Brain Tumor Detection with VGG-16 Model

author: Ruslan Klymentiev

date: 10th June, 2019

Table of Contents

1. Project Overview and Objectives

The main purpose of this project was to build a CNN model that would classify if subject has a tumor or not base on MRI scan. I used the VGG-16 model architecture and weights to train the model for this binary problem. I used `accuracy` as a metric to justify the model performance which can be defined as:

$\textrm{Accuracy} = \frac{\textrm{Number of correclty predicted images}}{\textrm{Total number of tested images}} \times 100\%$

Final results look as follows:

Set	Accuracy
Validation Set*	~88%
Test Set*	~80%

* *Note: there might be some misunderstanding in terms of set names so I want to describe what do I mean by `test` and `validation` set:* * *`validation set` - is the set used during the model training to adjust the hyperparameters. * * *`test set` - is the small set that I don't touch for the whole training process at all. It's been used for final model performance evaluation.*

1.1. Data Set Description

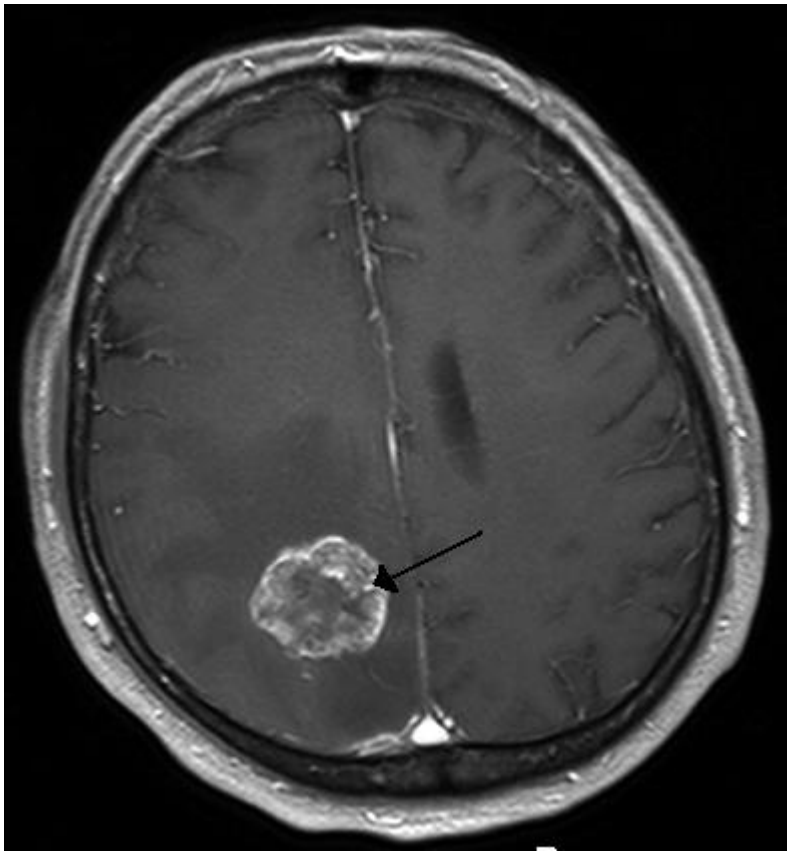
The image data that was used for this problem is Brain MRI Images for Brain Tumor Detection. It conists of MRI scans of two classes:

- `NO` - no tumor, encoded as `0`
- `YES` - tumor, encoded as `1`

Unfortunately, the data set description doesn't hold any information where this MRI scans come from and so on.

1.2. What is Brain Tumor?

> A brain tumor occurs when abnormal cells form within the brain. There are two main types of tumors: cancerous (malignant) tumors and benign tumors. Cancerous tumors can be divided into primary tumors, which start within the brain, and secondary tumors, which have spread from elsewhere, known as brain metastasis tumors. All types of brain tumors may produce symptoms that vary depending on the part of the brain involved. These symptoms may include headaches, seizures, problems with vision, vomiting and mental changes. The headache is classically worse in the morning and goes away with vomiting. Other symptoms may include difficulty walking, speaking or with sensations. As the disease progresses, unconsciousness may occur.



Brain metastasis in the right cerebral hemisphere from lung cancer, shown on magnetic resonance imaging.

Source: Wikipedia

2. Setting up the Environment

```
from IPython.display import clear_output
!pip install imutils
clear_output()

import numpy as np
from tqdm import tqdm
import cv2
import os
import shutil
import itertools
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

# Updated imports from tensorflow.keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras import layers
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Flatten, Dropout, Dense

init_notebook_mode(connected=True)
RANDOM_SEED = 123
```

Right now all images are in one folder with `yes` and `no` subfolders. I will split the data into `train`, `val` and `test` folders which makes its easier to work for me. The new folder heirarchy will look as follows:

```
!apt-get install tree
clear_output()
# create new folders
!mkdir TRAIN TEST VAL TRAIN/YES TRAIN/NO TEST/YES TEST/NO VAL/YES VAL/NO
!tree -d
```

```
.
 TEST
   NO
   YES
 TRAIN
   NO
   YES
 VAL
     NO
     YES

9 directories
```

```
IMG_PATH = '../input/brain-mri-images-for-brain-tumor-detection/brain_tumor_dataset/'
# split the data by train/val/test
for CLASS in os.listdir(IMG_PATH):
   if not CLASS.startswith('.'):
       IMG_NUM = len(os.listdir(IMG_PATH + CLASS))
       for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
           img = IMG_PATH + CLASS + '/' + FILE_NAME
           if n < 5:
               shutil.copy(img, 'TEST/' + CLASS.upper() + '/' + FILE_NAME)
           elif n < 0.8*IMG_NUM:
               shutil.copy(img, 'TRAIN/'+ CLASS.upper() + '/' + FILE_NAME)
           else:
               shutil.copy(img, 'VAL/'+ CLASS.upper() + '/' + FILE_NAME)
```

3. Data Import and Preprocessing

```
import cv2
import numpy as np
from tqdm import tqdm

def load_data(dir_path, img_size=(100,100)):
   """
   Load resized images as np.arrays with consistent shapes
   """
   X = []
   y = []
   labels = dict()
   for idx, path in enumerate(tqdm(sorted(os.listdir(dir_path)))):
       if not path.startswith('.'):
           labels[idx] = path
           full_path = os.path.join(dir_path, path)
           for file in os.listdir(full_path):
               if not file.startswith('.'):
                   img_path = os.path.join(full_path, file)
                   try:
                       # Read image and handle grayscale/color
                       img = cv2.imread(img_path, cv2.IMREAD_COLOR)
                       if img is None:
                           raise ValueError(f"Failed to load {img_path}")

                       # Resize to target size
                       img = cv2.resize(img, img_size)

                       # Ensure 3 channels (convert grayscale to RGB)
                       if img.ndim == 2:  # Grayscale
                           img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
                       else:  # Color (BGR to RGB)
                           img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

                       X.append(img)
```

```
                    y.append(idx)
                except Exception as e:
                    print(f"Skipping {img_path}: {str(e)}")
    X = np.array(X)
    y = np.array(y)
    print(f'{len(X)} images loaded from {dir_path} directory.')
    return X, y, labels

TRAIN_DIR = 'TRAIN/'
TEST_DIR = 'TEST/'
VAL_DIR = 'VAL/'
IMG_SIZE = (224,224)

# use predefined function to load the image data into workspace
X_train, y_train, labels = load_data(TRAIN_DIR, IMG_SIZE)
X_test, y_test, _ = load_data(TEST_DIR, IMG_SIZE)
X_val, y_val, _ = load_data(VAL_DIR, IMG_SIZE)
```

```
100%| | 2/2 [00:00<00:00,  6.32it/s]

193 images loaded from TRAIN/ directory.

100%| | 2/2 [00:00<00:00, 73.62it/s]

10 images loaded from TEST/ directory.

100%| | 2/2 [00:00<00:00, 24.26it/s]

50 images loaded from VAL/ directory.
```

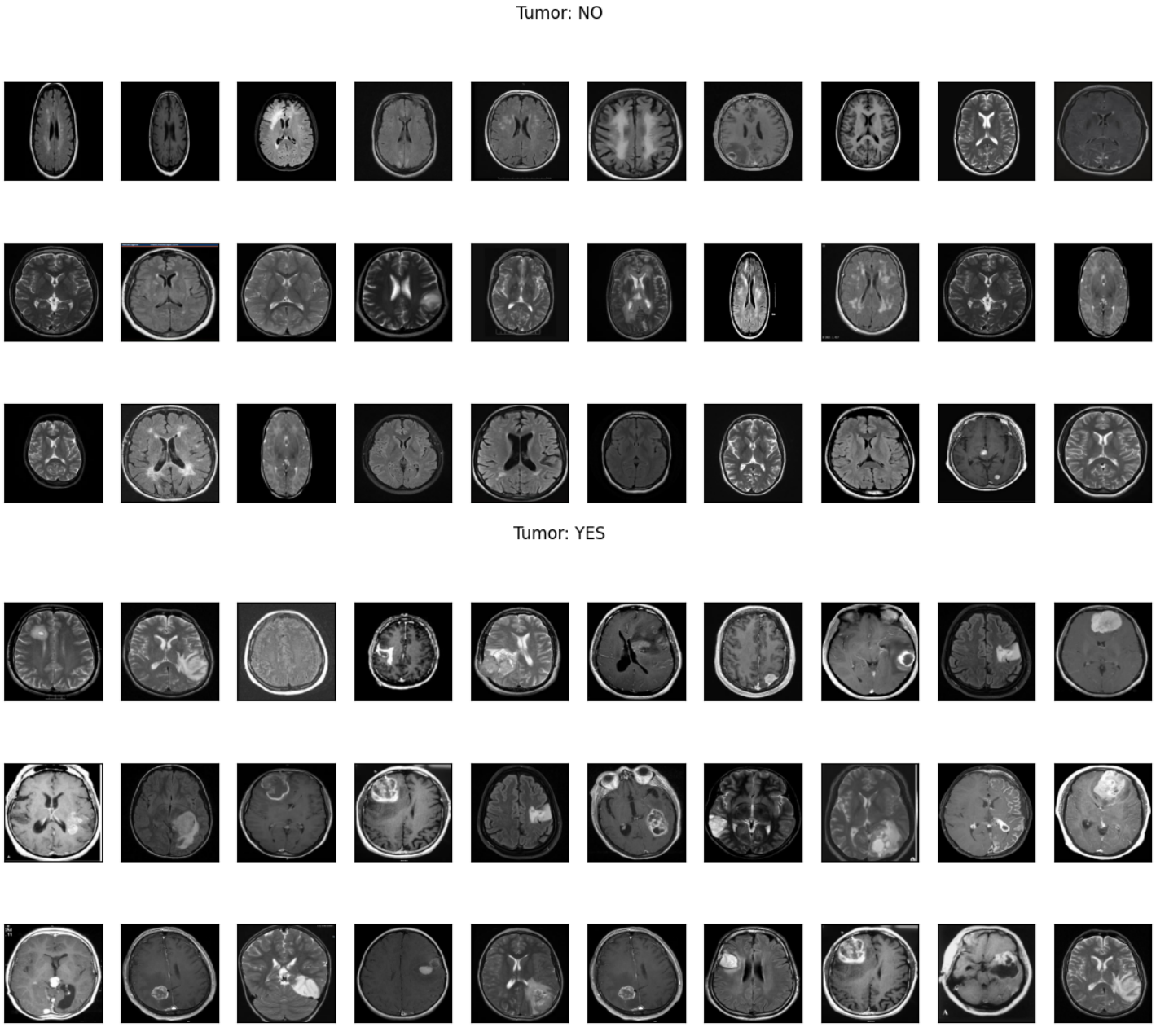
Let's take a look at the distribution of classes among sets:

```
y = dict()
y[0] = []
y[1] = []
for set_name in (y_train, y_val, y_test):
    y[0].append(np.sum(set_name == 0))
    y[1].append(np.sum(set_name == 1))

trace0 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[0],
    name='No',
    marker=dict(color='#33cc33'),
    opacity=0.7
)
trace1 = go.Bar(
    x=['Train Set', 'Validation Set', 'Test Set'],
    y=y[1],
    name='Yes',
    marker=dict(color='#ff3300'),
    opacity=0.7
)
data = [trace0, trace1]
layout = go.Layout(
    title='Count of classes in each set',
    xaxis={'title': 'Set'},
    yaxis={'title': 'Count'}
)
fig = go.Figure(data, layout)
iplot(fig)
```

```
print("Training labels:", np.unique(y_train))
print("Validation labels:", np.unique(y_val))
print("Test labels:", np.unique(y_test))
```

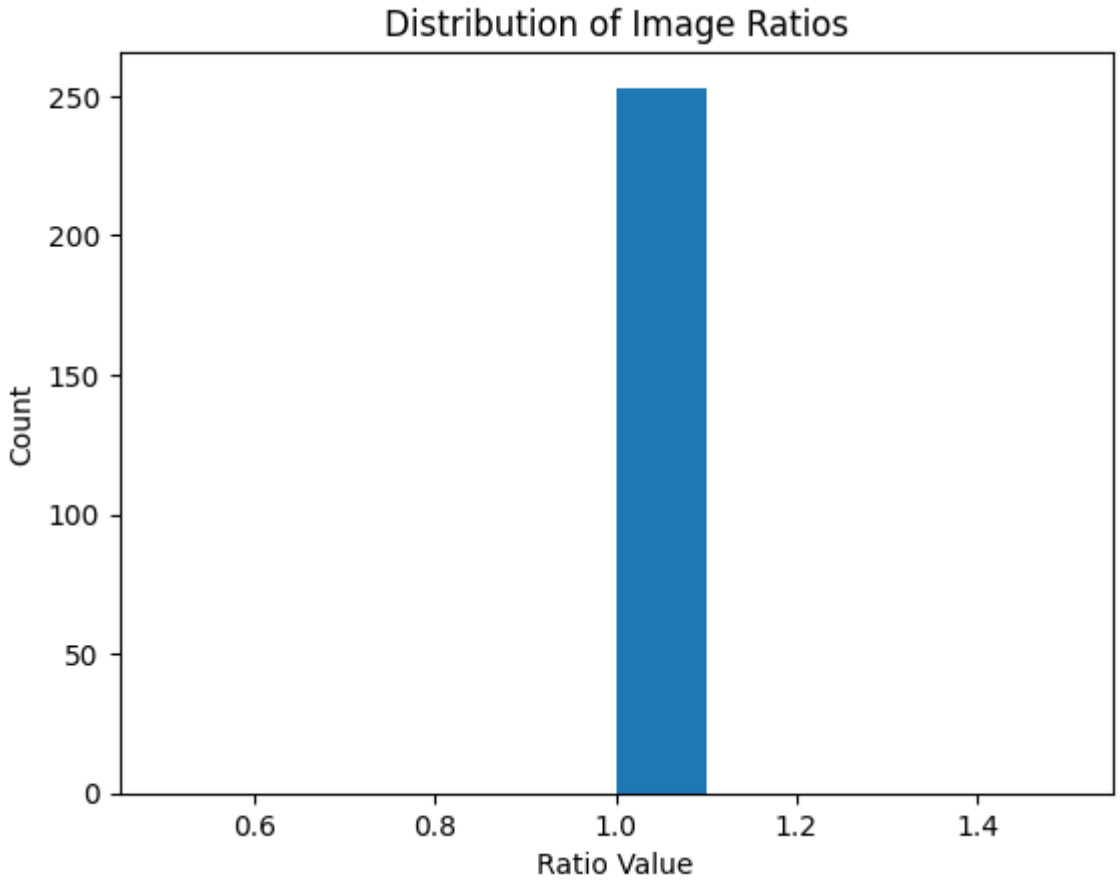
```
Training labels: [0 1]
Validation labels: [0 1]
Test labels: [0 1]
```

```
def plot_samples(X, y, labels_dict, n=50):
    """
    Creates a gridplot for desired number of images (n) from the specified set
    """
    for index in range(len(labels_dict)):
        imgs = X[np.argwhere(y == index)][:n]
        j = 10
        i = int(n/j)

        plt.figure(figsize=(15,6))
        c = 1
        for img in imgs:
            plt.subplot(i,j,c)
            plt.imshow(img[0])

            plt.xticks([])
            plt.yticks([])
            c += 1
        plt.suptitle('Tumor: {}'.format(labels_dict[index]))
        plt.show()

plot_samples(X_train, y_train, labels, 30)
```



As you can see, images have different `width` and `height` and diffent size of "black corners". Since the image size for VGG-16 imput layer is `(224,224)` some wide images may look weird after resizing. Histogram of ratio distributions (`ratio = width/height`):

```
RATIO_LIST = []
for set in (X_train, X_test, X_val):
   for img in set:
       RATIO_LIST.append(img.shape[1]/img.shape[0])

plt.hist(RATIO_LIST)
plt.title('Distribution of Image Ratios')
plt.xlabel('Ratio Value')
plt.ylabel('Count')
plt.show()
```



The first step of "normalization" would be to crop the brain out of the images. I used technique which was perfectly described in pyimagesearch blog and I highly suggest to looks deeper into it.

The images in the dataset have inconsistent shapes after cropping. This happens when the bounding box (calculated using extreme points) results in crops of different sizes for different images. To fix this, we need to ensure that all cropped images have the same shape.

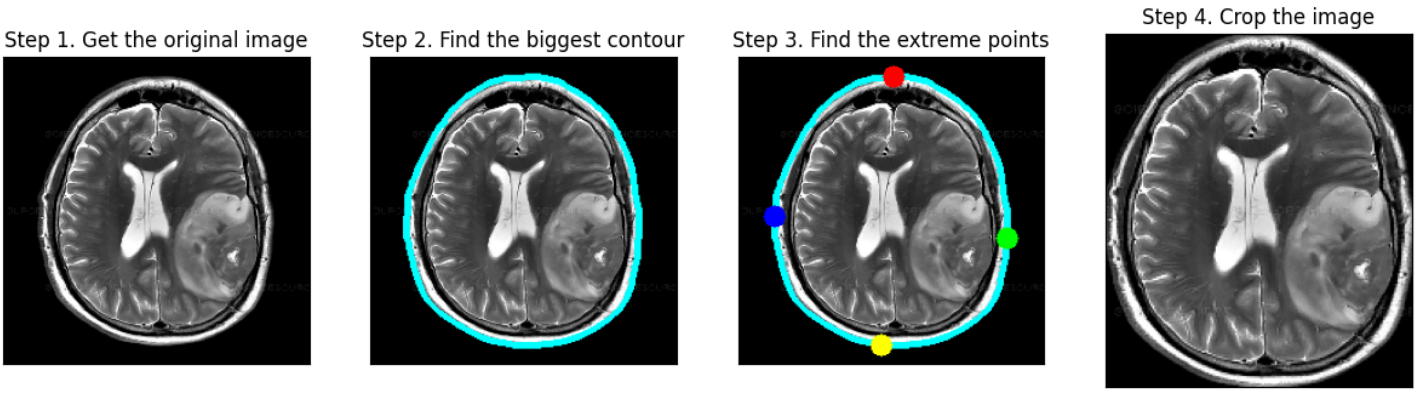
```
def crop_imgs(set_name, add_pixels_value=0, target_size=(224, 224)):
   """
   Finds the extreme points on the image and crops the rectangular out of them.
   Ensures all cropped images are resized to a consistent target size.
   """
   set_new = []
   for img in set_name:
       try:
           gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
           gray = cv2.GaussianBlur(gray, (5, 5), 0)

           # Threshold the image, then perform erosion + dilation
           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 the thresholded image
           cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
           cnts = imutils.grab_contours(cnts)

           if len(cnts) == 0:
```

```
                # No contours found, use the original image
                new_img = cv2.resize(img, target_size)
            else:
                # Find the largest contour
                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])

                ADD_PIXELS = add_pixels_value
                # Crop the image
                new_img = img[
                    max(extTop[1] - ADD_PIXELS, 0):min(extBot[1] + ADD_PIXELS, img.shape[0]),
                    max(extLeft[0] - ADD_PIXELS, 0):min(extRight[0] + ADD_PIXELS, img.shape[1])
                ].copy()

                # Resize to target size
                new_img = cv2.resize(new_img, target_size)

            set_new.append(new_img)
        except Exception as e:
            print(f"Error processing image: {str(e)}")
            # Append the original image if there's an issue
            set_new.append(cv2.resize(img, target_size))

    return np.array(set_new)
```

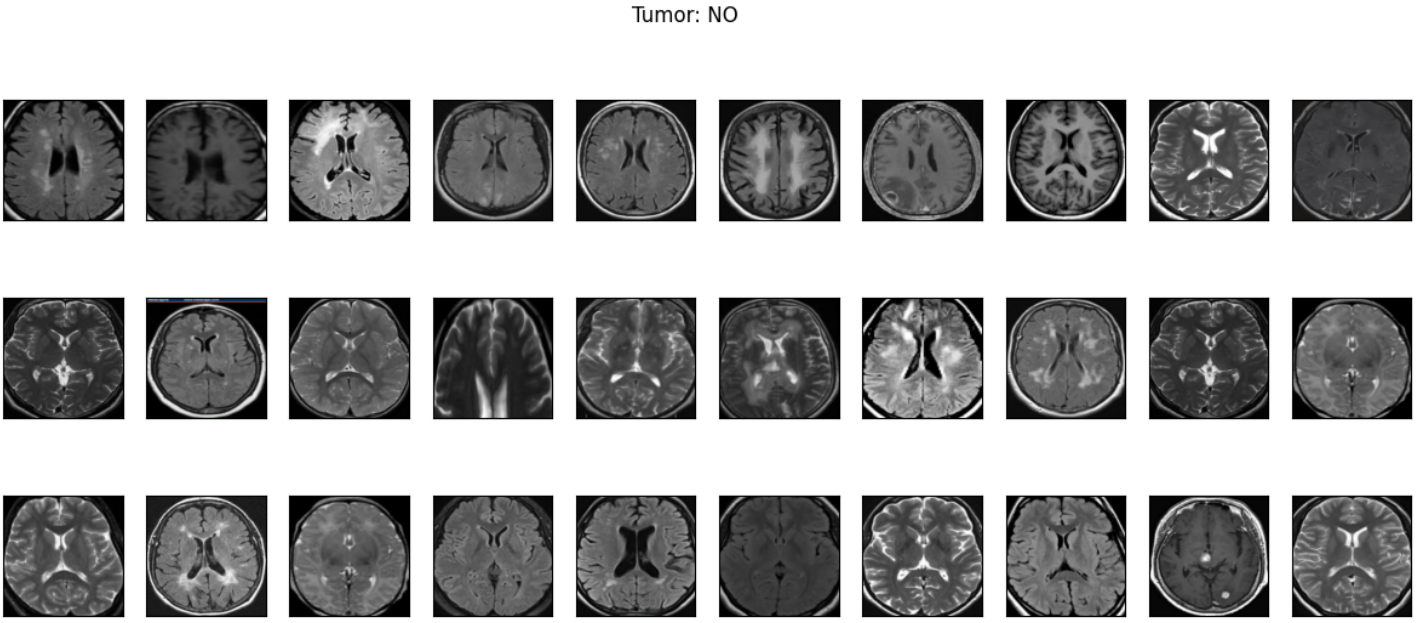
Let's look at example what this function will do with MRI scans:

```
img = cv2.imread('../input/brain-mri-images-for-brain-tumor-detection/brain_tumor_dataset/yes/Y108.jpg')
img = cv2.resize(
            img,
            dsize=IMG_SIZE,
            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])

# add contour on the image
img_cnt = cv2.drawContours(img.copy(), [c], -1, (0, 255, 255), 4)

# add extreme points
img_pnt = cv2.circle(img_cnt.copy(), extLeft, 8, (0, 0, 255), -1)
img_pnt = cv2.circle(img_pnt, extRight, 8, (0, 255, 0), -1)
img_pnt = cv2.circle(img_pnt, extTop, 8, (255, 0, 0), -1)
img_pnt = cv2.circle(img_pnt, extBot, 8, (255, 255, 0), -1)

# crop
ADD_PIXELS = 0
```

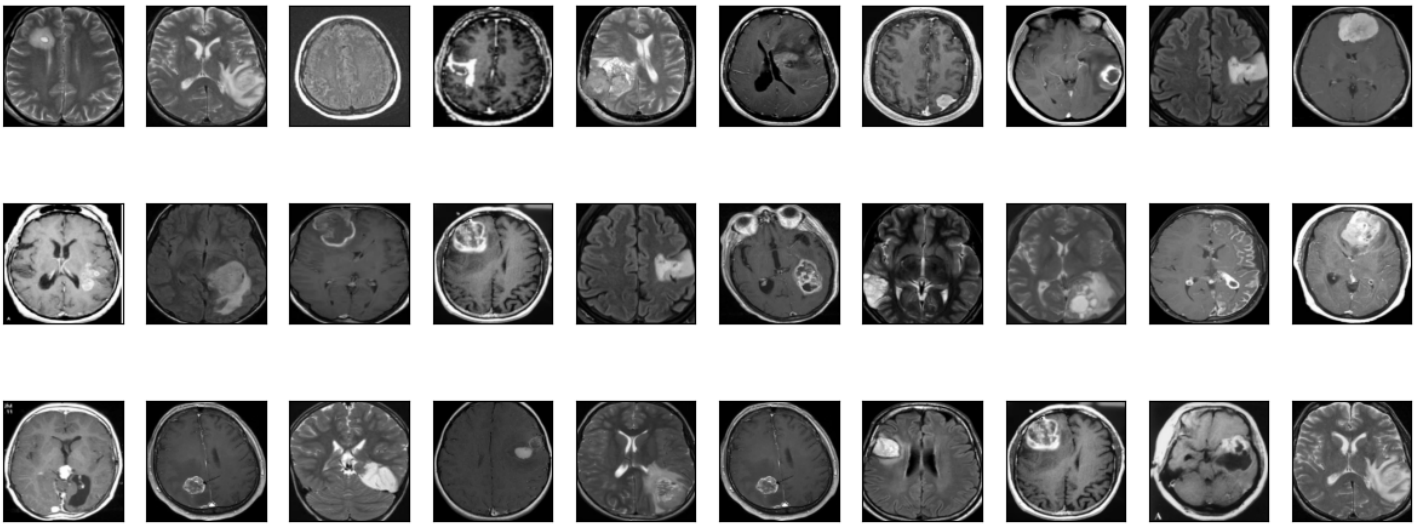
```
new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()

plt.figure(figsize=(15,6))
plt.subplot(141)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.title('Step 1. Get the original image')
plt.subplot(142)
plt.imshow(img_cnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 2. Find the biggest contour')
plt.subplot(143)
plt.imshow(img_pnt)
plt.xticks([])
plt.yticks([])
plt.title('Step 3. Find the extreme points')
plt.subplot(144)
plt.imshow(new_img)
plt.xticks([])
plt.yticks([])
plt.title('Step 4. Crop the image')
plt.show()
```




```
# apply this for each set
X_train_crop = crop_imgs(set_name=X_train)
X_val_crop = crop_imgs(set_name=X_val)
X_test_crop = crop_imgs(set_name=X_test)

plot_samples(X_train_crop, y_train, labels, 30)
```



Tumor: YES

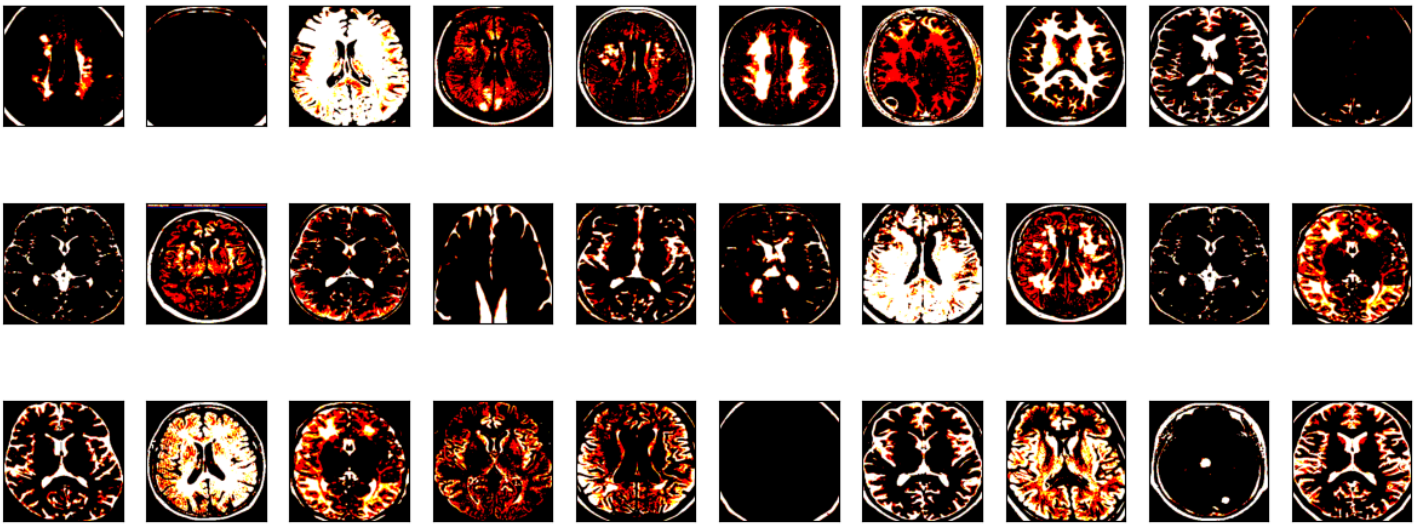


```
def save_new_images(x_set, y_set, folder_name):
   i = 0
   for (img, imclass) in zip(x_set, y_set):
       if imclass == 0:
           cv2.imwrite(folder_name+'NO/'+str(i)+'.jpg', img)
       else:
           cv2.imwrite(folder_name+'YES/'+str(i)+'.jpg', img)
       i += 1

# saving new images to the folder
!mkdir TRAIN_CROP TEST_CROP VAL_CROP TRAIN_CROP/YES TRAIN_CROP/NO TEST_CROP/YES TEST_CROP/NO VAL_CROP/YES VAL_CROP/NO

save_new_images(X_train_crop, y_train, folder_name='TRAIN_CROP/')
save_new_images(X_val_crop, y_val, folder_name='VAL_CROP/')
save_new_images(X_test_crop, y_test, folder_name='TEST_CROP/')
```

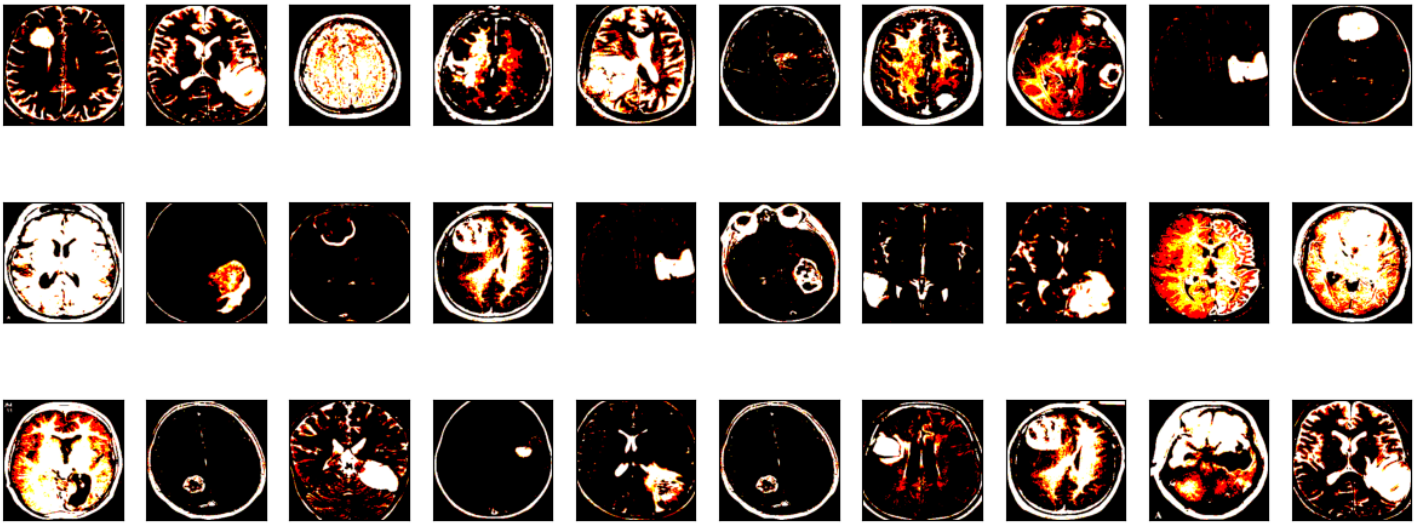
The next step would be resizing images to `(224,224)` and applying preprocessing needed for VGG-16 model input.

```
def preprocess_imgs(set_name, img_size):
   """
   Resize and apply VGG-15 preprocessing
   """
   set_new = []
   for img in set_name:
       img = cv2.resize(
           img,
           dsize=img_size,
           interpolation=cv2.INTER_CUBIC
       )
       set_new.append(preprocess_input(img))
   return np.array(set_new)

X_train_prep = preprocess_imgs(set_name=X_train_crop, img_size=IMG_SIZE)
X_test_prep = preprocess_imgs(set_name=X_test_crop, img_size=IMG_SIZE)
X_val_prep = preprocess_imgs(set_name=X_val_crop, img_size=IMG_SIZE)

plot_samples(X_train_prep, y_train, labels, 30)
```

Tumor: NO



Tumor: YES



4. CNN Model

I was using Transfer Learning with VGG-16 architecture and weights as a base model.

4.1. Data Augmentation

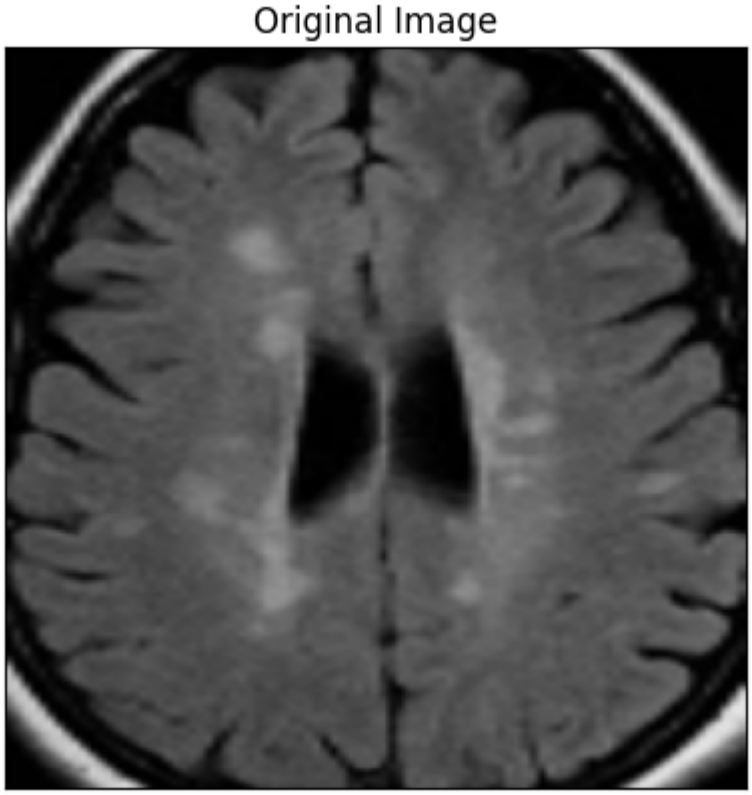
Since I had small data set I used the technique called Data Augmentation which helps to "increase" the size of training set.

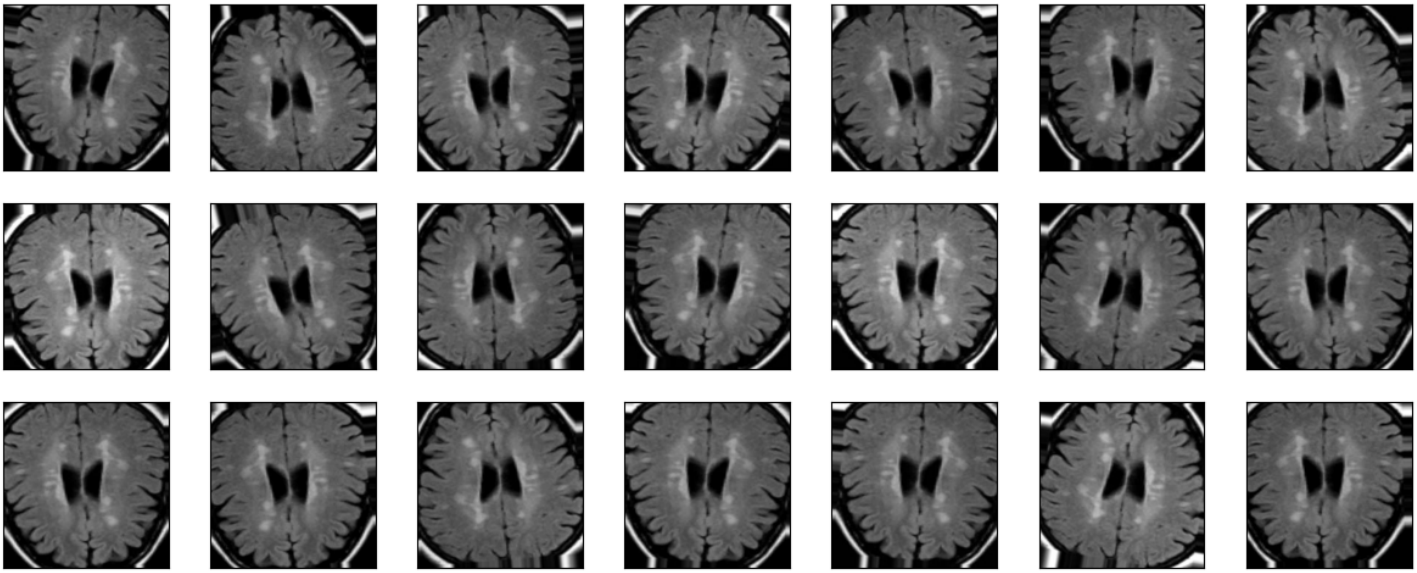
4.1.1. Demo

That's the example from one image how does augmentation look like.

```
# set the paramters we want to change randomly
demo_datagen = ImageDataGenerator(
   rotation_range=15,
   width_shift_range=0.05,
   height_shift_range=0.05,
   rescale=1./255,
   shear_range=0.05,
   brightness_range=[0.1, 1.5],
   horizontal_flip=True,
   vertical_flip=True
)
```

```
os.mkdir('preview')
x = X_train_crop[0]
x = x.reshape((1,) + x.shape)

i = 0
for batch in demo_datagen.flow(x, batch_size=1, save_to_dir='preview', save_prefix='aug_img', save_format='jpg'):
   i += 1
   if i > 20:
       break

# Display the original image
plt.imshow(X_train_crop[0])
plt.xticks([])
plt.yticks([])
plt.title('Original Image')
plt.show()

# Display augmented images
plt.figure(figsize=(15, 6))
i = 1
for img_name in os.listdir('preview/'):
   img_path = os.path.join('preview', img_name)  # Construct full path
   img = cv2.imread(img_path)  # Corrected: Use cv2.imread instead of cv2.cv2.imread
   img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
   plt.subplot(3, 7, i)
   plt.imshow(img)
   plt.xticks([])
   plt.yticks([])
   i += 1
   if i > 3 * 7:  # Limit to 21 images
       break

plt.suptitle('Augmented Images')
plt.show()
```






```
!rm -rf preview/
```

4.1.2. Apply

```
TRAIN_DIR = 'TRAIN_CROP/'
VAL_DIR = 'VAL_CROP/'

train_datagen = ImageDataGenerator(
   rotation_range=15,
   width_shift_range=0.1,
   height_shift_range=0.1,
   shear_range=0.1,
   brightness_range=[0.5, 1.5],
   horizontal_flip=True,
   vertical_flip=True,
   preprocessing_function=preprocess_input
)

test_datagen = ImageDataGenerator(
   preprocessing_function=preprocess_input
)


train_generator = train_datagen.flow_from_directory(
   TRAIN_DIR,
   color_mode='rgb',
   target_size=IMG_SIZE,
   batch_size=32,
   class_mode='binary',
   seed=RANDOM_SEED
)


validation_generator = test_datagen.flow_from_directory(
   VAL_DIR,
   color_mode='rgb',
   target_size=IMG_SIZE,
   batch_size=16,
   class_mode='binary',
   seed=RANDOM_SEED
)
```

```
Found 193 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
```

4.2. Model Building

```
# Load base model
vgg16_weight_path = '../input/keras-pretrained-models/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
base_model = VGG16(
   weights=vgg16_weight_path,
   include_top=False,
   input_shape=IMG_SIZE + (3,)
)

NUM_CLASSES = 1

# Create the model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES, activation='sigmoid'))

# Freeze the base model
model.layers[0].trainable = False

# Compile the model
model.compile(
   loss='binary_crossentropy',
   optimizer=RMSprop(learning_rate=1e-4),  # Fixed: Use 'learning_rate'
   metrics=['accuracy']
)

# Print model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten (Flatten)	(None, 25088)	0
dropout (Dropout)	(None, 25088)	0
dense (Dense)	(None, 1)	25,089

Total params: 14,739,777 (56.23 MB)

Trainable params: 25,089 (98.00 KB)

Non-trainable params: 14,714,688 (56.13 MB)

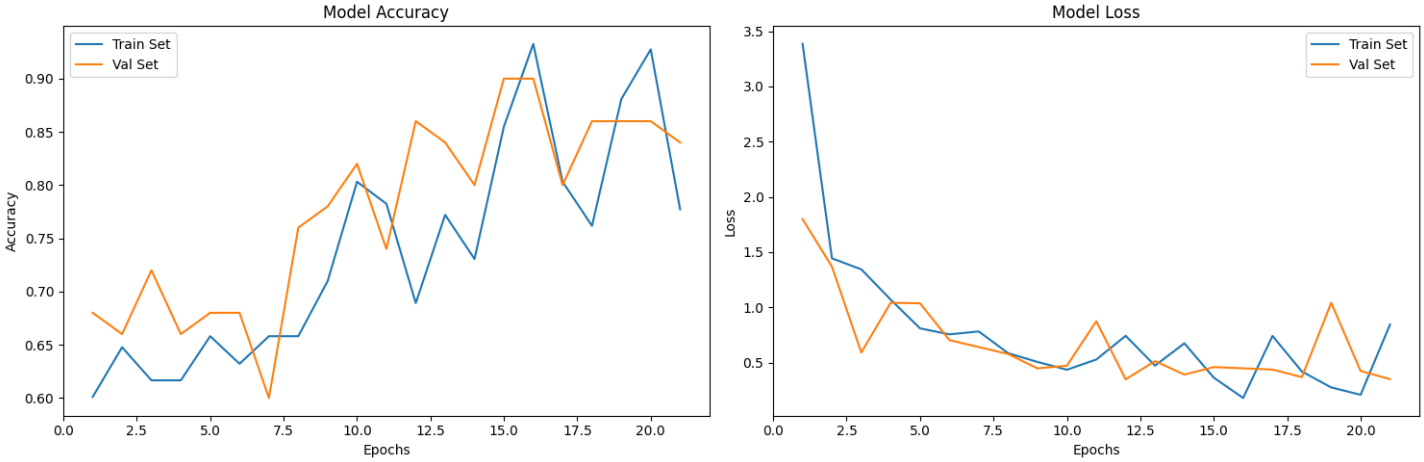
```
# Define Early Stopping
EPOCHS = 30
es = EarlyStopping(
   monitor='val_accuracy',  # Use 'val_accuracy' instead of 'val_acc'
   mode='max',
   patience=6
)

# Train the model
history = model.fit(
   train_generator,
   steps_per_epoch=50,
   epochs=EPOCHS,
   validation_data=validation_generator,
   validation_steps=25,
   callbacks=[es]
)

Epoch 1/30

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning:

Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_mul


 7/50    6:44 9s/step - accuracy: 0.6238 - loss: 4.3310
```

```
/usr/lib/python3.10/contextlib.py:153: UserWarning:

Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_e


50/50   93s 2s/step - accuracy: 0.6042 - loss: 3.5184 - val_accuracy: 0.6800 - val_loss: 1.7987
Epoch 2/30
50/50   85s 1s/step - accuracy: 0.6461 - loss: 1.4822 - val_accuracy: 0.6600 - val_loss: 1.3685
Epoch 3/30
50/50   85s 2s/step - accuracy: 0.6259 - loss: 1.3460 - val_accuracy: 0.7200 - val_loss: 0.5911
Epoch 4/30
50/50   91s 2s/step - accuracy: 0.6208 - loss: 1.0745 - val_accuracy: 0.6600 - val_loss: 1.0419
Epoch 5/30
50/50   85s 1s/step - accuracy: 0.6580 - loss: 0.8171 - val_accuracy: 0.6800 - val_loss: 1.0366
Epoch 6/30
50/50   148s 2s/step - accuracy: 0.6358 - loss: 0.7589 - val_accuracy: 0.6800 - val_loss: 0.7035
Epoch 7/30
50/50   85s 2s/step - accuracy: 0.6672 - loss: 0.7615 - val_accuracy: 0.6000 - val_loss: 0.6402
Epoch 8/30
50/50   141s 1s/step - accuracy: 0.6585 - loss: 0.5835 - val_accuracy: 0.7600 - val_loss: 0.5775
Epoch 9/30
50/50   91s 2s/step - accuracy: 0.7150 - loss: 0.4955 - val_accuracy: 0.7800 - val_loss: 0.4472
Epoch 10/30
50/50   85s 1s/step - accuracy: 0.8039 - loss: 0.4334 - val_accuracy: 0.8200 - val_loss: 0.4719
Epoch 11/30
50/50   84s 1s/step - accuracy: 0.7815 - loss: 0.5260 - val_accuracy: 0.7400 - val_loss: 0.8748
Epoch 12/30
50/50   85s 1s/step - accuracy: 0.6945 - loss: 0.7430 - val_accuracy: 0.8600 - val_loss: 0.3485
Epoch 13/30
50/50   84s 1s/step - accuracy: 0.7730 - loss: 0.4678 - val_accuracy: 0.8400 - val_loss: 0.5127
Epoch 14/30
50/50   85s 1s/step - accuracy: 0.7389 - loss: 0.6739 - val_accuracy: 0.8000 - val_loss: 0.3912
Epoch 15/30
50/50   85s 1s/step - accuracy: 0.8545 - loss: 0.3645 - val_accuracy: 0.9000 - val_loss: 0.4590
Epoch 16/30
50/50   142s 1s/step - accuracy: 0.9320 - loss: 0.1778 - val_accuracy: 0.9000 - val_loss: 0.4482
Epoch 17/30
50/50   141s 2s/step - accuracy: 0.8130 - loss: 0.7049 - val_accuracy: 0.8000 - val_loss: 0.4366
Epoch 18/30
50/50   85s 1s/step - accuracy: 0.7536 - loss: 0.4282 - val_accuracy: 0.8600 - val_loss: 0.3685
Epoch 19/30
50/50   84s 2s/step - accuracy: 0.8833 - loss: 0.2700 - val_accuracy: 0.8600 - val_loss: 1.0410
Epoch 20/30
50/50   85s 1s/step - accuracy: 0.9252 - loss: 0.2097 - val_accuracy: 0.8600 - val_loss: 0.4238
Epoch 21/30
50/50   84s 2s/step - accuracy: 0.7929 - loss: 0.8037 - val_accuracy: 0.8400 - val_loss: 0.3496
```

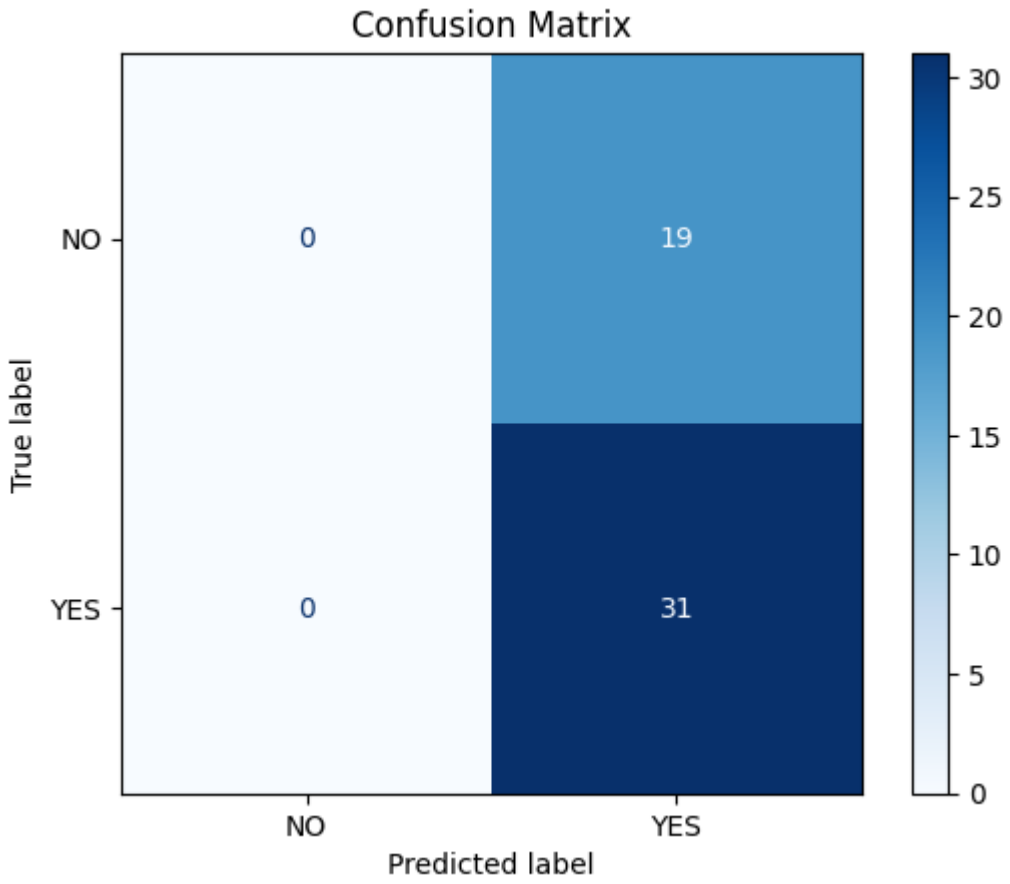
4.3. Model Performance

```
# plot model performance
acc = history.history['accuracy']  # Use 'accuracy' instead of 'acc'
val_acc = history.history['val_accuracy']  # Use 'val_accuracy' instead of 'val_acc'
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(1, len(history.epoch) + 1)

plt.figure(figsize=(15,5))

plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Train Set')
plt.plot(epochs_range, val_acc, label='Val Set')
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Train Set')
plt.plot(epochs_range, val_loss, label='Val Set')
```

```
plt.legend(loc="best")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')

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

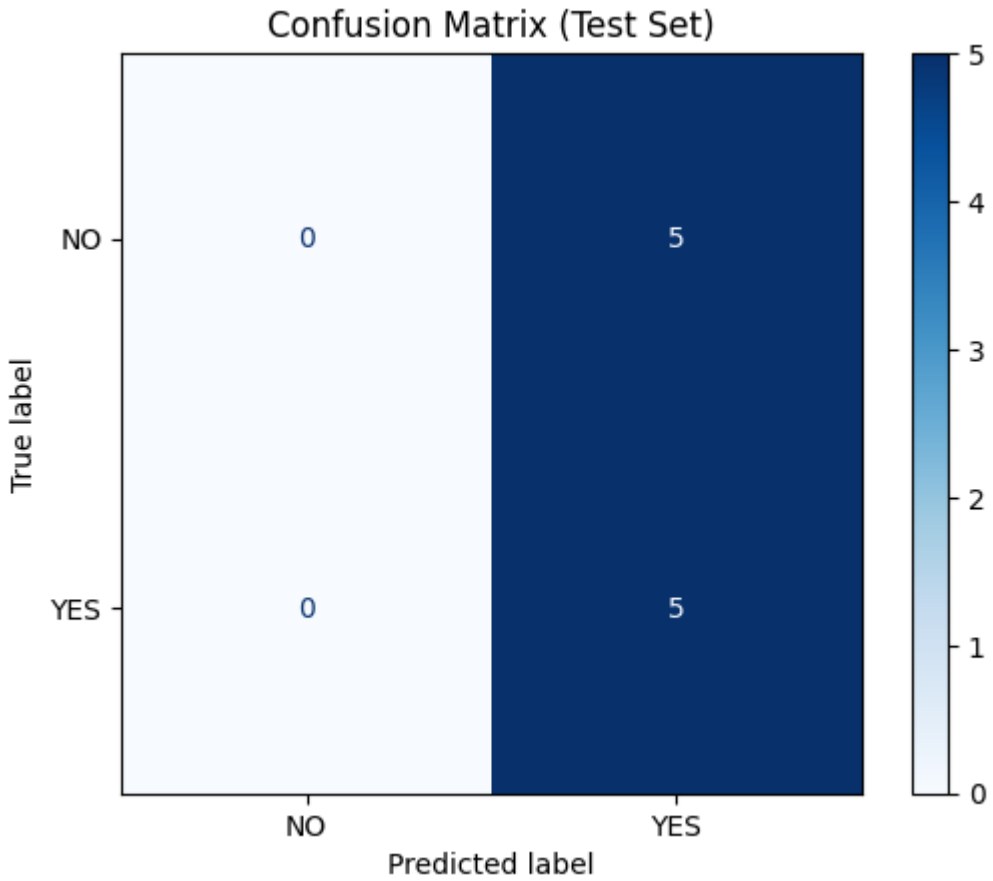



```
from sklearn.metrics import ConfusionMatrixDisplay
# Validate on validation set
predictions = model.predict(X_val_prep)
predictions = [1 if x > 0.5 else 0 for x in predictions]

# Calculate accuracy
accuracy = accuracy_score(y_val, predictions)
print('Val Accuracy = %.2f' % accuracy)

# Compute confusion matrix
confusion_mtx = confusion_matrix(y_val, predictions)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_mtx, display_labels=list(labels.values()))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()

2/2   14s 5s/step
Val Accuracy = 0.62
```




```
# Validate on test set
predictions = model.predict(X_test_prep)
predictions = [1 if x > 0.5 else 0 for x in predictions]

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print('Test Accuracy = %.2f' % accuracy)

# Compute confusion matrix
confusion_mtx = confusion_matrix(y_test, predictions)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_mtx, display_labels=list(labels.values()))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix (Test Set)')
plt.show()
```

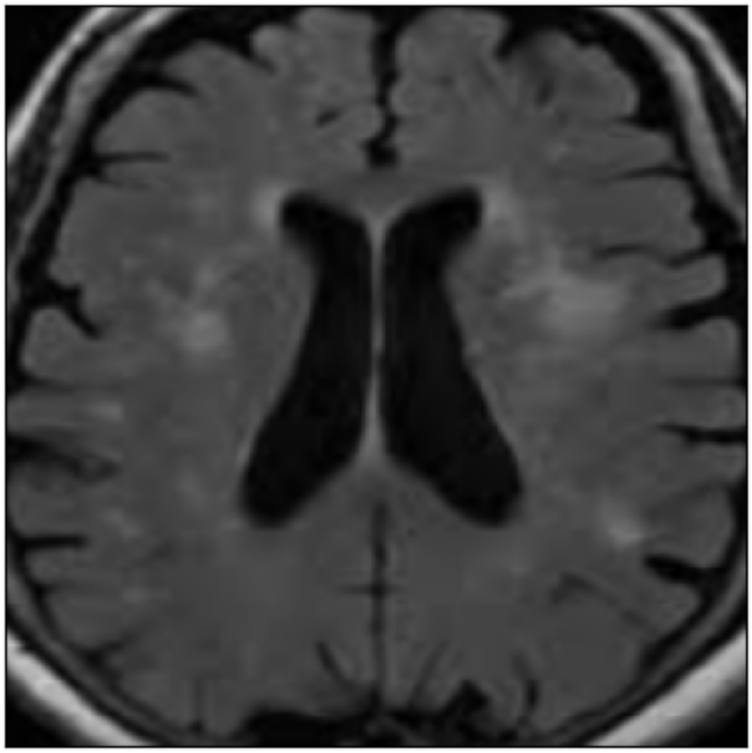
```
1/1   3s 3s/step
Test Accuracy = 0.50
```



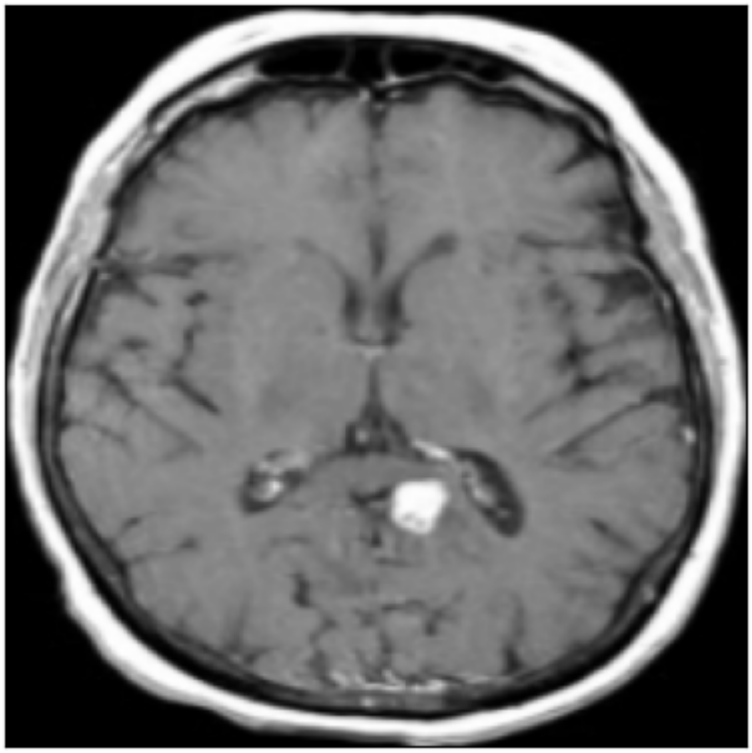
Now let's take a look at the images from the `test set` that were misclassified:

```
ind_list = np.argwhere((y_test == predictions) == False)[:, -1]
if ind_list.size == 0:
   print('There are no missclassified images.')
else:
   for i in ind_list:
       plt.figure()
       plt.imshow(X_test_crop[i])
       plt.xticks([])
       plt.yticks([])
       plt.title(f'Actual class: {y_val[i]}\nPredicted class: {predictions[i]}')
       plt.show()
```

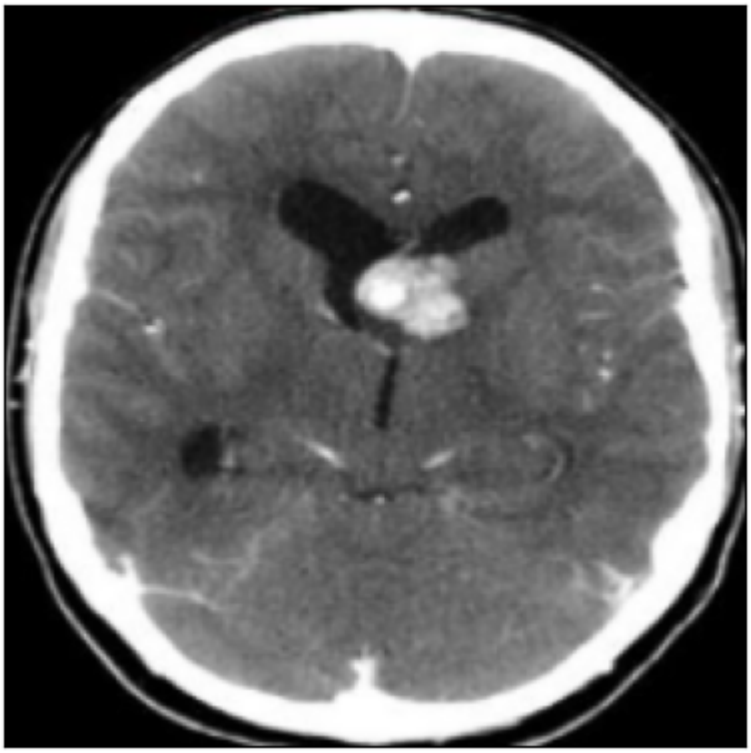
Actual class: 0
Predicted class: 1



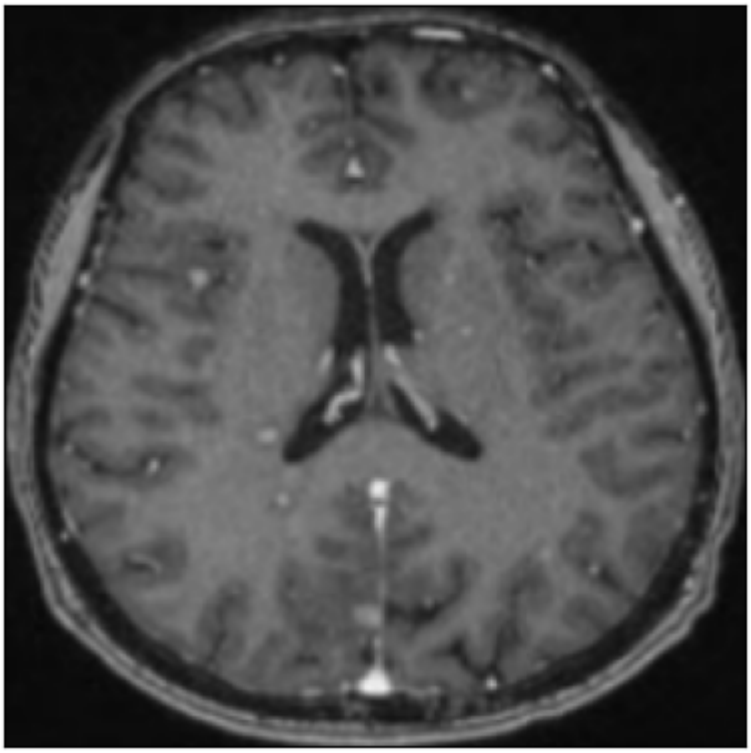
Actual class: 0
Predicted class: 1

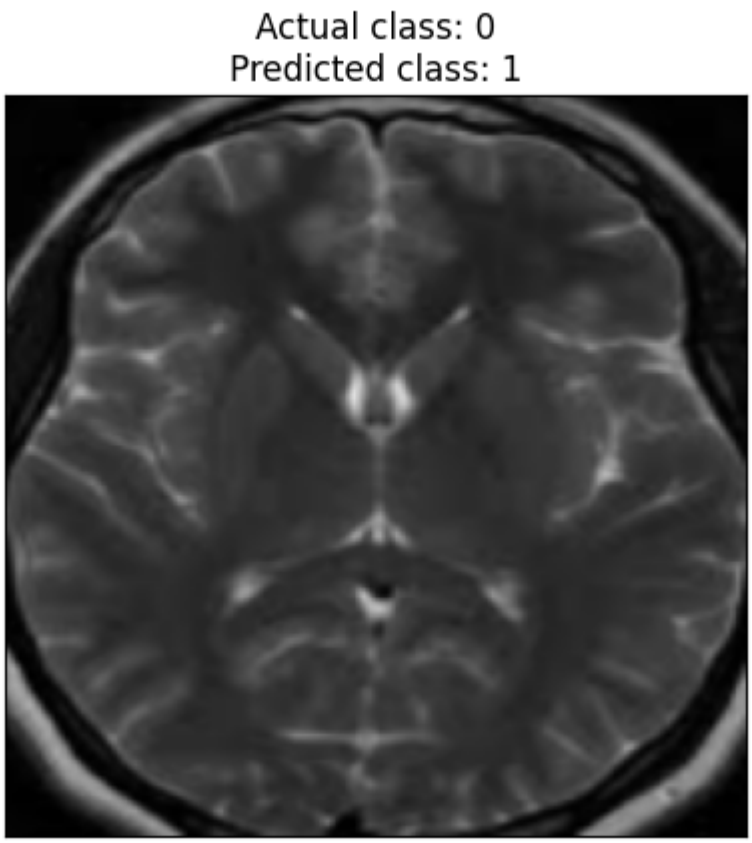


Actual class: 0
Predicted class: 1



Actual class: 0
Predicted class: 1





First scan looks a bit misleading - what is that light spot in the middle? I can clearly see why model classified it as `Tumor`.

5. Conclusions

This project was a combination of CNN model classification problem (to predict wheter the subject has brain tumor or not) & Computer Vision problem (to automate the process of brain cropping from MRI scans). The final accuracy is much higher than 50% baseline (random guess). However, it could be increased by larger number of train images or through model hyperparameters tuning.

```
# clean up the space
!rm -rf TRAIN TEST VAL TRAIN_CROP TEST_CROP VAL_CROP
# save the model
model.save('2019-06-07_VGG_model.h5')
```

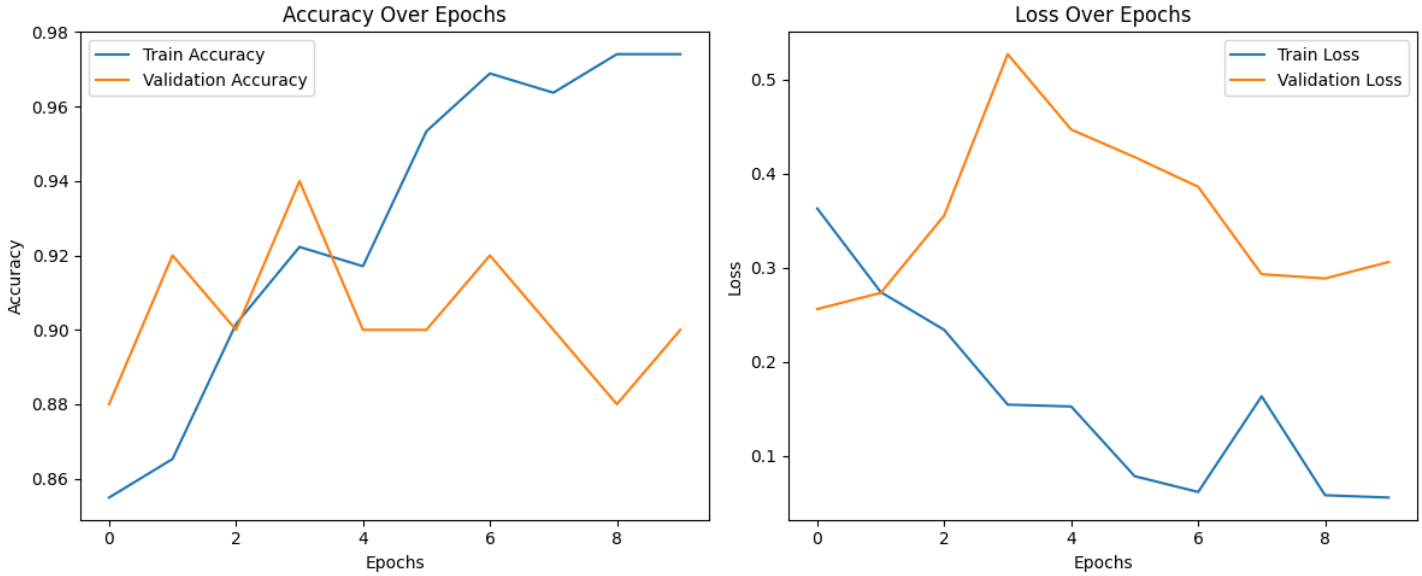
Improving Accuracy

```
train_datagen = ImageDataGenerator(
   rotation_range=30,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   vertical_flip=True,
   brightness_range=[0.5, 1.5],
   fill_mode='nearest'
)

X_train_prep = X_train_crop / 255.0
X_val_prep = X_val_crop / 255.0
X_test_prep = X_test_crop / 255.0

# Unfreeze the last few layers of VGG16
for layer in base_model.layers[-4:]:
   layer.trainable = True

# Recompile the model
model.compile(
   loss='binary_crossentropy',
   optimizer=RMSprop(learning_rate=1e-5),  # Lower learning rate for fine-tuning
```

```
   metrics=['accuracy']
)

model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(NUM_CLASSES, activation='sigmoid'))

model.compile(
   loss='binary_crossentropy',
   optimizer=RMSprop(learning_rate=1e-4),
   metrics=['accuracy']
)

from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(
   class_weight='balanced',
   classes=np.unique(y_train),
   y=y_train
)
class_weights_dict = dict(enumerate(class_weights))

history = model.fit(
   train_generator,
   steps_per_epoch=50,
   epochs=EPOCHS,
   validation_data=validation_generator,
   validation_steps=25,
   callbacks=[es],
   class_weight=class_weights_dict
)

Epoch 1/30
7/50    8:03 11s/step - accuracy: 0.6462 - loss: 0.8602

/usr/lib/python3.10/contextlib.py:153: UserWarning:

Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_e


50/50   87s 2s/step - accuracy: 0.6519 - loss: 0.8238 - val_accuracy: 0.8000 - val_loss: 0.3305
Epoch 2/30
50/50   87s 1s/step - accuracy: 0.6827 - loss: 0.8214 - val_accuracy: 0.8600 - val_loss: 0.3870
Epoch 3/30
50/50   84s 1s/step - accuracy: 0.7288 - loss: 0.9240 - val_accuracy: 0.8600 - val_loss: 0.4361
Epoch 4/30
50/50   91s 2s/step - accuracy: 0.6968 - loss: 0.5351 - val_accuracy: 0.8600 - val_loss: 0.4072
Epoch 5/30
50/50   85s 1s/step - accuracy: 0.8467 - loss: 0.4448 - val_accuracy: 0.8800 - val_loss: 0.3350
Epoch 6/30
50/50   84s 1s/step - accuracy: 0.8867 - loss: 0.2312 - val_accuracy: 0.8400 - val_loss: 0.4044

from tensorflow.keras.optimizers import Adam

model.compile(
   loss='binary_crossentropy',
   optimizer=Adam(learning_rate=1e-4),
   metrics=['accuracy']
)

model.add(Dropout(0.5))  # After dense layers

train_datagen = ImageDataGenerator(
   rotation_range=20,
   width_shift_range=0.2,
```

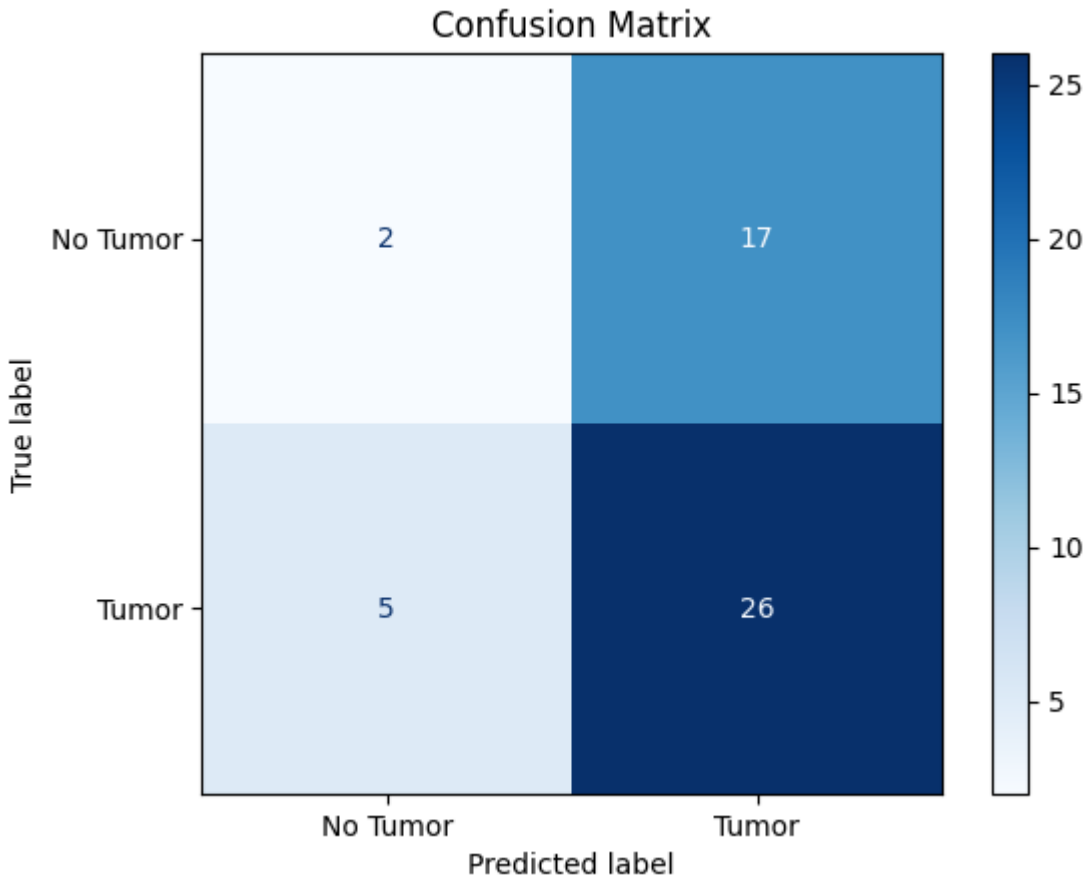
```
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   fill_mode='nearest'
)

es = EarlyStopping(
   monitor='val_loss',  # Stop when validation loss stops improving
   patience=5,
   restore_best_weights=True
)

for layer in base_model.layers[-10:]:  # Unfreeze last 10 layers
   layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5))  # Smaller LR for fine-tuning

class_weights = class_weight.compute_class_weight(
   'balanced',
   classes=np.unique(y_train),
   y=y_train
)

from tensorflow.keras.applications import EfficientNetB0
base_model = EfficientNetB0(
   include_top=False,
   input_shape=IMG_SIZE + (3,),
   weights='imagenet'
)

Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208   1s 0us/step

from tensorflow.keras.applications import EfficientNetB0

base_model = EfficientNetB0(
   include_top=False,
   input_shape=IMG_SIZE + (3,),
   weights='imagenet'
)

model = Sequential([
   base_model,
   layers.GlobalAveragePooling2D(),
   layers.Dense(512, activation='relu'),
   layers.BatchNormalization(),
   layers.Dropout(0.5),
   layers.Dense(1, activation='sigmoid')
])

model.compile(
   loss='binary_crossentropy',
   optimizer=Adam(learning_rate=1e-4),
   metrics=['accuracy']
)

# Plot accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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




```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate predictions
y_pred = model.predict(validation_generator)
y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary labels

# Compute confusion matrix
cm = confusion_matrix(validation_generator.classes, y_pred)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No Tumor', 'Tumor'])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

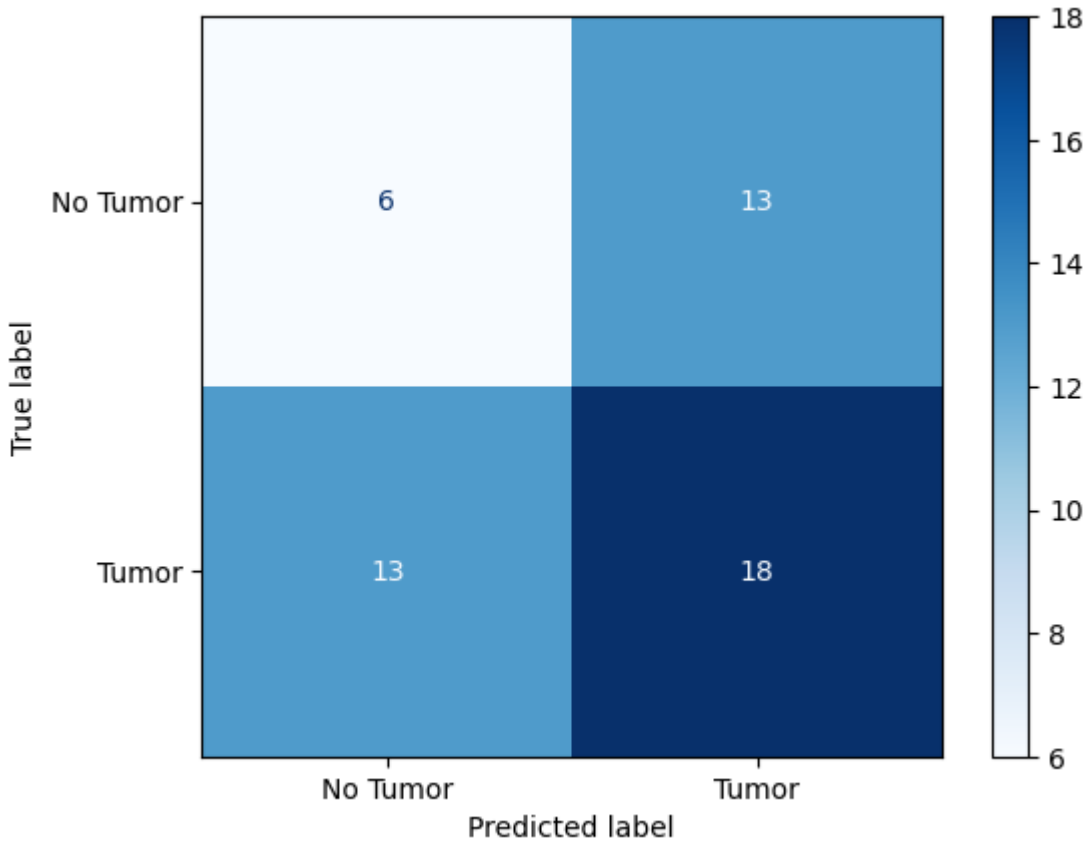
```
4/4   6s 1s/step
```




```
# Train the model
history = model.fit(
   train_generator,
   steps_per_epoch=50,
   epochs=EPOCHS,
   validation_data=validation_generator,
   validation_steps=25,
   callbacks=[es, lr_scheduler]
)

# Plot training results
plt.figure(figsize=(12, 5))

# Evaluate on the validation set
y_pred = model.predict(validation_generator)
y_pred = (y_pred > 0.5).astype(int)
cm = confusion_matrix(validation_generator.classes, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=['No Tumor', 'Tumor'])
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

```
Epoch 1/30
7/50    3:45 5s/step - accuracy: 0.5597 - loss: 0.8672

/usr/lib/python3.10/contextlib.py:153: UserWarning:

Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_e


50/50   91s 738ms/step - accuracy: 0.5507 - loss: 0.9093 - val_accuracy: 0.6200 - val_loss: 0.6581 - learning_rate: 1.0000e-04
Epoch 2/30
50/50   44s 831ms/step - accuracy: 0.7335 - loss: 0.5931 - val_accuracy: 0.6800 - val_loss: 0.6369 - learning_rate: 1.0000e-04
Epoch 3/30
50/50   42s 675ms/step - accuracy: 0.7827 - loss: 0.4677 - val_accuracy: 0.6800 - val_loss: 0.6162 - learning_rate: 1.0000e-04
Epoch 4/30
50/50   42s 670ms/step - accuracy: 0.7852 - loss: 0.4750 - val_accuracy: 0.7200 - val_loss: 0.5976 - learning_rate: 1.0000e-04
Epoch 5/30
50/50   40s 646ms/step - accuracy: 0.8290 - loss: 0.3637 - val_accuracy: 0.7800 - val_loss: 0.5822 - learning_rate: 1.0000e-04
```

```
Epoch 6/30
50/50   40s 638ms/step - accuracy: 0.8719 - loss: 0.3220 - val_accuracy: 0.8000 - val_loss: 0.5561 - learning_rate: 1.0000e-04
Epoch 7/30
50/50   40s 634ms/step - accuracy: 0.9084 - loss: 0.2651 - val_accuracy: 0.8200 - val_loss: 0.5344 - learning_rate: 1.0000e-04
Epoch 8/30
50/50   68s 1s/step - accuracy: 0.9128 - loss: 0.2719 - val_accuracy: 0.8200 - val_loss: 0.5212 - learning_rate: 1.0000e-04
Epoch 9/30
50/50   41s 645ms/step - accuracy: 0.9230 - loss: 0.2155 - val_accuracy: 0.8200 - val_loss: 0.5009 - learning_rate: 1.0000e-04
Epoch 10/30
50/50   40s 645ms/step - accuracy: 0.9239 - loss: 0.2014 - val_accuracy: 0.8200 - val_loss: 0.4909 - learning_rate: 1.0000e-04
Epoch 11/30
50/50   41s 649ms/step - accuracy: 0.9388 - loss: 0.1884 - val_accuracy: 0.8200 - val_loss: 0.4838 - learning_rate: 1.0000e-04
Epoch 12/30
50/50   41s 655ms/step - accuracy: 0.9685 - loss: 0.1296 - val_accuracy: 0.8000 - val_loss: 0.4846 - learning_rate: 1.0000e-04
Epoch 13/30
50/50   41s 651ms/step - accuracy: 0.9740 - loss: 0.0867 - val_accuracy: 0.8200 - val_loss: 0.4768 - learning_rate: 1.0000e-04
Epoch 14/30
50/50   41s 664ms/step - accuracy: 0.9684 - loss: 0.0884 - val_accuracy: 0.8200 - val_loss: 0.4586 - learning_rate: 1.0000e-04
Epoch 15/30
50/50   41s 778ms/step - accuracy: 0.9301 - loss: 0.1480 - val_accuracy: 0.8800 - val_loss: 0.4170 - learning_rate: 1.0000e-04
Epoch 16/30
50/50   44s 714ms/step - accuracy: 0.9881 - loss: 0.0474 - val_accuracy: 0.8800 - val_loss: 0.3943 - learning_rate: 1.0000e-04
Epoch 17/30
50/50   79s 653ms/step - accuracy: 0.9491 - loss: 0.1344 - val_accuracy: 0.8800 - val_loss: 0.3843 - learning_rate: 1.0000e-04
Epoch 18/30
50/50   41s 650ms/step - accuracy: 0.9642 - loss: 0.0911 - val_accuracy: 0.8800 - val_loss: 0.3825 - learning_rate: 1.0000e-04
Epoch 19/30
50/50   41s 769ms/step - accuracy: 0.9486 - loss: 0.1590 - val_accuracy: 0.8800 - val_loss: 0.3755 - learning_rate: 1.0000e-04
Epoch 20/30
50/50   41s 659ms/step - accuracy: 0.9898 - loss: 0.0407 - val_accuracy: 0.8800 - val_loss: 0.3652 - learning_rate: 1.0000e-04
Epoch 21/30
50/50   40s 640ms/step - accuracy: 0.9488 - loss: 0.0951 - val_accuracy: 0.8800 - val_loss: 0.3444 - learning_rate: 1.0000e-04
Epoch 22/30
50/50   66s 1s/step - accuracy: 0.9648 - loss: 0.0508 - val_accuracy: 0.8800 - val_loss: 0.3298 - learning_rate: 1.0000e-04
Epoch 23/30
50/50   42s 669ms/step - accuracy: 0.9946 - loss: 0.0369 - val_accuracy: 0.8800 - val_loss: 0.3185 - learning_rate: 1.0000e-04
Epoch 24/30
50/50   42s 664ms/step - accuracy: 0.9798 - loss: 0.0586 - val_accuracy: 0.8800 - val_loss: 0.3073 - learning_rate: 1.0000e-04
Epoch 25/30
50/50   41s 659ms/step - accuracy: 0.9848 - loss: 0.0610 - val_accuracy: 0.8800 - val_loss: 0.2965 - learning_rate: 1.0000e-04
Epoch 26/30
50/50   41s 654ms/step - accuracy: 0.9731 - loss: 0.0682 - val_accuracy: 0.8800 - val_loss: 0.2884 - learning_rate: 1.0000e-04
Epoch 27/30
50/50   45s 735ms/step - accuracy: 0.9795 - loss: 0.0571 - val_accuracy: 0.9000 - val_loss: 0.2827 - learning_rate: 1.0000e-04
Epoch 28/30
50/50   42s 789ms/step - accuracy: 0.9690 - loss: 0.0546 - val_accuracy: 0.9200 - val_loss: 0.2729 - learning_rate: 1.0000e-04
Epoch 29/30
50/50   41s 653ms/step - accuracy: 0.9734 - loss: 0.0767 - val_accuracy: 0.9200 - val_loss: 0.2641 - learning_rate: 1.0000e-04
Epoch 30/30
50/50   41s 654ms/step - accuracy: 0.9952 - loss: 0.0418 - val_accuracy: 0.9200 - val_loss: 0.2535 - learning_rate: 1.0000e-04
4/4   2s 384ms/step

<Figure size 1200x500 with 0 Axes>
```




```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(
   'balanced',
   classes=np.unique(y_train),
   y=y_train
)

from tensorflow.keras.applications import EfficientNetB0

base_model = EfficientNetB0(
   include_top=False,
   input_shape=IMG_SIZE + (3,),
   weights='imagenet'
)

from tensorflow.keras.layers import BatchNormalization

model.add(Dropout(0.5))
model.add(BatchNormalization())

train_datagen = ImageDataGenerator(
   rotation_range=30,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   vertical_flip=True,
   brightness_range=[0.5, 1.5]
)

es = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6)

# Load base model
base_model = VGG16(
   include_top=False,
   input_shape=IMG_SIZE + (3,),
   weights='imagenet'
```

```
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kern
58889256/58889256   2s 0us/step
```

```
# Create the model
model = Sequential([
   base_model,
   Flatten(),
   Dense(512, activation='relu'),
   BatchNormalization(),  # Add batch normalization
   Dropout(0.5),          # Add dropout for regularization
   Dense(256, activation='relu'),
   BatchNormalization(),  # Add another batch normalization layer
   Dropout(0.5),
   Dense(1, activation='sigmoid')  # Output layer
])
```

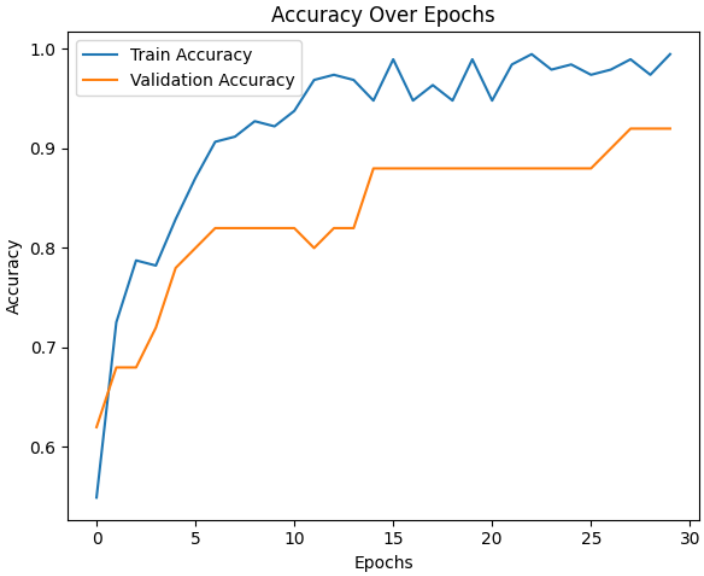
```
# Freeze the base model
base_model.trainable = False
```

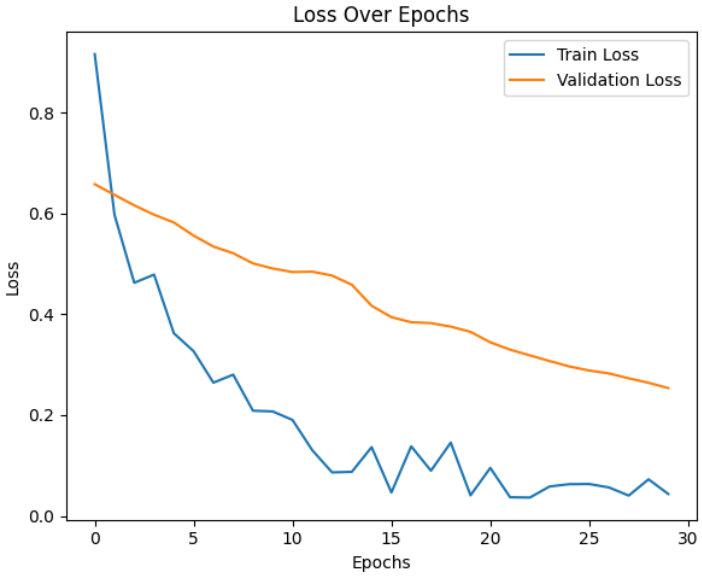
```
# Compile the model
model.compile(
   loss='binary_crossentropy',
   optimizer='adam',
   metrics=['accuracy']
)
```

```
import matplotlib.pyplot as plt
```

```
# Plot accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

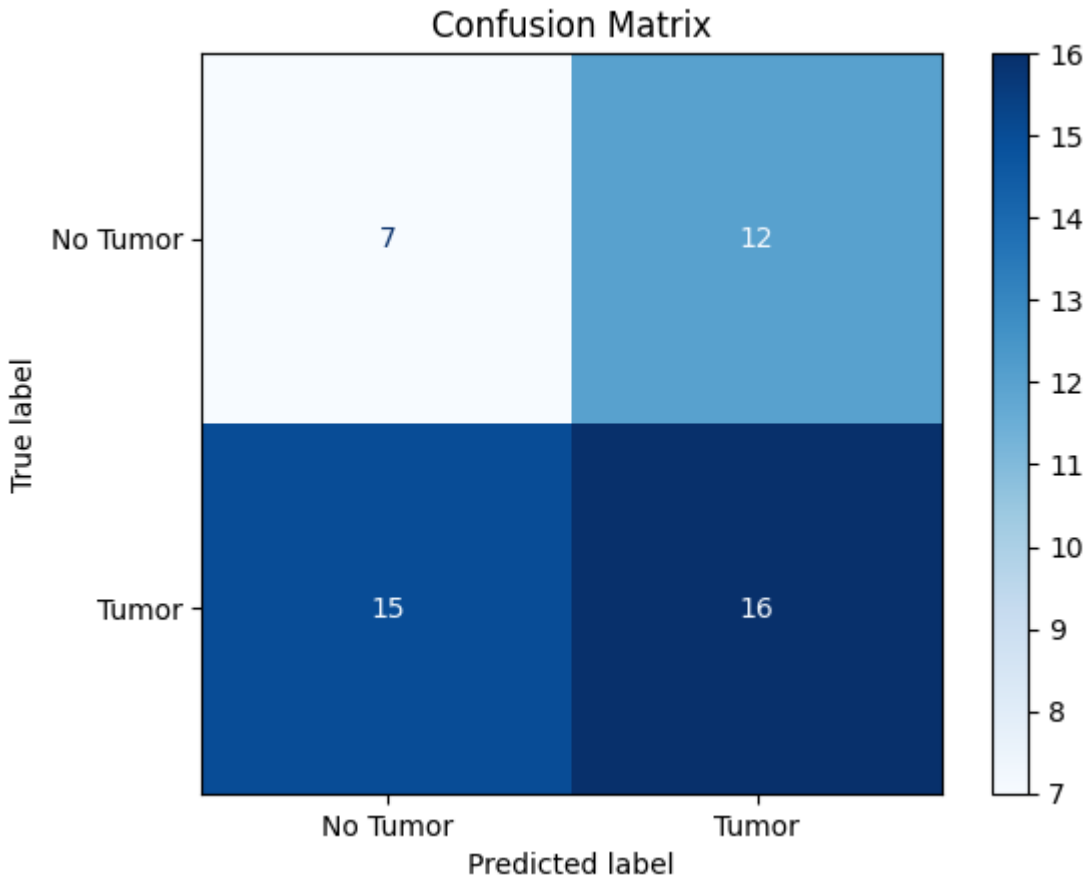
```
# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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






```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
import numpy as np

# Generate predictions
y_pred = model.predict(validation_generator)
y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary labels

# Compute confusion matrix
cm = confusion_matrix(validation_generator.classes, y_pred)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No Tumor', 'Tumor'])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()

# Print classification report
print(classification_report(validation_generator.classes, y_pred, target_names=['No Tumor', 'Tumor']))
```

```
4/4   15s 3s/step
```



	precision	recall	f1-score	support
No Tumor	0.32	0.37	0.34	19
Tumor	0.57	0.52	0.54	31
accuracy			0.46	50
macro avg	0.44	0.44	0.44	50
weighted avg	0.48	0.46	0.47	50

```
# Evaluate on validation set (as a proxy for test set)
test_loss, test_accuracy = model.evaluate(validation_generator)
print(f"Validation Accuracy: {test_accuracy:.2f}")
```

```
4/4   15s 3s/step - accuracy: 0.5312 - loss: 1.9575
Validation Accuracy: 0.50
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define test data generator
test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
   'TEST_CROP',               # Path to test data
   target_size=IMG_SIZE,
   batch_size=32,
   class_mode='binary',
   shuffle=False               # Don't shuffle for evaluation
)

# Evaluate on test set
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {test_accuracy:.2f}")
```

```
Found 10 images belonging to 2 classes.

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning:

Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_mul
```

```
1/1   3s 3s/step - accuracy: 0.4000 - loss: 0.7456
Test Accuracy: 0.40
```

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight(
   'balanced',
   classes=np.unique(y_train),
   y=y_train
)
class_weights_dict = dict(enumerate(class_weights))

history = model.fit(
   train_generator,
   steps_per_epoch=50,
   epochs=EPOCHS,
   validation_data=validation_generator,
   validation_steps=25,
   callbacks=[es, lr_scheduler],
   class_weight=class_weights_dict
)
```

```
Epoch 1/30
5/50    5:27 7s/step - accuracy: 0.5892 - loss: 0.9215
```

```
train_datagen = ImageDataGenerator(
   rotation_range=20,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   vertical_flip=True,
   brightness_range=[0.5, 1.5]
)

# Unfreeze the last few layers of the base model
for layer in base_model.layers[-10:]:
   layer.trainable = True

# Recompile the model with a lower learning rate
model.compile(
   loss='binary_crossentropy',
   optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
   metrics=['accuracy']
)

# Continue training
history = model.fit(
   train_generator,
   steps_per_epoch=50,
   epochs=10,  # Train for a few more epochs
   validation_data=validation_generator,
   validation_steps=25,
   callbacks=[es, lr_scheduler]
)

from tensorflow.keras.applications import EfficientNetB0

base_model = EfficientNetB0(
   include_top=False,
   input_shape=IMG_SIZE + (3,),
   weights='imagenet'
)

# Freeze the base model
base_model.trainable = False
```

```
# Build the model
model = Sequential([
   base_model,
   Flatten(),
   Dense(512, activation='relu'),
   BatchNormalization(),
   Dropout(0.5),
   Dense(1, activation='sigmoid')
])

model.compile(
   loss='binary_crossentropy',
   optimizer='adam',
   metrics=['accuracy']
)
```