In [33]:
```python
dataset_path = 'C:\\Users\\sd616\\Downloads\\bt\\'
```

In [45]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing import image  # Import the image functi
import numpy as np
```

In [35]:
```python


# Set your dataset path
dataset_path = 'C:\\Users\\sd616\\Downloads\\bt'
# dataset_path = '/home/username/Downloads/archive 3/'  # Linux or Mac

# Define image dimensions
img_width, img_height = 150, 150

# Create ImageDataGenerator for training and testing
train_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Load training data
train_data = train_datagen.flow_from_directory(
    os.path.join(dataset_path, 'training'),
    target_size=(img_width, img_height),
    batch_size=32,
    class_mode='categorical'
)

# Load testing data
test_data = test_datagen.flow_from_directory(
    os.path.join(dataset_path, 'testing'),
    target_size=(img_width, img_height),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 2870 images belonging to 4 classes.
Found 394 images belonging to 4 classes.
```

In [36]:
```python
# Define the CNN model
model = Sequential()

# First convolutional layer
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolutional layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third convolutional layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten layer
model.add(Flatten())

# Fully connected layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout to avoid overfitting

# Output layer (4 classes for tumor types)
model.add(Dense(4, activation='softmax'))
```

In [37]:
```python
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [38]:

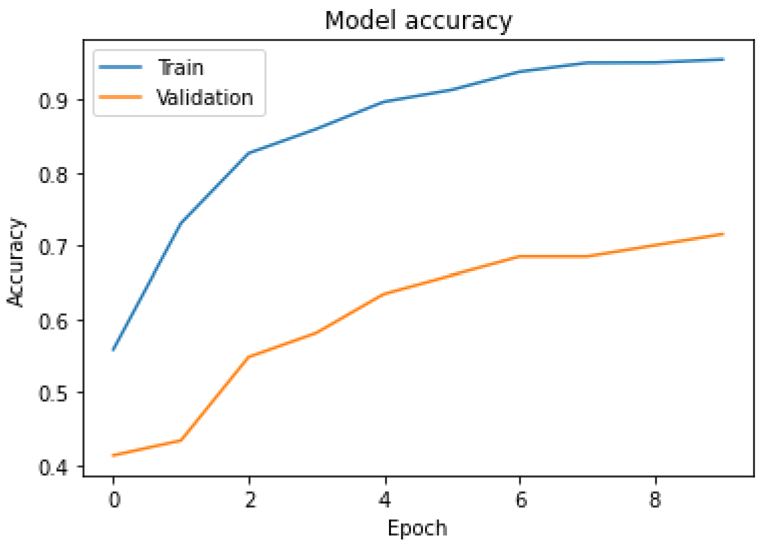
```python
# Train the model with automatic steps_per_epoch
history = model.fit(
    train_data,
    validation_data=test_data,
    epochs=10  # You can adjust the number of epochs
)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_data)
print(f"Test Accuracy: {test_acc}")
```

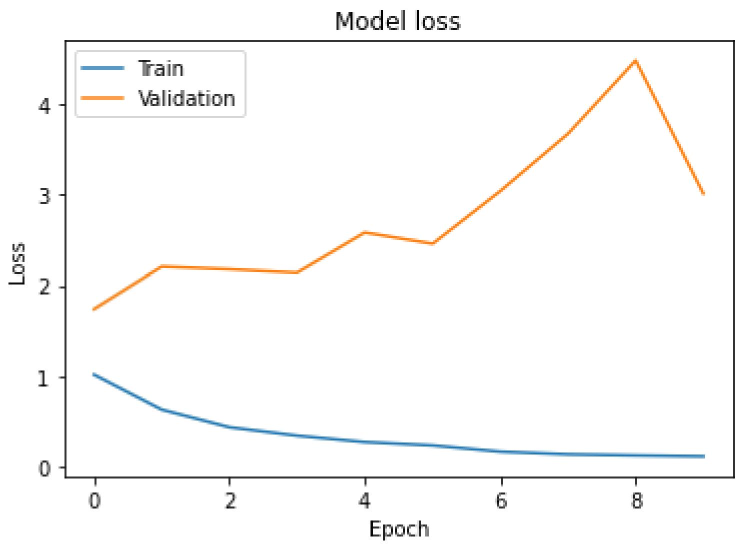
```
Epoch 1/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 37s 388ms/step - accuracy: 0.4798 - loss: 1.214
3 - val_accuracy: 0.4137 - val_loss: 1.7408
Epoch 2/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 38s 411ms/step - accuracy: 0.7048 - loss: 0.692
9 - val_accuracy: 0.4340 - val_loss: 2.2071
Epoch 3/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 42s 459ms/step - accuracy: 0.8135 - loss: 0.479
5 - val_accuracy: 0.5482 - val_loss: 2.1777
Epoch 4/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 36s 398ms/step - accuracy: 0.8641 - loss: 0.342
6 - val_accuracy: 0.5812 - val_loss: 2.1392
Epoch 5/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 38s 417ms/step - accuracy: 0.8972 - loss: 0.279
2 - val_accuracy: 0.6345 - val_loss: 2.5802
Epoch 6/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 41s 451ms/step - accuracy: 0.9137 - loss: 0.233
7 - val_accuracy: 0.6599 - val_loss: 2.4572
Epoch 7/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 47s 517ms/step - accuracy: 0.9436 - loss: 0.170
0 - val_accuracy: 0.6853 - val_loss: 3.0377
Epoch 8/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 48s 524ms/step - accuracy: 0.9573 - loss: 0.131
5 - val_accuracy: 0.6853 - val_loss: 3.6682
Epoch 9/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 54s 588ms/step - accuracy: 0.9492 - loss: 0.131
2 - val_accuracy: 0.7005 - val_loss: 4.4743
Epoch 10/10
90/90 ━━━━━━━━━━━━━━━━━━━━ 43s 467ms/step - accuracy: 0.9595 - loss: 0.106
0 - val_accuracy: 0.7157 - val_loss: 3.0097
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 171ms/step - accuracy: 0.7262 - loss: 2.8871
Test Accuracy: 0.7157360315322876
```

In [32]:
```python
# Make predictions
predictions = model.predict(test_data)

# Print the predicted class for the first batch of images
predicted_classes = tf.argmax(predictions, axis=1)
print(predicted_classes)
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 126ms/step
tf.Tensor(
[1 1 3 2 1 1 2 2 2 1 2 2 2 1 1 1 2 2 2 1 2 1 2 3 1 3 0 1 1 2 3 2 2 3 2 3 2
 2 2 2 1 2 0 2 1 1 1 3 1 2 1 2 2 1 2 1 2 2 1 2 2 2 1 0 3 3 0 1 1 2 2 2 3 3
 1 2 2 1 1 1 2 3 3 3 3 1 1 3 1 2 2 3 2 1 1 1 1 1 2 1 2 1 2 0 3 2 1 2 2 2 1
 2 2 2 3 1 0 1 2 1 1 2 2 2 1 1 2 1 1 1 2 1 1 0 2 1 1 1 3 2 1 3 1 1 2 1 3 2
 3 3 2 3 2 2 3 3 1 2 1 1 2 2 2 1 3 3 1 3 1 1 2 1 0 1 1 0 1 0 1 2 0 2 2 3 1
 1 3 2 2 2 1 2 1 0 1 1 2 2 3 1 3 2 0 1 1 2 2 2 1 1 3 1 2 2 0 3 1 2 1 2 3 2
 2 2 1 1 1 1 1 1 3 1 1 2 2 2 1 2 2 2 3 0 2 2 2 3 1 1 2 3 0 1 3 2 3 2 1 2 2
 2 2 2 2 2 2 2 2 1 1 1 2 2 3 2 0 1 1 1 1 2 1 1 1 1 1 2 2 1 1 2 2 2 1 2 2 1
 2 1 2 3 2 2 1 2 2 1 1 3 2 2 1 1 1 3 1 1 2 2 1 2 3 3 1 1 2 2 1 2 1 2 2 2 1
 2 2 2 3 1 1 1 1 1 2 3 3 1 3 1 1 2 2 3 1 1 2 1 3 1 2 2 1 2 2 1 1 2 0 1 3 3
 1 3 1 3 1 2 1 2 2 3 2 1 3 2 3 1 1 2 1 2 2 1 2 1], shape=(394,), dtype=int
64)
```

In [48]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





In [42]:
```
def preprocess_image(img_path, img_width, img_height):
    img = image.load_img(img_path, target_size=(img_width, img_height))  #
    img_array = image.img_to_array(img)  # Convert image to array
    img_array = np.expand_dims(img_array, axis=0)  # Add a batch dimension
    img_array /= 255.0  # Normalize the image
    return img_array
```

In [43]:
```
img_path = r'C:\\Users\\sd616\\Downloads\\bt\\Training\\meningioma_tumor\m3
```

In [46]:
```
preprocessed_img = preprocess_image(img_path, img_width, img_height)
```

In [47]:
```
plt.imshow(image.load_img(img_path, target_size=(img_width, img_height)))
plt.axis('off')  # Turn off axis
plt.show()

# Make predictions
prediction = model.predict(preprocessed_img)

# Get the predicted class index
predicted_class = np.argmax(prediction, axis=1)

# Class labels
class_labels = ['glioma_tumor', 'meningioma_tumor', 'pituitary_tumor', 'no_

# Get the class label for the predicted class
predicted_label = class_labels[predicted_class[0]]

# Output the prediction
print(f'The model predicts: {predicted_label}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 432ms/step
The model predicts: meningioma_tumor
```

In [ ]: