Assessment Report

On

“Brain Tumor Detection”

Submitted as partial fulfilment for the award of

BACHELOR OF TECHNOLOGY

DEGREE

SESSION: 2024 – 2025

CSE (Artificial Intelligence)

By

Rachit Garg

Priyanshi

Mayank Chaudhary

Parth Rathore

Mohd. Suleman Aqdam

KIET Group of Institutions, Ghaziabad

Affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow

1. Introduction

This project focuses on developing a deep learning model to automatically detect brain tumors from MRI images using Convolutional Neural Networks (CNNs). By training the model on a dataset of labeled brain scans, it learns to classify images as either "tumor" or "no tumor." The goal is to assist in early diagnosis by providing fast and accurate predictions. The project also includes visualizing model performance and predictions to ensure transparency and effectiveness in real-world applications.

---

2. Problem Statement

Build a Convolutional Neural Network (CNN) model that can classify MRI images into **"Tumor"** or **"No Tumor"** categories. Visualize predictions and evaluate performance using standard metrics.

---

3. Objectives

- **To develop a CNN-based model** that can accurately classify MRI images as containing a brain tumor or not.
- **To preprocess and augment MRI image data** for improved training and generalization of the model.
- **To evaluate the model's performance** using metrics like accuracy, precision, recall, and F1-score.
- **To visualize training results and predictions** for better interpretability and model analysis.
- **To reduce diagnostic time and human error** by providing an automated solution for brain tumor detection.
- **To build a scalable and deployable model** that can assist medical professionals in real-world settings.

---

4. Methodology

- **Data Collection**: MRI images of brains with and without tumors are collected from a publicly available dataset.

- **Data Preprocessing**:

    - Encoding labels as binary (0 = No Tumor, 1 = Tumor).

    - Classifying the dataset into tumor and no tumor.

    - Resizing all images to a consistent dimension (e.g., 150×150 pixels).

- **Model Building**:

    - Designing a Convolutional Neural Network (CNN) architecture suitable for binary image classification.

    - Adding layers like Conv2D, MaxPooling2D, Flatten, Dense, and Dropout for optimal performance.

    - Compiling the model with the Adam optimizer and binary cross-entropy loss function.

- **Model Training**:

    - Designing a Convolutional Neural Network (CNN) architecture suitable for binary image classification.

    - Adding layers like Conv2D, MaxPooling2D, Flatten, Dense, and Dropout for optimal performance.

- **Model Evaluation**:
    - Evaluating performance using metrics such as accuracy, precision, recall, and F1-score.
    - Visualizing training and validation accuracy/loss curves.
    - Displaying sample predictions and generating a confusion matrix heatmap for performance insight.

---

5. Data Preprocessing:

The MRI image dataset is cleaned and prepared as follows:

- **Images are resized** to a fixed dimension (e.g., 150×150 pixels) to ensure uniform input size for the model.
- **Pixel values are normalized** by scaling them between 0 and 1 to improve training performance.
- **Class labels are encoded** as binary values: 0 for “No Tumor” and 1 for “Tumor.”
- **Data augmentation** techniques like rotation, zoom, and flipping are applied to increase dataset diversity.
- The dataset is **split into 80% training and 20% testing** to evaluate model performance on unseen data.

---

6. Model Implementation

A **Convolutional Neural Network (CNN)** is implemented due to its high effectiveness in image classification tasks. The model is built using several convolutional and pooling layers, followed by fully connected layers to learn patterns from MRI scans. The CNN is trained on the preprocessed image dataset to predict whether a brain tumor is present. The model is evaluated on the test set to determine its accuracy and diagnostic reliability.

---

7. Evaluation Metrics

The CNN model is evaluated using the following metrics:

- **Accuracy**: Overall correctness of predictions.
- **Precision**: Proportion of predicted tumor cases that are actually tumors.
- **Recall**: Proportion of actual tumor cases correctly detected.
- **F1 Score**: Harmonic mean of precision and recall, useful for imbalanced data.
- **Confusion Matrix**: Visualized with a Seaborn heatmap to show true/false positives and negatives clearly.

---

8. Results and Analysis

- The CNN model showed good performance on the test set for classifying MRI images.

- The confusion matrix heatmap helped visualize the distribution of true positives, false positives, true negatives, and false negatives.

- Precision and recall reflected the model's ability to detect actual tumor cases while minimizing false alarms.

---

9. Conclusion

The CNN model effectively detected brain tumors from MRI scans with satisfactory performance metrics. This project highlights the potential of deep learning in medical image analysis and early diagnosis. Future improvements could involve using more complex architectures, data augmentation, and addressing any class imbalance for enhanced accuracy.

---

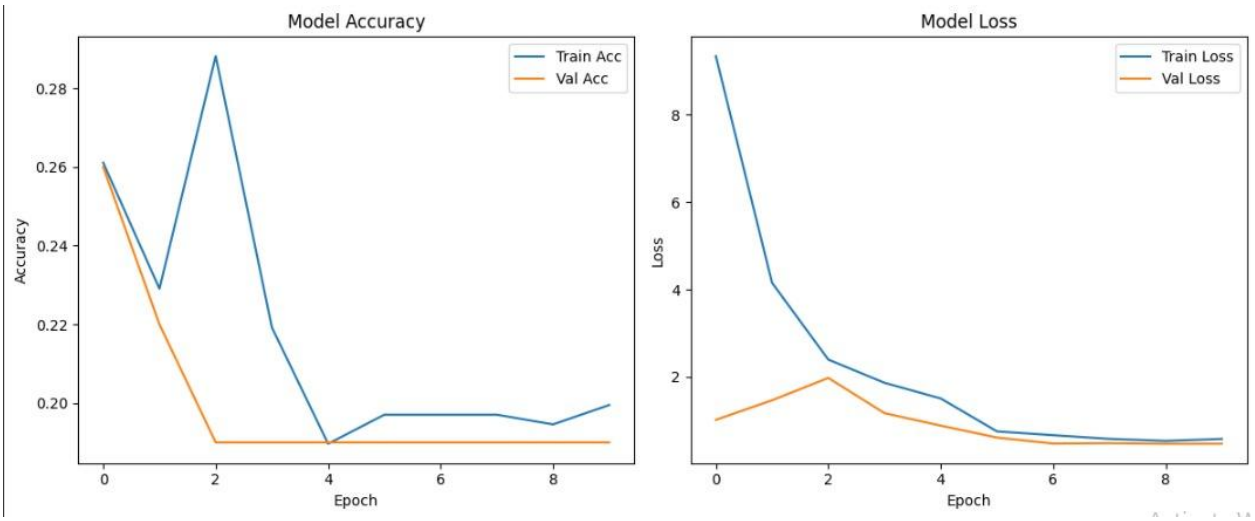
10. References

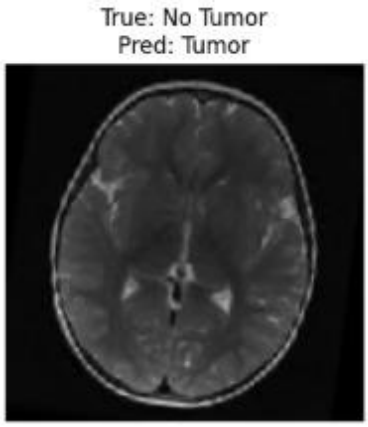
- TensorFlow and Keras documentation
- OpenCV and NumPy documentation
- Seaborn visualization library
- Research articles on brain tumor detection and medical image classification using deep learning

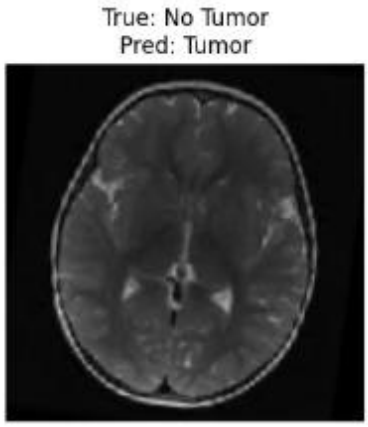
---

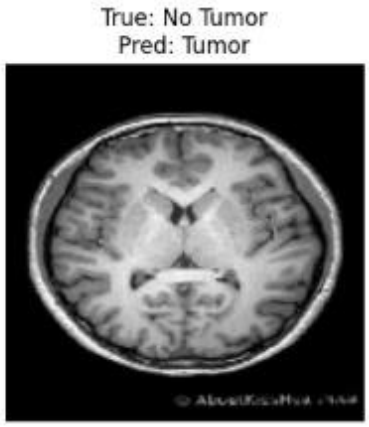
```python
# Import necessary libraries
import os
import numpy as np      #for calculations
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from sklearn.metrics import classification_report, confusion_matrix
import random

# Suppress warnings for clean output
import warnings
warnings.filterwarnings("ignore")

# Set dataset directory path
base_dir = '/content/drive/MyDrive/Brain Tumour Detection'  # Update this path if needed

# Initialize ImageDataGenerator with augmentation for training and validation split
datagen = ImageDataGenerator(
    rescale=1./255,          # Normalize image pixels between 0 and 1
    validation_split=0.2,    # Split data into 80% train and 20% validation
    rotation_range=10,       # Randomly rotate images for augmentation
    zoom_range=0.1,          # Random zoom
    horizontal_flip=True     # Random horizontal flip
)

# Create training data generator
train_generator = datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),      # Resize all images to 150x150
    batch_size=32,               # Process 32 images at a time
    class_mode='binary',         # Binary classification (0 or 1)
    subset='training'            # Use 80% of data
)

# Create validation data generator
val_generator = datagen.flow_from_directory(
    base_dir,
    target_size=(150, 150),
```

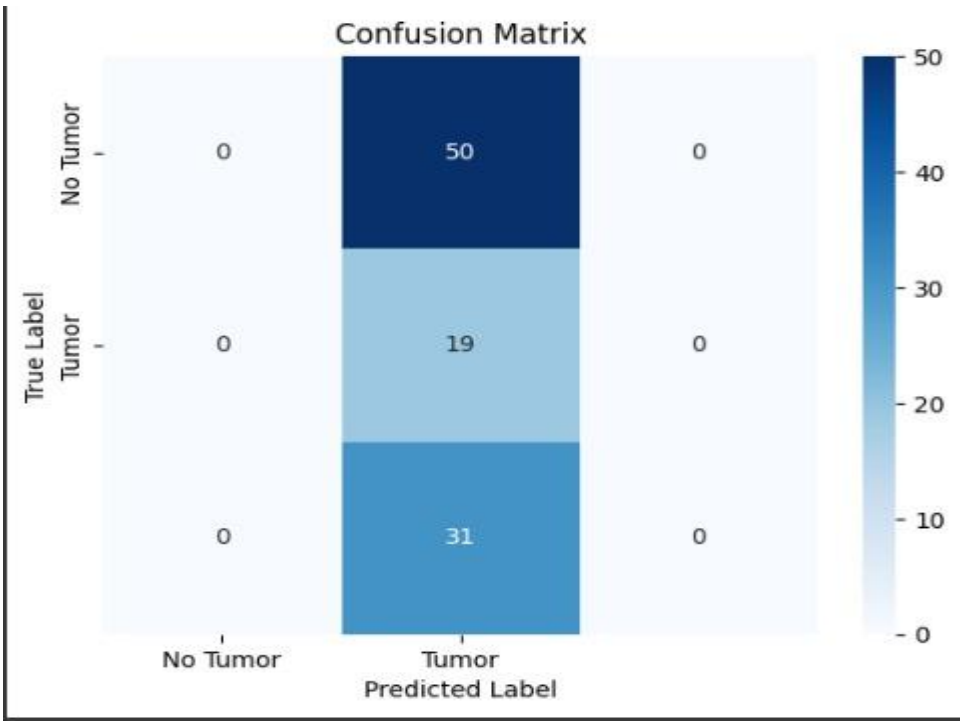
```python
# Train the model on training data and validate on validation data
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    validation_data=val_generator,
    validation_steps=len(val_generator),
    epochs=10  # You can increase for better accuracy
)

# Plot training and validation accuracy/loss
plt.figure(figsize=(12,5))

# Accuracy plot
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate the model: Predictions and Confusion Matrix
val_generator.reset()
predictions = model.predict(val_generator, verbose=1)
y_pred = np.round(predictions).astype(int)   # Convert probabilities to binary labels
y_true = val_generator.classes               # Actual labels

# Confusion matrix visualization
cm = confusion_matrix(y_true, y_pred)
```

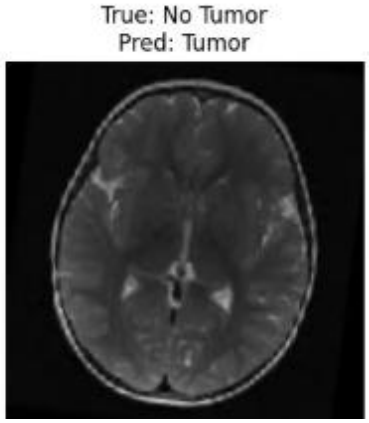

Model Accuracy
Train Acc
Val Acc
Accuracy
Epoch
Model Loss
Train Loss
Val Loss
Loss
Epoch


True: No Tumor
Pred: Tumor


True: No Tumor
Pred: Tumor


True: No Tumor
Pred: Tumor


Confusion Matrix
No Tumor
Tumor
True Label
0
50
0
0
19
0
0
31
0
No Tumor
Tumor
Predicted Label
50
40
30
20
10
0


True: No Tumor
Pred: Tumor