bt

August 2, 2023

Brain tumor classification plays a critical role in medical diagnosis and treatment planning. With the advent of ***deep learning, transfer learning*** has emerged as a powerful technique for leveraging pre-trained models on large datasets to solve new tasks with limited labeled data. In this study, we employ the **VGG16** architecture as the foundation for our brain tumor classification model. VGG16 is a well-known convolutional neural network (CNN) model that has been pre-trained on the ImageNet dataset, making it highly adept at extracting meaningful features from images. By fine-tuning the VGG16 model on a specialized brain tumor dataset, we aim to harness its exceptional ability to discern intricate patterns and discern between different tumor types. Through this research, we seek to contribute to the development of accurate and efficient brain tumor classification models that can potentially enhance diagnostic accuracy and streamline treatment decisions.

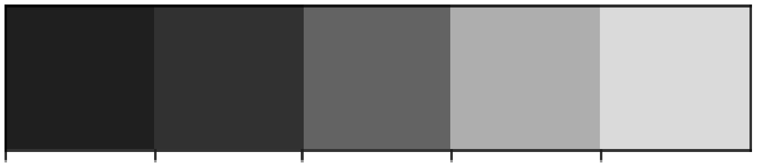
IMPORTING NECESSARY LIBRARIES

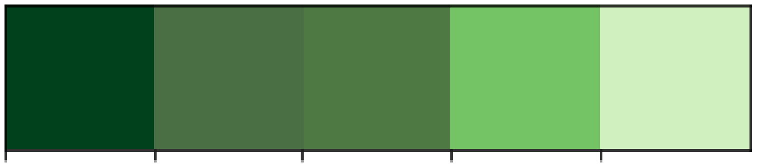
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tqdm import tqdm
import os
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
 ↪TensorBoard, ModelCheckpoint
from sklearn.metrics import classification_report,confusion_matrix
import ipywidgets as widgets
import io
from PIL import Image
from IPython.display import display,clear_output
```

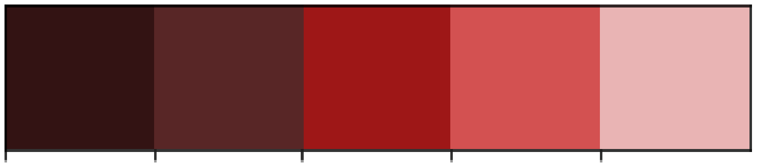
SOME VISUALS

[2]:
```
colors_dark = ["#1F1F1F", "#313131", '#636363', '#AEAEAE', '#DADADA']
colors_red = ["#331313", "#582626", '#9E1717', '#D35151', '#E9B4B4']
colors_green = ['#01411C','#4B6F44','#4F7942','#74C365','#D0F0C0']
```

```
sns.palplot(colors_dark)
sns.palplot(colors_green)
sns.palplot(colors_red)
```







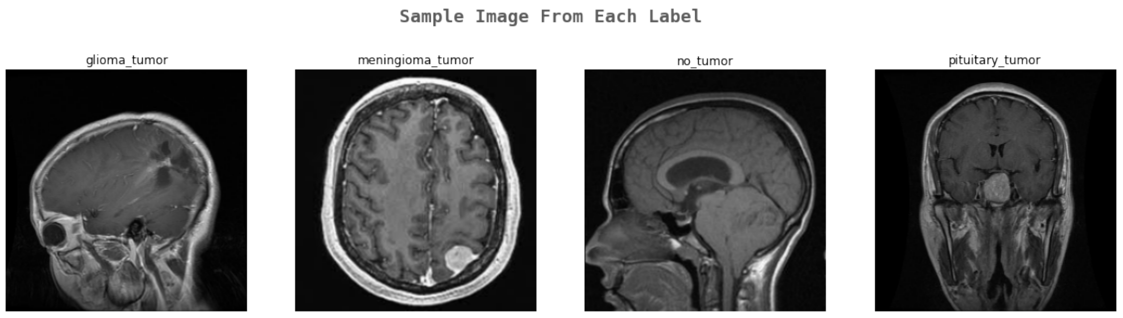
DATA PREPARATION & PREPROCESSING

[3]:
```
X_train = []
Y_train = []
image_size = 224
labels = ['glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor']
for i in labels:
    folderPath = os.path.join('/kaggle/input/
 ↪brain-tumor-classification-mri','Training',i)
    for j in tqdm(os.listdir(folderPath)):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size, image_size))
        X_train.append(img)
        Y_train.append(i)

for i in labels:
    folderPath = os.path.join('/kaggle/input/
 ↪brain-tumor-classification-mri','Testing',i)
```

```
    for j in tqdm(os.listdir(folderPath)):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size,image_size))
        X_train.append(img)
        Y_train.append(i)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
```

```
100%|      | 826/826 [00:06<00:00, 131.62it/s]
100%|      | 822/822 [00:06<00:00, 126.70it/s]
100%|      | 395/395 [00:02<00:00, 144.77it/s]
100%|      | 827/827 [00:08<00:00, 100.04it/s]
100%|      | 100/100 [00:00<00:00, 130.18it/s]
100%|      | 115/115 [00:00<00:00, 157.93it/s]
100%|      | 105/105 [00:00<00:00, 218.71it/s]
100%|      | 74/74 [00:00<00:00, 83.80it/s]
```

[4]:
```
k=0
fig, ax = plt.subplots(1,4,figsize=(20,20))
fig.text(s='Sample Image From Each Label',size=18,fontweight='bold',
             fontname='monospace',color=colors_dark[1],y=0.62,x=0.4,alpha=0.8)
for i in labels:
    j=0
    while True :
        if Y_train[j]==i:
            ax[k].imshow(X_train[j])
            ax[k].set_title(Y_train[j])
            ax[k].axis('off')
            k+=1
            break
        j+=1
```



SHUFFLING & SPLITTING THE DATA INTO TRAINING AND TESTING SETS

```
[5]: X_train,Y_train = shuffle(X_train,Y_train,random_state=14)
     X_train.shape
```

```
[5]: (3264, 224, 224, 3)
```

```
[6]: X_train,X_test,y_train,y_test = train_test_split(X_train,Y_train,test_size=0.
      ↪2,random_state=14)
```

PERFORMING ONE HOT ENCODING : This helps in converting labes into Numerical Values

BELOW IS THE PSEUDOCODE FOR ANY GENERAL ONE HOT ENCODING ALOGRITHM

```
function oneHotEncode(data):
    unique_values = findUniqueValues(data)
    encoded_data = createEmptyMatrix(len(data), len(unique_values))

    for i = 0 to len(data) - 1:
            value = data[i]

   for j = 0 to len(unique_values) - 1:
            if unique_values[j] == value:
                    encoded_data[i][j] = 1
            else:
                encoded_data[i][j] = 0

   return encoded_data
```

```
[7]: y_train_new = []
     for i in y_train:
         y_train_new.append(labels.index(i))
     y_train=y_train_new
     y_train = tf.keras.utils.to_categorical(y_train)

     y_test_new = []
     for i in y_test:
         y_test_new.append(labels.index(i))
     y_test=y_test_new
     y_test = tf.keras.utils.to_categorical(y_test)
```

TRANSFER LEARNING

Deep convolutional neural network models often require a significant amount of time to train, especially on large datasets. To expedite this process, a common approach is to leverage pre-trained models that have been developed and fine-tuned on standard computer vision benchmark datasets, such as the ImageNet image recognition tasks. By reusing the weights from these pre-trained models, one can benefit from their learned representations and accelerate the training process for their own computer vision problems.

In this notebook, I will utilize the VGG16 model, which comes with pre-trained weights from the ImageNet dataset. By setting the include_top parameter to False, we exclude the top layer or output layer of the pre-built model. This allows us to add our own output layer, tailored to our specific use case.

BELOW IS THE PSEUDOCODE

```
pretrained_model = load_pretrained_model()
pretrained_model.last_layer = new_layer()

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

transfer_model = create_model(pretrained_model)
transfer_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

new_dataset = load_new_dataset()
preprocessed_dataset = preprocess_dataset(new_dataset)

num_epochs = 10
batch_size = 32

for epoch in range(num_epochs):
    for batch in range(0, len(preprocessed_dataset), batch_size):
        batch_data = preprocessed_dataset[batch:batch+batch_size]
        transfer_model.train_on_batch(batch_data)

test_dataset = load_test_dataset()
preprocessed_test_dataset = preprocess_dataset(test_dataset)
loss, accuracy = transfer_model.evaluate(preprocessed_test_dataset)

save_model(transfer_model)
```

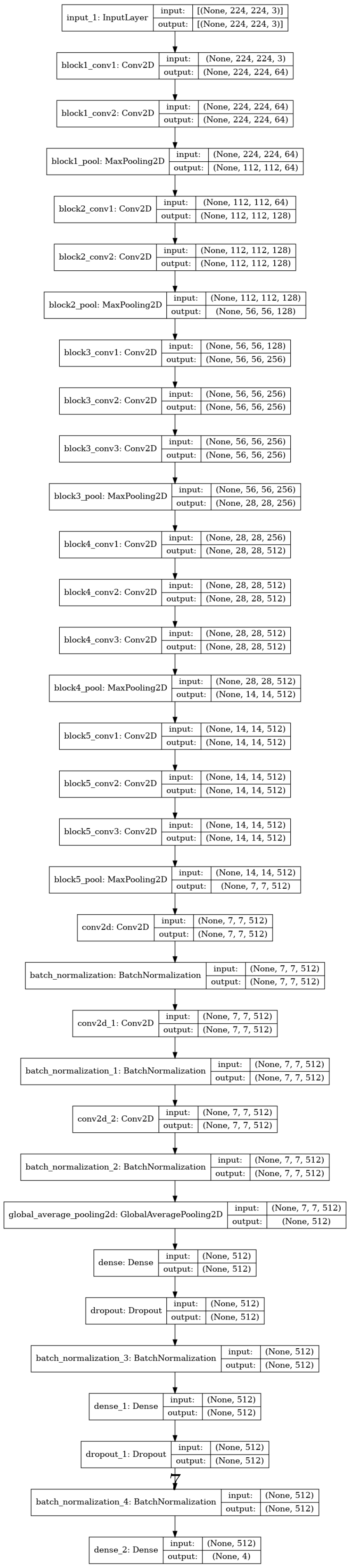
VISUAL GEOMETRY GROUP (VGG16) BASE MODEL ARCHITECTURE

[8]:
```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,␣
 ↪GlobalAveragePooling2D, Dropout, Dense, BatchNormalization

# Load VGG16 with pre-trained weights and without the top classification layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

# Freeze the VGG16 layers
for layer in base_model.layers:
    layer.trainable = False

# Adding our custom classification layers on top of VGG16
```

```
x = base_model.output
x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)

x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)

x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)

x = GlobalAveragePooling2D()(x)

x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)

x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)


output = Dense(4, activation='softmax')(x)

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

[9]:
```
from tensorflow.keras.utils import plot_model


plot_model(model, to_file='model_architecture.png', show_shapes=True,␣
  ↪show_layer_names=True)
```

[9]:


input_1: InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
block1_conv1: Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 64)
block1_conv2: Conv2D | input: (None, 224, 224, 64) | output: (None, 224, 224, 64)
block1_pool: MaxPooling2D | input: (None, 224, 224, 64) | output: (None, 112, 112, 64)
block2_conv1: Conv2D | input: (None, 112, 112, 64) | output: (None, 112, 112, 128)
block2_conv2: Conv2D | input: (None, 112, 112, 128) | output: (None, 112, 112, 128)
block2_pool: MaxPooling2D | input: (None, 112, 112, 128) | output: (None, 56, 56, 128)
block3_conv1: Conv2D | input: (None, 56, 56, 128) | output: (None, 56, 56, 256)
block3_conv2: Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_conv3: Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
block3_pool: MaxPooling2D | input: (None, 56, 56, 256) | output: (None, 28, 28, 256)
block4_conv1: Conv2D | input: (None, 28, 28, 256) | output: (None, 28, 28, 512)
block4_conv2: Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_conv3: Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
block4_pool: MaxPooling2D | input: (None, 28, 28, 512) | output: (None, 14, 14, 512)
block5_conv1: Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv2: Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_conv3: Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
block5_pool: MaxPooling2D | input: (None, 14, 14, 512) | output: (None, 7, 7, 512)
conv2d: Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
batch_normalization: BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
conv2d_1: Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
batch_normalization_1: BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
conv2d_2: Conv2D | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
batch_normalization_2: BatchNormalization | input: (None, 7, 7, 512) | output: (None, 7, 7, 512)
global_average_pooling2d: GlobalAveragePooling2D | input: (None, 7, 7, 512) | output: (None, 512)
dense: Dense | input: (None, 512) | output: (None, 512)
dropout: Dropout | input: (None, 512) | output: (None, 512)
batch_normalization_3: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_1: Dense | input: (None, 512) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
batch_normalization_4: BatchNormalization | input: (None, 512) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 4)

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

Callbacks -> Callbacks can help you fix bugs more quickly, and can help you build better models. They can help you visualize how your model's training is going, and can even help prevent overfitting by implementing early stopping or customizing the learning rate on each iteration.

By definition, "A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training."

In this notebook, We'll be using **TensorBoard, ModelCheckpoint and ReduceLROnPlateau** callback functions

```
[11]: tensorboard = TensorBoard(log_dir = 'logs')
      checkpoint = ModelCheckpoint("vgg16.
        ↪h5",monitor="val_accuracy",save_best_only=True,mode="auto",verbose=1)
      reduce_lr = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.3, patience␣
        ↪= 2, min_delta = 0.001,
                                    mode='auto',verbose=1)
```

```
[12]: history = model.fit(X_train,y_train,validation_split=0.2, epochs =15,␣
        ↪verbose=1, batch_size=64,
                         callbacks=[tensorboard,checkpoint,reduce_lr])
```

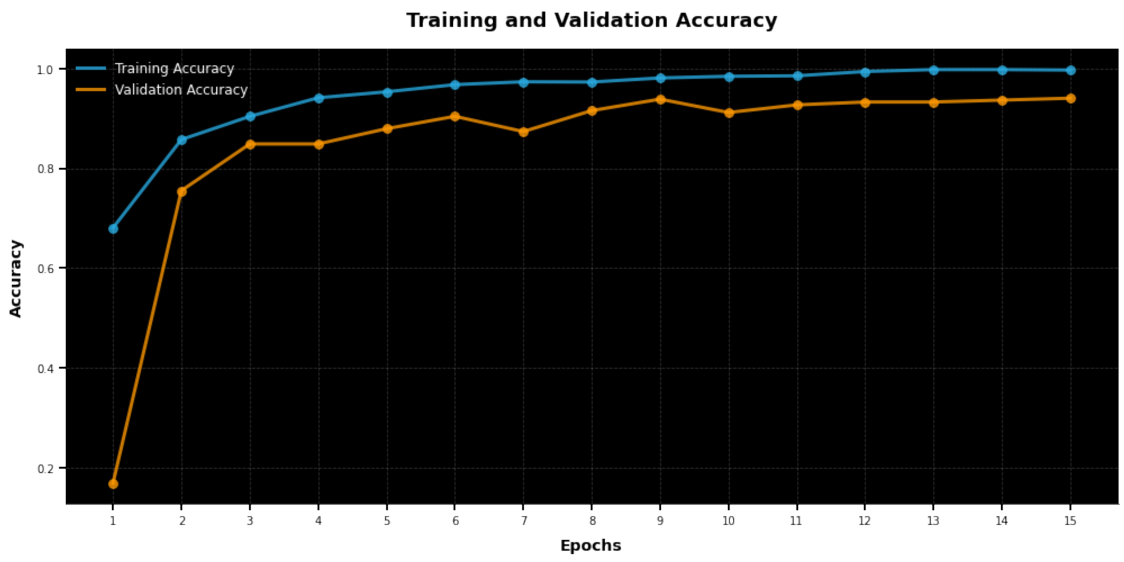
```
Epoch 1/15
33/33 [==============================] - 38s 690ms/step - loss: 0.8957 -
accuracy: 0.6806 - val_loss: 21.9285 - val_accuracy: 0.1683

Epoch 00001: val_accuracy improved from -inf to 0.16826, saving model to
vgg16.h5
Epoch 2/15
33/33 [==============================] - 12s 369ms/step - loss: 0.4277 -
accuracy: 0.8578 - val_loss: 1.4708 - val_accuracy: 0.7553

Epoch 00002: val_accuracy improved from 0.16826 to 0.75526, saving model to
vgg16.h5
Epoch 3/15
33/33 [==============================] - 12s 377ms/step - loss: 0.2735 -
accuracy: 0.9042 - val_loss: 0.6725 - val_accuracy: 0.8489

Epoch 00003: val_accuracy improved from 0.75526 to 0.84895, saving model to
vgg16.h5
Epoch 4/15
33/33 [==============================] - 12s 368ms/step - loss: 0.1690 -
accuracy: 0.9416 - val_loss: 0.8311 - val_accuracy: 0.8489
```

```
Epoch 00004: val_accuracy did not improve from 0.84895
Epoch 5/15
33/33 [==============================] - 12s 353ms/step - loss: 0.1313 -
accuracy: 0.9535 - val_loss: 0.5843 - val_accuracy: 0.8795

Epoch 00005: val_accuracy improved from 0.84895 to 0.87954, saving model to
vgg16.h5
Epoch 6/15
33/33 [==============================] - 11s 349ms/step - loss: 0.0996 -
accuracy: 0.9679 - val_loss: 0.4517 - val_accuracy: 0.9044

Epoch 00006: val_accuracy improved from 0.87954 to 0.90440, saving model to
vgg16.h5
Epoch 7/15
33/33 [==============================] - 12s 359ms/step - loss: 0.0763 -
accuracy: 0.9737 - val_loss: 0.6184 - val_accuracy: 0.8738

Epoch 00007: val_accuracy did not improve from 0.90440
Epoch 8/15
33/33 [==============================] - 12s 362ms/step - loss: 0.0751 -
accuracy: 0.9732 - val_loss: 0.4336 - val_accuracy: 0.9159

Epoch 00008: val_accuracy improved from 0.90440 to 0.91587, saving model to
vgg16.h5
Epoch 9/15
33/33 [==============================] - 12s 365ms/step - loss: 0.0614 -
accuracy: 0.9813 - val_loss: 0.3808 - val_accuracy: 0.9388

Epoch 00009: val_accuracy improved from 0.91587 to 0.93881, saving model to
vgg16.h5
Epoch 10/15
33/33 [==============================] - 12s 359ms/step - loss: 0.0509 -
accuracy: 0.9847 - val_loss: 0.3998 - val_accuracy: 0.9120

Epoch 00010: val_accuracy did not improve from 0.93881
Epoch 11/15
33/33 [==============================] - 12s 359ms/step - loss: 0.0446 -
accuracy: 0.9856 - val_loss: 0.3726 - val_accuracy: 0.9273

Epoch 00011: val_accuracy did not improve from 0.93881

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
Epoch 12/15
33/33 [==============================] - 12s 366ms/step - loss: 0.0163 -
accuracy: 0.9943 - val_loss: 0.3310 - val_accuracy: 0.9331

Epoch 00012: val_accuracy did not improve from 0.93881
Epoch 13/15
```

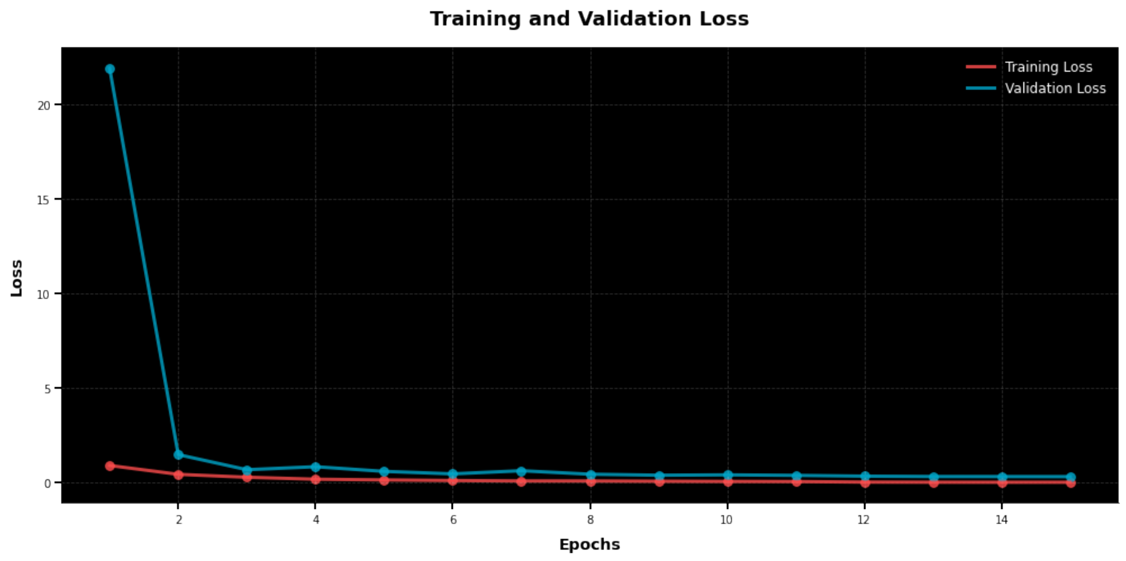
```
33/33 [==============================] - 12s 355ms/step - loss: 0.0100 -
accuracy: 0.9981 - val_loss: 0.3122 - val_accuracy: 0.9331

Epoch 00013: val_accuracy did not improve from 0.93881

Epoch 00013: ReduceLROnPlateau reducing learning rate to 9.000000427477062e-05.
Epoch 14/15
33/33 [==============================] - 12s 363ms/step - loss: 0.0079 -
accuracy: 0.9981 - val_loss: 0.3095 - val_accuracy: 0.9369

Epoch 00014: val_accuracy did not improve from 0.93881
Epoch 15/15
33/33 [==============================] - 12s 353ms/step - loss: 0.0084 -
accuracy: 0.9971 - val_loss: 0.3053 - val_accuracy: 0.9407

Epoch 00015: val_accuracy improved from 0.93881 to 0.94073, saving model to
vgg16.h5
```

[13]:
```python
model.save('braintumor.h5')

import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)

fig, ax = plt.subplots(figsize=(14, 7))
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(True)
ax.spines['left'].set_visible(True)
ax.tick_params(axis='both', which='both', length=6, width=2)
ax.set_facecolor('#000000')

ax.plot(epochs, acc, linewidth=3, label="Training Accuracy", color='#29abe2',␣
 ↪alpha=0.8)
ax.scatter(epochs, acc, s=60, color='#29abe2', alpha=0.8)

ax.plot(epochs, val_acc, linewidth=3, label="Validation Accuracy",␣
 ↪color='#ff9900', alpha=0.8)
ax.scatter(epochs, val_acc, s=60, color='#ff9900', alpha=0.8)

ax.set_xlabel('Epochs', fontsize=14, labelpad=10, color='#000000',␣
 ↪fontweight='bold')
ax.set_ylabel('Accuracy', fontsize=14, labelpad=10, color='#000000',␣
 ↪fontweight='bold')
```

```
ax.set_title('Training and Validation Accuracy', fontsize=18, pad=20,␣
↪color='#000000', fontweight='bold')

legend = ax.legend(loc='upper left', fontsize=12, frameon=False)
legend.set_title(None)
for handle in legend.legendHandles:
    handle.set_markersize(50)
legend.get_frame().set_facecolor('none')
for text in legend.get_texts():
    text.set_color('#ffffff')

ax.grid(color='#ffffff', linestyle='--', alpha=0.2)

ax.set_xticks(list(epochs))
ax.set_xticklabels(list(epochs))

plt.tight_layout()

plt.show()
```

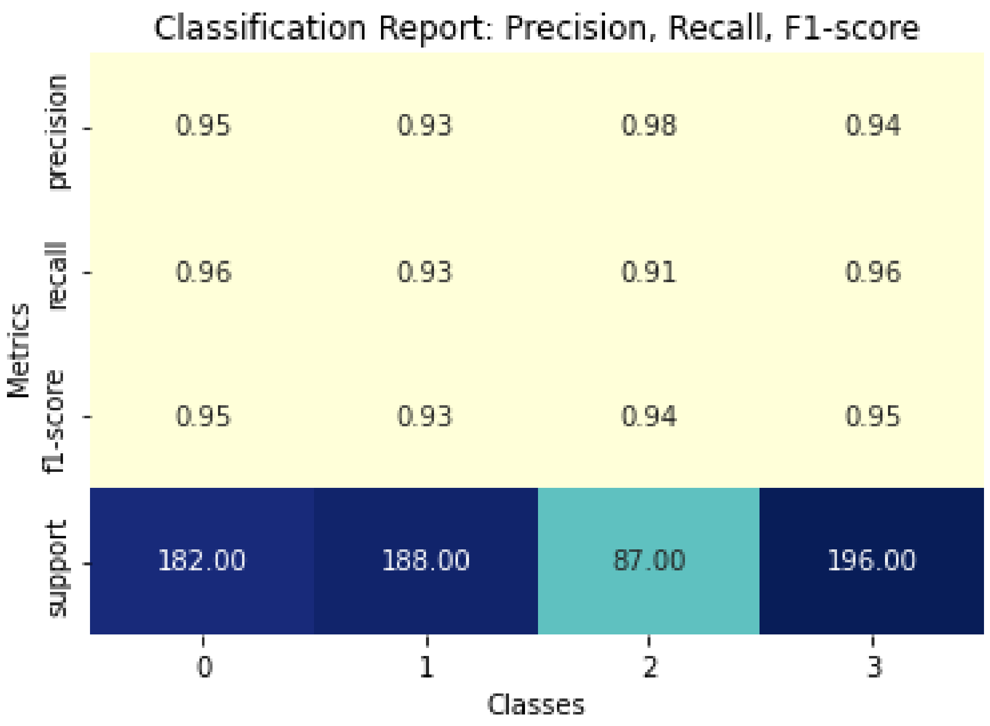


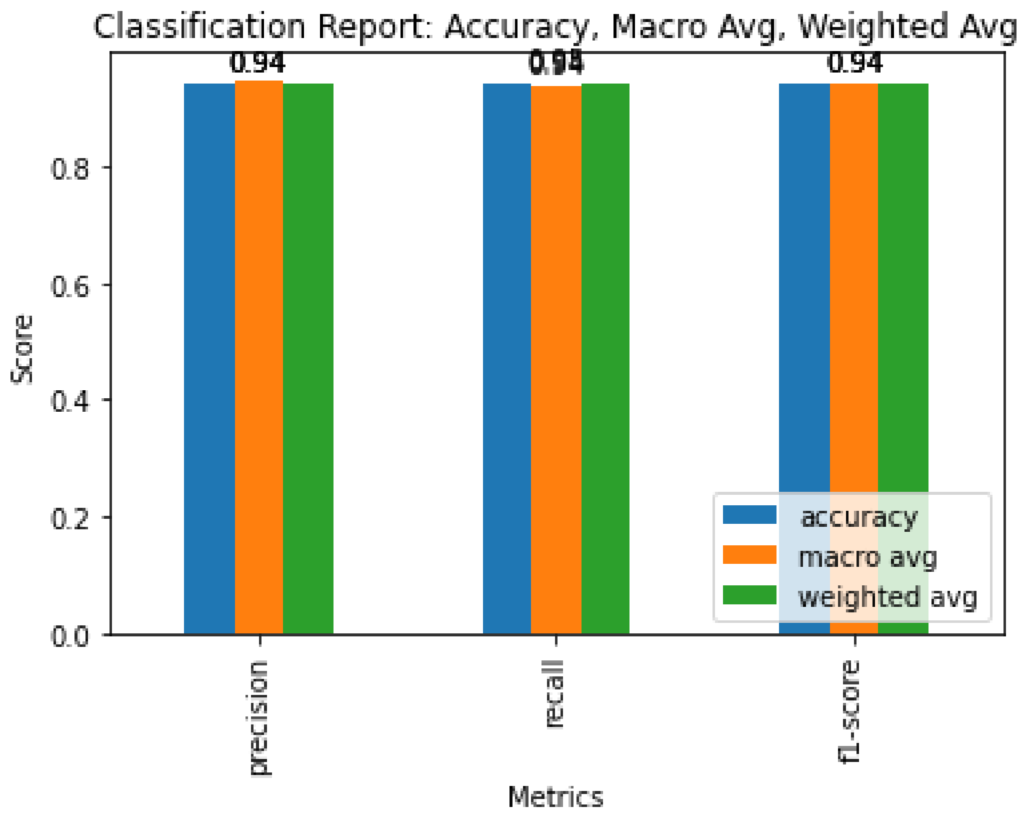
[14]:
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)

fig, ax = plt.subplots(figsize=(14, 7))
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(True)
ax.spines['left'].set_visible(True)
ax.tick_params(axis='both', which='both', length=6, width=2)
ax.set_facecolor('#000000')

ax.plot(epochs, loss, linewidth=3, label="Training Loss", color='#ff4e4e',␣
 ↪alpha=0.8)
ax.scatter(epochs, loss, s=60, color='#ff4e4e', alpha=0.8)

ax.plot(epochs, val_loss, linewidth=3, label="Validation Loss",␣
 ↪color='#00a8cc', alpha=0.8)
ax.scatter(epochs, val_loss, s=60, color='#00a8cc', alpha=0.8)

ax.set_xlabel('Epochs', fontsize=14, labelpad=10, color='#000000',␣
 ↪fontweight='bold')
ax.set_ylabel('Loss', fontsize=14, labelpad=10, color='#000000',␣
 ↪fontweight='bold')
ax.set_title('Training and Validation Loss', fontsize=18, pad=20,␣
 ↪color='#000000', fontweight='bold')

legend = ax.legend(loc='upper right', fontsize=12, frameon=False)
legend.set_title(None)
for handle in legend.legendHandles:
    handle.set_markersize(50)
legend.get_frame().set_facecolor('none')
for text in legend.get_texts():
    text.set_color('#ffffff')

ax.grid(color='#ffffff', linestyle='--', alpha=0.2)

plt.tight_layout()

plt.show()
```



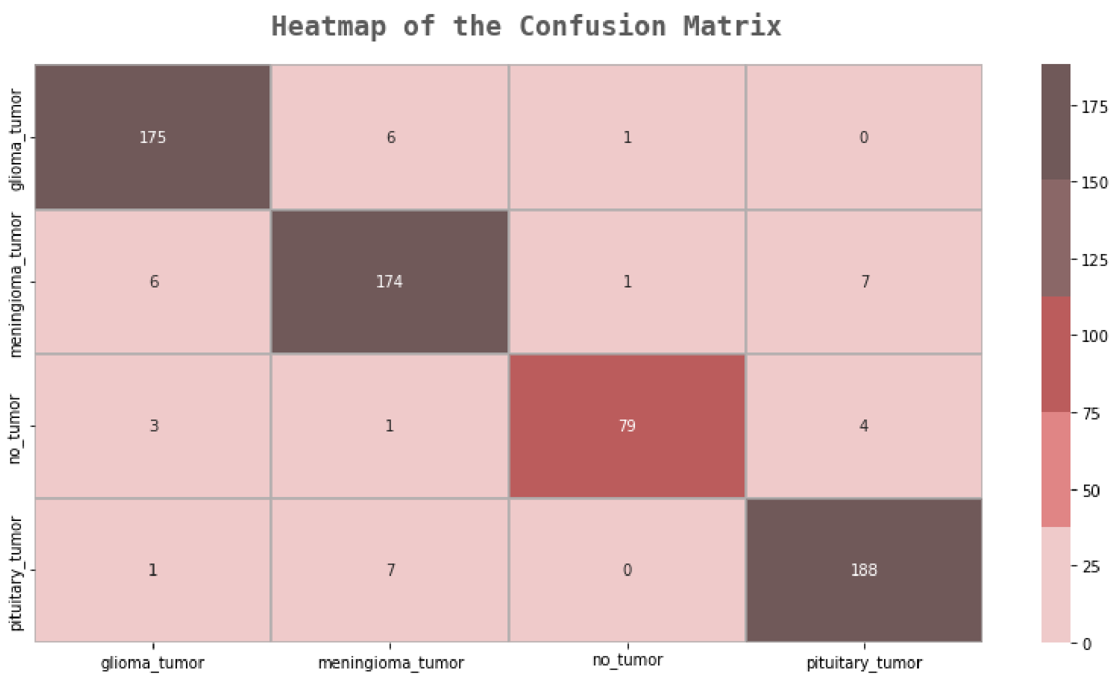
[15]:
```
pred = model.predict(X_test)
pred = np.argmax(pred,axis=1)
y_test_new = np.argmax(y_test,axis=1)
```

[16]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728"]

# Assuming you have already defined 'y_test_new' and 'pred' variables␣
 ↪containing the true labels and predictions.

report = classification_report(y_test_new, pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

sns.heatmap(report_df.iloc[:-3, :].T, annot=True, fmt='.2f', cmap="YlGnBu",␣
 ↪cbar=False)
plt.xlabel('Classes')
plt.ylabel('Metrics')
plt.title('Classification Report: Precision, Recall, F1-score')
plt.show()

metrics = ['accuracy', 'macro avg', 'weighted avg']
scores = report_df.loc[metrics, ['precision', 'recall', 'f1-score']].T
scores.plot(kind='bar', color=colors)

for i, score in enumerate(scores.values):
    for j, value in enumerate(score):
        plt.text(j, value + 0.01, '{:.2f}'.format(value), ha='center',␣
↪va='bottom')

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Classification Report: Accuracy, Macro Avg, Weighted Avg')
plt.legend(loc='lower right')

plt.subplots_adjust(hspace=0.5)

plt.show()
```





```
[17]: import matplotlib.pyplot as plt
      import seaborn as sns
      import numpy as np
      from sklearn.metrics import classification_report,confusion_matrix

      fig, ax = plt.subplots(1, 1, figsize=(14, 7))
      sns.heatmap(confusion_matrix(y_test_new, pred), ax=ax, xticklabels=labels,␣
       ↪yticklabels=labels, annot=True,
                  fmt='d', cmap=colors_red[::-1], alpha=0.7, linewidths=2,␣
       ↪linecolor=colors_dark[3])
      fig.text(s='Heatmap of the Confusion Matrix', size=18, fontweight='bold',
               fontname='monospace', color=colors_dark[1], y=0.92, x=0.28, alpha=0.8)

      plt.show()
```

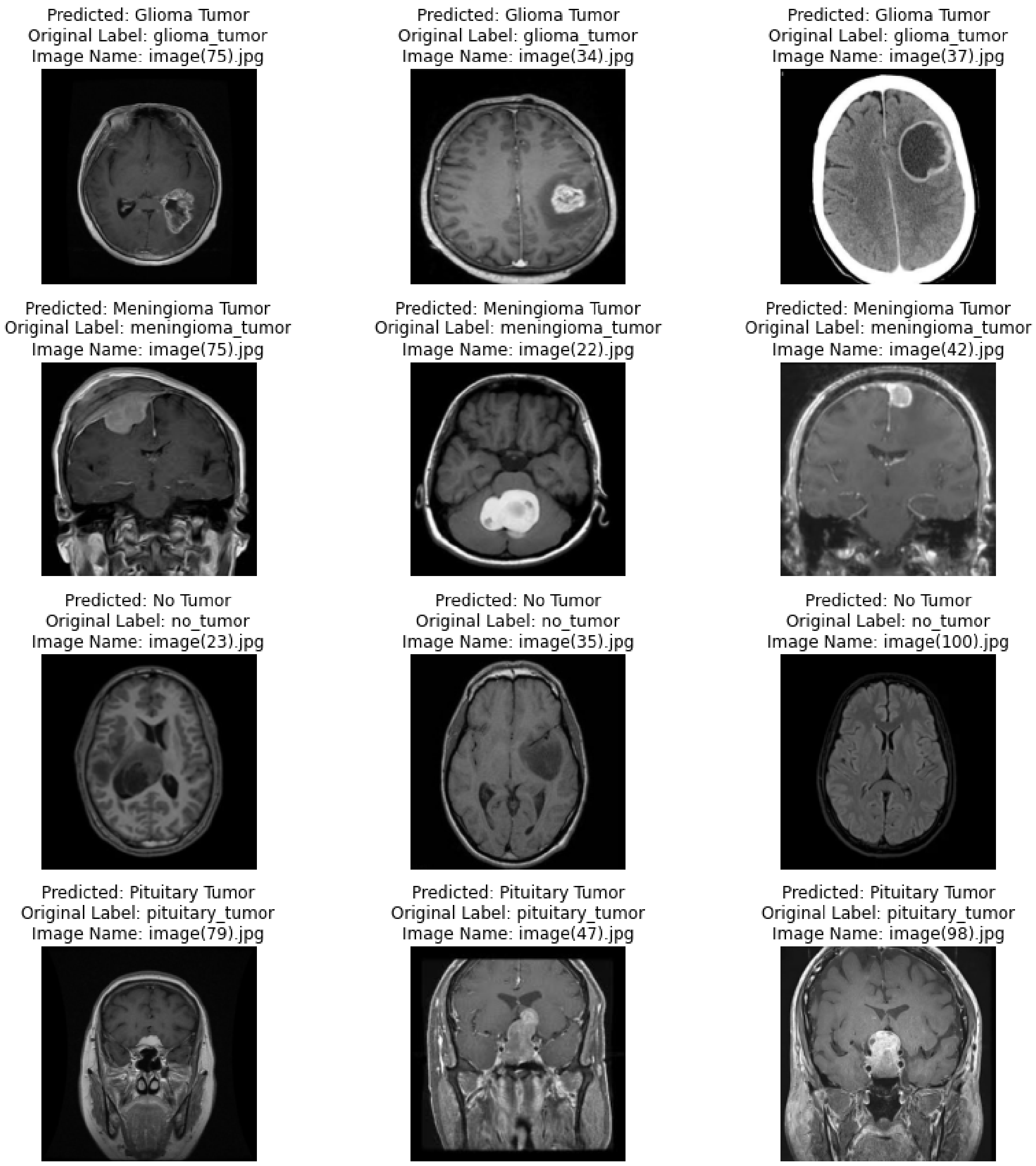


Prediction

[19]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
import os
import random


directory_path = '/kaggle/input/brain-tumor-classification-mri/Testing/'

tumor_classes = ['glioma_tumor', 'meningioma_tumor', 'no_tumor',␣
 ↪'pituitary_tumor']

selected_files_with_labels = {tumor_class: [] for tumor_class in tumor_classes}

num_files_to_select = 3

for tumor_class in tumor_classes:
    class_directory = os.path.join(directory_path, tumor_class)
    class_files = os.listdir(class_directory)
    selected_files = random.sample(class_files, num_files_to_select)
    selected_files_with_labels[tumor_class] = [(file, tumor_class) for file in␣
 ↪selected_files]
```

```
selected_files_with_labels = [item for sublist in selected_files_with_labels.
↪values() for item in sublist]


image_paths = [os.path.join(directory_path, tumor_class, file) for file,␣
↪tumor_class in selected_files_with_labels]
class_mapping = {
    0: 'Glioma Tumor',
    1: 'Meningioma Tumor',
    2: 'No Tumor',
    3: 'Pituitary Tumor'
}
```

[21]:
```
num_images = len(image_paths)
num_rows = int(np.ceil(np.sqrt(num_images)))
num_cols = int(np.ceil(num_images / num_rows))

def shorten_folder_name(folder_name, length=10):
    if len(folder_name) > length:
        return '...' + folder_name[-(length-3):]
    else:
        return folder_name

fig, axes = plt.subplots(4, num_cols, figsize=(12, 12))
plt.subplots_adjust(hspace=0.8, wspace=0.8)


class_counters = {class_name: 0 for class_name in class_mapping.values()}


class_images = {class_name: [] for class_name in class_mapping.values()}
class_image_indices = {class_name: [] for class_name in class_mapping.values()}

for i, (path, label) in enumerate(zip(image_paths, selected_files_with_labels)):
    img = cv2.imread(path)

    if img is None:
        print(f"Failed to read image: {path}")
        continue

    img = cv2.resize(img, (224, 224))

    img_array = np.array(img)
    img_array = img_array.reshape(1, 224, 224, 3)
    predictions = model.predict(img_array)
    predicted_class = class_mapping[predictions.argmax()]


    class_images[predicted_class].append(img)
    class_image_indices[predicted_class].append(i)

for i, class_name in enumerate(class_mapping.values()):
    row_images = class_images[class_name]
    row_indices = class_image_indices[class_name]
    num_images = len(row_images)

    for j in range(min(num_images, num_cols)):
        img_index = row_indices[j]
        original_path = image_paths[img_index]
        original_label = selected_files_with_labels[img_index][1]
        axes[i, j].imshow(cv2.cvtColor(row_images[j], cv2.COLOR_BGR2RGB))
        axes[i, j].axis('off')
        axes[i, j].set_title(f"Predicted: {class_name}\nOriginal Label:␣
 ↪{original_label}\nImage Name: {os.path.basename(original_path)}")  # Use␣
 ↪'\n' for a new line


    for j in range(num_images, num_cols):
        fig.delaxes(axes[i, j])


    class_counters[class_name] += num_cols

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


Predicted: Glioma Tumor
Original Label: glioma_tumor
Image Name: image(75).jpg
Predicted: Glioma Tumor
Original Label: glioma_tumor
Image Name: image(34).jpg
Predicted: Glioma Tumor
Original Label: glioma_tumor
Image Name: image(37).jpg
Predicted: Meningioma Tumor
Original Label: meningioma_tumor
Image Name: image(75).jpg
Predicted: Meningioma Tumor
Original Label: meningioma_tumor
Image Name: image(22).jpg
Predicted: Meningioma Tumor
Original Label: meningioma_tumor
Image Name: image(42).jpg
Predicted: No Tumor
Original Label: no_tumor
Image Name: image(23).jpg
Predicted: No Tumor
Original Label: no_tumor
Image Name: image(35).jpg
Predicted: No Tumor
Original Label: no_tumor
Image Name: image(100).jpg
Predicted: Pituitary Tumor
Original Label: pituitary_tumor
Image Name: image(79).jpg
Predicted: Pituitary Tumor
Original Label: pituitary_tumor
Image Name: image(47).jpg
Predicted: Pituitary Tumor
Original Label: pituitary_tumor
Image Name: image(98).jpg