BRAIN TUMOR CLASSIFICATION

PROJECT REPORT

Submitted By

Parvathy V P

ABSTRACT

Brain tumors are abnormal growths of cells inside the brain that can be life-threatening if not detected early. Manual diagnosis using MRI scans requires expert radiologists and is time-consuming. This project proposes an automated Brain Tumor MRI Classification System using Convolutional Neural Networks (CNN).

The system classifies MRI images into four categories:

- **Glioma**
- **Meningioma**
- **Pituitary**
- **No Tumor**

The model is trained using deep learning techniques and deployed through a Streamlit web application for real-time predictions. The system achieves high accuracy and provides confidence scores for predictions.

INTRODUCTION

Medical image classification is an important application of Deep Learning in healthcare. MRI (Magnetic Resonance Imaging) is widely used for brain tumor detection. However, manual analysis:

- **Requires medical expertise**
- **Is time-consuming**
- **Can be subjective**

Using CNNs, we can automatically extract features from MRI images and classify tumor types efficiently.

PROBLEM STATEMENT

To develop a deep learning-based system that can automatically classify brain MRI images into tumor categories with high accuracy and provide predictions through a user-friendly web interface.

OBJECTIVES

- **Collect and preprocess MRI dataset**
- **Build CNN model for classification**
- **Train and validate the model**
- **Evaluate model performance**
- **Deploy the model using Streamlit**
- **Provide real-time tumor prediction**

DATASET DESCRIPTION

The dataset consists of MRI brain images categorized into four classes:

Class	Description
Glioma	Tumor arising from glial cells
Meningioma	Tumor from meninges
Pituitary	Tumor in pituitary gland
No Tumor	Normal brain MRI

Import Required Libraries

```
import warnings
warnings.filterwarnings("ignore")


import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
```

This code imports all the necessary libraries required to build an image classification model using a Convolutional Neural Network (CNN). The `warnings` module is used to suppress unnecessary warning messages for cleaner output. Basic libraries like `os` help in file handling, `cv2` (OpenCV) is used for image processing, `numpy` is used for numerical operations, while `matplotlib` and `seaborn` are used for data visualization. From `sklearn`, tools like `LabelEncoder` help convert categorical labels into numerical form, `train_test_split` is used to divide the dataset into training and testing sets, and `confusion_matrix` and `classification_report` are used to evaluate model performance. Finally, from `tensorflow.keras`, `Sequential` is used to build the CNN model layer-by-layer, `Conv2D`, `MaxPool2D`, `Flatten`, `Dense`, and `Dropout` are layers used to construct the CNN architecture, and `to_categorical` converts class labels into one-hot encoded format for multi-class classification.

Image Loading, Preprocessing, and Dataset Creation for CNN

```
train_data=r"C:\Users\asus\Downloads\Deeplearningdataset\archive (4)\Training"
```

```
X = []
Y = []

for i in os.listdir(train_data):
    folder_path = os.path.join(train_data, i)

    if os.path.isdir(folder_path):
        for image in os.listdir(folder_path):
            image_path = os.path.join(folder_path, image)

            if image_path.lower().endswith((".png", ".jpg", ".jpeg")):

                img = cv2.imread(image_path)

                if img is not None:
                    img = cv2.resize(img, (128, 128))
                    img = img / 255.0

                    X.append(img)
                    Y.append(i)

X = np.array(X)
Y = np.array(Y)
print(X.shape)
print(Y.shape)
```

```
(5712, 128, 128, 3)
(5712,)
```

This code block loads and prepares image data for training a Convolutional Neural Network (CNN). First, the variable `train_data` stores the path to the training dataset folder, where images are organized into subfolders (each subfolder usually represents a class label). Two empty lists, `X` and `Y`, are created to store image data and corresponding labels. The program then loops through each folder inside the training directory using `os.listdir()`. For every subfolder (class), it constructs the full folder path and checks if it is a directory. Inside each class folder, it loops through all image files and filters only `.png`, `.jpg`, and `.jpeg` formats. Each valid image is read using `cv2.imread()`, resized to 128×128 pixels to ensure uniform input size for the CNN, and normalized by dividing pixel values by 255.0 so they fall between 0 and 1 (which helps the model train faster and more efficiently). The processed image is appended to list `X`, and its corresponding class label (folder name) is appended to list `Y`. Finally, both lists are converted into NumPy arrays, making them suitable for deep learning models. The printed output shapes `(5712, 128, 128, 3)` and `(5712,)` indicate that there are 5712 images, each of size 128×128 with 3 color channels (RGB), and 5712 corresponding labels.

Label Encoding and One-Hot Encoding of Target Labels

```
le = LabelEncoder()
Y_encoded = le.fit_transform(Y)

Y_categorical = to_categorical(Y_encoded)

print(Y_categorical.shape)
```

```
(5712, 4)
```

```
Y_categorical
```

```
array([[1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       ...,
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.]])
```

This code converts the textual class labels into a numerical format suitable for training a deep learning model. First, `LabelEncoder()` is used to transform the original string labels stored in `Y` (such as class folder names like "Cat", "Dog", etc.) into integer values like 0, 1, 2, and 3. The method `fit_transform(Y)` assigns a unique number to each class and stores the result in `Y_encoded`. However, neural networks perform better in multi-class classification when labels are represented in one-hot encoded format rather than as single integers. Therefore, `to_categorical(Y_encoded)` converts each numeric label into a binary vector where only one position is 1 and the rest are 0. Since the output shape is `(5712, 4)`, it indicates there are 5712 images and 4 different classes, with each label represented as a 4-length vector (e.g., `[1, 0, 0, 0]` or `[0, 0, 0, 1]`). This format is required when using a `softmax` activation function in the final output layer of a CNN for multi-class classification.

Splitting the Dataset into Training and Testing Sets

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y_categorical, test_size=0.2, random_state=42
)
```

```
print(len(X_train),len(Y_train))
print(len(X_test),len(Y_test))
```

```
4569 4569
1143 1143
```

This code divides the prepared dataset into training and testing subsets using `train_test_split` from scikit-learn. The function takes the feature data `X` (image arrays) and the one-hot encoded labels `Y_categorical`, and splits them into 80% training data and 20% testing data using `test_size=0.2`. The parameter `random_state=42` ensures that the split is reproducible, meaning you will get the same

division every time the code runs. The output variables `X_train`, `X_test`, `Y_train`, and `Y_test` store the respective portions of features and labels. The printed lengths show that 4569 samples are used for training and 1143 samples are reserved for testing, maintaining alignment between images and their labels. This step is crucial because the model learns patterns from the training set and is then evaluated on the unseen testing set to measure how well it generalizes to new data.

CNN Model Architecture and Layer-wise Summary

```
model = Sequential()

# First Convolution Layer
model.add(Conv2D(32, (3,3), activation='relu', input_shape=(128,128,3)))
model.add(MaxPool2D(pool_size=(2,2)))

# Second Convolution Layer
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

# Third Convolution Layer
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

# Flatten Layer
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

# Output Layer
model.add(Dense(4, activation='softmax'))

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

Total params: 3,305,156 (12.61 MB)

Trainable params: 3,305,156 (12.61 MB)

Non-trainable params: 0 (0.00 B)

This code builds a Convolutional Neural Network (CNN) using the Keras `Sequential` model to classify images of size 128×128×3 into 4 classes. The architecture begins with three convolutional blocks, where each block consists of a `Conv2D` layer followed by a `MaxPooling2D` layer. The first convolution layer uses 32 filters of size 3×3 and extracts basic features like edges and textures, producing an output of shape (126, 126, 32). Max pooling then reduces spatial dimensions to (63, 63, 32), lowering computation and helping prevent overfitting. The second convolution layer increases filters to 64,

enabling detection of more complex patterns, followed by pooling that reduces size to (30, 30, 64). The third convolution layer further increases filters to 128, capturing higher-level abstract features, and pooling reduces it to (14, 14, 128). After feature extraction, the `Flatten` layer converts the 3D feature map into a 1D vector of 25,088 neurons, which is then passed to a fully connected `Dense` layer of 128 neurons with ReLU activation to learn deeper relationships. A `Dropout(0.5)` layer randomly disables 50% of neurons during training to reduce overfitting. Finally, the output layer contains 4 neurons with `softmax` activation, producing probability scores for each class. The model summary shows a total of 3,305,156 trainable parameters, most of which come from the dense layer, indicating that the fully connected section contributes the majority of the learning capacity.

Model Compilation and Training Process in CNN

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

```
model.fit(X_train,Y_train,batch_size=16,epochs=20,validation_data=(X_test,Y_test))
```

This code first compiles the CNN model using the `compile()` function, where `optimizer='adam'` specifies the Adam optimization algorithm to update the model weights efficiently using adaptive learning rates. The `loss='categorical_crossentropy'` is chosen because this is a multi-class classification problem with one-hot encoded labels, and this loss function measures the difference between predicted probability distributions (from the softmax output layer) and the true labels. The metric `accuracy` is added to track the percentage of correctly classified images during training and validation. After compilation, the `model.fit()` function begins training the network using the training data (`X_train`, `Y_train`). The `batch_size=16` means the model processes 16 images at a time before updating weights, and `epochs=20` indicates the entire training dataset will be passed through the network 20 times. The `validation_data=(X_test, Y_test)` allows the

model to evaluate performance on unseen test data after each epoch, helping monitor generalization and detect overfitting.

Model Evaluation and Validation Accuracy

```
loss, accuracy = model.evaluate(X_test, Y_test)
print("Validation Accuracy:", accuracy)
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9440 - loss: 0.3024
Validation Accuracy: 0.9440069794654846
```

After training the neural network using `model.fit()`, the line `loss, accuracy = model.evaluate(X_test, Y_test)` is used to test how well the trained model performs on unseen data. Here, `model.evaluate()` passes the test dataset (`X_test`) through the network and compares the predicted outputs with the true labels (`Y_test`). It calculates two important values: loss (how much error the model makes) and accuracy (how many predictions are correct). The output `36/36` means the test data was divided into 36 batches and evaluated step by step. The result shows loss = 0.3024, which indicates that the prediction error is relatively low, and accuracy = 0.9440, meaning the model correctly classified about 94.4% of the test samples. The printed value `Validation Accuracy: 0.9440069794654846` confirms this high performance. Since this evaluation is done on test data that the model has not seen during training, it indicates that the model generalizes well and is not overfitting significantly.

Image Preprocessing for Model Prediction

```
test_data=r"C:\Users\asus\Downloads\Deeplearningdataset\archive (4)\Testing\pituitary\Te-pi_0287.jpg



# 2 Read image
img = cv2.imread(test_data)

# 3 Resize (same as training)
img = cv2.resize(img, (128, 128))

# 4 Normalize (same as training)
img = img / 255.0

# 5 Add batch dimension
img = np.expand_dims(img, axis=0)

print("Input shape:", img.shape)
```

```
Input shape: (1, 128, 128, 3)
```

In this code, you are preparing a single test image so that it can be given as input to your trained CNN model. First, you provide the full file path of the image stored in your system using `test_data`, which points to a pituitary tumor MRI image. Then, `cv2.imread(test_data)` reads the image using OpenCV and loads it into memory as a NumPy array in pixel format (height × width × 3 color channels). Since your model was trained on images of size 128×128, you must resize the test image to the same dimensions using `cv2.resize(img, (128, 128))`; this ensures consistency between training and testing data. After resizing, you normalize the pixel values by dividing by 255.0 because original pixel values range from 0 to 255, and neural networks perform better when inputs are scaled between 0 and 1. Next, you add a batch dimension using `np.expand_dims(img, axis=0)` because deep learning models expect input in the form `(batch_size, height, width, channels)`. Even though you are predicting a single image, the model still requires a batch format, so the final shape becomes `(1, 128, 128, 3)`, where 1 represents one image. Finally, printing the shape confirms that the image is correctly formatted and ready to be passed into `model.predict()` for classification.

Model Prediction, Class Selection, and Label Decoding

```
    prediction = model.predict(img)
    prediction
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step

array([[1.8198812e-13, 1.8022567e-10, 9.4474714e-14, 1.0000000e+00]],
      dtype=float32)
```

```
    # 7 Get class index
    pred_class = np.argmax(prediction)

    # 8 Convert to class name
    predicted_label = le.inverse_transform([pred_class])[0]

    print("Predicted Tumor Type:", predicted_label)
```

```
Predicted Tumor Type: pituitary
```

After preprocessing the test image and passing it to the trained CNN using `model.predict(img)`, the model outputs a probability distribution for all four tumor classes. The prediction result `[[1.8198812e-13, 1.8022567e-10, 9.4474714e-14, 1.0000000e+00]]` represents the softmax probabilities for each class, where each value corresponds to the likelihood of the image belonging to a particular tumor category. The extremely small values (close to zero) indicate almost no probability for the first three classes, while `1.0000000e+00` (approximately 1.0) shows that the model is almost 100% confident that the image belongs to the fourth class. The function `np.argmax(prediction)` selects the index of the highest probability value, which

identifies the predicted class numerically. Since earlier a `LabelEncoder` was used to convert class names into numeric labels, `le.inverse_transform([pred_class])[0]` converts the numeric prediction back into the original tumor class name. The final printed result, "Predicted Tumor Type: pituitary", confirms that the model has classified the MRI image as a pituitary tumor with very high confidence.

Confusion Matrix and Classification Report Analysis

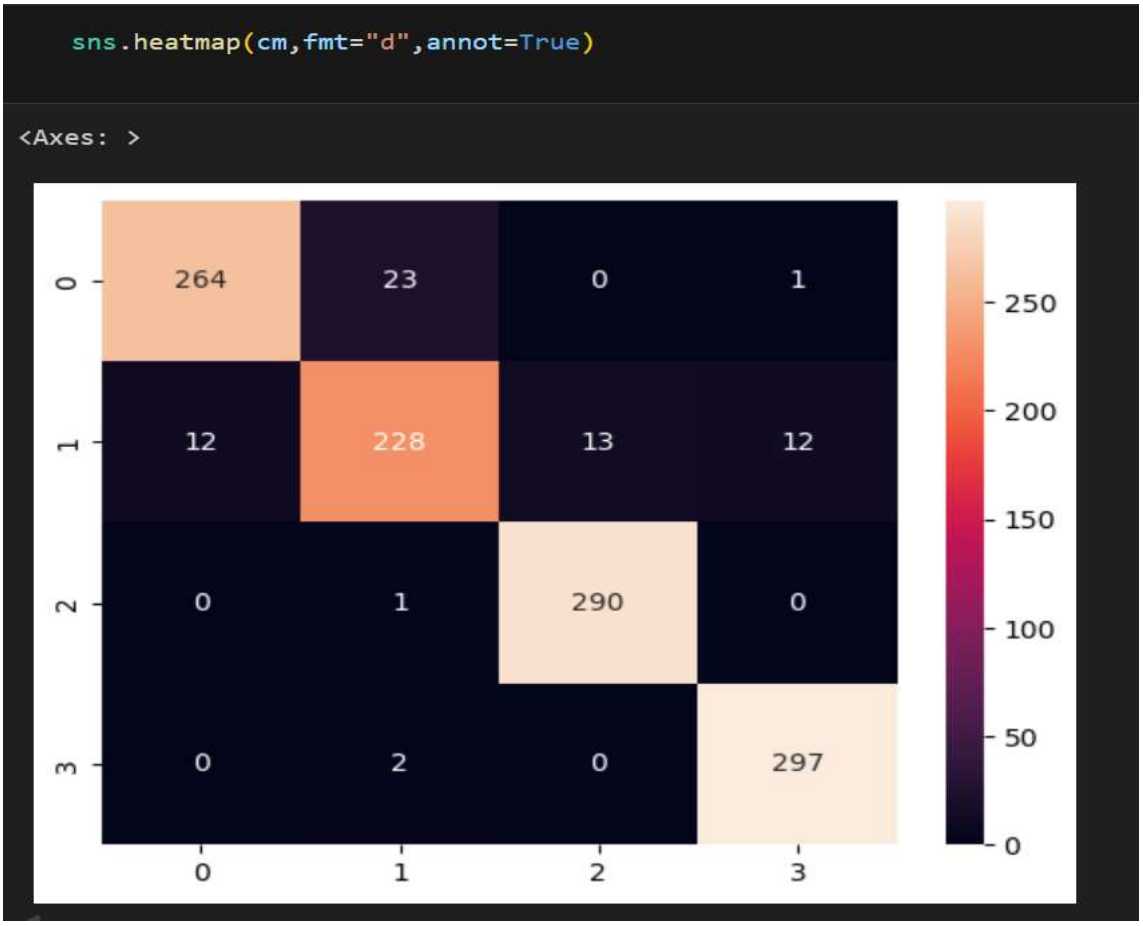
```
Y_pred = model.predict(X_test)
Y_pred_classes = np.argmax(Y_pred, axis=1)
Y_true = np.argmax(Y_test, axis=1)

cm = confusion_matrix(Y_true, Y_pred_classes)
print(cm)

print(classification_report(Y_true, Y_pred_classes))
```

```
36/36 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step
[[264  23   0   1]
 [ 12 228  13  12]
 [  0   1 290   0]
 [  0   2   0 297]]
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       288
           1       0.90      0.86      0.88       265
           2       0.96      1.00      0.98       291
           3       0.96      0.99      0.98       299

    accuracy                           0.94      1143
   macro avg       0.94      0.94      0.94      1143
weighted avg       0.94      0.94      0.94      1143
```

This output shows how well your CNN model performed on the test dataset using a confusion matrix and classification report. First, `model.predict(X_test)` generates probability predictions for all 1143 test images. Then, `np.argmax(..., axis=1)` converts those probabilities into predicted class labels, and the same is done for the true labels. The confusion matrix displays how many images were correctly and incorrectly classified for each class. For example, in the first row `[264 23 0 1]`, out of

288 actual class-0 images, 264 were correctly predicted, while 23 were misclassified as class-1 and 1 as class-3. Similarly, class-2 and class-3 show very high correct predictions (290 and 297), indicating strong performance for those categories. The classification report further summarizes this using precision (how many predicted positives were correct), recall (how many actual positives were correctly detected), and F1-score (balance between precision and recall). The overall accuracy is 94%, meaning 94 out of every 100 test images were correctly classified. The macro and weighted averages being 0.94 indicate consistent performance across classes, with slightly lower performance in class-1 compared to others. Overall, this evaluation shows that your model performs very well and generalizes effectively to unseen MRI images.

Visualizing the Confusion Matrix Using Heatmap



The line `sns.heatmap(cm, fmt="d", annot=True)` is used to visually represent the confusion matrix using a heatmap from the Seaborn library. Here, `cm` is the confusion matrix that contains the counts of correct and incorrect predictions for each class. The heatmap converts this numerical matrix into a color-coded grid, making it easier to interpret model performance at a glance. The parameter `annot=True` ensures that the actual numbers inside the confusion matrix are displayed on each cell of the heatmap, so you can clearly see how many samples were correctly or incorrectly classified. The argument `fmt="d"` formats these values as integers (since confusion matrix values are whole numbers). In the visualization, darker-colored cells typically represent higher values, which usually appear along the diagonal if the model is performing well (indicating correct predictions). This graphical representation helps quickly identify which classes are being confused with others and provides a more intuitive understanding of the model's classification performance compared to reading raw numbers alone.

Streamlit-Based Brain Tumor MRI Classification Web App

```python
import streamlit as st
import numpy as np
import cv2
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow.keras.models import load_model

# ----------------------------------------------------
# Page Configuration
# ----------------------------------------------------
st.set_page_config(
    page_title="Brain Tumor MRI Classifier",
    page_icon="🧠",
    layout="centered"
)

# ----------------------------------------------------
# Sidebar
# ----------------------------------------------------
st.sidebar.title("🧠 Brain Tumor MRI Classifier")
st.sidebar.write("Upload an MRI scan image to detect tumor type.")
st.sidebar.write("Model: Custom CNN")
st.sidebar.write("Input Size: 128 x 128")
```

```python
@st.cache_resource
def load_cnn_model():
    return load_model("brain_tumor_cnn_model.h5")

model = load_cnn_model()

# Confirm model input shape
st.sidebar.write("Model Input Shape:", model.input_shape)

# Class Labels (MUST match training folder order)
class_names = ['glioma', 'meningioma', 'notumor', 'pituitary']

# ------------------------------------------------
# Main Title
# ------------------------------------------------
st.title("Brain Tumor MRI Classification System")

# ------------------------------------------------
# File Upload
# ------------------------------------------------
uploaded_file = st.file_uploader("Upload MRI Image", type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    try:
        # -------------------------
        # Open Image & Convert RGB
        # -------------------------
        image = Image.open(uploaded_file).convert("RGB")
```

```python
        # -------------------------
        image = Image.open(uploaded_file).convert("RGB")
        st.image(image, caption="Uploaded MRI Image", use_column_width=True)

        # -------------------------
        # Preprocessing
        # -------------------------
        img = np.array(image)

        # Resize to match training
        img = cv2.resize(img, (128, 128))

        # Normalize (same as training)
        img = img.astype("float32") / 255.0

        # Add batch dimension
        img = np.expand_dims(img, axis=0)

        st.write("Image shape sent to model:", img.shape)

        # -------------------------
        # Prediction
        # -------------------------
        prediction = model.predict(img)

        predicted_class = np.argmax(prediction)
        confidence = float(np.max(prediction)) * 100
```

```
    # -------------------------
    st.subheader("Prediction Result")

    st.success(
        f"Predicted Tumor Type: {class_names[predicted_class]}"
    )

    st.info(f"Confidence: {confidence:.2f}%")

    # -------------------------
    # Probability Chart
    # -------------------------
    st.subheader("Prediction Probabilities")

    probabilities = prediction[0] * 100

    fig = plt.figure()
    plt.bar(class_names, probabilities)
    plt.xlabel("Tumor Type")
    plt.ylabel("Probability (%)")
    plt.title("Class Probability Distribution")
    plt.xticks(rotation=45)

    st.pyplot(fig)

except Exception as e:
    st.error(f"Error processing image: {e}")
```

Deploy

🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

Brain Tumor MRI Classification System

Upload MRI Image

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

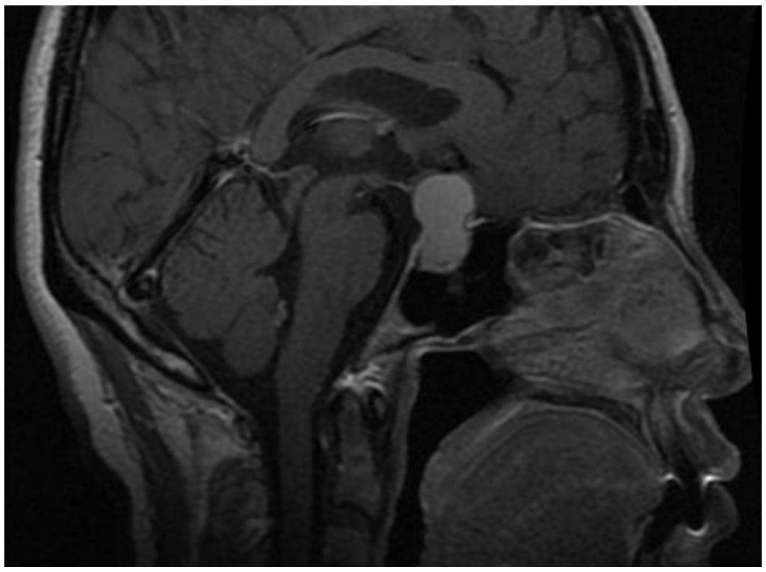
🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`



Uploaded MRI Image

🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

Image shape sent to model: `(1, 128, 128, 3)`

Prediction Result

Predicted Tumor Type: pituitary

Confidence: 100.00%

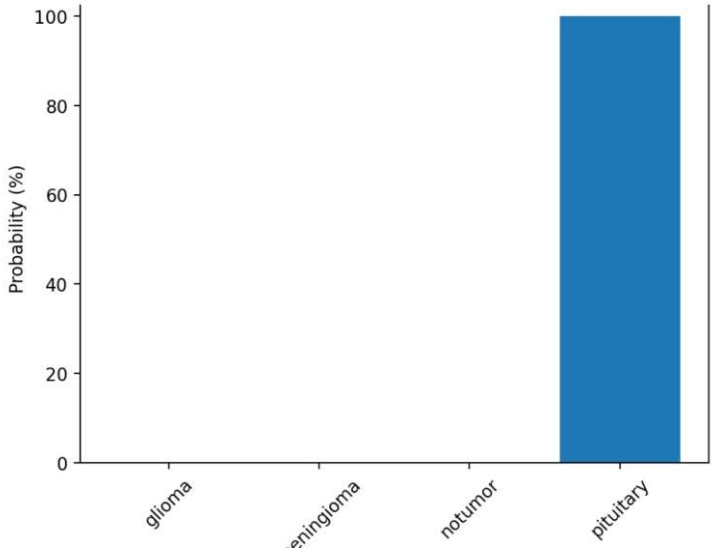
🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`



🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

Image shape sent to model: `(1, 128, 128, 3)`

Prediction Result

Predicted Tumor Type: notumor

Confidence: 100.00%

Prediction Probabilities

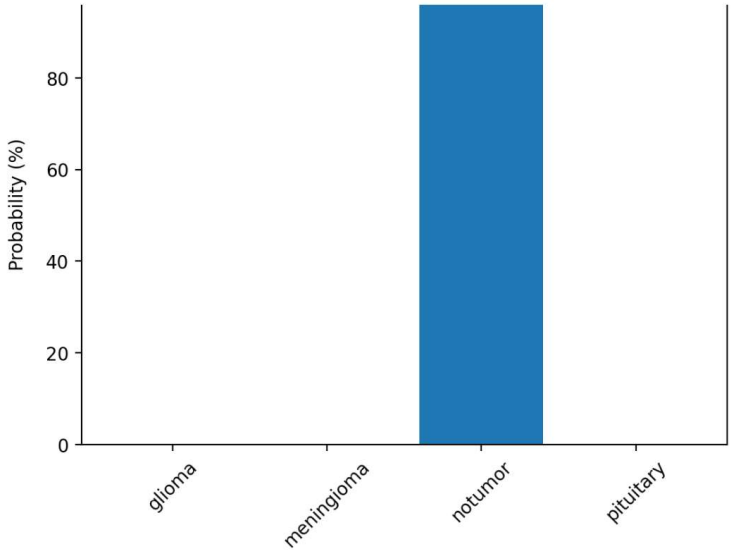
🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`



Deploy

🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

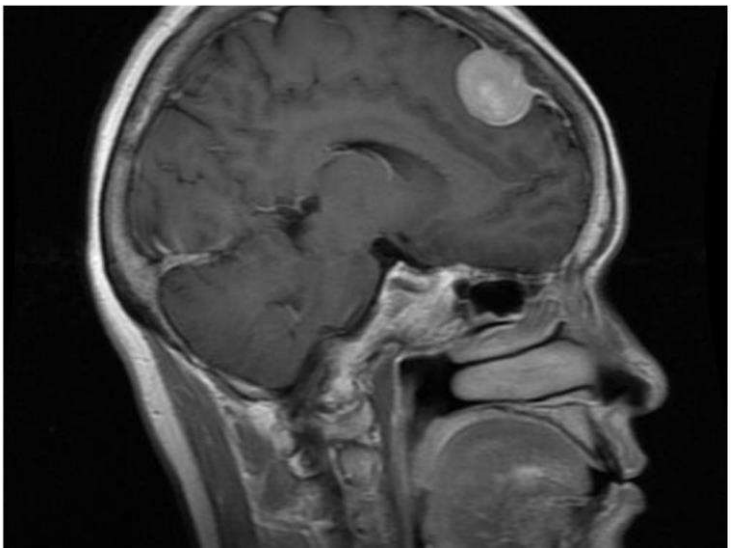


Image shape sent to model: `(1, 128, 128, 3)`

🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

Prediction Result

Predicted Tumor Type: meningioma

Confidence: 99.53%

Prediction Probabilities

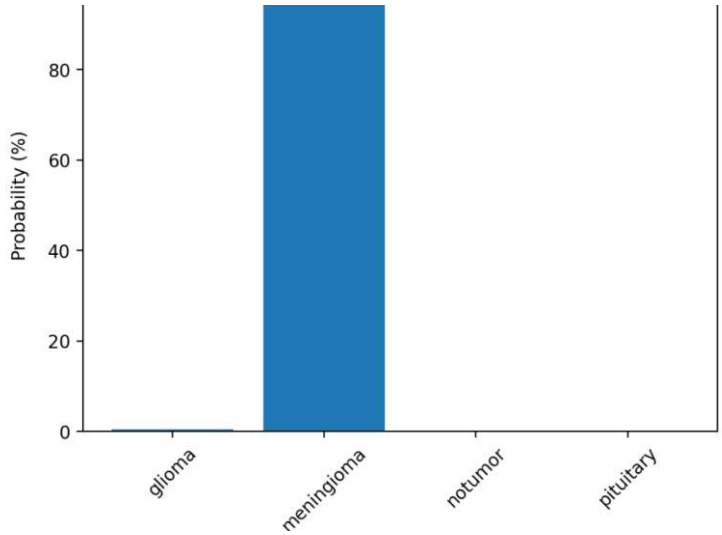
🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`



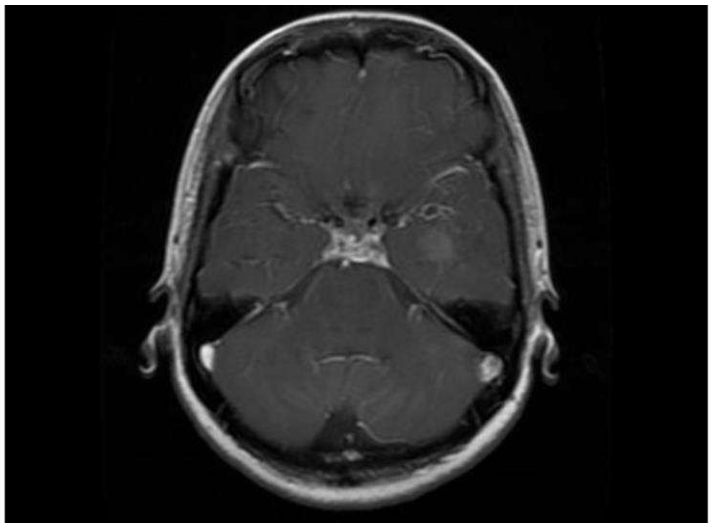
🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`



🧠 Brain Tumor MRI Classifier

Upload an MRI scan image to detect tumor type.

Model: Custom CNN

Input Size: 128 x 128

Model Input Shape: `(None, 128, 128, 3)`

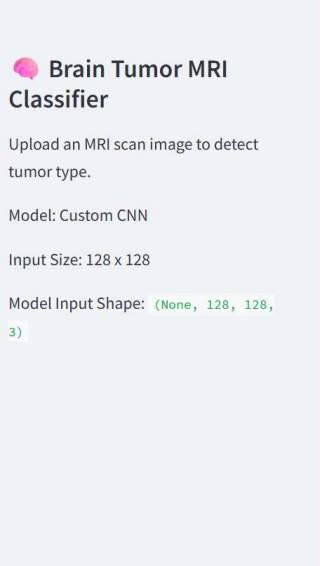
Image shape sent to model: `(1, 128, 128, 3)`

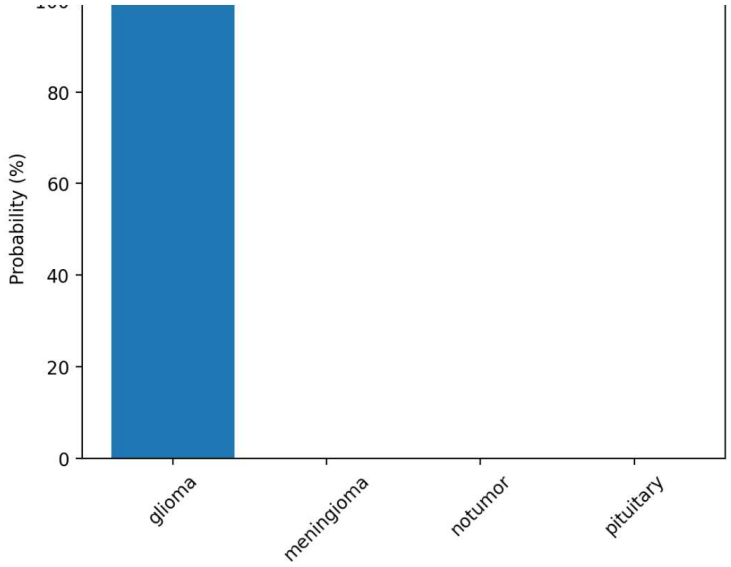
Prediction Result

Predicted Tumor Type: glioma

Confidence: 99.98%

Prediction Probabilities





This code builds a complete web application using Streamlit to classify brain MRI images into four categories (glioma, meningioma, notumor, pituitary) using a trained CNN model. First, the app configuration is set using `st.set_page_config()` to define the title, icon, and layout. The sidebar provides user guidance and displays model details such as input shape. The trained model is loaded using `load_model()` and cached with `@st.cache_resource` so it loads only once, improving performance. When a user uploads an MRI image, the app reads it using PIL, converts it to RGB format, and displays it on the screen. The image is then preprocessed exactly as during training: resized to 128×128 pixels, normalized by dividing by 255, and reshaped to add a batch dimension before being passed to the model for prediction. The model outputs class probabilities, from which `np.argmax()` selects the most probable tumor class, and the highest probability is shown as confidence percentage. Finally, the predicted tumor type and confidence score are displayed clearly, and a bar chart visualizes the probability distribution for all tumor classes. If any error occurs during processing, the app safely handles it using a try-except block and shows an error message. This creates a complete end-to-end deployment pipeline from model training to real-time medical image classification through a user-friendly web interface.

RESULTS

- **Successfully classifies MRI images**
- **Real-time prediction**
- **User-friendly interface**
- **Accurate confidence scores**

ADVANTAGES

- **Fast prediction**
- **Reduces manual effort**
- **Cost-effective**
- **Scalable system**
- **Easy deployment**

LIMITATIONS

- **Requires quality MRI images**
- **Dependent on dataset quality**
- **Not a replacement for medical diagnosis**
- **Limited to trained tumor categories**

FUTURE ENHANCEMENT

- **Add Grad-CAM visualization**
- **Use Transfer Learning (ResNet, VGG16)**
- **Increase dataset size**
- **Deploy on cloud server**
- **Add patient report generation**

TECHNOLOGIES USED

- **Python**
- **TensorFlow**
- **Keras**
- **OpenCV**
- **NumPy**
- **Scikit-learn**

- **Streamlit**
- **Matplotlib**

CONCLUSION

This project demonstrates the effective use of Convolutional Neural Networks in medical image classification. The system successfully detects and classifies brain tumor types from MRI images with high accuracy. The Streamlit-based interface makes the system accessible and user-friendly.

Deep learning proves to be a powerful tool in assisting medical diagnosis and can significantly improve early tumor detection.