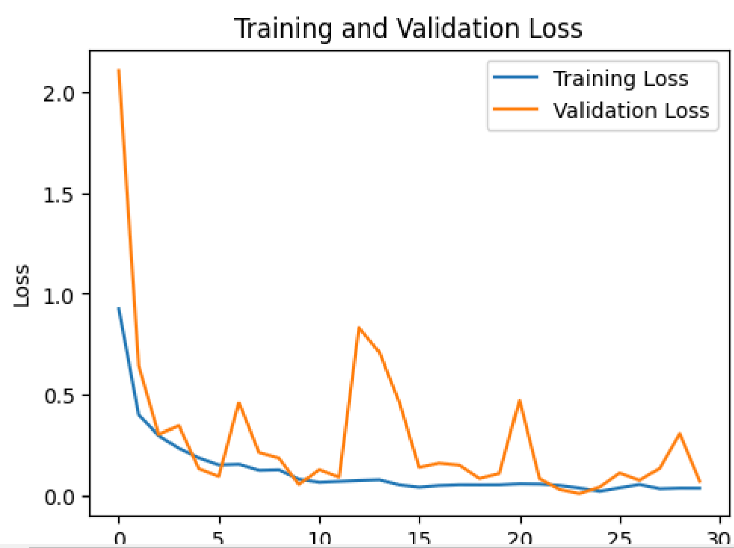
## compile fit and evalution of CNN model

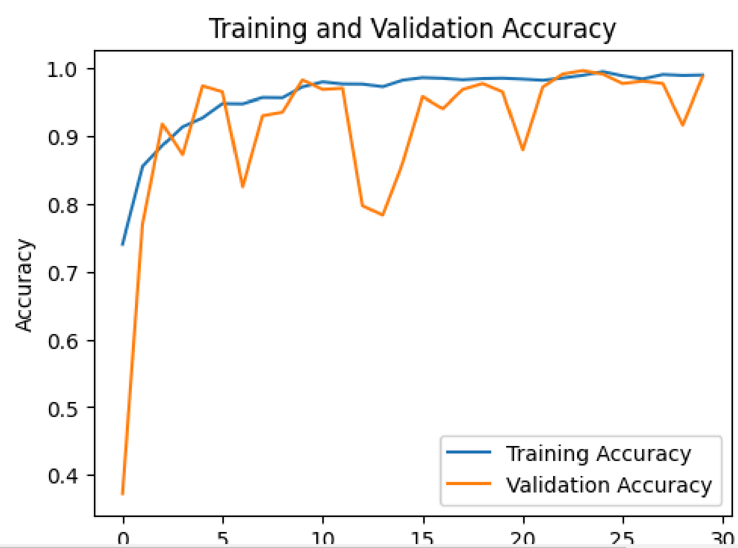
```

# Display the model summary
model_with_attention.summary()

# Compile the model
model_with_attention.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model_with_attention.fit(train_data, train_labels, epochs=30, validation_split=0.1)

# Evaluate the model performance on the testing dataset
test_loss, test_accuracy = model_with_attention.evaluate(test_data, test_labels)
```

```python
print(f"Test accuracy: {test_accuracy * 100:.2f}%")

# Plot the training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_4 (InputLayer)        [(None, 128, 128, 3)]        0         []

 conv2d_20 (Conv2D)          (None, 126, 126, 32)         896       ['input_4[0][0]']

 batch_normalization_20 (Ba  (None, 126, 126, 32)         128       ['conv2d_20[0][0]']
 tchNormalization)

 max_pooling2d_20 (MaxPooli  (None, 63, 63, 32)           0         ['batch_normalization_20[0][0]
 ng2D)                                                              ']

 conv2d_21 (Conv2D)          (None, 61, 61, 64)           18496     ['max_pooling2d_20[0][0]']

 batch_normalization_21 (Ba  (None, 61, 61, 64)           256       ['conv2d_21[0][0]']
 tchNormalization)

 max_pooling2d_21 (MaxPooli  (None, 30, 30, 64)           0         ['batch_normalization_21[0][0]
 ng2D)                                                              ']

 conv2d_22 (Conv2D)          (None, 28, 28, 128)          73856     ['max_pooling2d_21[0][0]']

 batch_normalization_22 (Ba  (None, 28, 28, 128)          512       ['conv2d_22[0][0]']
 tchNormalization)

 max_pooling2d_22 (MaxPooli  (None, 14, 14, 128)          0         ['batch_normalization_22[0][0]
 ng2D)                                                              ']

 conv2d_23 (Conv2D)          (None, 12, 12, 128)          147584    ['max_pooling2d_22[0][0]']

 batch_normalization_23 (Ba  (None, 12, 12, 128)          512       ['conv2d_23[0][0]']
 tchNormalization)

 max_pooling2d_23 (MaxPooli  (None, 6, 6, 128)            0         ['batch_normalization_23[0][0]
 ng2D)                                                              ']

 flatten_5 (Flatten)         (None, 4608)                 0         ['max_pooling2d_23[0][0]']

 reshape_2 (Reshape)         (None, 1, 4608)              0         ['flatten_5[0][0]']

 attention_2 (Attention)     (None, 1, 4608)              0         ['reshape_2[0][0]',
                                                                     'reshape_2[0][0]']

 global_average_pooling1d_2  (None, 4608)                 0         ['attention_2[0][0]']
  (GlobalAveragePooling1D)

 dropout_5 (Dropout)         (None, 4608)                 0         ['global_average_pooling1d_2[0
                                                                    ][0]']

 dense_8 (Dense)             (None, 512)                  2359808   ['dropout_5[0][0]']

 dense_9 (Dense)             (None, 4)                    2052      ['dense_8[0][0]']

==================================================================================================
Total params: 2604100 (9.93 MB)
Trainable params: 2603396 (9.93 MB)
Non-trainable params: 704 (2.75 KB)
__________________________________________________________________________________________________
Epoch 1/30
161/161 [==============================] - 7s 27ms/step - loss: 0.9269 - accuracy: 0.7407 - val_loss: 2.1102 - val_accuracy
Epoch 2/30
161/161 [==============================] - 4s 23ms/step - loss: 0.3979 - accuracy: 0.8547 - val_loss: 0.6440 - val_accuracy
Epoch 3/30
161/161 [==============================] - 4s 23ms/step - loss: 0.2958 - accuracy: 0.8860 - val_loss: 0.3019 - val_accuracy
Epoch 4/30
161/161 [==============================] - 4s 25ms/step - loss: 0.2326 - accuracy: 0.9136 - val_loss: 0.3443 - val_accuracy
Epoch 5/30
161/161 [==============================] - 4s 23ms/step - loss: 0.1847 - accuracy: 0.9268 - val_loss: 0.1305 - val_accuracy
Epoch 6/30
161/161 [==============================] - 5s 29ms/step - loss: 0.1494 - accuracy: 0.9481 - val_loss: 0.0919 - val_accuracy
Epoch 7/30
```

```
161/161 [==============================] - 5s 31ms/step - loss: 0.1525 - accuracy: 0.9477 - val_loss: 0.4571 - val_accuracy
Epoch 8/30
161/161 [==============================] - 4s 26ms/step - loss: 0.1224 - accuracy: 0.9566 - val_loss: 0.2116 - val_accuracy
Epoch 9/30
161/161 [==============================] - 4s 25ms/step - loss: 0.1247 - accuracy: 0.9562 - val_loss: 0.1838 - val_accuracy
Epoch 10/30
161/161 [==============================] - 6s 36ms/step - loss: 0.0780 - accuracy: 0.9724 - val_loss: 0.0516 - val_accuracy
Epoch 11/30
161/161 [==============================] - 5s 32ms/step - loss: 0.0626 - accuracy: 0.9798 - val_loss: 0.1257 - val_accuracy
Epoch 12/30
161/161 [==============================] - 5s 31ms/step - loss: 0.0670 - accuracy: 0.9767 - val_loss: 0.0883 - val_accuracy
Epoch 13/30
161/161 [==============================] - 5s 33ms/step - loss: 0.0715 - accuracy: 0.9765 - val_loss: 0.8318 - val_accuracy
Epoch 14/30
161/161 [==============================] - 5s 29ms/step - loss: 0.0749 - accuracy: 0.9726 - val_loss: 0.7106 - val_accuracy
Epoch 15/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0496 - accuracy: 0.9823 - val_loss: 0.4612 - val_accuracy
Epoch 16/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0378 - accuracy: 0.9860 - val_loss: 0.1374 - val_accuracy
Epoch 17/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0465 - accuracy: 0.9850 - val_loss: 0.1582 - val_accuracy
Epoch 18/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0499 - accuracy: 0.9829 - val_loss: 0.1482 - val_accuracy
Epoch 19/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0495 - accuracy: 0.9846 - val_loss: 0.0818 - val_accuracy
Epoch 20/30
161/161 [==============================] - 4s 24ms/step - loss: 0.0497 - accuracy: 0.9852 - val_loss: 0.1065 - val_accuracy
Epoch 21/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0548 - accuracy: 0.9839 - val_loss: 0.4705 - val_accuracy
Epoch 22/30
161/161 [==============================] - 4s 26ms/step - loss: 0.0536 - accuracy: 0.9821 - val_loss: 0.0810 - val_accuracy
Epoch 23/30
161/161 [==============================] - 5s 31ms/step - loss: 0.0469 - accuracy: 0.9854 - val_loss: 0.0267 - val_accuracy
Epoch 24/30
161/161 [==============================] - 5s 28ms/step - loss: 0.0336 - accuracy: 0.9895 - val_loss: 0.0059 - val_accuracy
Epoch 25/30
161/161 [==============================] - 5s 31ms/step - loss: 0.0172 - accuracy: 0.9949 - val_loss: 0.0387 - val_accuracy
Epoch 26/30
161/161 [==============================] - 5s 29ms/step - loss: 0.0346 - accuracy: 0.9887 - val_loss: 0.1090 - val_accuracy
Epoch 27/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0516 - accuracy: 0.9840 - val_loss: 0.0725 - val_accuracy
Epoch 28/30
161/161 [==============================] - 4s 23ms/step - loss: 0.0300 - accuracy: 0.9907 - val_loss: 0.1329 - val_accuracy
Epoch 29/30
161/161 [==============================] - 4s 25ms/step - loss: 0.0331 - accuracy: 0.9893 - val_loss: 0.3066 - val_accuracy
Epoch 30/30
161/161 [==============================] - 4s 27ms/step - loss: 0.0329 - accuracy: 0.9899 - val_loss: 0.0680 - val_accuracy
41/41 [==============================] - 0s 10ms/step - loss: 0.0937 - accuracy: 0.9786
Test accuracy: 97.86%
```

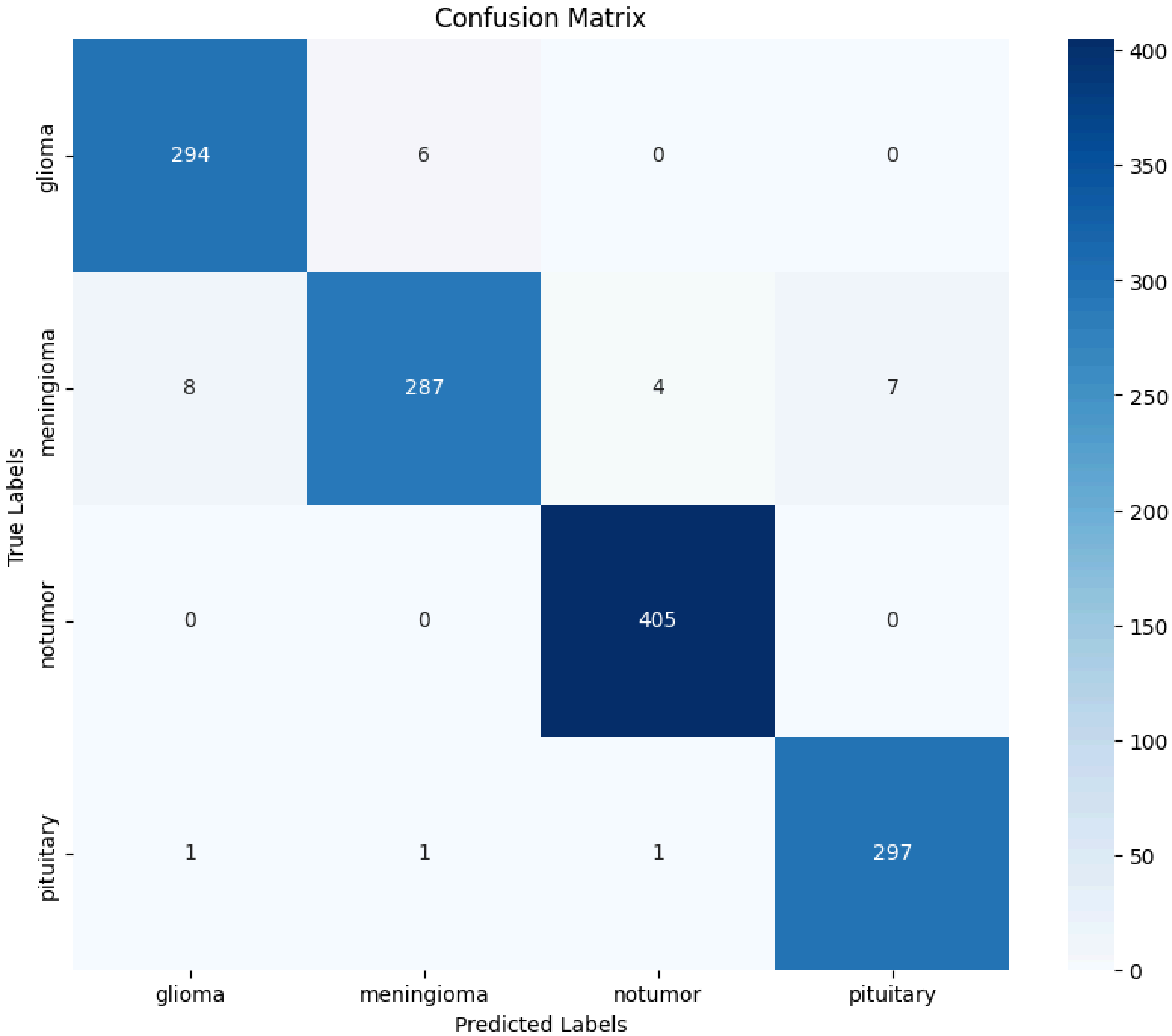


## plot classification report confusion matrix

```
# Predict the labels for the test data
test_predictions = model_with_attention.predict(test_data)
test_predictions_classes = np.argmax(test_predictions, axis=1)
test_true_classes = np.argmax(test_labels, axis=1)

# Generate the classification report
report = classification_report(test_true_classes, test_predictions_classes, target_names=label_folders)
print(report)

# Compute the confusion matrix
conf_matrix = confusion_matrix(test_true_classes, test_predictions_classes)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_folders, yticklabels=label_
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
41/41 [==============================] - 0s 6ms/step
              precision    recall  f1-score   support

      glioma       0.97      0.98      0.98       300
  meningioma       0.98      0.94      0.96       306
     notumor       0.99      1.00      0.99       405
   pituitary       0.98      0.99      0.98       300

    accuracy                           0.98      1311
   macro avg       0.98      0.98      0.98      1311
weighted avg       0.98      0.98      0.98      1311
```




## ResNet50

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.models import Model

def create_resnet50_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)

    for layer in base_model.layers:
        layer.trainable = False

    return model

resnet50_model = create_resnet50_model(input_shape, num_classes)
resnet50_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
resnet50_model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_6 (InputLayer)           [(None, 128, 128, 3)]  0         []

 conv1_pad (ZeroPadding2D)      (None, 134, 134, 3)  0           ['input_6[0][0]']

 conv1_conv (Conv2D)            (None, 64, 64, 64)   9472        ['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati     (None, 64, 64, 64)   256         ['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)        (None, 64, 64, 64)   0           ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)      (None, 66, 66, 64)   0           ['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)      (None, 32, 32, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2     (None, 32, 32, 64)   4160        ['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo     (None, 32, 32, 64)   256         ['conv2_block1_1_conv[0][0]']
 rmalization)

 conv2_block1_1_relu (Activ     (None, 32, 32, 64)   0           ['conv2_block1_1_bn[0][0]']
 ation)

 conv2_block1_2_conv (Conv2     (None, 32, 32, 64)   36928       ['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo     (None, 32, 32, 64)   256         ['conv2_block1_2_conv[0][0]']
 rmalization)

 conv2_block1_2_relu (Activ     (None, 32, 32, 64)   0           ['conv2_block1_2_bn[0][0]']
 ation)

 conv2_block1_0_conv (Conv2     (None, 32, 32, 256)  16640       ['pool1_pool[0][0]']
 D)

 conv2_block1_3_conv (Conv2     (None, 32, 32, 256)  16640       ['conv2_block1_2_relu[0][0]']
 D)

 conv2_block1_0_bn (BatchNo     (None, 32, 32, 256)  1024        ['conv2_block1_0_conv[0][0]']
 rmalization)

 conv2_block1_3_bn (BatchNo     (None, 32, 32, 256)  1024        ['conv2_block1_3_conv[0][0]']
```

```
rmalization)

conv2_block1_add (Add)      (None, 32, 32, 256)         0         ['conv2_block1_0_bn[0][0]',
                                                                   'conv2_block1_3_bn[0][0]']

conv2_block1_out (Activati  (None, 32, 32, 256)         0         ['conv2_block1_add[0][0]']
on)

conv2_block2_1_conv (Conv2  (None, 32, 32, 64)          16448     ['conv2_block1_out[0][0]']
D)
```

## Fit and Evalute the model

```python

# Train the model (replace `model_with_attention` with your selected model)
history = resnet50_model.fit(train_data, train_labels, epochs=30, validation_split=0.1, batch_size=32)

# Evaluate the model performance on the testing dataset
test_loss, test_accuracy = resnet50_model.evaluate(test_data, test_labels)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")

```

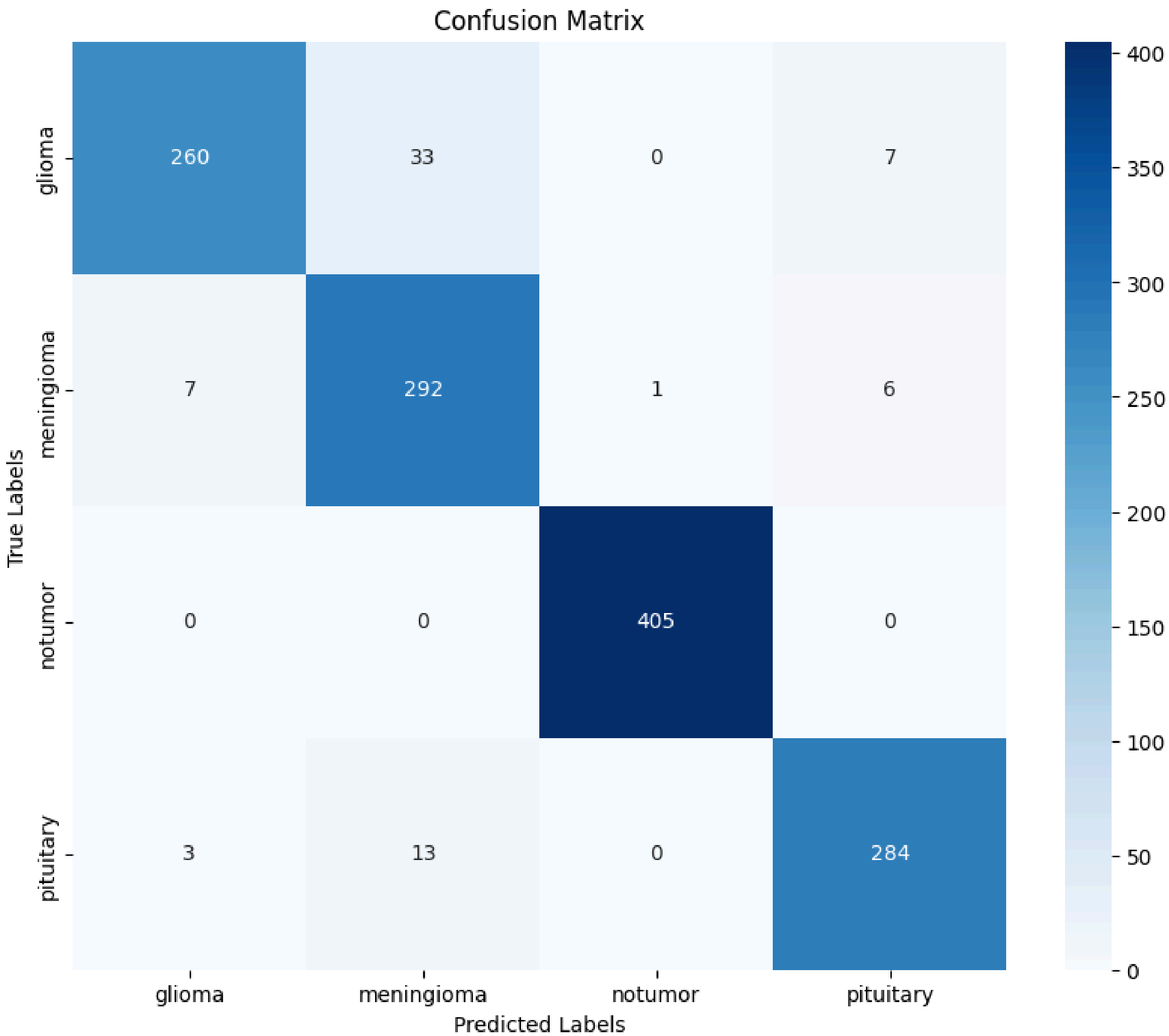
```
161/161 [==============================] - 7s 41ms/step - loss: 0.2883 - accuracy: 0.8907 - val_loss: 0.1116 - val_accur
Epoch 4/30
161/161 [==============================] - 7s 45ms/step - loss: 0.2556 - accuracy: 0.8977 - val_loss: 0.2443 - val_accur
Epoch 5/30
161/161 [==============================] - 7s 46ms/step - loss: 0.2290 - accuracy: 0.9086 - val_loss: 0.3078 - val_accur
Epoch 6/30
161/161 [==============================] - 8s 50ms/step - loss: 0.2091 - accuracy: 0.9169 - val_loss: 0.3288 - val_accur
Epoch 7/30
161/161 [==============================] - 8s 47ms/step - loss: 0.1837 - accuracy: 0.9346 - val_loss: 0.1078 - val_accur
Epoch 8/30
161/161 [==============================] - 7s 46ms/step - loss: 0.1849 - accuracy: 0.9282 - val_loss: 0.1689 - val_accur
Epoch 9/30
161/161 [==============================] - 7s 45ms/step - loss: 0.1637 - accuracy: 0.9377 - val_loss: 0.1476 - val_accur
Epoch 10/30
161/161 [==============================] - 7s 42ms/step - loss: 0.1476 - accuracy: 0.9451 - val_loss: 0.1763 - val_accur
Epoch 11/30
161/161 [==============================] - 6s 39ms/step - loss: 0.1475 - accuracy: 0.9418 - val_loss: 0.0779 - val_accur
Epoch 12/30
161/161 [==============================] - 6s 40ms/step - loss: 0.1456 - accuracy: 0.9416 - val_loss: 0.1084 - val_accur
Epoch 13/30
161/161 [==============================] - 7s 41ms/step - loss: 0.1287 - accuracy: 0.9498 - val_loss: 0.1123 - val_accur
Epoch 14/30
161/161 [==============================] - 7s 41ms/step - loss: 0.1255 - accuracy: 0.9521 - val_loss: 0.0694 - val_accur
Epoch 15/30
161/161 [==============================] - 7s 43ms/step - loss: 0.1204 - accuracy: 0.9564 - val_loss: 0.2073 - val_accur
Epoch 16/30
161/161 [==============================] - 7s 40ms/step - loss: 0.1064 - accuracy: 0.9595 - val_loss: 0.0883 - val_accur
Epoch 17/30
161/161 [==============================] - 6s 39ms/step - loss: 0.1072 - accuracy: 0.9599 - val_loss: 0.0595 - val_accur
Epoch 18/30
161/161 [==============================] - 6s 40ms/step - loss: 0.0906 - accuracy: 0.9652 - val_loss: 0.0730 - val_accur
Epoch 19/30
161/161 [==============================] - 7s 40ms/step - loss: 0.0928 - accuracy: 0.9667 - val_loss: 0.0943 - val_accur
Epoch 20/30
161/161 [==============================] - 7s 41ms/step - loss: 0.0968 - accuracy: 0.9599 - val_loss: 0.0579 - val_accur
Epoch 21/30
161/161 [==============================] - 6s 39ms/step - loss: 0.0944 - accuracy: 0.9611 - val_loss: 0.0667 - val_accur
Epoch 22/30
161/161 [==============================] - 6s 40ms/step - loss: 0.0776 - accuracy: 0.9700 - val_loss: 0.0786 - val_accur
Epoch 23/30
161/161 [==============================] - 6s 40ms/step - loss: 0.1067 - accuracy: 0.9586 - val_loss: 0.1182 - val_accur
```

```
Epoch 27/30
161/161 [==============================] - 6s 40ms/step - loss: 0.0685 - accuracy: 0.9739 - val_loss: 0.0372 - val_accur
Epoch 28/30
161/161 [==============================] - 7s 41ms/step - loss: 0.0813 - accuracy: 0.9687 - val_loss: 0.0738 - val_accur
Epoch 29/30
161/161 [==============================] - 6s 39ms/step - loss: 0.0823 - accuracy: 0.9698 - val_loss: 0.1266 - val_accur
Epoch 30/30
161/161 [==============================] - 7s 41ms/step - loss: 0.0742 - accuracy: 0.9726 - val_loss: 0.1959 - val_accur
41/41 [==============================] - 3s 85ms/step - loss: 0.1533 - accuracy: 0.9466
Test accuracy: 94.66%
```

## PLot the classiifcation report and confusion matrix

```
# Predict the labels for the test data
test_predictions = resnet50_model.predict(test_data)
test_predictions_classes = np.argmax(test_predictions, axis=1)
test_true_classes = np.argmax(test_labels, axis=1)

# Generate the classification report
report = classification_report(test_true_classes, test_predictions_classes, target_names=label_folders)
print(report)

# Compute the confusion matrix
conf_matrix = confusion_matrix(test_true_classes, test_predictions_classes)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_folders, yticklabels=label_
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

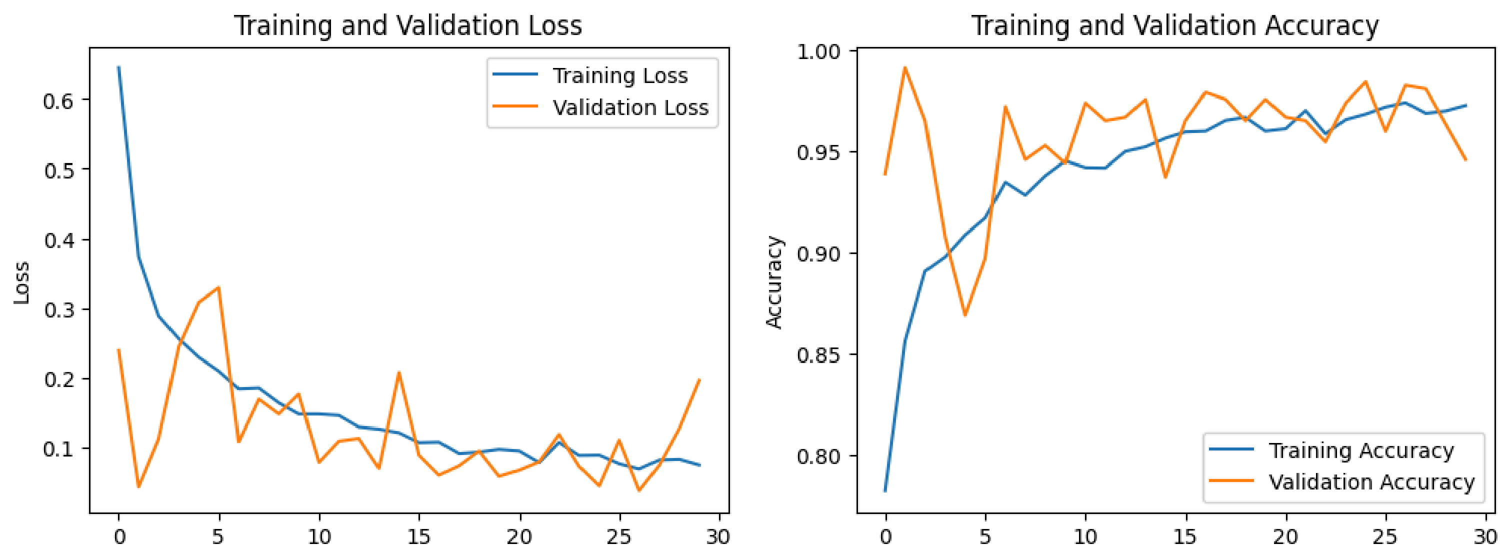
```
41/41 [==============================] - 4s 35ms/step
              precision    recall  f1-score   support

      glioma       0.96      0.87      0.91       300
  meningioma       0.86      0.95      0.91       306
     notumor       1.00      1.00      1.00       405
   pituitary       0.96      0.95      0.95       300

    accuracy                           0.95      1311
   macro avg       0.95      0.94      0.94      1311
weighted avg       0.95      0.95      0.95      1311
```




**Plot the Histoy graph**

```
# Plot the training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

```python
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```




## MobileNetV2

```python
from tensorflow.keras.applications import MobileNetV2

def create_mobilenetv2_model(input_shape, num_classes):
    base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=input_shape)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)

    for layer in base_model.layers:
        layer.trainable = False

    return model

mobilenetv2_model = create_mobilenetv2_model(input_shape, num_classes)
mobilenetv2_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
mobilenetv2_model.summary()
```

```
 block_1_project_BN (BatchN  (None, 32, 32, 24)          96        ['block_1_project[0][0]']
 ormalization)

 block_2_expand (Conv2D)     (None, 32, 32, 144)         3456      ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNo  (None, 32, 32, 144)         576       ['block_2_expand[0][0]']
 rmalization)

 block_2_expand_relu (ReLU)  (None, 32, 32, 144)         0         ['block_2_expand_BN[0][0]']

 block_2_depthwise (Depthwi  (None, 32, 32, 144)         1296      ['block_2_expand_relu[0][0]']
 seConv2D)

 block_2_depthwise_BN (Batc  (None, 32, 32, 144)         576       ['block_2_depthwise[0][0]']
 hNormalization)

 block_2_depthwise_relu (Re  (None, 32, 32, 144)         0         ['block_2_depthwise_BN[0][0]']
 LU)

 block_2_project (Conv2D)    (None, 32, 32, 24)          3456      ['block_2_depthwise_relu[0][0]
                                                                   ']

 block_2_project_BN (BatchN  (None, 32, 32, 24)          96        ['block_2_project[0][0]']
 ormalization)

 block_2_add (Add)           (None, 32, 32, 24)          0         ['block_1_project_BN[0][0]',
                                                                    'block_2_project_BN[0][0]']

 block_3_expand (Conv2D)     (None, 32, 32, 144)         3456      ['block_2_add[0][0]']

 block_3_expand_BN (BatchNo  (None, 32, 32, 144)         576       ['block_3_expand[0][0]']
 rmalization)

 block_3_expand_relu (ReLU)  (None, 32, 32, 144)         0         ['block_3_expand_BN[0][0]']

 block_3_pad (ZeroPadding2D  (None, 33, 33, 144)         0         ['block_3_expand_relu[0][0]']
 )

 block_3_depthwise (Depthwi  (None, 16, 16, 144)         1296      ['block_3_pad[0][0]']
 seConv2D)
```

## Fit and Evalute the model

```python

# Train the model (replace `model_with_attention` with your selected model)
history = mobilenetv2_model.fit(train_data, train_labels, epochs=30, validation_split=0.1, batch_size=32)

# Evaluate the model performance on the testing dataset
test_loss, test_accuracy = mobilenetv2_model.evaluate(test_data, test_labels)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")

```

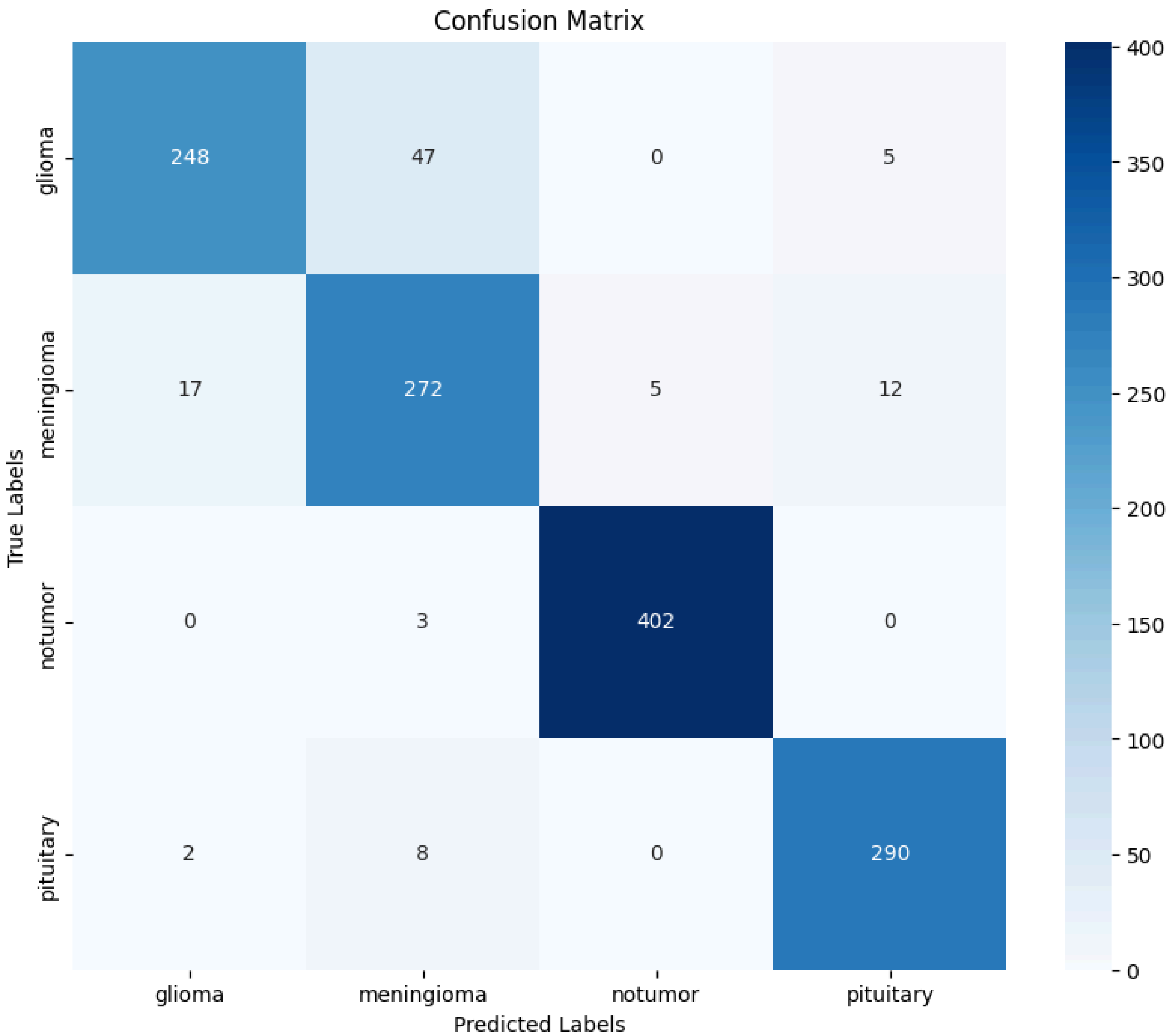
```
Epoch 1/30
161/161 [==============================] - 9s 31ms/step - loss: 0.6287 - accuracy: 0.7691 - val_loss: 0.2680 - val_accur
Epoch 2/30
161/161 [==============================] - 3s 20ms/step - loss: 0.4103 - accuracy: 0.8432 - val_loss: 0.1893 - val_accur
Epoch 3/30
161/161 [==============================] - 3s 18ms/step - loss: 0.3637 - accuracy: 0.8625 - val_loss: 0.2912 - val_accur
Epoch 4/30
161/161 [==============================] - 3s 18ms/step - loss: 0.3283 - accuracy: 0.8800 - val_loss: 0.3318 - val_accur
Epoch 5/30
161/161 [==============================] - 3s 17ms/step - loss: 0.2999 - accuracy: 0.8899 - val_loss: 0.2798 - val_accur
Epoch 6/30
161/161 [==============================] - 3s 19ms/step - loss: 0.2847 - accuracy: 0.8946 - val_loss: 0.3211 - val_accur
Epoch 7/30
```

```
161/161 [==============================] - 3s 18ms/step - loss: 0.2700 - accuracy: 0.8977 - val_loss: 0.2461 - val_accur
Epoch 8/30
161/161 [==============================] - 3s 18ms/step - loss: 0.2562 - accuracy: 0.9004 - val_loss: 0.1547 - val_accur
Epoch 9/30
161/161 [==============================] - 3s 17ms/step - loss: 0.2291 - accuracy: 0.9089 - val_loss: 0.4886 - val_accur
Epoch 10/30
161/161 [==============================] - 3s 19ms/step - loss: 0.2226 - accuracy: 0.9179 - val_loss: 0.4252 - val_accur
Epoch 11/30
161/161 [==============================] - 3s 21ms/step - loss: 0.2176 - accuracy: 0.9189 - val_loss: 0.1300 - val_accur
Epoch 12/30
161/161 [==============================] - 3s 18ms/step - loss: 0.2065 - accuracy: 0.9208 - val_loss: 0.1618 - val_accur
Epoch 13/30
161/161 [==============================] - 3s 18ms/step - loss: 0.1968 - accuracy: 0.9233 - val_loss: 0.2546 - val_accur
Epoch 14/30
161/161 [==============================] - 3s 19ms/step - loss: 0.2012 - accuracy: 0.9235 - val_loss: 0.0977 - val_accur
Epoch 15/30
161/161 [==============================] - 3s 21ms/step - loss: 0.1848 - accuracy: 0.9315 - val_loss: 0.2084 - val_accur
Epoch 16/30
161/161 [==============================] - 4s 24ms/step - loss: 0.1810 - accuracy: 0.9317 - val_loss: 0.1494 - val_accur
Epoch 17/30
161/161 [==============================] - 3s 22ms/step - loss: 0.1811 - accuracy: 0.9292 - val_loss: 0.1647 - val_accur
Epoch 18/30
161/161 [==============================] - 3s 18ms/step - loss: 0.1634 - accuracy: 0.9375 - val_loss: 0.0439 - val_accur
Epoch 19/30
161/161 [==============================] - 3s 19ms/step - loss: 0.1705 - accuracy: 0.9339 - val_loss: 0.1238 - val_accur
Epoch 20/30
161/161 [==============================] - 3s 19ms/step - loss: 0.1519 - accuracy: 0.9412 - val_loss: 0.1074 - val_accur
Epoch 21/30
161/161 [==============================] - 3s 18ms/step - loss: 0.1518 - accuracy: 0.9420 - val_loss: 0.1217 - val_accur
Epoch 22/30
161/161 [==============================] - 4s 22ms/step - loss: 0.1548 - accuracy: 0.9407 - val_loss: 0.1130 - val_accur
Epoch 23/30
161/161 [==============================] - 4s 23ms/step - loss: 0.1469 - accuracy: 0.9469 - val_loss: 0.1537 - val_accur
Epoch 24/30
161/161 [==============================] - 5s 28ms/step - loss: 0.1340 - accuracy: 0.9473 - val_loss: 0.1382 - val_accur
Epoch 25/30
161/161 [==============================] - 4s 25ms/step - loss: 0.1309 - accuracy: 0.9496 - val_loss: 0.1661 - val_accur
Epoch 26/30
161/161 [==============================] - 4s 22ms/step - loss: 0.1264 - accuracy: 0.9500 - val_loss: 0.2163 - val_accur
Epoch 27/30
161/161 [==============================] - 5s 29ms/step - loss: 0.1277 - accuracy: 0.9514 - val_loss: 0.0737 - val_accur
Epoch 28/30
161/161 [==============================] - 4s 22ms/step - loss: 0.1143 - accuracy: 0.9549 - val_loss: 0.2091 - val_accur
```

## plot classification report and Confusion Matrix

```
# Predict the labels for the test data
test_predictions = mobilenetv2_model.predict(test_data)
test_predictions_classes = np.argmax(test_predictions, axis=1)
test_true_classes = np.argmax(test_labels, axis=1)

# Generate the classification report
report = classification_report(test_true_classes, test_predictions_classes, target_names=label_folders)
print(report)

# Compute the confusion matrix
conf_matrix = confusion_matrix(test_true_classes, test_predictions_classes)

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_folders, yticklabels=label_folders)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
41/41 [==============================] - 2s 16ms/step
              precision    recall  f1-score   support

      glioma       0.93      0.83      0.87       300
  meningioma       0.82      0.89      0.86       306
     notumor       0.99      0.99      0.99       405
   pituitary       0.94      0.97      0.96       300

    accuracy                           0.92      1311
   macro avg       0.92      0.92      0.92      1311
weighted avg       0.93      0.92      0.92      1311
```



```python
# Plot the training history
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.show()
```


Start coding or generate with AI.

## DenseNet121

```python
from tensorflow.keras.applications import DenseNet121

def create_densenet121_model(input_shape, num_classes):
    base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=input_shape)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)

    for layer in base_model.layers:
        layer.trainable = False

    return model

densenet121_model = create_densenet121_model(input_shape, num_classes)
densenet121_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
densenet121_model.summary()
```