# Install necessary Libraries

In [ ]:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adamax
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image
import warnings
warnings.filterwarnings('ignore')
```

# Load Train Data

In [ ]:

```
Train_df='/content/drive/MyDrive/archive/Training'

filepaths = []
labels = []
folds = os.listdir(Train_df)
for fold in folds:
    FoldPath = os.path.join(Train_df, fold)
    files = os.listdir(FoldPath)
    for file in tqdm(files):
        filepath = os.path.join(FoldPath,file)
        filepaths.append(filepath)
        labels.append(fold)

print(len(filepaths))
print(len(labels))
print(np.unique(labels))


df_train = pd.DataFrame(
    data = {
        'filepath': filepaths,
        'label': labels
    }
)
df_train.head()
```

```
100%|██████████| 1321/1321 [00:00<00:00, 538818.98it/s]
100%|██████████| 1602/1602 [00:00<00:00, 533360.45it/s]
100%|██████████| 1477/1477 [00:00<00:00, 504683.26it/s]
100%|██████████| 1389/1389 [00:00<00:00, 495440.79it/s]
```

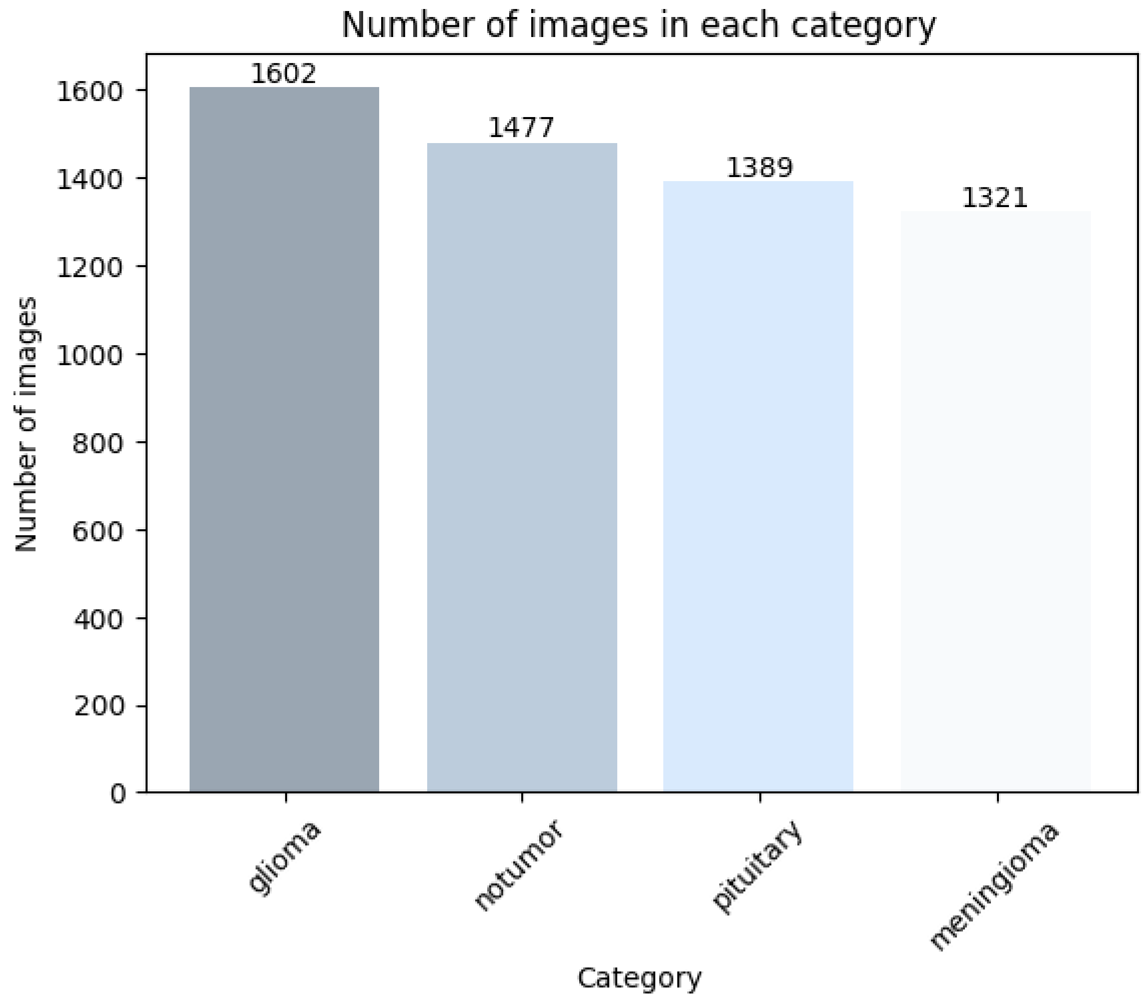
```
5789
5789
['glioma' 'meningioma' 'notumor' 'pituitary']
Out[ ]:
```

| | filepath | label |
|---|---|---|
| **0** | /content/drive/MyDrive/archive/Training/glioma... | glioma |
| **1** | /content/drive/MyDrive/archive/Training/glioma... | glioma |
| **2** | /content/drive/MyDrive/archive/Training/glioma... | glioma |
| **3** | /content/drive/MyDrive/archive/Training/glioma... | glioma |
| **4** | /content/drive/MyDrive/archive/Training/glioma... | glioma |

# Visualise Train Data

In [ ]:

```
color = ['#9AA6B2','#BCCCDC','#D9EAFD','#F8FAFC']
fig, ax = plt.subplots()
bars = ax.bar(df_train['label'].unique(), df_train['label'].value_counts(),color=color)
ax.bar_label(bars)
plt.title('Number of images in each category')
plt.xlabel('Category')
plt.ylabel('Number of images')
plt.xticks(rotation=45)
plt.show()
```




# Load Test Data

In [ ]:

```
Test_df = '/content/drive/MyDrive/archive/Testing'

filepaths = []
labels = []
folds = os.listdir(Test_df)
for fold in folds:
    FoldPath = os.path.join(Test_df, fold)
    files = os.listdir(FoldPath)
    for file in tqdm(files):
        filepath = os.path.join(FoldPath,file)
        filepaths.append(filepath)
        labels.append(fold)

print(len(filepaths))
print(len(labels))
print(np.unique(labels))

df_test = pd.DataFrame(
    data = {
        'filepath': filepaths,
        'label': labels
    }
```

```
)
df_test.head()
```

```
100%|██████████| 405/405 [00:00<00:00, 451299.98it/s]
100%|██████████| 300/300 [00:00<00:00, 315519.36it/s]
100%|██████████| 300/300 [00:00<00:00, 515059.84it/s]
100%|██████████| 306/306 [00:00<00:00, 369659.28it/s]
1311
1311
['glioma' 'meningioma' 'notumor' 'pituitary']
```

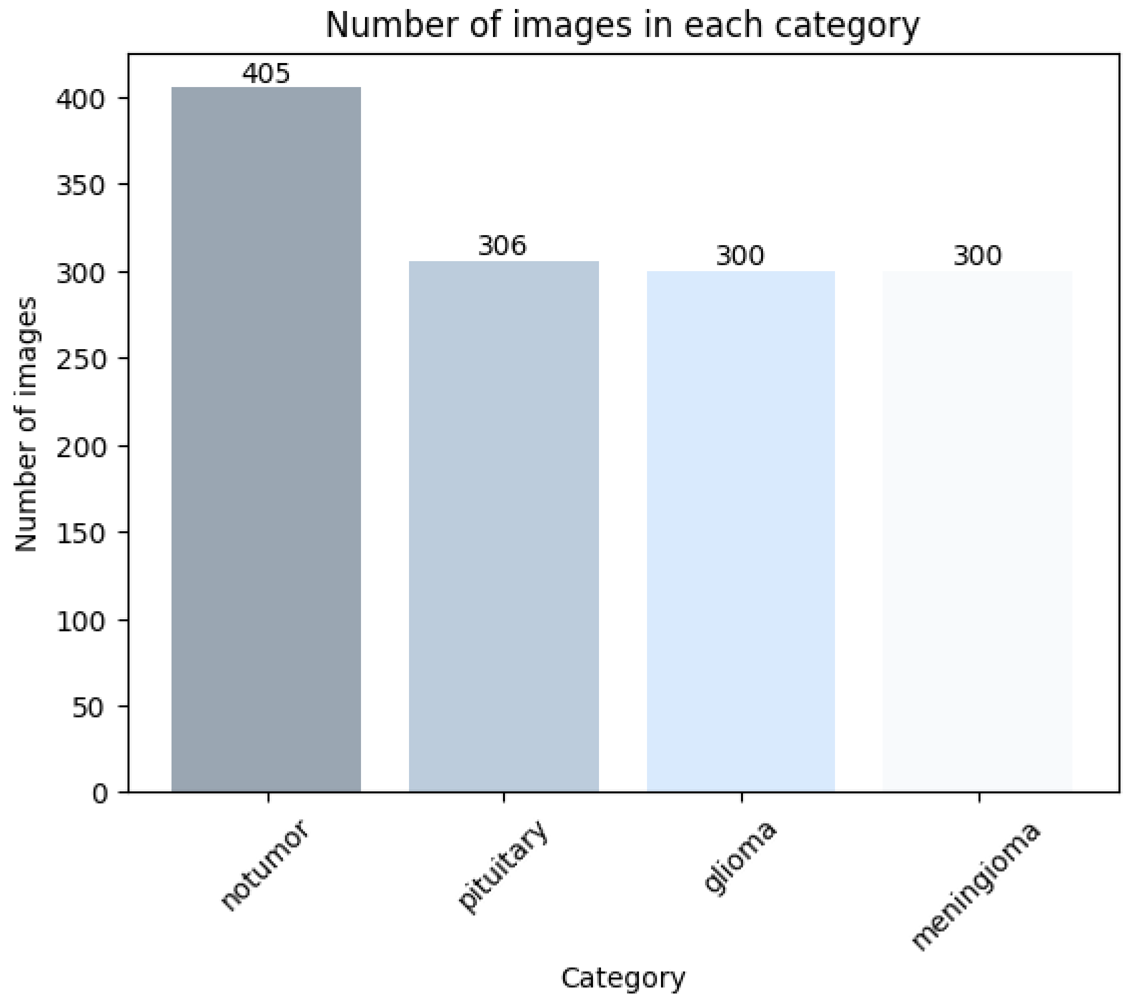
Out[ ]:

| | filepath | label |
|---|---|---|
| **0** | /content/drive/MyDrive/archive/Testing/notumor... | notumor |
| **1** | /content/drive/MyDrive/archive/Testing/notumor... | notumor |
| **2** | /content/drive/MyDrive/archive/Testing/notumor... | notumor |
| **3** | /content/drive/MyDrive/archive/Testing/notumor... | notumor |
| **4** | /content/drive/MyDrive/archive/Testing/notumor... | notumor |

# Visualise Test Data

In [ ]:

```
fig, ax = plt.subplots()
bars = ax.bar(df_test['label'].unique(), df_test['label'].value_counts(),color=color)
ax.bar_label(bars)
plt.title('Number of images in each category')
plt.xlabel('Category')
plt.ylabel('Number of images')
plt.xticks(rotation=45)

plt.show()
```




# Split Data into Train, Validation and Test

In [ ]:

```
valid_ts, df_test = train_test_split(df_test, test_size=0.5, random_state=42)

tr_gen = ImageDataGenerator(rescale=1/255)
ts_gen = ImageDataGenerator(rescale=1/255)

batchsize = 32
img_size = (224, 224)

gen_train = tr_gen.flow_from_dataframe(df_train,
                                       x_col='filepath',
                                       y_col='label',
                                       target_size=img_size,
                                       class_mode='categorical',
                                       batch_size=batchsize,
                                       shuffle=True,
                                       color_mode='rgb')
gen_valid = ts_gen.flow_from_dataframe(valid_ts,
                                       x_col='filepath',
                                       y_col='label',
                                       target_size=img_size,
                                       class_mode='categorical',
                                       batch_size=batchsize,
                                       shuffle=True,
```

```
                                             color_mode='rgb')
gen_test = ts_gen.flow_from_dataframe(df_test,
                                      x_col='filepath',
                                      y_col='label',
                                      target_size=img_size,
                                      class_mode='categorical',
                                      batch_size=batchsize,
                                      shuffle=False,
                                      color_mode='rgb')

class_dict = gen_train.class_indices
```

```
Found 5789 validated image filenames belonging to 4 classes.
Found 164 validated image filenames belonging to 4 classes.
Found 164 validated image filenames belonging to 4 classes.
```

# Model Building...

In [ ]:

```
Model = Sequential([
    Conv2D(64, kernel_size= (3,3), activation='relu', input_shape=(img_size[0],img_size[
    Conv2D(64, kernel_size= (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Conv2D(128, kernel_size= (3,3), activation='relu'),
    Conv2D(128, kernel_size= (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Conv2D(256, kernel_size= (3,3), activation='relu'),
    Conv2D(255, kernel_size= (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Conv2D(512, kernel_size= (3,3), activation='relu'),
    Conv2D(512, kernel_size= (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Conv2D(512, kernel_size= (3,3), activation='relu'),
    Conv2D(512, kernel_size= (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(256, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(4, activation='softmax') ])

Model.compile(optimizer=Adamax(learning_rate=0.001), loss='categorical_crossentropy', me

history = Model.fit(
    gen_train,
    epochs=15,
    validation_data=gen_valid,
    verbose=1,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore
)
```

```
Epoch 1/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 1724s 9s/step - accuracy: 0.3640 - loss: 1.2599 - val_accur
```

```
acy: 0.6524 - val_loss: 0.7799
Epoch 2/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 60s 328ms/step - accuracy: 0.6481 - loss: 0.8082 - val_accu
racy: 0.7622 - val_loss: 0.5891
Epoch 3/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 328ms/step - accuracy: 0.7389 - loss: 0.6342 - val_accu
racy: 0.8110 - val_loss: 0.4537
Epoch 4/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 328ms/step - accuracy: 0.8081 - loss: 0.5002 - val_accu
racy: 0.8232 - val_loss: 0.4190
Epoch 5/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 60s 330ms/step - accuracy: 0.8577 - loss: 0.3891 - val_accu
racy: 0.8293 - val_loss: 0.3476
Epoch 6/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 60s 331ms/step - accuracy: 0.8975 - loss: 0.2907 - val_accu
racy: 0.8598 - val_loss: 0.3417
Epoch 7/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 82s 332ms/step - accuracy: 0.9114 - loss: 0.2552 - val_accu
racy: 0.8598 - val_loss: 0.3065
Epoch 8/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 327ms/step - accuracy: 0.9240 - loss: 0.2048 - val_accu
racy: 0.8598 - val_loss: 0.3326
Epoch 9/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 328ms/step - accuracy: 0.9442 - loss: 0.1676 - val_accu
racy: 0.9085 - val_loss: 0.1962
Epoch 10/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 60s 330ms/step - accuracy: 0.9468 - loss: 0.1370 - val_accu
racy: 0.9207 - val_loss: 0.1887
Epoch 11/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 82s 333ms/step - accuracy: 0.9501 - loss: 0.1378 - val_accu
racy: 0.9024 - val_loss: 0.2082
Epoch 12/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 327ms/step - accuracy: 0.9668 - loss: 0.1043 - val_accu
racy: 0.9390 - val_loss: 0.1453
Epoch 13/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 59s 328ms/step - accuracy: 0.9724 - loss: 0.0758 - val_accu
racy: 0.9451 - val_loss: 0.1223
Epoch 14/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 60s 330ms/step - accuracy: 0.9642 - loss: 0.0938 - val_accu
racy: 0.9451 - val_loss: 0.1347
Epoch 15/15
181/181 ━━━━━━━━━━━━━━━━━━━━ 82s 332ms/step - accuracy: 0.9821 - loss: 0.0533 - val_accu
racy: 0.9390 - val_loss: 0.1289
```

# Accuracy and saving

In [ ]:

```
print(Model.evaluate(gen_train))
print(Model.evaluate(gen_valid))
print(Model.evaluate(gen_test))
Model.save('Model.h5')
```

```
181/181 ━━━━━━━━━━━━━━━━━━━━ 40s 219ms/step - accuracy: 0.9730 - loss: 0.0792
[0.08014791458845139, 0.9730523228645325]
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.9595 - loss: 0.0925
[0.12229700674057, 0.9451219439506531]
6/6 ━━━━━━━━━━━━━━━━━━━━ 42s 8s/step - accuracy: 0.9228 - loss: 0.3221
```

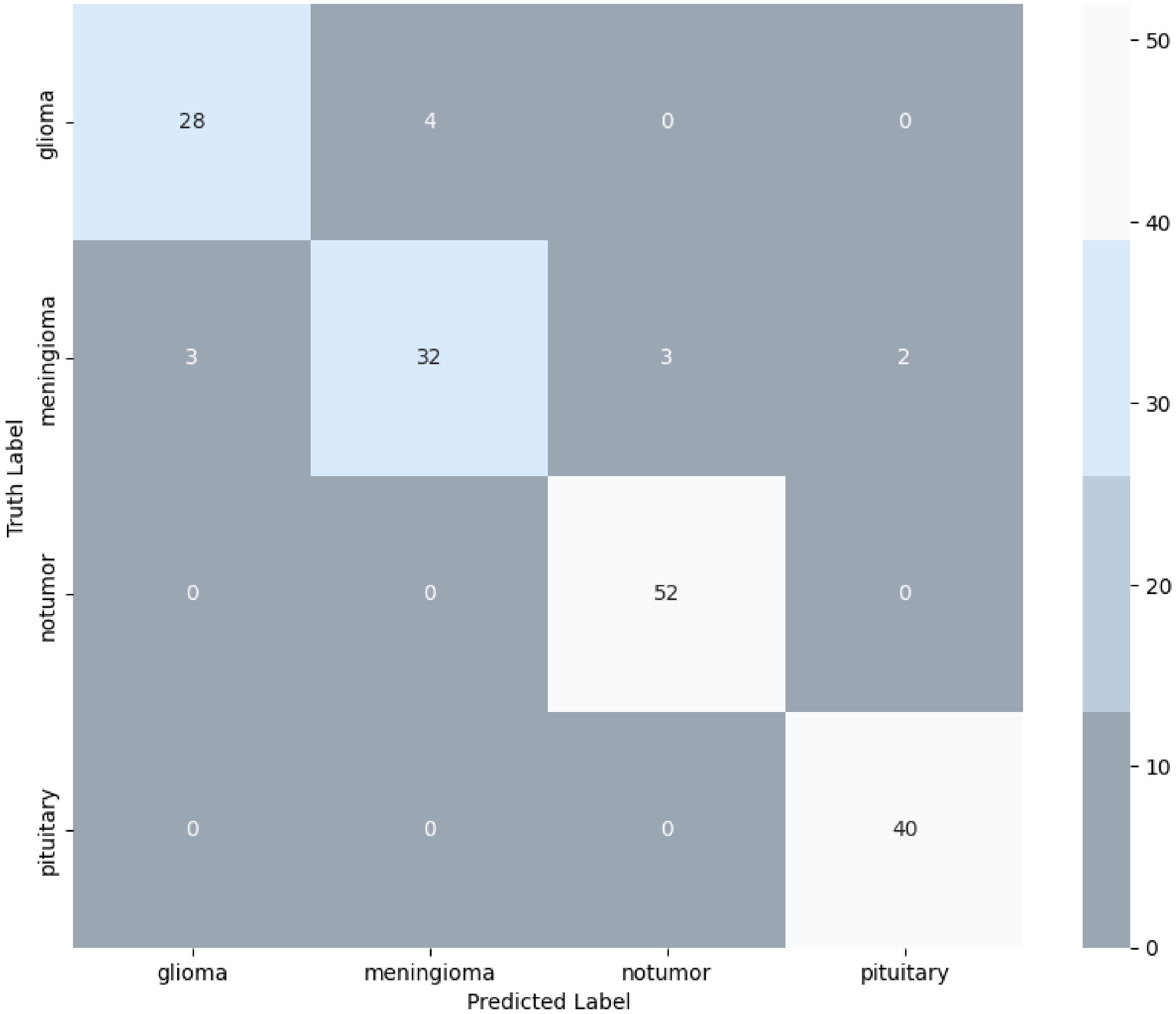
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
[0.27277714014053345, 0.9268292784690857]
```

# Confusion Matrix and Graphs

In [ ]:

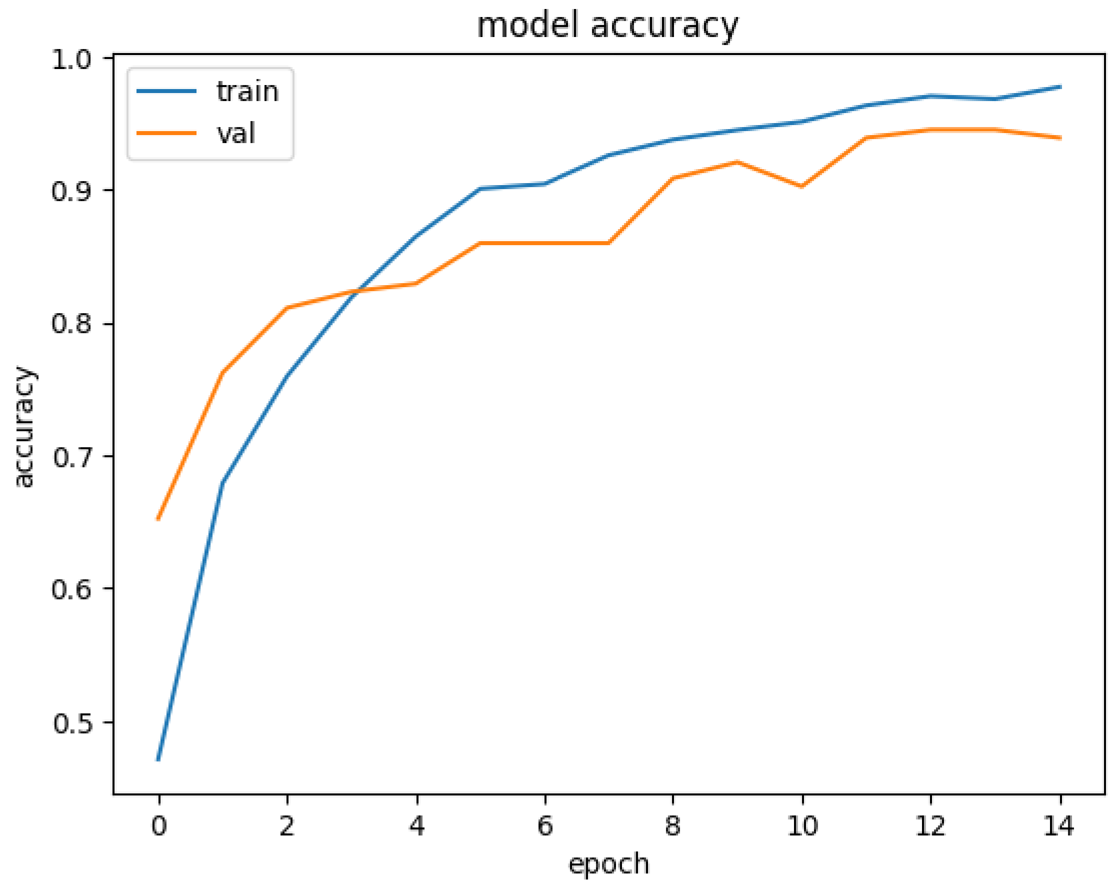
```
preds = Model.predict(gen_test)
y_pred = np.argmax(preds, axis=1)
cm = confusion_matrix(gen_test.classes, y_pred)
labels = list(class_dict.keys())
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap=color, xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Label')
plt.ylabel('Truth Label')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 171ms/step
```




Out[ ]:

```
<matplotlib.legend.Legend at 0x7b14c818d090>
```

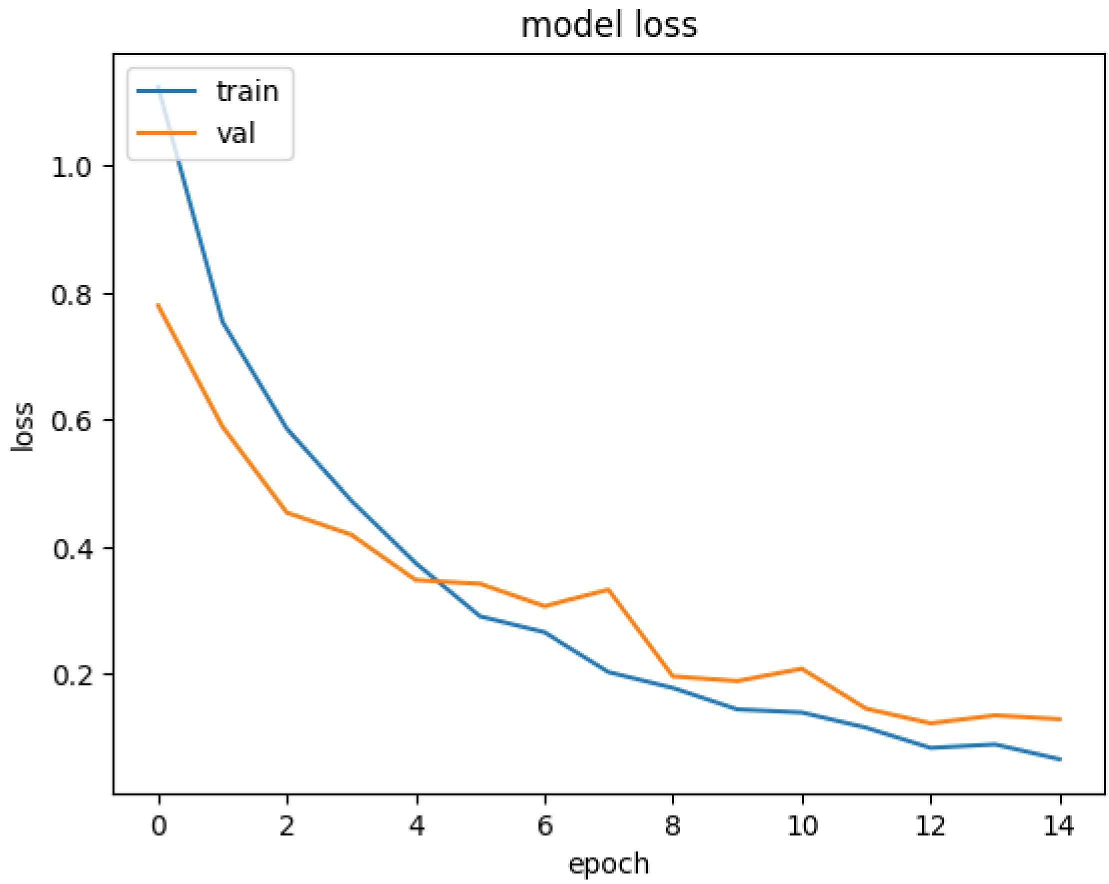



In [ ]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7b14c81d5e50>
```




# Classify With the model

In [ ]:

```
def predict(img_path):
    img = Image.open(img_path).resize((224, 224))
    img_array = np.asarray(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    predictions = Model.predict(img_array)[0]
    predicted_index = np.argmax(predictions)
    predicted_label = list(class_dict.keys())[predicted_index]
    confidence = predictions[predicted_index]

    plt.figure(figsize=(5, 5))
    plt.imshow(img)
    plt.axis("off")
    plt.title(f"{predicted_label} ({confidence:.2f})", fontsize=14)
    plt.show()

predict('/content/drive/MyDrive/archive/Testing/notumor/Te-no_0119.jpg')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

notumor (1.00)