```
# Install required packages
!pip -q install kaggle streamlit pyngrok==4.1.1 tensorflow==2.15 pillow scikit-learn

# Upload kaggle.json
from google.colab import files
print("Upload kaggle.json you just downloaded from Kaggle...")
uploaded = files.upload()  # choose kaggle.json

# Set up Kaggle API
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Create data folders
!mkdir -p /content/data
```

```
  Preparing metadata (setup.py) ... done
ERROR: Ignored the following versions that require a different python version: 0.55.2 Requires-Python <3.5
ERROR: Could not find a version that satisfies the requirement tensorflow==2.15 (from versions: 2.16.0rc0, 2.16.1, 2.16.2, 2.17.0rc0, 2.
ERROR: No matching distribution found for tensorflow==2.15
Upload kaggle.json you just downloaded from Kaggle...
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 71 bytes, last modified: 8/22/2025 - 100% done

```
Saving kaggle.json to kaggle (1).json
```

```
import os, zipfile, pathlib, shutil, random, glob, json
import numpy as np
from PIL import Image
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
print("TF:", tf.__version__)

# Download datasets
# 1. Classification dataset (yes/no)
!kaggle datasets download -d navoneel/brain-mri-images-for-brain-tumor-detection -p /content/data
!unzip -q -o /content/data/brain-mri-images-for-brain-tumor-detection.zip -d /content/data/cls_raw

# 2. Segmentation dataset (LGG with masks)
!kaggle datasets download -d mateuszbuda/lgg-mri-segmentation -p /content/data
!unzip -q -o /content/data/lgg-mri-segmentation.zip -d /content/data/seg_raw
```

```
TF: 2.19.0
Dataset URL: https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection
License(s): copyright-authors
brain-mri-images-for-brain-tumor-detection.zip: Skipping, found more recently modified local copy (use --force to force download)
Dataset URL: https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation
License(s): CC-BY-NC-SA-4.0
lgg-mri-segmentation.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
# Find the correct path for classification data
SRC = pathlib.Path('/content/data/cls_raw')
brain_tumor_path = None

# Search for the actual folder structure
for root, dirs, files in os.walk('/content/data/cls_raw'):
    if 'yes' in dirs and 'no' in dirs:
        brain_tumor_path = pathlib.Path(root)
        break

if brain_tumor_path:
    SRC = brain_tumor_path
else:
    # Check common folder names
    possible_paths = [
        '/content/data/cls_raw/brain_tumor_dataset',
        '/content/data/cls_raw/Brain Tumor Data Set',
        '/content/data/cls_raw'
    ]
    for path in possible_paths:
        if pathlib.Path(path).exists():
            SRC = pathlib.Path(path)
            break
```

```python
print("Classification data root:", SRC)
print("Contents:", list(SRC.iterdir()) if SRC.exists() else "Path not found")

# Create split folders
BASE = pathlib.Path('/content/data/cls_split')
for sub in ['train','val','test']:
    for cls in ['yes','no']:
        (BASE/sub/cls).mkdir(parents=True, exist_ok=True)

# Collect files
files_yes = sorted([*SRC.rglob('yes/*.jpg')]) + sorted([*SRC.rglob('yes/*.png')])
files_no  = sorted([*SRC.rglob('no/*.jpg')])  + sorted([*SRC.rglob('no/*.png')])

print(f"Found {len(files_yes)} tumor images and {len(files_no)} normal images")

def split_and_copy(files, cls):
    random.seed(42)
    random.shuffle(files)
    n = len(files)
    n_train = int(0.7*n); n_val = int(0.15*n)
    splits = [('train', files[:n_train]),
              ('val',   files[n_train:n_train+n_val]),
              ('test',  files[n_train+n_val:])]
    for split, flist in splits:
        for src in flist:
            dst = BASE/split/cls/src.name
            shutil.copy2(src, dst)
    print(f"Split {cls}: {n_train} train, {n_val} val, {n-n_train-n_val} test")

split_and_copy(files_yes, 'yes')
split_and_copy(files_no,  'no')
```

```
Classification data root: /content/data/cls_raw
Contents: [PosixPath('/content/data/cls_raw/yes'), PosixPath('/content/data/cls_raw/brain_tumor_dataset'), PosixPath('/content/data/cls_
Found 174 tumor images and 172 normal images
Split yes: 121 train, 26 val, 27 test
Split no: 120 train, 25 val, 27 test
```

```python
IMG_SIZE = (224,224)
BATCH = 32

train_ds = tf.keras.utils.image_dataset_from_directory(
    BASE/'train', image_size=IMG_SIZE, batch_size=BATCH, label_mode='categorical', seed=42)
val_ds   = tf.keras.utils.image_dataset_from_directory(
    BASE/'val', image_size=IMG_SIZE, batch_size=BATCH, label_mode='categorical', seed=42)
test_ds  = tf.keras.utils.image_dataset_from_directory(
    BASE/'test', image_size=IMG_SIZE, batch_size=BATCH, label_mode='categorical', shuffle=False)

# Get class names BEFORE applying transformations
class_names = train_ds.class_names
print("Classes:", class_names)

# Normalize + cache
def prep(x,y):
    # Handle grayscale images by converting to RGB if needed
    if x.shape[-1] == 1:
        x = tf.image.grayscale_to_rgb(x)
    elif x.shape[-1] == 4:  # Handle RGBA images
        x = x[:,:,:,:3]
    return tf.cast(x, tf.float32)/255.0, y

# Apply transformations
train_ds = train_ds.map(prep).cache().prefetch(tf.data.AUTOTUNE)
val_ds   = val_ds.map(prep).cache().prefetch(tf.data.AUTOTUNE)
test_ds  = test_ds.map(prep).cache().prefetch(tf.data.AUTOTUNE)

print(f"Training batches: {tf.data.experimental.cardinality(train_ds)}")
print(f"Validation batches: {tf.data.experimental.cardinality(val_ds)}")
print(f"Test batches: {tf.data.experimental.cardinality(test_ds)}")
```

```
Found 156 files belonging to 2 classes.
Found 44 files belonging to 2 classes.
Found 49 files belonging to 2 classes.
Classes: ['no', 'yes']
Training batches: 5
Validation batches: 2
Test batches: 2
```

```python
from tensorflow.keras.applications import EfficientNetB0, ResNet50, InceptionV3

def build_classifier(model_name='efficientnetb0', num_classes=2, input_shape=(224,224,3)):
    if model_name=='efficientnetb0':
        base = EfficientNetB0(include_top=False, weights='imagenet', input_shape=input_shape)
        preprocess = tf.keras.applications.efficientnet.preprocess_input
    elif model_name=='resnet50':
        base = ResNet50(include_top=False, weights='imagenet', input_shape=input_shape)
        preprocess = tf.keras.applications.resnet50.preprocess_input
    else: # 'inceptionv3' (GoogLeNet family)
        base = InceptionV3(include_top=False, weights='imagenet', input_shape=input_shape)
        preprocess = tf.keras.applications.inception_v3.preprocess_input

    base.trainable = False
    inputs = layers.Input(shape=input_shape)
    x = layers.Lambda(preprocess)(inputs)
    x = base(x, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.2)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = keras.Model(inputs, outputs)
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Train classifier
clf = build_classifier('efficientnetb0')
print("Training classifier...")
history = clf.fit(train_ds, validation_data=val_ds, epochs=10)
test_results = clf.evaluate(test_ds)
print(f"Test Accuracy: {test_results[1]:.4f}")

# Save
os.makedirs('/content/models', exist_ok=True)
clf.save('/content/models/classifier_efficientnet.keras')
```

```
Training classifier...
Epoch 1/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 30s 4s/step - accuracy: 0.4537 - loss: 0.7345 - val_accuracy: 0.4773 - val_loss: 0.6946
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 14s 3s/step - accuracy: 0.5039 - loss: 0.6934 - val_accuracy: 0.5227 - val_loss: 0.6937
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.5242 - loss: 0.7047 - val_accuracy: 0.5227 - val_loss: 0.6934
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.4701 - loss: 0.7038 - val_accuracy: 0.4773 - val_loss: 0.6944
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.5424 - loss: 0.7006 - val_accuracy: 0.4773 - val_loss: 0.6938
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 21s 3s/step - accuracy: 0.5428 - loss: 0.6831 - val_accuracy: 0.5227 - val_loss: 0.6927
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.5043 - loss: 0.6893 - val_accuracy: 0.5227 - val_loss: 0.6926
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 3s/step - accuracy: 0.5337 - loss: 0.6849 - val_accuracy: 0.5227 - val_loss: 0.6928
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 22s 3s/step - accuracy: 0.5088 - loss: 0.7017 - val_accuracy: 0.4773 - val_loss: 0.6938
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 15s 3s/step - accuracy: 0.4537 - loss: 0.7007 - val_accuracy: 0.5227 - val_loss: 0.6927
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 923ms/step - accuracy: 0.6006 - loss: 0.6909
Test Accuracy: 0.5102
```

```python
def build_unet(input_shape=(256, 256, 3)):
    inputs = layers.Input(shape=input_shape)

    # Encoder (downsampling)
    c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(inputs)
    c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D(2)(c1)

    c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D(2)(c2)

    c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(256, 3, activation='relu', padding='same')(c3)
    p3 = layers.MaxPooling2D(2)(c3)
```

```python
    c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(p3)
    c4 = layers.Conv2D(512, 3, activation='relu', padding='same')(c4)
    p4 = layers.MaxPooling2D(2)(c4)

    # Bridge
    c5 = layers.Conv2D(1024, 3, activation='relu', padding='same')(p4)
    c5 = layers.Conv2D(1024, 3, activation='relu', padding='same')(c5)

    # Decoder (upsampling)
    u6 = layers.UpSampling2D(2)(c5)
    u6 = layers.concatenate([u6, c4])
    c6 = layers.Conv2D(512, 3, activation='relu', padding='same')(u6)
    c6 = layers.Conv2D(512, 3, activation='relu', padding='same')(c6)

    u7 = layers.UpSampling2D(2)(c6)
    u7 = layers.concatenate([u7, c3])
    c7 = layers.Conv2D(256, 3, activation='relu', padding='same')(u7)
    c7 = layers.Conv2D(256, 3, activation='relu', padding='same')(c7)

    u8 = layers.UpSampling2D(2)(c7)
    u8 = layers.concatenate([u8, c2])
    c8 = layers.Conv2D(128, 3, activation='relu', padding='same')(u8)
    c8 = layers.Conv2D(128, 3, activation='relu', padding='same')(c8)

    u9 = layers.UpSampling2D(2)(c8)
    u9 = layers.concatenate([u9, c1])
    c9 = layers.Conv2D(64, 3, activation='relu', padding='same')(u9)
    c9 = layers.Conv2D(64, 3, activation='relu', padding='same')(c9)

    outputs = layers.Conv2D(1, 1, activation='sigmoid')(c9)

    model = keras.Model(inputs=[inputs], outputs=[outputs])
    return model


import cv2

def load_segmentation_data(data_path, img_size=(256, 256)):
    images = []
    masks = []

    seg_path = pathlib.Path(data_path)

    # Find all TIFF files (images)
    image_files = list(seg_path.rglob('*_mask.tif'))  # Find mask files first

    for mask_file in image_files:
        # Get corresponding image file
        img_file = str(mask_file).replace('_mask.tif', '.tif')

        if os.path.exists(img_file):
            # Load image
            img = cv2.imread(img_file, cv2.IMREAD_GRAYSCALE)
            if img is not None:
                img = cv2.resize(img, img_size)
                img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
                images.append(img)

                # Load mask
                mask = cv2.imread(str(mask_file), cv2.IMREAD_GRAYSCALE)
                mask = cv2.resize(mask, img_size)
                mask = mask / 255.0  # Normalize to 0-1
                masks.append(mask)

    return np.array(images), np.array(masks)

# Load segmentation data
print("Loading segmentation data...")
seg_images, seg_masks = load_segmentation_data('/content/data/seg_raw')
print(f"Loaded {len(seg_images)} images for segmentation")

# Split segmentation data
n = len(seg_images)
n_train_seg = int(0.8 * n)

seg_train_x, seg_test_x = seg_images[:n_train_seg], seg_images[n_train_seg:]
seg_train_y, seg_test_y = seg_masks[:n_train_seg], seg_masks[n_train_seg:]
```

```
# Normalize
seg_train_x = seg_train_x.astype('float32') / 255.0
seg_test_x = seg_test_x.astype('float32') / 255.0
```

```
Loading segmentation data...
```

```
# Build and train U-Net
unet = build_unet((256, 256, 3))
unet.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print("Training U-Net...")
unet_history = unet.fit(seg_train_x, seg_train_y,
                       validation_split=0.2,
                       epochs=10,
                       batch_size=8)

# Evaluate
unet_results = unet.evaluate(seg_test_x, seg_test_y)
print(f"Segmentation Test Accuracy: {unet_results[1]:.4f}")

# Save
unet.save('/content/models/unet_segmentation.keras')
```

```
%%writefile app.py
import streamlit as st
import tensorflow as tf
import numpy as np
from PIL import Image
import cv2

st.title("🧠 Brain Tumor Detection & Segmentation")
st.write("Upload an MRI image to detect and segment brain tumors")

# Load models
@st.cache_resource
def load_models():
    classifier = tf.keras.models.load_model('/content/models/classifier_efficientnet.keras')
    segmentor = tf.keras.models.load_models('/content/models/unet_segmentation.keras')
    return classifier, segmentor

try:
    classifier, segmentor = load_models()
    st.success("Models loaded successfully!")
except:
    st.error("Please train the models first!")
    st.stop()

# File uploader
uploaded_file = st.file_uploader("Choose an MRI image...", type=['jpg', 'jpeg', 'png'])

if uploaded_file is not None:
    # Display uploaded image
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded MRI Image", use_column_width=True)

    # Preprocess for classification
    img_array = np.array(image.resize((224, 224)))
    if len(img_array.shape) == 2:  # Grayscale
        img_array = cv2.cvtColor(img_array, cv2.COLOR_GRAY2RGB)
    img_array = np.expand_dims(img_array, axis=0) / 255.0

    # Classification
    st.subheader("🔍 Tumor Detection")
    pred = classifier.predict(img_array)
    class_names = ['No Tumor', 'Tumor Present']
    predicted_class = class_names[np.argmax(pred)]
    confidence = np.max(pred) * 100

    st.write(f"**Prediction:** {predicted_class}")
    st.write(f"**Confidence:** {confidence:.2f}%")

    # Segmentation if tumor detected
    if np.argmax(pred) == 1:  # Tumor present
        st.subheader("🎯 Tumor Segmentation")
```

```
        # Preprocess for segmentation
        seg_img = np.array(image.resize((256, 256)))
        if len(seg_img.shape) == 2:
            seg_img = cv2.cvtColor(seg_img, cv2.COLOR_GRAY2RGB)
        seg_img = np.expand_dims(seg_img, axis=0) / 255.0

        # Predict mask
        mask_pred = segmentor.predict(seg_img)
        mask = mask_pred[0, :, :, 0]

        # Display segmentation
        col1, col2 = st.columns(2)
        with col1:
            st.image(image, caption="Original Image", use_column_width=True)
        with col2:
            st.image(mask, caption="Tumor Segmentation", use_column_width=True)

        # Calculate tumor area
        tumor_pixels = np.sum(mask > 0.5)
        total_pixels = mask.shape[0] * mask.shape[1]
        tumor_percentage = (tumor_pixels / total_pixels) * 100
        st.write(f"**Tumor Area:** {tumor_percentage:.2f}% of image")

st.sidebar.info("This app uses EfficientNet for classification and U-Net for segmentation")


# Simple way to run locally in Colab
from pyngrok import ngrok

# Run Streamlit
!streamlit run app.py &

# Create tunnel
public_url = ngrok.connect(port='8501')
print(f'Your app is live at: {public_url}')
```