```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt


# Define paths to your dataset
train_data_dir = 'C:/Users/Mrinmayee/Downloads/Brain tumor'
test_data_dir = 'C:/Users/Mrinmayee/Downloads/Brain tumor'


# Set image dimensions and batch size
img_width, img_height = 150, 150
batch_size = 32


# Create data generators for training and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 3264 images belonging to 2 classes.
Found 3264 images belonging to 2 classes.
```

```
# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Binary classification (tumor or not)
])

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

# Display the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 flatten (Flatten)           (None, 36992)             0
 dense (Dense)               (None, 128)               4735104

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 4828481 (18.42 MB)
Trainable params: 4828481 (18.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

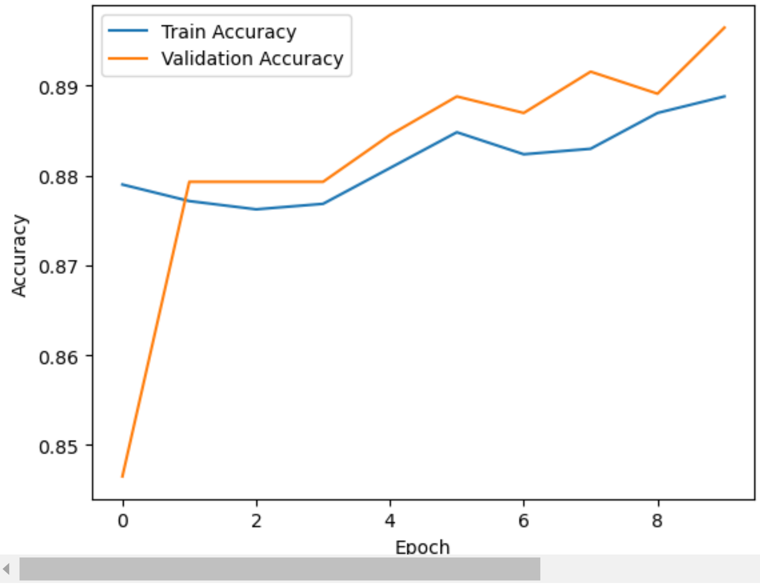
```
# Train the model
history = model.fit(train_generator, epochs=10, validation_data=test_generator)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_generator)
print(f"\nTest Accuracy: {test_acc * 100:.2f}%")

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/10
102/102 [==============================] - 125s 1s/step - loss: 0.3639 - accuracy: 0.
Epoch 2/10
102/102 [==============================] - 105s 1s/step - loss: 0.2937 - accuracy: 0.
Epoch 3/10
102/102 [==============================] - 125s 1s/step - loss: 0.2837 - accuracy: 0.
Epoch 4/10
102/102 [==============================] - 124s 1s/step - loss: 0.2697 - accuracy: 0.
Epoch 5/10
102/102 [==============================] - 129s 1s/step - loss: 0.2629 - accuracy: 0.
Epoch 6/10
102/102 [==============================] - 100s 984ms/step - loss: 0.2576 - accuracy:
Epoch 7/10
102/102 [==============================] - 94s 917ms/step - loss: 0.2547 - accuracy:
Epoch 8/10
102/102 [==============================] - 95s 928ms/step - loss: 0.2456 - accuracy:
Epoch 9/10
102/102 [==============================] - 137s 1s/step - loss: 0.2452 - accuracy: 0.
Epoch 10/10
102/102 [==============================] - 98s 961ms/step - loss: 0.2394 - accuracy:
102/102 [==============================] - 21s 200ms/step - loss: 0.2340 - accuracy:

Test Accuracy: 89.64%
```



```
import numpy as np
import os
# Choose a specific image to plot
image_index = 0

# Get class labels
class_labels = list(test_generator.class_indices.keys())

# Reset the test generator to the beginning
test_generator.reset()

# Generate predictions for the test set
predictions = model.predict(test_generator)
predictions = model.predict(test_generator)

# Get the true label, filename, and predicted probabilities for the chosen image
true_label = class_labels[test_generator.classes[image_index]]
filename = test_generator.filenames[image_index]
predicted_probabilities = predictions[image_index]

# Load and plot the image
img_path = os.path.join(test_data_dir, filename)
img = plt.imread(img_path)

# Plot the image along with true label and predicted probabilities
plt.imshow(img)
plt.title(f'True: {true_label}\nPredicted Probabilities: {predicted_probabilities}')
plt.axis('off')
plt.show()
```

```
102/102 [==============================] - 23s 224ms/step
```

True: Testing
Predicted Probabilities: [0.9907801]