Report: CNN Model for MRI Image Classification

1. Introduction

This report documents the development of a convolutional neural network (CNN) designed to classify MRI images as having a tumor or no tumor. The project leverages deep learning techniques for binary image classification, demonstrating the practical application of artificial intelligence in healthcare.

2. Objective

The primary objective of this project is to build and evaluate a CNN model capable of distinguishing between MRI images with and without tumors, ensuring high accuracy and reliability to support medical diagnostics.

3. Dataset Details

The dataset includes MRI images divided into training, validation, and test sets:

Data Organization

The dataset was pre-processed and organized into the following structure:

1. **Training Data**: Used for model training.
2. **Validation Data**: Used to tune the model and monitor performance during training.
3. **Test Data**: Used for final evaluation of the model.

Dataset Source and Pre-processing

This dataset, titled **TumorSegmentation**, was provided by Roboflow and exported on August 19, 2023. Additionally, the dataset is available on Kaggle under the name **Brain Tumor Image Dataset: Semantic Segmentation**. The following details pertain to the Kaggle dataset:

- **Project Type**: Semantic Segmentation
- **Classes**:
    - **Class 0 (Non-Tumor)**: Pixels labeled as non-tumor represent areas without any tumor presence in the medical images.
    - **Class 1 (Tumor)**: Pixels labeled as tumors correspond to areas where tumors are detected.
- **Purpose**: The dataset is intended for semantic segmentation tasks, where each pixel is classified as part of a tumor or non-tumor region. This approach provides an accurate spatial understanding of tumor distribution within medical images.
- **Last Updated**: 9 months ago

- **License**: CC BY 4.0

Preprocessing Steps

The following preprocessing steps were applied to the dataset:

1. **Auto-orientation**: Adjusted pixel data to correct orientation and stripped EXIF data.
2. **Resizing**: All images were resized to 150x150 (as specified in the implementation).
3. **No augmentation**: The dataset did not include additional augmentation techniques at the source.
4. **Data Splitting**: Organized raw data into structured directories for training, validation, and testing.
5. **Dataset Organization**: Images were organized into class-based directories using COCO annotations for tumor and non-tumor classes. The script ensured proper mapping and structure.

5.**Methodology**

5.1 Library Imports

```
import os

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report
```

5.2 Provide Addresses of training dataset and validation dataset

```
train_dir = r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\train"

val_dir = r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\valid"

test_dir=r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\test"
```

5.3 Organize the data into folders

As our Dataset in not in a desirable format we need to sepearate the images in train,tes

and validation sets into images which have tumor or not .

This code is simply to organize the data.

```
import os
```

```
import json
import shutil

# Define the base directory for the dataset
base_dir = r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)"

# Manually specify the exact paths to the annotation files
annotation_paths = {
    "train": r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\train\_annotations.coco.json",  # Replace with the correct path
    "valid": r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\valid\_annotations.coco.json",  # Replace with the correct path
    "test": r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\test\_annotations.coco.json",  # Replace with the correct path
}

# Classes
classes = ["tumor", "no_tumor"]

# Function to organize images based on COCO annotations
def organize_images_from_coco(dataset_name, annotation_file):
    dataset_dir = os.path.join(base_dir, dataset_name)
    image_dir = dataset_dir  # Images are in the same folder

    # Create class-based directories
    for class_name in classes:
        class_dir = os.path.join(dataset_dir, class_name)
        os.makedirs(class_dir, exist_ok=True)

    # Read the COCO annotation file
    with open(annotation_file, "r") as f:
        data = json.load(f)
```

```
    # Map image IDs to their respective classes
    image_classes = {}
    for annotation in data["annotations"]:
        image_id = annotation["image_id"]
        category_id = annotation["category_id"]
        if category_id == 1:  # Tumor class
            image_classes[image_id] = "tumor"
        else:
            image_classes[image_id] = "no_tumor"

    # Move images to respective class folders
    for image_info in data["images"]:
        image_id = image_info["id"]
        filename = image_info["file_name"]
        class_name = image_classes.get(image_id, "no_tumor")  # Default to "no_tumor"
        src_path = os.path.join(image_dir, filename)
        dest_dir = os.path.join(dataset_dir, class_name)
        dest_path = os.path.join(dest_dir, filename)

        if os.path.exists(src_path):
            shutil.move(src_path, dest_path)
        else:
            print(f"Warning: {src_path} not found.")

# Organize images for each dataset
for dataset_name, annotation_file in annotation_paths.items():
    print(f"Organizing {dataset_name} dataset...")
    if not os.path.exists(annotation_file):
        print(f"Error: Annotation file not found at {annotation_file}")
        continue
```

```
    organize_images_from_coco(dataset_name, annotation_file)

print("Dataset organization complete.")
```

5.4 Preprocess the data using ImageDataGenerator

```
train_datagen = ImageDataGenerator(rescale=1.0/255.0)
val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'  # Binary classification (tumor or no tumor)
)

val_data = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'  # Binary classification
)
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Maintain order for evaluation
)
```

5.5 Defining the model Architecture

```
# Initialize the model
model = Sequential()

# Add an Input layer
model.add(Input(shape=(150, 150, 3)))

# Convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

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

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

# Flatten the feature maps into a vector
model.add(Flatten())

# Fully connected (dense) layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout to reduce overfitting
model.add(Dense(1, activation='sigmoid'))  # Sigmoid activation for binary classification
```

5.6 Compile the model

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

5.7 Generate Model summary

model.summary()

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 128)	4,735,104
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Additional Parameters

Total Parameters: 4,828,481 (18.42 MB)

Trainable Parameters: 4,828,481 (18.42 MB)

Non-trainable Parameters: 0 (0.00 B)

5.8 Train the model

```
history = model.fit(
    train_data,
    epochs=10,
    batch_size=32,
    validation_data=val_data)
```

5.9 Save the trained model

```
model.save('tumor_detection_model.keras')
print("Model saved as tumor_detection_model.keras")
```

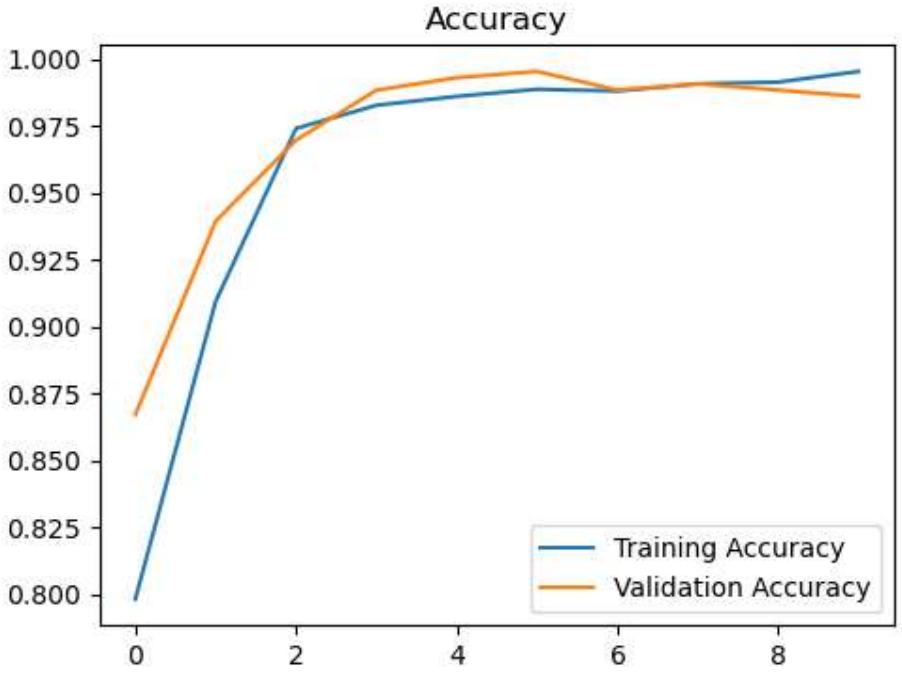
5.10 Plot training history (accuracy)

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.legend()

plt.title('Accuracy')



5.11 Calculate Training and Validation accuracy after each epoch

print("Training Accuracy: ", history.history['accuracy'])

print("Validation Accuracy: ", history.history['val_accuracy'])

Training Accuracy: [0.7982689738273621, 0.9094540476799011, 0.9740346074104309, 0.9826897382736206, 0.9860186576843262, 0.9886817336082458, 0.9880159497261047, 0.9906790852546692, 0.9913448691368103, 0.995339572429657]

Validation Accuracy: [0.867132842540741, 0.939393937587738, 0.9696969985961914, 0.9883449673652649, 0.9930070042610168, 0.9953380227088928, 0.9883449673652649, 0.9906759858131409, 0.9883449673652649, 0.9860140085220337]

5.12 Plot training history(loss)

plt.subplot(1, 2, 2)

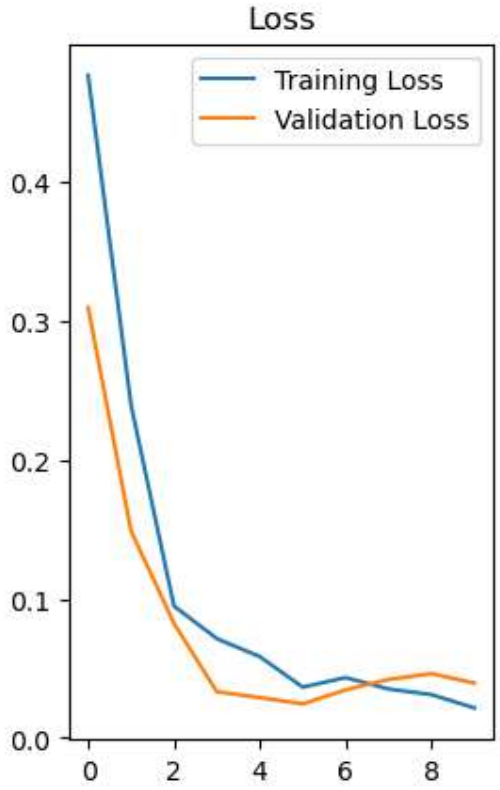
plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.legend()

plt.title('Loss')

plt.show()



5.13 Predict the class labels for the validation dataset and compare with actual labels

```
def predict_image(image_path, model):
    from tensorflow.keras.preprocessing import image
    img = image.load_img(image_path, target_size=(150, 150))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    prediction = model.predict(img_array)
    return "Tumor Present" if prediction[0][0] > 0.5 else "No Tumor"
```

5.14 Test the model with a new image

```
test_image_path = r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\test\tumor\1864_jpg.rf.6708a8791c3ad5158aec5efed476784c.jpg"
result = predict_image(test_image_path, model)
print(f"Prediction for test image: {result}")
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

5.15 Define the test data generator

```
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Specify the test data directory (adjust path)
test_dir = r"C:\Users\LENOVO\OneDrive\Desktop\tumor classification cnn\archive (10)\test"  # Replace with your actual test directory path

# Load the test data
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),  # Adjust according to your input size
    batch_size=32,
    class_mode='binary'  # Assuming binary classification (tumor/no_tumor)
)
```

5.16 Evaluate the model on the test data

```
test_loss, test_accuracy = model.evaluate(test_data, steps=len(test_data))

print("Test Accuracy: ", test_accuracy)



# Data generators for preprocessing and augmentation
train_datagen = ImageDataGenerator(rescale=1.0/255.0)
val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_data = train_datagen.flow_from_directory(
    train_dir,
```

```
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

Output: Found 1502 images belonging to 2 classes.

```
val_data = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'
)
```

Output: Found 429 images belonging to 2 classes.

```
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

Output: Found 215 images belonging to 2 classes.

```
from sklearn.metrics import classification_report
import numpy as np
```

5.17 Get the true labels for the test dataset

```
y_true = val_data.classes  # For validation set (use test_data.classes for test set)
```

5.18 Get the predicted labels

```
y_pred = model.predict(val_data)
```

y_pred = (y_pred > 0.5).astype("int32") # Convert probabilities to binary labels (0 or 1)

5.19 Generate the classification report

report = classification_report(y_true, y_pred)

print(report)

5.20 Classification report

	Precision	**F1-Score** **Recall**	**Support** **F1-Score**	**Support**
0	0.51	0.51	0.51	219
1	0.49	0.49	0.49	210
Accuracy			0.50	429
Macro avg	0.50	0.50	0.50	429
Weighted avg	0.50	0.50	0.50	429

Results

The model achieved a test accuracy of

Test Accuracy: 0.9813953638076782

Block diagram