```
from google.colab import drive
drive.mount('/content/drive')
```

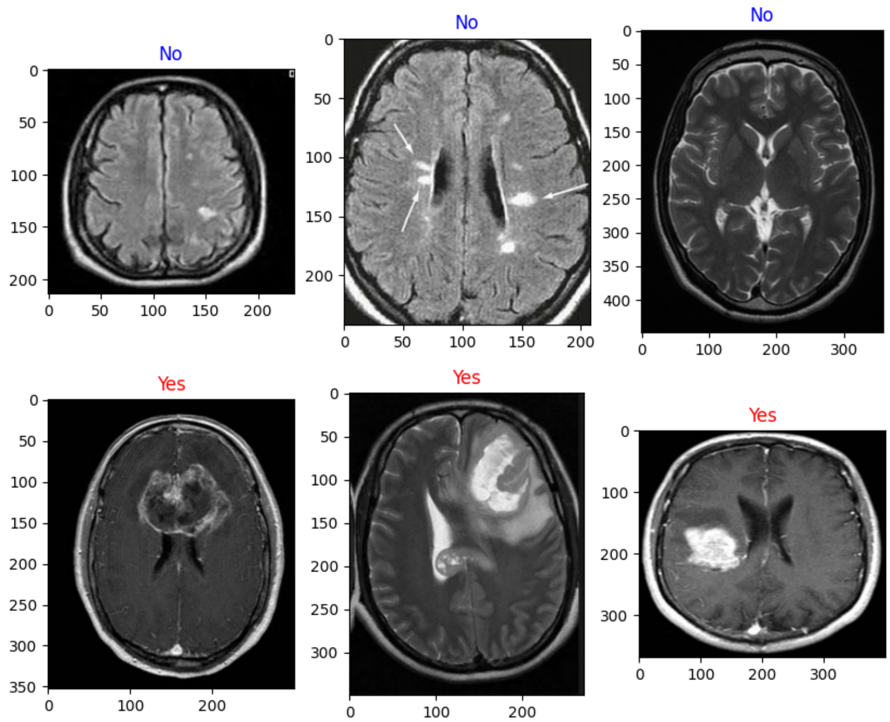
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Import necessary libraries
import os
import numpy as np
import cv2
import imutils
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,Flatten,BatchNormalization,Dropout
from tensorflow.keras.models import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

```
# Define the path to the dataset
IMG_PATH = '/content/drive/MyDrive/MINOR_TESTPHASE/brain_tumor_dataset'

# Create a list of all the image filenames
all_images = []
for folder in ['yes', 'no']:
    folder_path = os.path.join(IMG_PATH, folder)
    for filename in os.listdir(folder_path):

         all_images.append(os.path.join(folder_path, filename))

# Create a list of corresponding labels (0 for 'no', 1 for 'yes')
labels = [1 if 'Y' in filename else 0 for filename in all_images]

# Split the dataset into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(all_images, labels, test_size=0.2, random_state=123)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=123)

# Print the sizes of the resulting datasets
print(f'Train set size: {len(X_train)}')
print(f'Validation set size: {len(X_val)}')
print(f'Test set size: {len(X_test)}')
```

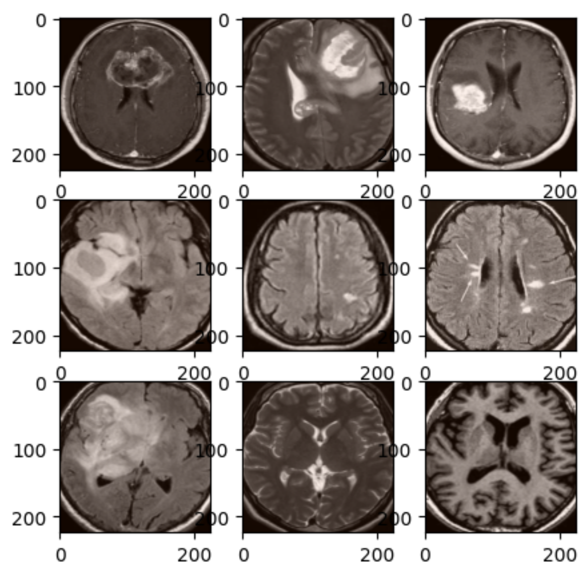
```
Train set size: 151
Validation set size: 51
Test set size: 51
```

```
%matplotlib inline

# Define the path to the dataset
IMG_PATH = '/content/drive/MyDrive/MINOR_TESTPHASE/brain_tumor_dataset'


# Define the labels and their corresponding colors
labels = {0: 'No', 1: 'Yes'}
colors = {0: 'blue', 1: 'red'}

# Plot some images from the train set for each label
fig, axs = plt.subplots(2, 3, figsize=(10, 8))
for i, label in enumerate([0, 1]):
    images = [x for x, y in zip(X_train, y_train) if y == label][:3]
    for j, image_path in enumerate(images):
        img = plt.imread(image_path)
        axs[i, j].imshow(img)
        axs[i, j].set_title(labels[label], color=colors[label])
plt.show()
```




```
def preprocess_images(images):
    preprocessed_images = []
    for i, img_path in enumerate(images):
        img = cv2.imread(img_path)
        img = cv2.resize(
            img,
            dsize=(224,224),
            interpolation=cv2.INTER_CUBIC
        )
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)

        # threshold the image, then perform a series of erosions +
        # dilations to remove any small regions of noise
        thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
        thresh = cv2.erode(thresh, None, iterations=2)
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in thresholded image, then grab the largest one
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        c = max(cnts, key=cv2.contourArea)

        # find the extreme points
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0])

        # crop
        ADD_PIXELS = 0
        new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()
        new_img = cv2.resize(
            new_img,
            dsize=(224,224))
        preprocessed_images.append(new_img)
    return np.array(preprocessed_images)


# Apply the preprocessing to all the data subsets
X_train_pre = preprocess_images(X_train)
```

```
X_val_pre = preprocess_images(X_val)
X_test_pre = preprocess_images(X_test)


# Transform the subsets to numpy arrays
X_train_pre_vgg = np.array([preprocess_input(image) for image in X_train_pre])
X_val_pre_vgg = np.array([preprocess_input(image) for image in X_val_pre])
X_test_pre_vgg = np.array([preprocess_input(image) for image in X_test_pre])


# plot some images from X_train_pre
import matplotlib.pyplot as plt
from skimage import exposure

plt.figure(figsize=(5, 5))
for i in range(9):
    img = exposure.rescale_intensity(X_train_pre_vgg[i], out_range=(0, 1))
    plt.subplot(3, 3, i+1)
    plt.imshow(img, cmap='gray')
plt.show()
```



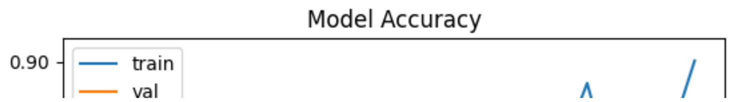

```
# Load the pre-trained VGG19 model
base_model = VGG19(
weights='imagenet',
include_top=False,
input_shape=(224,224) + (3,)
)

# Create a new model by adding a few layers on top of the pre-trained model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# Freeze the weights of the pre-trained model
model.layers[0].trainable = False

# Compile the model with appropriate loss function, optimizer and metrics
model.compile(
loss='binary_crossentropy',
optimizer=Adam(),
metrics=['accuracy']
)

# Print the summary of the model
model.summary()

# Define the training data generator with necessary data augmentation techniques
train_datagen = ImageDataGenerator(
rescale=1./255,
rotation_range=20,
```

```
width_shift_range=0.1,
height_shift_range=0.1,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest'
)

# Create the training data generator using the training dataset and the data generator
train_generator = train_datagen.flow(
X_train_pre_vgg,
y_train,
batch_size=32
)

# Define a callback to reduce the learning rate when the validation accuracy plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, patience=5, min_lr=1e-5)

# Define a callback to stop the training when validation accuracy reaches 90%
class StopOnAccuracy(Callback):
    def on_epoch_end(self, epoch, logs={}):
        if logs.get('accuracy') > 0.9:
            print("\nReached 90% accuracy, stopping training...")
            self.model.stop_training = True

# Define the validation data generator with appropriate data preprocessing
val_datagen = ImageDataGenerator(rescale=1./255)

# Create the validation data generator using the validation dataset and the data generator
val_generator = val_datagen.flow(X_val_pre_vgg, y_val, batch_size=18)

# Train the model using the training data generator, validation data generator and the defined callbacks
history = model.fit(
train_generator,
epochs=50,
validation_data=val_generator,
callbacks=[reduce_lr,StopOnAccuracy()]
)
```

```
5/5 [==============================] - 2s 412ms/step - loss: 0.6055 - accuracy: 0.6689 - val_loss: 0.6991 - val_accuracy: 0.6667 - lr
Epoch 3/50
5/5 [==============================] - 3s 583ms/step - loss: 0.5626 - accuracy: 0.7152 - val_loss: 0.6144 - val_accuracy: 0.6863 - lr
Epoch 4/50
5/5 [==============================] - 3s 465ms/step - loss: 0.5811 - accuracy: 0.6887 - val_loss: 0.5653 - val_accuracy: 0.7843 - lr
Epoch 5/50
5/5 [==============================] - 2s 414ms/step - loss: 0.4929 - accuracy: 0.7815 - val_loss: 0.5329 - val_accuracy: 0.7843 - lr
Epoch 6/50
5/5 [==============================] - 2s 435ms/step - loss: 0.4622 - accuracy: 0.7947 - val_loss: 0.5336 - val_accuracy: 0.7647 - lr
Epoch 7/50
5/5 [==============================] - 2s 487ms/step - loss: 0.4609 - accuracy: 0.7417 - val_loss: 0.5216 - val_accuracy: 0.7451 - lr
Epoch 8/50
5/5 [==============================] - 2s 403ms/step - loss: 0.4903 - accuracy: 0.7748 - val_loss: 0.5095 - val_accuracy: 0.7843 - lr
Epoch 9/50
5/5 [==============================] - 3s 558ms/step - loss: 0.4325 - accuracy: 0.8013 - val_loss: 0.4963 - val_accuracy: 0.7843 - lr
Epoch 10/50
5/5 [==============================] - 3s 502ms/step - loss: 0.3753 - accuracy: 0.7947 - val_loss: 0.5360 - val_accuracy: 0.7843 - lr
Epoch 11/50
5/5 [==============================] - 2s 475ms/step - loss: 0.4180 - accuracy: 0.8079 - val_loss: 0.4793 - val_accuracy: 0.7843 - lr
Epoch 12/50
5/5 [==============================] - 2s 432ms/step - loss: 0.3235 - accuracy: 0.8411 - val_loss: 0.4763 - val_accuracy: 0.7843 - lr
Epoch 13/50
```

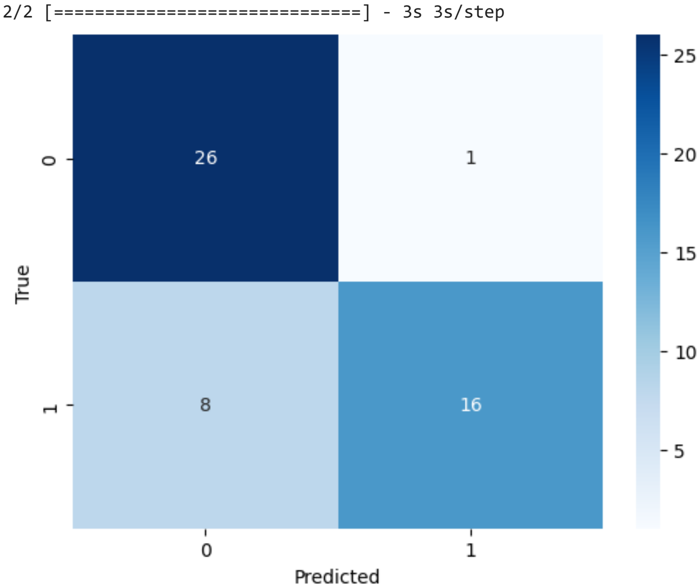
```
Epoch 24/50
5/5 [==============================] - 2s 494ms/step - loss: 0.2998 - accuracy: 0.8874 - val_loss: 0.4625 - val_accuracy: 0.7843 - lr
Epoch 25/50
5/5 [==============================] - 2s 414ms/step - loss: 0.3287 - accuracy: 0.8411 - val_loss: 0.4631 - val_accuracy: 0.7843 - lr
Epoch 26/50
5/5 [==============================] - 2s 419ms/step - loss: 0.3735 - accuracy: 0.8609 - val_loss: 0.4650 - val_accuracy: 0.7843 - lr
Epoch 27/50
5/5 [==============================] - 2s 411ms/step - loss: 0.3110 - accuracy: 0.8609 - val_loss: 0.4670 - val_accuracy: 0.7647 - lr
Epoch 28/50
5/5 [==============================] - 2s 513ms/step - loss: 0.3223 - accuracy: 0.8609 - val_loss: 0.4655 - val_accuracy: 0.7647 - lr
Epoch 29/50
5/5 [==============================] - ETA: 0s - loss: 0.3012 - accuracy: 0.9007
Reached 90% accuracy, stopping training...
```

```
# Plot the training and validation accuracy curves
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')
plt.show()

# Plot the training and validation loss curves
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')
plt.show()
```




```
# Make predictions on X_test_pre
y_pred = model.predict(X_test_pre)
y_pred = np.round(y_pred).astype(int)

# Plot confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
2/2 [==============================] - 3s 3s/step
```




```
#kapa score
from sklearn.metrics import cohen_kappa_score

# Calculate Cohen's kappa coefficient
kappa = cohen_kappa_score(y_test, y_pred)

# Print kappa score
print('Kappa score:', kappa)
```

```
Kappa score: 0.64
```

```
#F1.score
from sklearn.metrics import f1_score

# Make predictions on X_test_pre
y_pred = model.predict(X_test_pre)
y_pred = np.round(y_pred).astype(int)

# Calculate F1 score
f1 = f1_score(y_test, y_pred)

print('F1 score:', f1)
```

```
2/2 [==============================] - 0s 183ms/step
F1 score: 0.7804878048780487
```

```
#Recall
from sklearn.metrics import recall_score
```

```
recall = recall_score(y_test, y_pred)
print('Recall:', recall)
```

```
Recall: 0.6666666666666666
```

```
#precision
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred)
print('Precision:', precision)
```

```
Precision: 0.9411764705882353
```

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.8235294117647058
```