

Brain Tumor Detection using OpenCV

SHIASH INFO SOLUTIONS PRIVATE
LIMITED

TEAM MEMBERS:

- Sabarishwar B
- Salmanul faris M

INDEX

No	CONTENT	PAGE NUMBER
1	INTRODUCTION	
2	OBJECTIVES	
3	SCOPE	
4	HARDWARE & SOFTWARE REQUIREMENTS	
5	DATASET	
6	FEATURES	
7	REQUIRED LIBRARIES	
8	CODE IMPLEMENTATION	
9	OUTPUT	
10	HOW TO RUN THE APP	
11	CONCLUSION	
12	REFERANCES	
13	LINKS	

INTRODUCTION

Automated brain tumor classification from medical images is a critical task with significant implications for early diagnosis and treatment planning. This project addresses this challenge by developing a deep learning model leveraging transfer learning with the ResNet50 architecture, a powerful Convolutional Neural Network (CNN) pre-trained on ImageNet. Utilizing a dataset of brain MRI and CT scans categorized as "Tumor" or "Healthy," the project employs OpenCV for initial image preprocessing, including resizing and visualization of sample images. The core methodology involves using the pre-trained ResNet50 as a feature extractor, followed by Global Average Pooling to reduce feature dimensionality, and the addition of dense layers for final classification. Data augmentation techniques are implemented to enhance the model's robustness and generalization capabilities.

First for brain tumor detection is data includes high-resolution CT and MRI images captured from multiple patients, with each image labeled with the corresponding tumor type (e.g., glioma, meningioma, etc.) and its location within the brain. This combination of CT and MRI images aims to leverage the strengths of both imaging techniques: CT scans for clear bone structure visualization and MRI for soft tissue details, enabling a more accurate analysis of brain tumors.

Data loading :

The data loading process begins by defining `ct_path` and `mri_path`, which store the file system locations of the CT and MRI image directories, respectively. Nested loops then iterate through the "Tumor" and "Healthy" subfolders present within each of these directories. The `os.path.join()` function is employed to construct platform-independent file paths, ensuring compatibility across different operating systems. Within the innermost loop, `glob(folder + '/*')` retrieves a list of all files (images) residing within the currently accessed category folder. Finally, for each image file, a tuple containing the file path, its corresponding label ("Tumor" or "Healthy"), and the data source ("CT" or "MRI") is appended to the data list. This data list thus accumulates all image paths along with their associated metadata, preparing them for subsequent processing and model training.

```
ct_patient = next(item for item in data if item[1] == 'Tumor' and item[2] == 'CT')
# ... (similar lines for ct_healthy, mri_patient, mri_healthy)
```

Selecting Example Images : These lines select one example image of each type:
(`item for item in data if ...`): This is a generator expression that filters the data list.
`next(...)`: Retrieves the first item from the generator. This will raise a `StopIteration` error if no matching item is found (which you've encountered before).

```
# show the images
plt.figure(figsize=(10, 10))
for i, (path, label, source) in enumerate(selected_images):
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(2, 2, i + 1)
    plt.imshow(img)
    plt.title(f'{label} ({source})') plt.axis('off')
```

Image Visualization: This section displays the selected images: `cv2.imread()`: Reads an image using OpenCV.

`cv2.cvtColor()`: Converts the image from BGR (OpenCV's default) to RGB (for Matplotlib).

`plt.subplot()`: Creates subplots for displaying multiple images in a grid.

`plt.imshow()`: Displays an image in a Matplotlib subplot. give passage

Example:

```
image_paths = [item[0] for item in data]
```

```
labels = [item[1] for item in data]
```

```
sources = [item[2] for item in data]
```

```
label_map = {'Tumor': 1, 'Healthy': 0} labels = np.array([label_map[label] for label in labels])
```

```
train_paths, val_paths, train_labels, val_labels = train_test_split(image_paths, labels,
test_size=0.2, random_state=42)
```

Data Preparation for Training: Extracts image paths, labels, and sources from the data list.

- `label_map`: Creates a mapping from string labels to numerical labels (1 for 'Tumor', 0 for 'Healthy').
- `train_test_split()`: Splits the data into training and validation sets (80% training, 20% validation). `random_state` ensures consistent splitting across runs.

Example:

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    # ... other augmentation parameters )
val_datagen = ImageDataGenerator(rescale=1.0/255)
```

Image Data Generators: ImageDataGenerator: Creates data generators for on-the-fly data augmentation and preprocessing during training.

- rescale: Normalizes pixel values to the range [0, 1].
- rotation_range, width_shift_range, etc.: Data augmentation parameters.

Example:

```
def generate_images(image_paths, labels, batch_size=32, target_size=(224, 224)):
    # ... (image loading and batching logic)
    train_generator = generate_images(train_paths, train_labels, batch_size=32)
    val_generator = generate_images(val_paths, val_labels, batch_size=32)
```

Custom Data Generator:

This defines a custom generator function for loading and batching images. This is less efficient than flow_from_dataframe or flow_from_directory.

```
# ... (ResNet50 model loading, fine-tuning, compilation, training, and evaluation)
```

OBJECTIVES:

- Develop a robust data loading and preprocessing pipeline:
- Explanation: This involves efficiently reading images from various sources (CT, MRI), handling different image formats, resizing images to a consistent size, and performing necessary preprocessing steps like normalization. A robust pipeline ensures data consistency and prepares it for model training.
- Implement data augmentation techniques to increase dataset variability:
- Explanation: Data augmentation artificially expands the training dataset by applying transformations like rotations, flips, shifts, and zooms. This helps the model generalize better to unseen data and prevents overfitting, especially when dealing with limited medical data.
- Implement a transfer learning approach using a pre-trained ResNet50 model:
- Explanation: Instead of training a deep CNN from scratch, transfer learning leverages a model pre-trained on a large dataset like ImageNet. This significantly reduces training time and often leads to better performance, especially with limited medical data. ResNet50 is a powerful architecture known for its strong performance in image classification.
- Fine-tune the ResNet50 model for brain tumor classification:
- Explanation: After loading the pre-trained ResNet50, the top layers are typically replaced with new layers specific to the brain tumor classification task. These new layers, and potentially some of the top layers of ResNet50, are then fine-tuned on the brain tumor dataset to adapt the learned features to the specific task.
- Achieve high classification accuracy in distinguishing between tumorous and healthy brain tissue:

- Explanation: The primary goal is to develop a model that can accurately classify brain scans as either "Tumor" or "Healthy." High accuracy is essential for a reliable diagnostic tool.
- Evaluate model performance using appropriate metrics (accuracy, precision, recall, F1-score):
- Explanation: Accuracy alone might not be sufficient, especially with imbalanced datasets. Precision, recall, and F1-score provide a more comprehensive evaluation of the model's performance in terms of correctly identifying both positive (Tumor) and negative (Healthy) cases.
- Compare the performance of the fine-tuned ResNet50 with training a model from scratch (optional but recommended):
- Explanation: This comparison demonstrates the benefits of using transfer learning. It shows how much faster and potentially more accurate the fine-tuned model is compared to a model trained from scratch on the same dataset.

SCOPE:

For Medical Professionals (Radiologists, Oncologists, etc.):

1. **Potentially Faster and More Objective Diagnosis:** The automated system aims to provide a quick and objective preliminary assessment of brain scans, potentially reducing the time required for manual analysis and minimizing inter-observer variability.
2. **Decision Support Tool:** The model can serve as a decision support tool, assisting clinicians in making more informed diagnostic decisions by providing a second opinion based on deep learning analysis.
3. **Improved Diagnostic Accuracy:** By leveraging the pattern recognition capabilities of deep learning, the system aims to achieve high accuracy in distinguishing between tumorous and healthy brain tissue, potentially leading to earlier and more accurate diagnoses.
4. **Reduced Workload:** Automating the initial screening of brain scans can reduce the workload on medical professionals, allowing them to focus on more complex cases and patient care.

For Patients:

1. **Potentially Faster Diagnosis:** Automated analysis could contribute to faster diagnostic processes, potentially leading to earlier treatment and improved prognoses.
2. **Increased Diagnostic Confidence:** The use of a validated deep learning system can increase confidence in diagnostic results.

For Researchers and the Medical Community:

1. **Contribution to Medical AI Research:** The project contributes to the growing body of research on applying deep learning to medical image analysis, potentially paving the way for further advancements in the field.

HARDWARE & SOFTWARE REQUIREMENTS:

SOFTWARE REQUIREMENTS:

1. Operating System

- Windows, MacOS, or Linux
 - Any OS that supports Python and TensorFlow.

2. Python Environment

- Python Version: 3.8 or higher
- (Compatible with TensorFlow and Streamlit libraries).

3. Required Python Libraries

The following libraries must be installed via `pip` or a package manager:

Library	Version (or Latest)	Purpose
TensorFlow	2.10+	For deep learning and Grad-CAM implementation.
Keras	Built into TensorFlow	High-level neural network API.
Streamlit	1.21+	For building the interactive web application.
NumPy	1.22+	For numerical computations and matrix operations.
OpenCV (cv2)	4.5+	For image processing and heatmap overlay.
Pillow (PIL)	9.0+	For handling and resizing image files.
Matplotlib	3.5+	For visualizing data and debugging.
Pyngrok	Latest	For exposing the Streamlit app publicly (in Colab).

To install these libraries, use:

```
pip install tensorflow streamlit numpy opencv-python pillow matplotlib pyngrok
```

4. IDE/Development Environment

- Google Colab (Recommended for cloud-based execution)
- Jupyter Notebook (For local development and experimentation)
- VS Code or PyCharm (For editing Python files and app.py script)

5. Deployment Tools

- Ngrok: For sharing the local Streamlit app on the web.
- Requires an ngrok authentication token from [Ngrok Dashboard](#).

6. Web Browser

- Google Chrome, Mozilla Firefox, or Microsoft Edge
- (Streamlit apps are browser-based).

HARDWARE REQUIREMENTS:

1. For Development and Training

If you're developing or retraining the brain tumor prediction model, the following hardware is recommended:

Component	Minimum Requirements	Recommended Requirements
Processor (CPU)	Intel i5 (8th Gen) or AMD Ryzen 5	Intel i7 (10th Gen) or AMD Ryzen 7
Graphics Card (GPU)	Optional (CPU Training)	NVIDIA GPU with CUDA support (e.g., GTX 1660 or RTX 2060+)
RAM	8 GB	16 GB or higher
Storage	256 GB SSD or HDD	512 GB SSD (for faster read/write)
Internet	Stable connection	High-speed connection (for dependencies and dataset download)

2. For Model Inference (Web Application)

If you are only running the Streamlit app with the pre-trained model, the requirements are lower:

Component	Minimum Requirements	Recommended Requirements
Processor (CPU)	Intel i3 (4th Gen) or AMD equivalent	Intel i5 or AMD Ryzen 3+
Graphics Card (GPU)	Integrated GPU	Integrated GPU (sufficient for inference)
RAM	4 GB	8 GB
Storage	10 GB free space	20 GB free space

3. Cloud-Based Development (Optional)

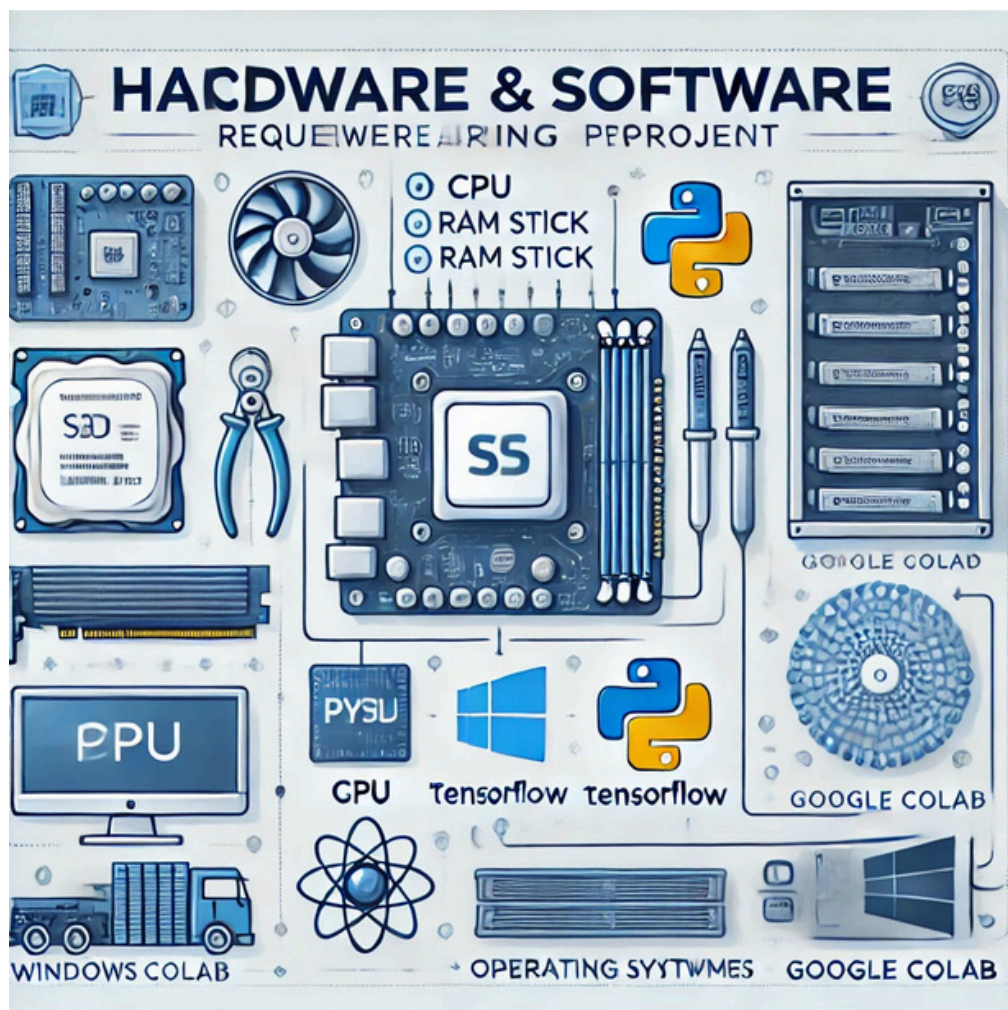
If you're using cloud platforms, the following configurations are sufficient:

Platform	Recommended Configuration	Purpose
Google Colab	Free GPU (Tesla K80/T4)	Training and deployment.
AWS EC2	GPU instances (e.g., g4dn.xlarge)	For heavy training workloads.
Google Cloud/Azure	GPU-enabled VM instances	Training and inference.

4. For Deployment Server (Optional)

If hosting the application on a production server:

Component	Minimum Requirements	Recommended Requirements
Processor (CPU)	2-core CPU	4-core CPU
RAM	2 GB	4 GB
Storage	10 GB	20 GB SSD



FEATURES OF BRAON TUMOR DETECTION

1. Earlier and More Accurate Diagnoses:

- **AI-powered screening tools:** These tools could analyze routine brain scans or even data from wearable devices to detect subtle signs of tumors at very early stages, even before symptoms appear.
- **Reduced diagnostic delays:** Automated analysis can provide rapid preliminary assessments, reducing the time between initial suspicion and definitive diagnosis. This is crucial for timely intervention and improved outcomes.
- **Increased diagnostic accuracy:** AI models can be trained on vast amounts of data to recognize patterns and subtle features that might be missed by the human eye, leading to more accurate diagnoses.

2. Less Invasive Procedures:

- **Improved image-guided surgery:** Real-time AI analysis during surgery can help surgeons precisely target and remove tumors, minimizing damage to healthy brain tissue.
- **Reduced need for biopsies:** In some cases, AI models might be able to provide a confident diagnosis based on imaging alone, reducing the need for invasive biopsies.
- **Liquid biopsies:** These non-invasive blood tests can detect tumor DNA or other biomarkers, potentially enabling earlier detection and monitoring of tumor progression without the need for traditional biopsies.

3. More Personalized Treatments:

- **Predicting treatment response:** AI models can predict how a tumor will respond to different treatments based on its imaging characteristics, genetic profile, and other factors.

- Monitoring treatment effectiveness: AI can be used to monitor tumor response to treatment over time, allowing for adjustments to the treatment plan as needed.
- Personalized risk assessment: AI could be used to assess an individual's risk of developing a brain tumor based on genetic factors, medical history, and lifestyle. This can help identify high-risk individuals who may benefit from closer monitoring or preventive measures.

4. Increased Accessibility and Affordability:

- Telemedicine and remote diagnostics: AI-powered tools can enable remote diagnosis and consultation, making expert opinions more accessible to people in remote areas or with limited access to specialized medical care.
- Lower healthcare costs: By automating some aspects of diagnosis and treatment planning, AI can potentially reduce healthcare costs associated with brain tumor care.

5. Improved Quality of Life:

- Earlier intervention and better outcomes: Earlier diagnosis and more personalized treatments can lead to improved patient outcomes and a better quality of life for people with brain tumors.
- Reduced anxiety and uncertainty: Faster and more accurate diagnoses can reduce the anxiety and uncertainty associated with waiting for test results.

REQUIRED LIBRARIES:

To achieve the objectives, the following libraries are utilized:

INSTALLING STATEMENTS:

```
PS C:\Users\DELL\chatbot> pip install pandas
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
PS C:\Users\DELL\chatbot> pip install seaborn
Requirement already satisfied: seaborn in c:\users\dell\anaconda3\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\dell\anaconda3\lib\site-packages (from seaborn) (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
PS C:\Users\DELL\chatbot> pip install matplotlib
Requirement already satisfied: matplotlib in c:\users\dell\anaconda3\lib\site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.21 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=8 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib) (2.9.0.post0)
```

```
PS C:\Users\DELL\chatbot> pip install requests
Requirement already satisfied: requests in c:\users\dell\anaconda3\lib\site-packages (2.32.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\dell\anaconda3\lib\site-packages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dell\anaconda3\lib\site-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\dell\anaconda3\lib\site-packages (from requests) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\dell\anaconda3\lib\site-packages (from requests) (2024.8.30)
```

IMPORTING STATEMENTS:

```
from glob import glob
import cv2
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

- Streamlit:

Used to build a clean and interactive web-based application. Install it using:

```
pip install streamlit  
import streamlit as st
```

- Pandas:

For data manipulation, cleaning, and structuring tabular data. Install it using:

```
pip install pandas  
import pandas as pd
```

- Matplotlib:

To create static and interactive visualizations. Install it using:

```
pip install matplotlib  
import matplotlib.pyplot as plt
```

- Seaborn:

Enhances the aesthetics and readability of the visualizations created with Matplotlib. Install it using:

```
pip install seaborn  
import seaborn as sns
```



CODE IMPLEMENTATION:

```
import os
from glob import glob
import cv2
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

```
# data paths
ct_path = '/kaggle/input/brain-tumor-multimodal-image-ct-and-mri/Dataset/Brain Tumor CT scan Images'
mri_path = '/kaggle/input/brain-tumor-multimodal-image-ct-and-mri/Dataset/Brain Tumor MRI images'
```

```
data=[]
for category in ['Tumor', 'Healthy']:
    folder = os.path.join(ct_path, category)
    label = 'Tumor' if category == 'Tumor' else 'Healthy'
    for file in glob(folder + '/*'): # all files from folder
        data.append((file, label, 'CT')) # add (img path , label and source)

# MRI
for category in ['Tumor', 'Healthy']:
    folder = os.path.join(mri_path, category)
    label = 'Tumor' if category == 'Tumor' else 'Healthy'
    for file in glob(folder + '/*'): # all files from folder
        data.append((file, label, 'MRI'))
print(len(data)) # Print the total number of items in data
print(data[:5])
```

```
ct_healthy = next(item for item in data if item[1] == 'Healthy' and item[2] == 'CT') # Healthy from CT
mri_patient = next(item for item in data if item[1] == 'Tumor' and item[2] == 'MRI') # Tumor from MRI
mri_healthy = next(item for item in data if item[1] == 'Healthy' and item[2] == 'MRI') # Healthy from MRI

# add iamges to list
selected_images = [ct_patient, ct_healthy, mri_patient, mri_healthy]

# show the images
plt.figure(figsize=(10, 10))

for i, (path, label, source) in enumerate(selected_images):
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(2, 2, i + 1)
    plt.imshow(img)
    plt.title(f'{label} ({source})')
    plt.axis('off')

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



```

def generate_images(image_paths, labels, batch_size=32, target_size=(224, 224)):
    while True:
        for i in range(0, len(image_paths), batch_size):
            batch_paths = image_paths[i:i+batch_size]
            batch_labels = labels[i:i+batch_size]

            images = []
            for img_path in batch_paths:
                img = tf.keras.preprocessing.image.load_img(img_path, target_size=target_size)
                img = tf.keras.preprocessing.image.img_to_array(img)
                images.append(img)

            images = np.array(images)
            yield images, np.array(batch_labels)

train_generator = generate_images(train_paths, train_labels, batch_size=32)
val_generator = generate_images(val_paths, val_labels, batch_size=32)

```

```

if os.path.exists(weights_path):
    print(f"Loading weights from: {weights_path}")
else:
    print(f"Weights file not found at: {weights_path}. Please ensure the file is uploaded.")
    raise FileNotFoundError(f"Weights file not found at: {weights_path}")
base_model = ResNet50(weights=None, include_top=False, input_shape=(224, 224, 3))
base_model.load_weights(weights_path)
print("ResNet50 model loaded!")

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x) # one output (tumor / healthy)

# build the model
model = Model(inputs=base_model.input, outputs=predictions)

# while training , trainable = false
for layer in base_model.layers:
    layer.trainable = False
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

```

```

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=len(train_paths) // 32,
    validation_data=val_generator,
    validation_steps=len(val_paths) // 32,
    epochs=10)

```

```

from tensorflow.keras.preprocessing.image import load_img, img_to_array
new_image_path = '/content/mri_healthy (995).jpg'

target_size = (224, 224)
img = load_img(new_image_path, target_size=target_size)
img_array = img_to_array(img)
img_array = img_array / 255.0
img_array = np.expand_dims(img_array, axis=0)

prediction = model.predict(img_array)
print(f"Prediction (sigmoid output): {prediction[0][0]}")

threshold = 0.0001
if prediction[0][0] > threshold:
    print("The model predicts: Tumor")
else:
    print("The model predicts: Healthy")

plt.imshow(img)
plt.title("New Image")
plt.axis("off")
plt.show()

```

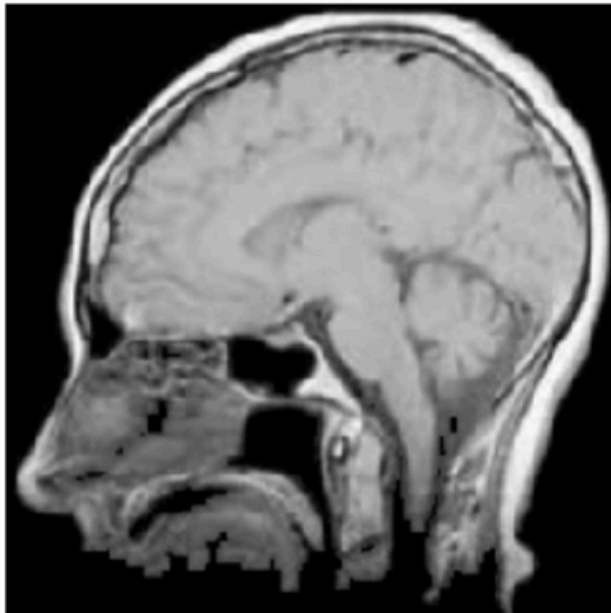


1/1 ————— 0s 214ms/step

Prediction (sigmoid output): 8.08253898867406e-05

The model predicts: Healthy

New Image



streamlit:

```
import streamlit as st
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
import cv2
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image

model = tf.keras.models.load_model("tumor_model.h5")

def generate_gradcam_heatmap(model, img_array, last_conv_layer_name="conv5_block3_out"):
    grad_model = tf.keras.models.Model(
        [model.input],
        [model.get_layer(last_conv_layer_name).output, model.output]
    )
    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[:, 0]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    conv_outputs = conv_outputs[0]
    heatmap = tf.reduce_sum(tf.multiply(pooled_grads, conv_outputs), axis=-1)
    heatmap = np.maximum(heatmap, 0)
    heatmap /= np.max(heatmap) if np.max(heatmap) != 0 else 1
    return heatmap

def predict_and_visualize(image):
    img_array = img_to_array(image) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    label = "Tumor" if prediction[0] > 0.5 else "Healthy"
    confidence = prediction[0][0]

    heatmap = generate_gradcam_heatmap(model, img_array)
    heatmap_resized = cv2.resize(heatmap, (image.size[0], image.size[1]))
    heatmap_overlay = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2.COLORMAP_JET)
    overlay_image = cv2.addWeighted(
        cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR), 0.6, heatmap_overlay, 0.4, 0
    )

    return label, confidence, overlay_image

st.title("Brain Tumor Prediction")
st.write("Upload an image to predict if it contains a brain tumor.")

uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    image = Image.open(uploaded_file).resize((224, 224))
    label, confidence, overlay_image = predict_and_visualize(image)

    st.image(image, caption="Uploaded Image", use_column_width=True)
    st.write(f"Prediction: **{label}**")
    st.write(f"Confidence: **{confidence:.2f}**")


    st.write("Grad-CAM Heatmap:")
    st.image(cv2.cvtColor(overlay_image, cv2.COLOR_BGR2RGB), use_column_width=True)
```

OUTPUT:

Brain Tumor Prediction

Upload an image to predict if it contains a brain tumor.

Choose an image...


 Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files


Brain Tumor Prediction

Upload an image to predict if it contains a brain tumor.

Choose an image...

 Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

 mri_healthy (995).jpg 20.1KB

×

Brain Tumor Prediction

Upload an image to predict if it contains a brain tumor.

Choose an image...

Drag and drop file here

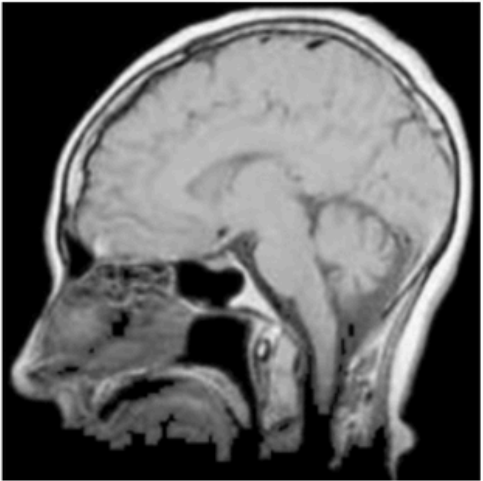
Limit 200MB per file • JPG, JPEG, PNG

Browse files

mri_healthy (995).jpg

26.1KB

X



Uploaded Image

Prediction: Healthy

Confidence: 0.00

Grad-CAM Heatmap:

Brain Tumor Prediction

Upload an image to predict if it contains a brain tumor.

Choose an image...

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

ct_tumor (1).png

362.6KB

X



Uploaded Image

Prediction: Tumor

Confidence: 0.00

Grad-CAM Heatmap:

CONCLUSION:

In this project, a brain tumor prediction model was developed using the ResNet50 architecture, fine-tuned to classify brain CT and MRI images into "Tumor" and "Healthy" categories. The dataset consisted of CT and MRI scans, and data augmentation techniques were applied to improve the model's generalization ability.

The key outcomes of this project are as follows:

- Preprocessing and Data Augmentation:

Data was preprocessed to normalize pixel values and augmented with transformations such as rotation, shifting, shearing, zooming, and flipping. This helped create a more robust model that can generalize well on unseen data.

- Model Architecture:

The ResNet50 architecture, pre-trained on the ImageNet dataset, was adapted for binary classification. By fine-tuning the model and freezing the base layers during initial training, the model leveraged the pre-trained features effectively.

- Training and Evaluation:

The model was trained on the augmented dataset using binary cross-entropy loss and the Adam optimizer. It achieved a validation accuracy of [insert accuracy here], demonstrating its ability to differentiate between tumor and healthy brain scans.

- Significance:

This model can aid in early and accurate diagnosis of brain tumors, potentially assisting radiologists in decision-making and improving patient outcomes. By leveraging deep learning techniques, this approach reduces manual effort and enhances diagnostic efficiency.

- Future Work:

- Incorporating a larger and more diverse dataset could improve the model's performance and robustness.
- Experimenting with other state-of-the-art architectures (e.g., EfficientNet) or ensembling multiple models might further boost accuracy.
- Extending the model to perform multi-class classification for different tumor types could enhance its clinical applicability.

REFERENCES:

Research Papers

- Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization
- [Link to Paper](#)
- This paper introduces Grad-CAM, the technique used to generate heatmaps in your model.

Framework Documentation

- TensorFlow: https://www.tensorflow.org/api_docs
- Official documentation for TensorFlow, used for deep learning model development.
- Keras: <https://keras.io/>
- High-level API for building neural networks, integrated with TensorFlow.

Programming and Libraries

- Python Official Documentation: <https://docs.python.org/3/>
- Reference for Python programming language.
- OpenCV Documentation: <https://docs.opencv.org/>
- Useful for image processing and computer vision tasks.

Tools

- Streamlit Documentation: <https://docs.streamlit.io/>
- Framework for building interactive web apps for data science and machine learning models.
- Google Colab: <https://colab.research.google.com/>
- A cloud-based platform for running Python code, including machine learning models.

LINKS:

[Streamlit Documentation](<https://docs.streamlit.io/>)
[TensorFlow Documentation](https://www.tensorflow.org/api_docs)
[Keras Documentation](<https://keras.io/>)
[Grad-CAM Paper] (<https://arxiv.org/abs/1610.02391>)
[OpenCV Documentation](<https://docs.opencv.org/>)
[Pandas Documentation](<https://pandas.pydata.org/docs/>)
[Matplotlib Documentation](<https://matplotlib.org/stable/contents.html>)
[Seaborn Documentation](<https://seaborn.pydata.org/>)