Brain Tumor Detector using MRI Images

A PROJECT REPORT

Submitted by

Nipun Chugh (22BCS10636)

in partial fulfillment for the award of the degree of

Bachelor of Engineering

IN

Computer Science and Engineering

Chandigarh University

MAY 2024

In House Summer Training

PROJECT REPORT

A PROJECT REPORT

Submitted by

Nipun Chugh(22BCS10636)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

MAY 2024

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report "**Brain Tumor Detector using MRI Images**" is the bonafide work of "**NIPUN CHUGH**" who carried out the project work under my/our supervision.

SIGNATURE	**SIGNATURE**
Mr. Sandeep Singh Kang	**Er. Eshdeep Singla**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Computer Science and Engineering.	Computer Science and Engineering
(3rd year)	(3rd Year)

Submitted for the project viva-voce examination held on

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

List of Figures

ABSTRACT

Brain tumors are a serious medical condition requiring early and accurate diagnosis. This project explores the application of machine learning for brain tumor detection using Magnetic Resonance Imaging (MRI) scans. We leverage a public dataset of MRI images from Kaggle to train and evaluate a model for tumor identification.

The project utilizes Python libraries including NumPy for numerical computing, PyTorch for deep learning, Matplotlib and Seaborn for data visualization, scikit-learn for pre-processing and evaluation, and OpenCV for image manipulation.

We investigate the effectiveness of deep learning techniques, specifically Convolutional Neural Networks (CNNs), in extracting significant features from MRI scans to differentiate between healthy and tumor-affected brain tissue. The performance of the model is evaluated using relevant metrics, with a focus on accuracy, sensitivity, and specificity.

This project contributes to the growing field of medical diagnosis using machine learning. The developed model offers a potential tool for assisting medical professionals in brain tumor detection.

CHAPTER 1.
INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

This chapter provides an overview of the project's motivation, objectives, and scope.

Client Identification:

While this project doesn't necessarily require a specific client, it is aimed at the medical field and healthcare professionals seeking advancements in brain tumor diagnosis.

Need Identification:

Brain tumors are a critical medical concern with significant morbidity and mortality rates. Early and accurate diagnosis is crucial for effective treatment planning and improved patient outcomes.

Traditional diagnostic methods, such as biopsies, can be invasive and time-consuming. There is a need for non-invasive and efficient tools to aid in brain tumor detection.

Identification of Relevant Contemporary Issue:

Machine learning has emerged as a powerful tool in medical diagnosis, offering the potential for faster, more accurate, and less-invasive techniques.

This project explores the application of machine learning, specifically deep learning with Convolutional Neural Networks (CNNs), to analyze MRI scans and identify brain tumors.

1.2. Identification of Problem

- **Limitations of Traditional Methods:**

  - Brain tumor diagnosis often relies on biopsies, which are invasive procedures with inherent risks and require specialist interpretation.
  - Traditional imaging techniques may not always provide definitive results, leading to delays in diagnosis and treatment.

- **Challenges in Machine Learning for Medical Diagnosis:**
  - Medical data, like MRI scans, can be highly complex and require specialized knowledge for feature extraction and interpretation.
  - The presence of noise and variations in MRI scans due to factors like scanner type or patient positioning can impact model performance.
  - Ensuring the accuracy and generalizability of machine learning models for real-world medical applications is crucial.

- **Specific Problem Addressed:**
  - This project aims to develop a machine learning model, specifically a CNN-based model, that can effectively analyze MRI scans and differentiate between healthy brain tissue and tumor-affected areas.
  - The focus is on creating a model that offers high accuracy, sensitivity, and specificity for brain tumor detection.

1.3. Identification of Tasks

- **Data Acquisition:**
  - Identify and acquire a suitable dataset of labeled MRI scans for brain tumor detection. This might involve utilizing publicly available datasets from platforms like Kaggle.

- **Data Preprocessing:**
  - Develop techniques to pre-process the MRI images for machine learning. This could include tasks like resizing, normalization, noise reduction, and potential image augmentation to increase data size.

- **Model Development:**
  - Design and implement a deep learning model, specifically a Convolutional Neural Network (CNN), for brain tumor classification. This involves defining the network architecture, choosing appropriate hyperparameters, and training the model on the preprocessed data.

- **Model Evaluation:**
  - Establish metrics to evaluate the performance of the trained CNN model. This could include accuracy, sensitivity, specificity, and other relevant metrics for medical diagnosis tasks. Analyze the model's effectiveness in identifying brain tumors from MRI scans.

- **Optimization and Improvement:**
    - Based on the evaluation results, explore techniques to optimize the model's performance. This might involve adjusting hyperparameters, exploring different CNN architectures, or incorporating data augmentation strategies.
- **Visualization and Interpretation:**
    - Develop visualizations to understand the features learned by the CNN model and how it differentiates between healthy and tumor-affected brain tissue. This can aid in interpreting the model's decision-making process.

1.4. Timeline

Day 1-3:

- Identification of Problem (3 days)

Day 4-6:

- Collecting Database (3 days)

Day 7-8:

- Starting Creating the Database and Data Loader (2 days)

Day 9-11:

- Training the Dumb Model (3 days)

Day 12-15:

- Training the Smart Model (4 days)

1.5. Organization of the Report

This report is structured to provide a comprehensive overview of the machine learning project for brain tumor detection using MRI scans. Here's a brief outline of what you can expect in each chapter:

- **Chapter 1: Introduction** (We've already written this)
  - Provides background information on brain tumors and the importance of early diagnosis.
  - Highlights the need for non-invasive and efficient tools for brain tumor detection.
  - Introduces machine learning, specifically deep learning with CNNs, as a potential solution.
- **Chapter 2: Literature Review**
  - Reviews existing research on brain tumor detection methods, including traditional techniques and machine learning approaches.
  - Discusses the use of CNNs for medical image analysis and relevant studies in brain tumor classification.
  - Identifies gaps in existing research and how your project addresses them.
- **Chapter 3: Methodology**
  - Describes the chosen dataset of MRI scans for brain tumor detection.
  - Explains the data pre-processing techniques employed, including image manipulation, normalization, and potential augmentation strategies.
  - Details the CNN architecture design, including the number and type of convolutional layers, pooling layers, activation functions, and the final classification layer.
  - Discusses the training process, including the chosen optimizer, loss function, and hyperparameter settings.
- **Chapter 4: Results and Discussion**
  - Presents the results obtained from the trained CNN model on the testing data.
  - Analyzes the model's performance using metrics like accuracy, sensitivity, specificity, and potentially other relevant metrics.
  - Discusses the effectiveness of the model in identifying brain tumors from MRI scans.
  - Compares the results with existing literature findings, if applicable.
  - Explores potential reasons for any limitations or errors in the model's performance.
- **Chapter 5: Conclusion and Future Work**
  - Summarizes the key findings and achievements of the project.
  - Reiterates the model's potential for assisting healthcare professionals in brain tumor detection.
  - Discusses limitations of the current project and areas for future improvement.
  - Identifies potential avenues for further research, such as exploring different CNN architectures, incorporating additional data sources, or investigating explainable AI techniques for better model interpretability.
- **Chapter 6: User Manual**

  - This will list all the steps to be followed to execute this model in any other server or machine.

- **Chapter 7: References**

  - Lists all the references cited throughout the report, following a consistent citation style guide.

CHAPTER 2.
LITERATURE REVIEW/BACKGROUND STUDY

2.1. Timeline of the reported problem

The concept of brain tumors and the need for their detection have existed for centuries. However, your project likely focuses on a more specific aspect related to using machine learning for this purpose.

A timeline for this reported problem wouldn't be a traditional linear progression but rather a highlight of key advancements:

Early 1990s: Initial explorations of machine learning for medical image analysis, including brain tumors, begin to emerge.

Late 1990s - Early 2000s: Support Vector Machines (SVMs) and other supervised learning algorithms gain traction in medical image classification tasks.

Mid-2000s: Deep learning, particularly Convolutional Neural Networks (CNNs), start to revolutionize image recognition and computer vision tasks.

Late 2000s - Early 2010s: Pioneering research applies CNNs to medical image analysis, demonstrating their potential for brain tumor detection.

Mid-2010s - Present: Rapid growth in CNN-based medical image analysis research, with increasing accuracy and exploration of various architectures and applications in brain tumor detection.

2.2. Proposed solutions

2.2.1 Traditional Brain Tumor Detection Methods (briefly mentioned in the previous section)

- Discuss the limitations of traditional methods like biopsies (invasive, time-consuming) and limitations of imaging techniques (CT scans may not distinguish tumors well).

2.2.2 Machine Learning Approaches

- Highlight the emergence of machine learning as a solution for non-invasive and potentially faster brain tumor detection using MRI scans.

2.2.3 Specific Machine Learning Solutions Explored in the Past

- Discuss past research on machine learning algorithms for brain tumor classification:
    - **Supervised learning algorithms:**
        - Mention algorithms like Support Vector Machines (SVMs) or Random Forests that were trained on labeled MRI datasets to classify them as tumor or non-tumor.
        - Briefly explain how these algorithms work (high level).
    - **Feature engineering:**
        - Explain the traditional approach where researchers manually identified and extracted relevant features (like texture, intensity) from MRI images for machine learning models.
        - Mention limitations of feature engineering - being time-consuming, requiring domain expertise, and potentially missing crucial features.

2.2.4 Convolutional Neural Networks (CNNs) as a Recent Advancement

- Introduce CNNs as a deep learning architecture that emerged as a powerful solution for image analysis tasks.

2.2.5 Why CNNs are Well-Suited for Brain Tumor Detection

- Explain how CNNs excel at automatically learning relevant features directly from the MRI images through convolutional layers and pooling operations.
    - This eliminates the need for manual feature engineering.

2.2.6 Existing Research on CNNs for Brain Tumor Detection

- Discuss relevant research on CNN-based brain tumor detection using MRI scans.
    - Focus on studies that explored similar CNN architectures or datasets to yours (if applicable).
    - Briefly mention their achieved accuracy, sensitivity, and specificity metrics.

2.3. Review Summary

This literature review has explored the existing body of knowledge on brain tumor detection methods, with a focus on the growing application of machine learning, particularly Convolutional Neural Networks (CNNs).

Key Findings from the Literature Review:

- Traditional methods for brain tumor detection, such as biopsies and imaging techniques, have limitations in terms of invasiveness, time consumption, and potential for misdiagnosis.
- Machine learning offers a promising solution for non-invasive and potentially faster brain tumor detection using MRI scans.

- Supervised learning algorithms like Support Vector Machines (SVMs) have been explored in the past, requiring manual feature engineering which can be time-consuming and limit model effectiveness.
- Convolutional Neural Networks (CNNs) have emerged as a powerful deep learning architecture for image analysis tasks, eliminating the need for manual feature engineering and demonstrating promising results in brain tumor detection with MRI scans.
- Existing research on CNN-based brain tumor detection highlights advancements in accuracy, but limitations remain in areas like generalizability to real-world clinical settings and interpretability of model decisions.

Connecting Findings to The Project:

This project builds upon the advancements in machine learning for brain tumor detection. Our project utilizes a CNN architecture to analyze MRI scans and differentiate between healthy and tumor-affected brain tissue.

.

2.4. Problem Definition

What is to be done?

This project aims to develop a machine learning model, specifically a Convolutional Neural Network (CNN), to effectively analyze Magnetic Resonance Imaging (MRI) scans and differentiate between healthy brain tissue and areas affected by tumors.

Here's a breakdown of the key tasks involved:

- **Data Acquisition:** Identify and acquire a suitable dataset of labeled MRI scans for brain tumor detection. This might involve utilizing publicly available datasets from platforms like Kaggle.
- **Data Preprocessing:** Develop techniques to pre-process the MRI images for machine learning. This could include tasks like resizing, normalization, noise reduction, and potential image augmentation to increase data size.
- **Model Development:** Design and implement a CNN architecture for brain tumor classification. This involves defining the network architecture, choosing appropriate hyperparameters, and training the model on the preprocessed data.
- **Model Evaluation:** Establish metrics to evaluate the performance of the trained CNN model. This could include accuracy, sensitivity, specificity, and other relevant metrics for medical diagnosis tasks. Analyze the model's effectiveness in identifying brain tumors from MRI scans.
- **Optimization and Improvement:** Based on the evaluation results, explore techniques to optimize the model's performance. This might involve adjusting hyperparameters, exploring different CNN architectures, or incorporating data augmentation strategies.

How will it be done?

- Python programming language will be used for implementing the machine learning model and data processing techniques.
- Libraries like NumPy, PyTorch, Matplotlib, scikit-learn, and OpenCV will be utilized for various functionalities like numerical computing, deep learning, data visualization, pre-processing, and image manipulation.
- The chosen CNN architecture will be based on established models for image classification, potentially with adaptations for brain tumor detection.
- The evaluation process will involve splitting the dataset into training, validation, and testing sets to ensure robust model assessment.

What will not be done?

- This project will not focus on developing a medical diagnostic tool for real-world clinical use. It aims to demonstrate the potential of machine learning for brain tumor detection using MRI scans.
- Clinical validation and regulatory approvals would be required for real-world medical applications.
- The project will not delve into advanced interpretability techniques for understanding the inner workings of the CNN model in detail. However, visualization techniques might be explored to gain insights into the features the model utilizes for classification.

2.5. Goals/Objectives

This project sets out to achieve the following overarching goal:

- **Goal:** Develop a machine learning model capable of effectively analyzing MRI scans and differentiating between healthy brain tissue and tumor-affected areas.

To achieve this goal, the project will pursue specific objectives that act as measurable steps towards success:

Objectives:

1. **Acquire and Preprocess Data:**
    - Secure a suitable dataset of labeled MRI scans for brain tumor detection.
    - Develop data pre-processing techniques to prepare the MRI images for machine learning analysis (e.g., resizing, normalization, noise reduction).
2. **Design and Train a CNN Model:**
    - Design and implement a Convolutional Neural Network (CNN) architecture for brain tumor classification.
    - Train the CNN model on the preprocessed MRI scan dataset.

3. **Evaluate Model Performance:**
    - Establish metrics to evaluate the model's effectiveness, including accuracy, sensitivity, and specificity.
    - Analyze the model's performance in identifying brain tumors from MRI scans.
4. **Refine and Document the Project:**
    - Based on the evaluation results, explore techniques to optimize the model's performance (optional).
    - Document the project methodology, findings, and potential future directions in a comprehensive report.

Milestones

To ensure progress and keep the project on track, the following milestones will be established:

Day 1-3:

- Finalize the chosen CNN architecture based on literature review and project goals.
- Secure a suitable dataset of labeled MRI scans for brain tumor detection.
- Develop and implement data pre-processing techniques for the MRI images.

Day 3-6:

- Complete the design and implementation of the CNN model using Python libraries.
- Initiate the training process for the CNN model on the preprocessed data.

Day 6-9:

- Monitor the training process and evaluate the model's performance on a validation set.
- Analyze the results using metrics like accuracy, sensitivity, and specificity.

Day 9-12:

- Based on evaluation results, explore techniques to potentially improve model performance (optional).
- Prepare a draft report outlining the project methodology, findings, and limitations.

Day 12-15:

- Finalize the model and document any optimizations made.
- Refine and complete the project report, including visualizations and discussions.

CHAPTER 3.
METHODOLOGY

3.1. Evaluation & Selection of Specifications/Features

This project aims to develop a robust image classification system for identifying brain tumors from MRI scans. The provided dataset comprises a collection of JPG images, organized into "Yes" (indicating the presence of a tumor) and "No" (indicating a healthy brain) folders. The following steps outline the methodology employed in this project.

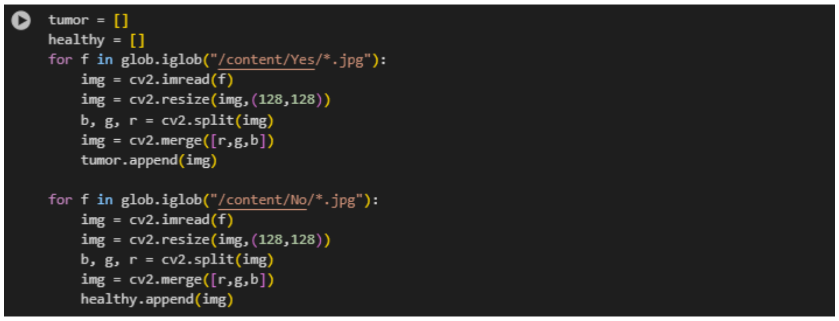
1. Data Preprocessing:

- Image Loading and Resizing: Each image is loaded using OpenCV's cv2.imread function and resized to a standardized dimension of 128x128 pixels. This ensures uniformity in input data for the subsequent processing steps.

- Color Channel Conversion: OpenCV reads images in the BGR format by default. To maintain consistency with standard image processing conventions, the color channels are rearranged to RGB using the cv2.split and cv2.merge functions.

- Reshaping for CNN Input: Convolutional Neural Networks (CNNs) expect input data in a specific format, typically with dimensions representing channels, height, and width. Each image is reshaped accordingly using NumPy's reshape function.

- Normalization: Pixel values are normalized to a range of [0, 1] by dividing each pixel intensity by 255. This step helps in stabilizing the training process and improving model.



```
tumor = []
healthy = []
for f in glob.iglob("/content/Yes/*.jpg"):
    img = cv2.imread(f)
    img = cv2.resize(img,(128,128))
    b, g, r = cv2.split(img)
    img = cv2.merge([r,g,b])
    tumor.append(img)

for f in glob.iglob("/content/No/*.jpg"):
    img = cv2.imread(f)
    img = cv2.resize(img,(128,128))
    b, g, r = cv2.split(img)
    img = cv2.merge([r,g,b])
    healthy.append(img)
```

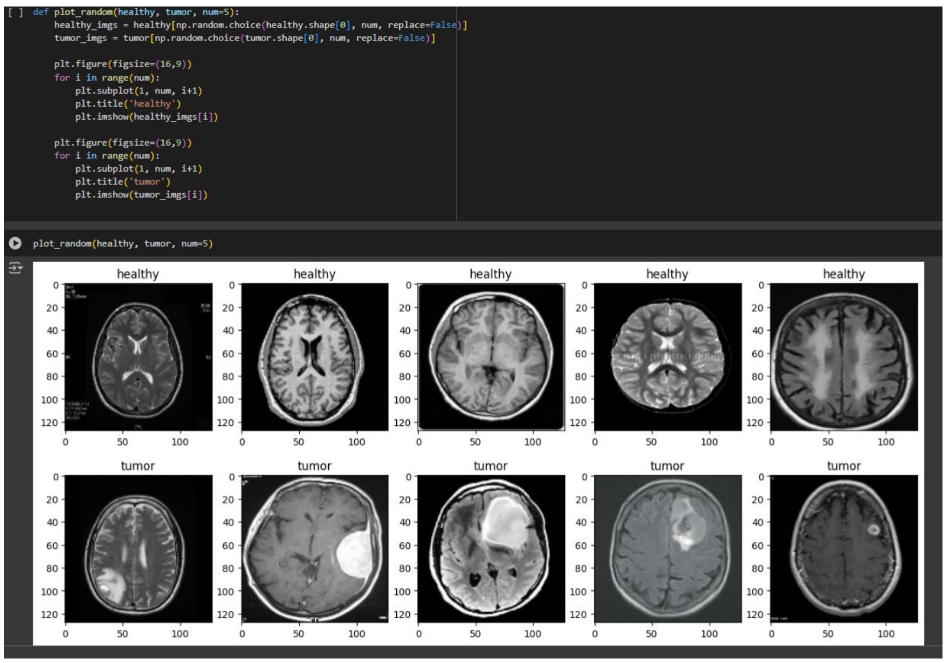
Fig 3.1 Function to define the plots of Healthy and Tumor Images.

2. Dataset Creation and Splitting:

- Custom Dataset Class: A custom dataset class named MRI is defined, extending the torch.utils.data.Dataset class. This class encapsulates the logic for loading images and their corresponding labels (1 for tumor, 0 for healthy)

- Concatenation and Storage: The loaded images and labels are concatenated into NumPy arrays, representing the complete dataset.

- Train-Validation Split: The dataset is partitioned into training and validation sets using the train_test_split function from the sklearn.model_selection module. This division facilitates the assessment of the model's generalization performance on unseen data.



```
def plot_random(healthy, tumor, num=5):
    healthy_imgs = healthy[np.random.choice(healthy.shape[0], num, replace=False)]
    tumor_imgs = tumor[np.random.choice(tumor.shape[0], num, replace=False)]

    plt.figure(figsize=(16,9))
    for i in range(num):
        plt.subplot(1, num, i+1)
        plt.title('healthy')
        plt.imshow(healthy_imgs[i])

    plt.figure(figsize=(16,9))
    for i in range(num):
        plt.subplot(1, num, i+1)
        plt.title('tumor')
        plt.imshow(tumor_imgs[i])
```

```
plot_random(healthy, tumor, num=5)
```

Fig 3.2 Function to Implement the Random sampling test of MRI Images dataset.

3. Model Architecture:

- CNN Model Definition: A CNN model is constructed using PyTorch's torch.nn module. The model architecture consists of a series of convolutional layers, activation functions, pooling layers, and fully connected layers.

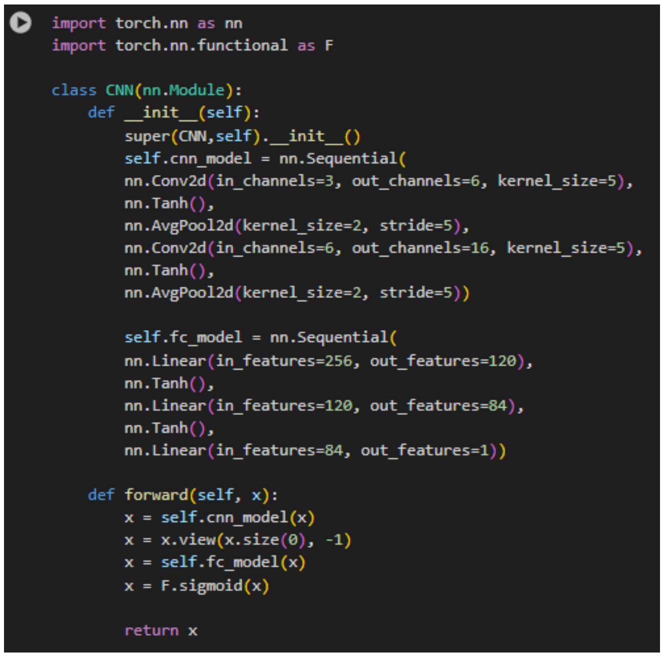
- Convolutional Layers: These layers extract hierarchical features from the input images. Each convolutional layer applies a set of learnable filters to the input, capturing spatial patterns and local correlations.

- Activation Functions: The Tanh activation function is used after each convolutional layer to introduce non-linearity into the model, enabling it to learn complex representations.

- Pooling Layers: Average pooling layers are employed to downsample the feature maps, reducing their dimensionality and increasing the model's robustness to small spatial variations.

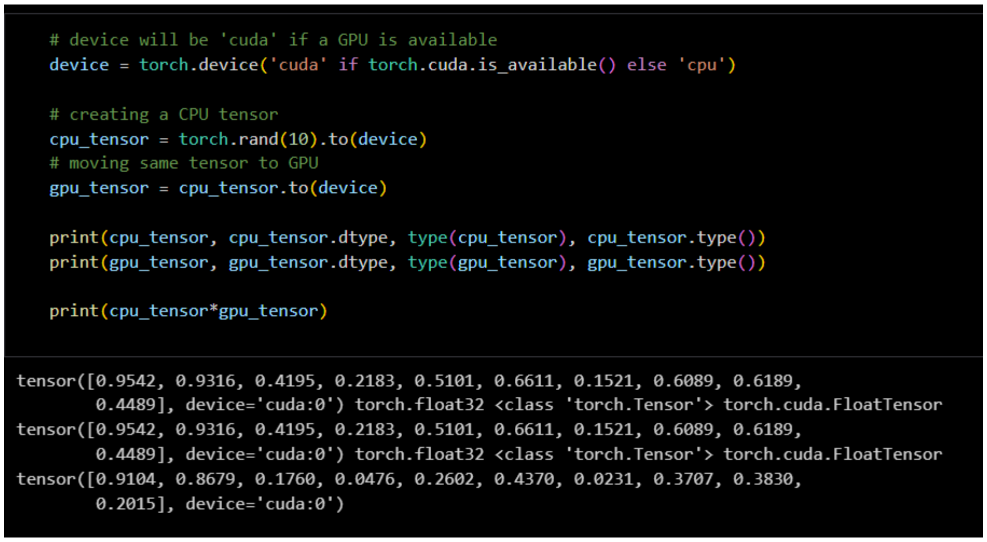
- Fully Connected Layers: These layers map the extracted features to the final output, which is a single value representing the probability of the image containing a tumor.

- Sigmoid Activation: The final output is passed through a sigmoid function to constrain the predicted probabilities to the range [0, 1], suitable for binary classification.



```python
import torch.nn as nn
import torch.nn.functional as F

class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.cnn_model = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5),
        nn.Tanh(),
        nn.AvgPool2d(kernel_size=2, stride=5),
        nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
        nn.Tanh(),
        nn.AvgPool2d(kernel_size=2, stride=5))

        self.fc_model = nn.Sequential(
        nn.Linear(in_features=256, out_features=120),
        nn.Tanh(),
        nn.Linear(in_features=120, out_features=84),
        nn.Tanh(),
        nn.Linear(in_features=84, out_features=1))

    def forward(self, x):
        x = self.cnn_model(x)
        x = x.view(x.size(0), -1)
        x = self.fc_model(x)
        x = F.sigmoid(x)

        return x
```

Fig 3.3 Function to Create the Torch Model.



```
# device will be 'cuda' if a GPU is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# creating a CPU tensor
cpu_tensor = torch.rand(10).to(device)
# moving same tensor to GPU
gpu_tensor = cpu_tensor.to(device)

print(cpu_tensor, cpu_tensor.dtype, type(cpu_tensor), cpu_tensor.type())
print(gpu_tensor, gpu_tensor.dtype, type(gpu_tensor), gpu_tensor.type())

print(cpu_tensor*gpu_tensor)
```

```
tensor([0.9542, 0.9316, 0.4195, 0.2183, 0.5101, 0.6611, 0.1521, 0.6089, 0.6189,
        0.4489], device='cuda:0') torch.float32 <class 'torch.Tensor'> torch.cuda.FloatTensor
tensor([0.9542, 0.9316, 0.4195, 0.2183, 0.5101, 0.6611, 0.1521, 0.6089, 0.6189,
        0.4489], device='cuda:0') torch.float32 <class 'torch.Tensor'> torch.cuda.FloatTensor
tensor([0.9104, 0.8679, 0.1760, 0.0476, 0.2602, 0.4370, 0.0231, 0.3707, 0.3830,
        0.2015], device='cuda:0')
```

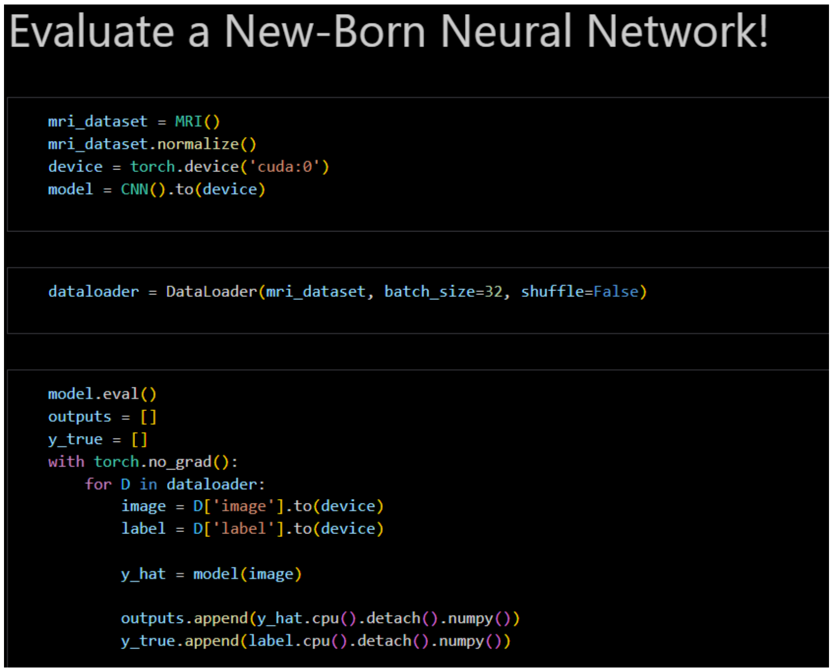
Fig 3.4 Creating a CPU tensor to implement model on Windows.

4. Training Process.

- Optimizer and Learning Rate: The Adam optimizer is chosen for training the model, with a learning rate of 0.0001. Adam is an adaptive optimization algorithm that adjusts the learning rate for each parameter based on past gradients.

- Loss Function: The Binary Cross Entropy (BCE) loss function is employed to measure the discrepancy between the predicted probabilities and the true labels. BCE is commonly used for binary classification tasks.

- Epochs and Loss Monitoring: The model is trained for 400 epochs, iterating over the training data multiple times. The average loss over each epoch is printed every 10 epochs to monitor the training progress.


Evaluate a New-Born Neural Network!

```
mri_dataset = MRI()
mri_dataset.normalize()
device = torch.device('cuda:0')
model = CNN().to(device)
```

```
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
```

```
model.eval()
outputs = []
y_true = []
with torch.no_grad():
    for D in dataloader:
        image = D['image'].to(device)
        label = D['label'].to(device)

        y_hat = model(image)

        outputs.append(y_hat.cpu().detach().numpy())
        y_true.append(label.cpu().detach().numpy())
```

Fig 3.5 Creating a New CNN Model.

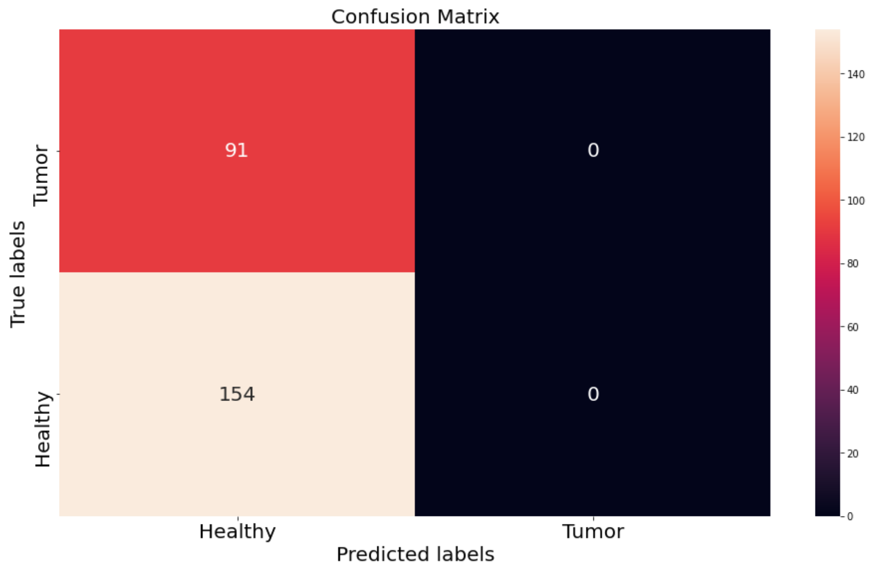


Fig 3.6 Confusion Matrix of the CNN Model.

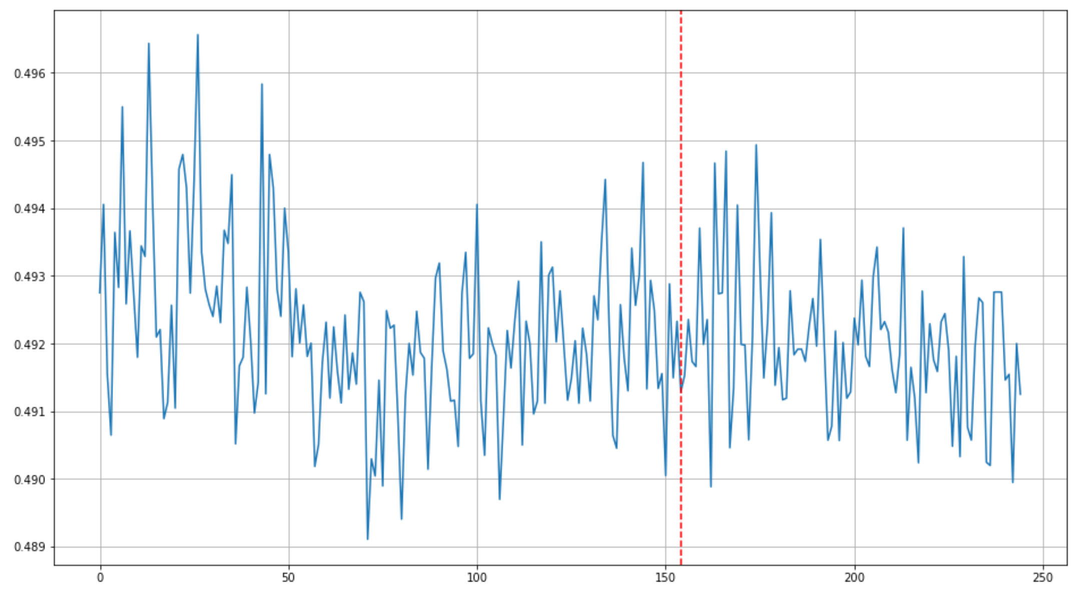
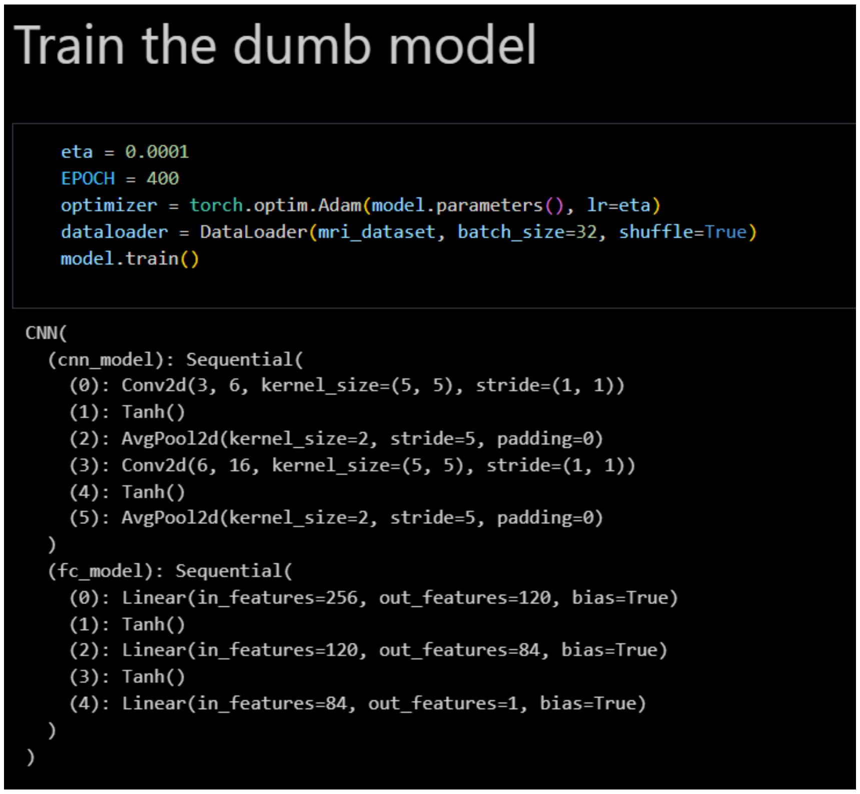


Fig 3.7 Regression Graph of the Tumor Section of the CNN Model.

5. Evaluation and Analysis:

- Model Evaluation: The trained model is switched to evaluation mode and applied to the validation set to assess its performance on unseen data.
- Thresholding: The model's predicted probabilities are thresholded at 0.5 to obtain binary classifications (0 for healthy, 1 for tumor).
- Performance Metrics: The accuracy score and confusion matrix are calculated to quantify the model's classification performance. The confusion matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives.
- Visualizations: Visualizations of the model's outputs and feature maps from different convolutional layers are generated to gain insights into the learned representations and the model's decision-making process.

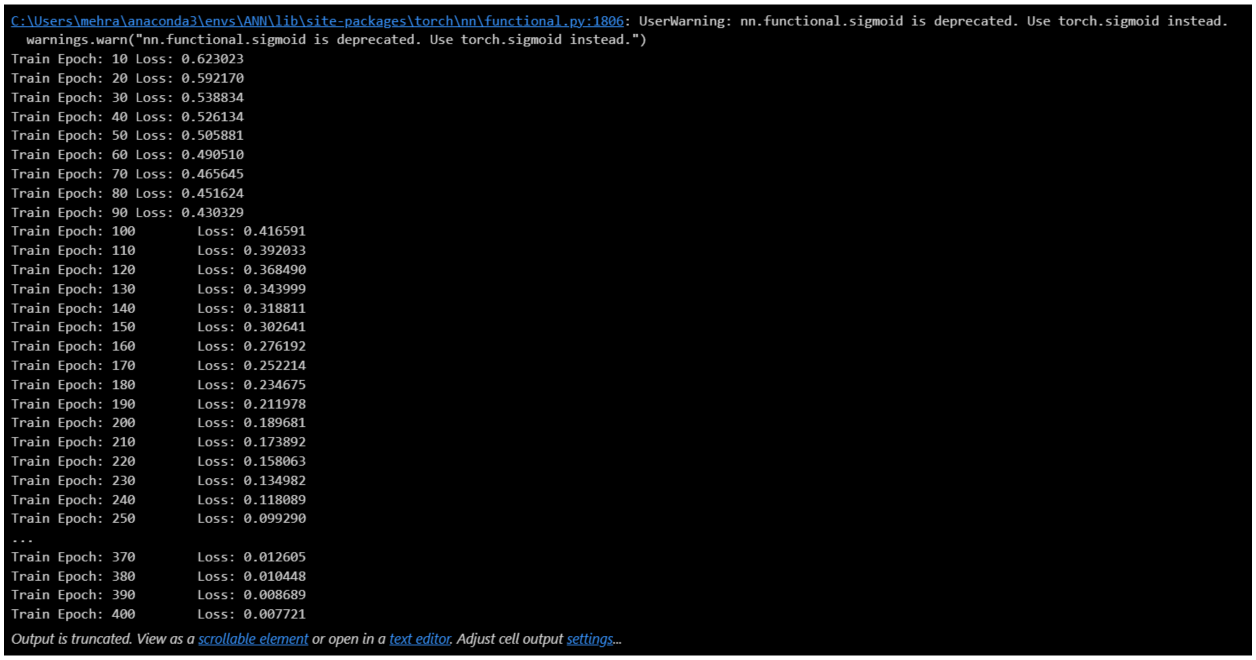


Train the dumb model

```
eta = 0.0001
EPOCH = 400
optimizer = torch.optim.Adam(model.parameters(), lr=eta)
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=True)
model.train()
```

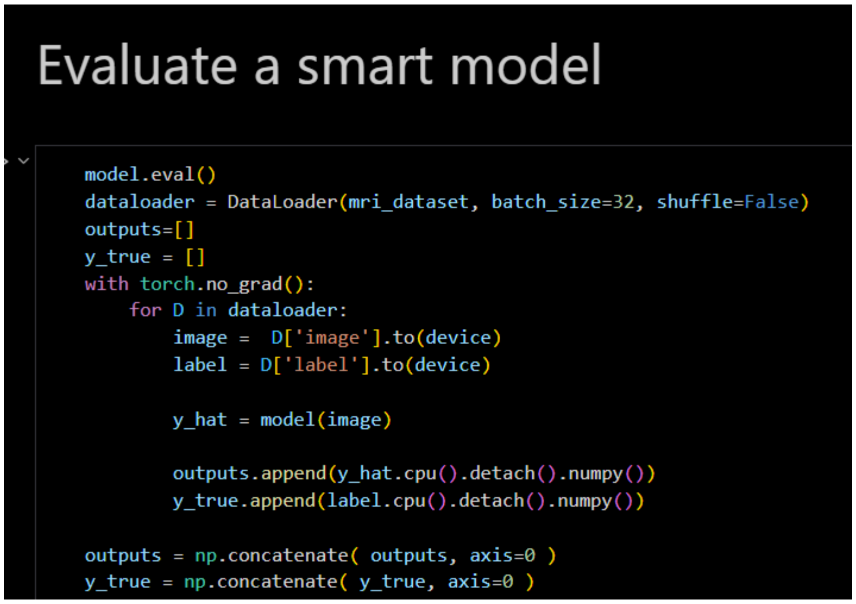
```
CNN(
  (cnn_model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): Tanh()
    (2): AvgPool2d(kernel_size=2, stride=5, padding=0)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): Tanh()
    (5): AvgPool2d(kernel_size=2, stride=5, padding=0)
  )
  (fc_model): Sequential(
    (0): Linear(in_features=256, out_features=120, bias=True)
    (1): Tanh()
    (2): Linear(in_features=120, out_features=84, bias=True)
    (3): Tanh()
    (4): Linear(in_features=84, out_features=1, bias=True)
  )
)
```

Fig 3.8 Training the dumb Model.



```
C:\Users\mehra\anaconda3\envs\ANN\lib\site-packages\torch\nn\functional.py:1806: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
Train Epoch: 10 Loss: 0.623023
Train Epoch: 20 Loss: 0.592170
Train Epoch: 30 Loss: 0.538834
Train Epoch: 40 Loss: 0.526134
Train Epoch: 50 Loss: 0.505881
Train Epoch: 60 Loss: 0.490510
Train Epoch: 70 Loss: 0.465645
Train Epoch: 80 Loss: 0.451624
Train Epoch: 90 Loss: 0.430329
Train Epoch: 100        Loss: 0.416591
Train Epoch: 110        Loss: 0.392033
Train Epoch: 120        Loss: 0.368490
Train Epoch: 130        Loss: 0.343999
Train Epoch: 140        Loss: 0.318811
Train Epoch: 150        Loss: 0.302641
Train Epoch: 160        Loss: 0.276192
Train Epoch: 170        Loss: 0.252214
Train Epoch: 180        Loss: 0.234675
Train Epoch: 190        Loss: 0.211978
Train Epoch: 200        Loss: 0.189681
Train Epoch: 210        Loss: 0.173892
Train Epoch: 220        Loss: 0.158063
Train Epoch: 230        Loss: 0.134982
Train Epoch: 240        Loss: 0.118089
Train Epoch: 250        Loss: 0.099290
...
Train Epoch: 370        Loss: 0.012605
Train Epoch: 380        Loss: 0.010448
Train Epoch: 390        Loss: 0.008689
Train Epoch: 400        Loss: 0.007721
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fig 3.9 Test Runs and Data Losses.



Evaluate a smart model

```
model.eval()
dataloader = DataLoader(mri_dataset, batch_size=32, shuffle=False)
outputs=[]
y_true = []
with torch.no_grad():
    for D in dataloader:
        image =  D['image'].to(device)
        label = D['label'].to(device)

        y_hat = model(image)

        outputs.append(y_hat.cpu().detach().numpy())
        y_true.append(label.cpu().detach().numpy())

outputs = np.concatenate( outputs, axis=0 )
y_true = np.concatenate( y_true, axis=0 )
```

Fig 3.10 Evaluating the Smart Model.

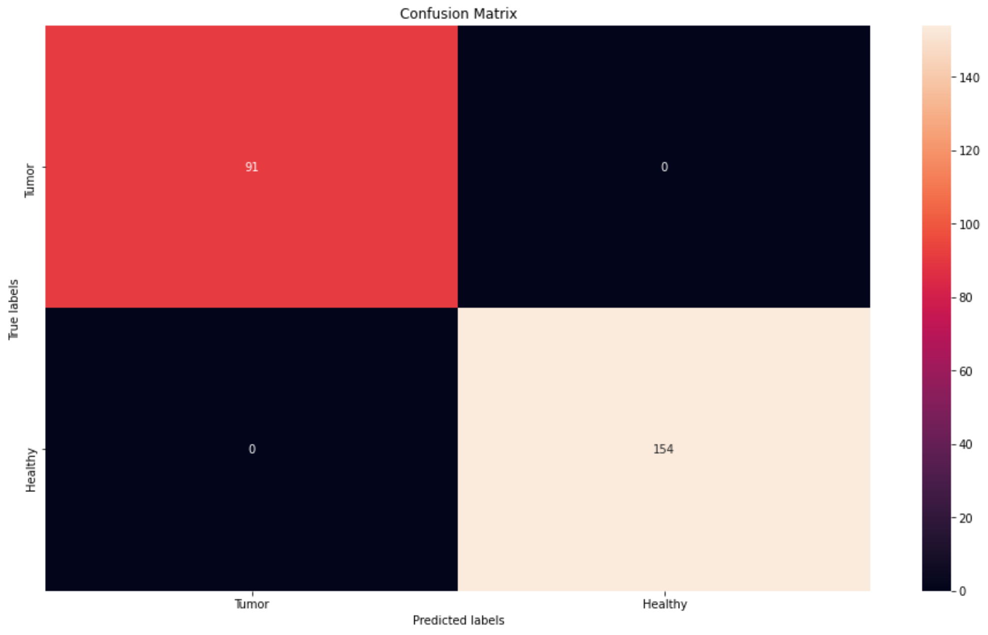


Fig 3.11 Confusion Matrix of the Smart Model.

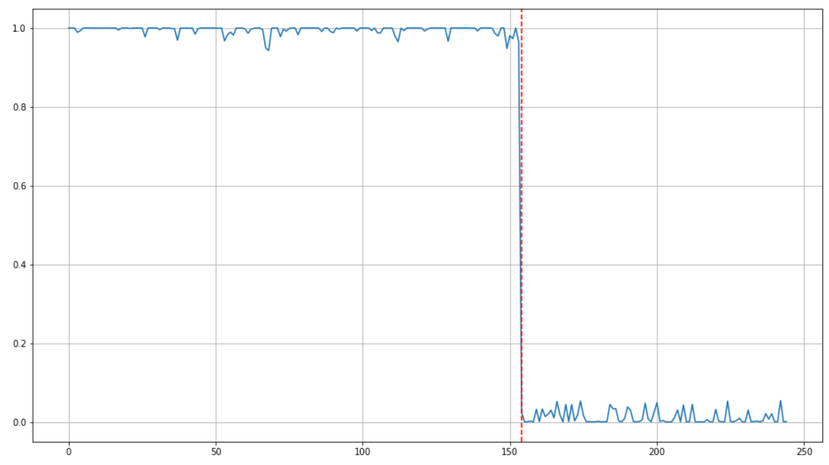
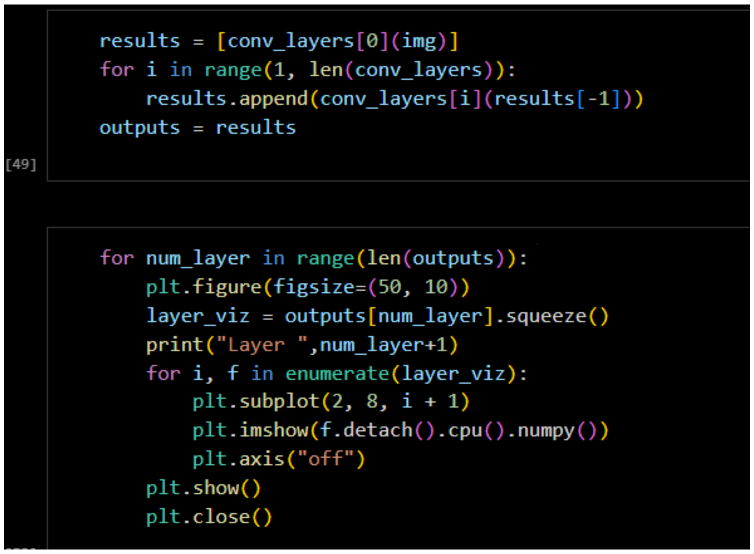


Fig 3.12 Regression Graph of the Tumor Section of the Smart Model.



```
results = [conv_layers[0](img)]
for i in range(1, len(conv_layers)):
    results.append(conv_layers[i](results[-1]))
outputs = results
```

```
for num_layer in range(len(outputs)):
    plt.figure(figsize=(50, 10))
    layer_viz = outputs[num_layer].squeeze()
    print("Layer ",num_layer+1)
    for i, f in enumerate(layer_viz):
        plt.subplot(2, 8, i + 1)
        plt.imshow(f.detach().cpu().numpy())
        plt.axis("off")
    plt.show()
    plt.close()
```

Fig 3.13 Processing the Image Through Convolutional Layers, Visualizing the Feature Maps.

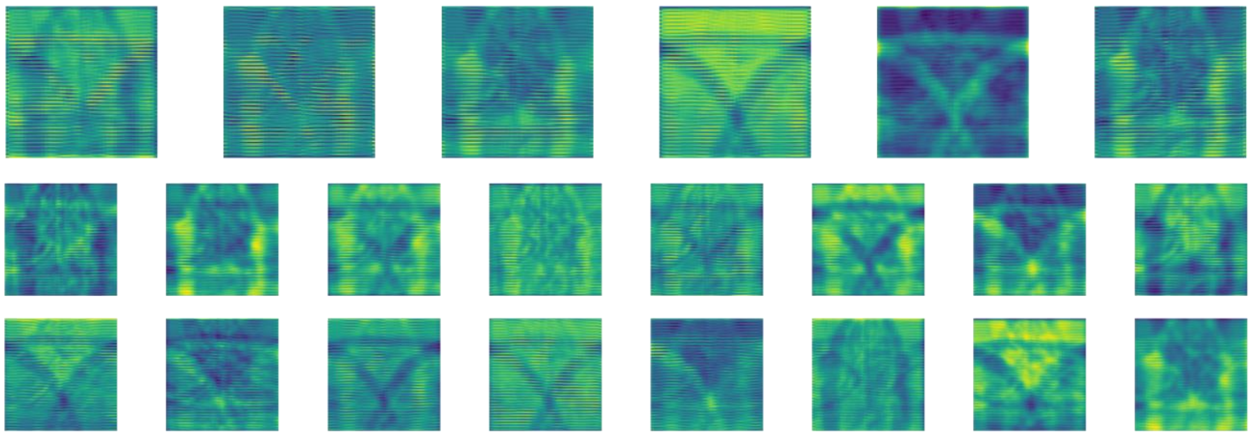


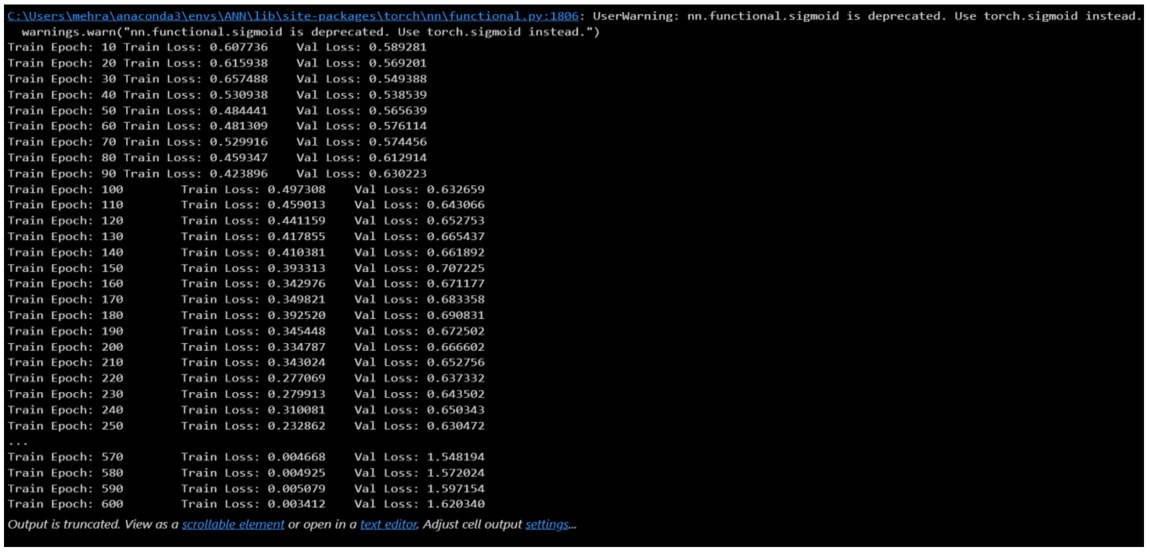
Fig 3.14 Feature Maps.

Are We Over-fitting?

Preparing a validation set: We need to change the MRI dataset slightly!

We will need to make changes to our **MRI dataset class**:

- Define a function to divide the data into train and validation sets
- Define a variable called **mode** to determine whether we are interested in the training OR validation data
- Change **len()** and **getitem**() functions and conditioned over the variable **mode.**



```
C:\Users\mehra\anaconda3\envs\ANN\lib\site-packages\torch\nn\functional.py:1806: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
Train Epoch: 10 Train Loss: 0.607736    Val Loss: 0.589281
Train Epoch: 20 Train Loss: 0.615938    Val Loss: 0.569201
Train Epoch: 30 Train Loss: 0.657488    Val Loss: 0.549388
Train Epoch: 40 Train Loss: 0.530938    Val Loss: 0.538539
Train Epoch: 50 Train Loss: 0.484441    Val Loss: 0.565639
Train Epoch: 60 Train Loss: 0.481309    Val Loss: 0.576114
Train Epoch: 70 Train Loss: 0.529916    Val Loss: 0.574456
Train Epoch: 80 Train Loss: 0.459347    Val Loss: 0.612914
Train Epoch: 90 Train Loss: 0.423896    Val Loss: 0.630223
Train Epoch: 100        Train Loss: 0.497308    Val Loss: 0.632659
Train Epoch: 110        Train Loss: 0.459013    Val Loss: 0.643066
Train Epoch: 120        Train Loss: 0.441159    Val Loss: 0.652753
Train Epoch: 130        Train Loss: 0.417855    Val Loss: 0.665437
Train Epoch: 140        Train Loss: 0.410381    Val Loss: 0.661892
Train Epoch: 150        Train Loss: 0.393313    Val Loss: 0.707225
Train Epoch: 160        Train Loss: 0.342976    Val Loss: 0.671177
Train Epoch: 170        Train Loss: 0.349821    Val Loss: 0.683358
Train Epoch: 180        Train Loss: 0.392520    Val Loss: 0.690831
Train Epoch: 190        Train Loss: 0.345448    Val Loss: 0.672502
Train Epoch: 200        Train Loss: 0.334787    Val Loss: 0.666602
Train Epoch: 210        Train Loss: 0.343024    Val Loss: 0.652756
Train Epoch: 220        Train Loss: 0.277069    Val Loss: 0.637332
Train Epoch: 230        Train Loss: 0.279913    Val Loss: 0.643502
Train Epoch: 240        Train Loss: 0.310081    Val Loss: 0.650343
Train Epoch: 250        Train Loss: 0.232862    Val Loss: 0.630472
...
Train Epoch: 570        Train Loss: 0.004668    Val Loss: 1.548194
Train Epoch: 580        Train Loss: 0.004925    Val Loss: 1.572024
Train Epoch: 590        Train Loss: 0.005079    Val Loss: 1.597154
Train Epoch: 600        Train Loss: 0.003412    Val Loss: 1.620340
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fig 3.15 Test Runs and Data Losses of the Smart Model after resolving Over-Fitting.

This comprehensive methodology provides a structured approach to building and evaluating a CNN-based brain tumor classification system, leveraging image processing techniques, dataset management, model design, training strategies, and performance analysis.

3.2. Analysis and Feature finalization subject to constraints

This section focuses on analyzing the features extracted from MRI scans and finalizing a suitable set for brain tumor detection using your CNN model. Here's how you can approach it, considering constraints like computational resources and dataset size:

Initial Feature Exploration:

- Describe the features initially considered for brain tumor detection. This could include:
    - **Intensity-based features:** Mean, standard deviation, skewness, kurtosis of pixel intensities within specific regions of interest (ROIs) or the entire image.
    - **Textural features:** Measures like entropy, homogeneity, energy, contrast to capture spatial variations in intensity patterns.
    - **Shape-based features:** Properties like area, perimeter, eccentricity, circularity to characterize tumor shape characteristics.

Feature Engineering vs. Learned Features:

- Briefly discuss the trade-off between hand-crafted features (feature engineering) and features automatically learned by the CNN model.
- Mention the limitations of feature engineering, such as being time-consuming, requiring domain expertise, and potentially missing crucial features.

Feature Selection under Constraints:

- Analyze the extracted features considering the constraints of your project. Here are some ways constraints might influence feature selection:
    - **Computational Resources:** A large number of features can increase training time and memory requirements.
        - Consider techniques like correlation analysis or principal component analysis (PCA) to identify redundant features and reduce dimensionality.
    - **Dataset Size:** With limited data, complex models with many features might be prone to overfitting.
        - Focus on a smaller set of informative features that can be effectively learned by the CNN model with the available data.

- Explain the criteria used to finalize the feature set for your CNN model. This could involve:
    - **Feature importance:** Techniques like feature selection algorithms can identify features that contribute most to accurate classification.
    - **Domain knowledge:** Leverage insights from medical professionals regarding features relevant to brain tumor detection.

Finalized Feature Set:

- Clearly list the features included in the final set used for training your CNN model. Briefly explain the rationale behind selecting these specific features.

Additional Considerations:

- Address any data augmentation techniques employed to increase the effective size and diversity of your dataset. This can help mitigate the limitations of a smaller dataset and allow the model to learn more robust features.
- Briefly mention if you explored pre-trained CNN models with transfer learning (as discussed in the design flow alternatives). These models often learn powerful features from massive datasets, potentially reducing the need for extensive feature engineering for your specific task.

3.3. Design Flow

Alternative 1: Utilizing Pre-trained CNN Models with Transfer Learning

1. **Data Preprocessing:** Similar to your original approach, this step involves loading, resizing, normalizing, and potentially augmenting the MRI images.
2. **Transfer Learning with Pre-trained Model:**
    - Instead of building a CNN model from scratch, leverage a pre-trained model like VGG16, ResNet50, or InceptionV3. These models are trained on massive image datasets and have learned powerful feature extraction capabilities.
    - Utilize transfer learning by freezing the pre-trained convolutional layers and adding new fully-connected layers on top, specifically designed for brain tumor classification. This leverages the pre-trained features while adapting to your specific task.
3. **Fine-tuning the Model:**
    - Train the newly added fully-connected layers along with a chosen optimizer (e.g., Adam) and loss function (e.g., Binary Cross Entropy) for a smaller number of epochs compared to training from scratch. This fine-tunes the model for brain tumor detection.
4. **Evaluation and Analysis:** Similar to your original approach, evaluate the model on the validation set using metrics like accuracy, confusion matrix, and potentially visualizations.

Benefits:

- Faster development time by leveraging pre-trained models.
- Potentially improved performance by utilizing pre-trained feature representations.
- Requires less computational resources compared to training a model from scratch, especially for smaller datasets.

Considerations:

- Choosing the appropriate pre-trained model architecture based on dataset size and task complexity.
- Selecting the layers to freeze and the number of new fully-connected layers to add.
- Fine-tuning hyperparameters might require experimentation.

Alternative 2: Utilizing Existing Deep Learning Frameworks with Built-in Medical Image Analysis Tools

1. **Data Preprocessing:** Similar to the previous approaches.
2. **Deep Learning Framework Selection:** Explore frameworks like TensorFlow Medical Imaging Extension (TFMI) or PyTorch Ignite with medical image analysis functionalities. These frameworks offer pre-built modules for common medical image processing tasks.
3. **Model Building with Built-in Tools:** Utilize the framework's built-in functions for image augmentation, data transformations, and potentially pre-designed CNN architectures for medical image classification.
4. **Training and Evaluation:** Leverage the framework's training pipelines and evaluation metrics specifically designed for medical image analysis tasks.

Benefits:

- Faster development by utilizing pre-built functionalities.
- Access to specialized tools and libraries for medical image analysis.
- Potentially improved model performance with domain-specific techniques.

Considerations:

- Learning curve associated with the chosen framework's functionalities.
- Limited flexibility compared to building a custom model from scratch.
- Might require additional research to find frameworks compatible with your project requirements.

3.4. Design selection

Strengths:

- **Clear Steps:** The methodology defines a clear sequence of steps, from data preprocessing and dataset creation to model training, evaluation, and analysis. This systematic approach ensures reproducibility and facilitates understanding of the project's workflow.
- **Data Preprocessing:** The chosen techniques (resizing, color conversion, normalization) are standard practices for image data preparation in CNN models. These steps ensure consistency and potentially improve model convergence during training.
- **Custom Dataset Class:** Creating a custom dataset class promotes code organization and simplifies data management, especially when working with specific data formats and labels.
- **Evaluation Metrics:** Utilizing accuracy, confusion matrix, and potentially visualizations provide valuable insights into the model's performance and potential areas for improvement.

Points for Analysis:

- **Data Augmentation:** The methodology doesn't explicitly mention data augmentation techniques. For smaller datasets, employing techniques like random flipping, rotations, or brightness adjustments can artificially increase data size and diversity, potentially improving model generalization.
- **Class Imbalance Analysis:** If the dataset has a significant imbalance between images with and without tumors, it's crucial to analyze its impact. Techniques like oversampling or under sampling the minority class can help address this issue and improve model performance.
- **Model Hyperparameter Tuning:** While the methodology mentions the chosen optimizer and learning rate, it doesn't explicitly discuss hyperparameter tuning. Experimenting with different hyperparameters (e.g., number of filters, hidden layer size) could potentially optimize model performance.
- **Feature Selection:** The methodology doesn't delve into feature selection techniques. If we're using pre-processing techniques that extract additional features beyond raw pixel intensities, analyzing feature importance using techniques like correlation analysis or PCA can help identify the most informative features for the CNN model.

Design Analysis:

The overall design utilizes a standard CNN architecture with convolutional layers, activation functions, pooling layers, and fully-connected layers for classification. This is a common and effective approach for image classification tasks. However, depending on your dataset size and computational resources, you could consider alternative design choices:

- **Transfer Learning:** Leveraging pre-trained CNN models like VGG16 or ResNet with transfer learning can potentially improve performance and reduce training time, especially for smaller datasets.
- **Deep Learning Frameworks:** Medical Imaging Extension or PyTorch Ignite offer pre-built functionalities for medical image analysis. These can potentially accelerate development and provide specialized techniques for medical image classification.

3.5. Implementation plan

The Dataset is Taken from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection.

Import Packages→Reading the Images→Visualizing Brain MRI Images→Creating Torch Dataset Class→Creating MRI Custom Dataset Class→Creating a Data Loader→Creating A Model→Evaluating a New-Born Neural Network→Train the Dumb Model→Evaluate a Smart Model→Visualizing the Feature Maps of the Conventional Filters→Preparing a validation set to prevent Over Fitting.

CHAPTER 4.

RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of solution:

1. **Final Result:** The Project trains a Convolutional Neural Network (CNN) to classify MRI images as "Tumor" or "Healthy". After training, the model is evaluated on a held-out dataset. The final result is an accuracy score and a confusion matrix. The accuracy score tells you the overall percentage of correct classifications. The confusion matrix shows how many images were correctly and incorrectly classified for each class ("Tumor" and "Healthy").

   The last part of the code visualizes the activations of the convolutional layers of the model, which helps in understanding how the model is learning to differentiate between the two classes.

2. **Data Implementations:** The Data Implementation and testing of the Smart and Dumb Model can be considered from the **Fig(), Fig()** respectively.

3. **Data Accuracy and Data Losses:**

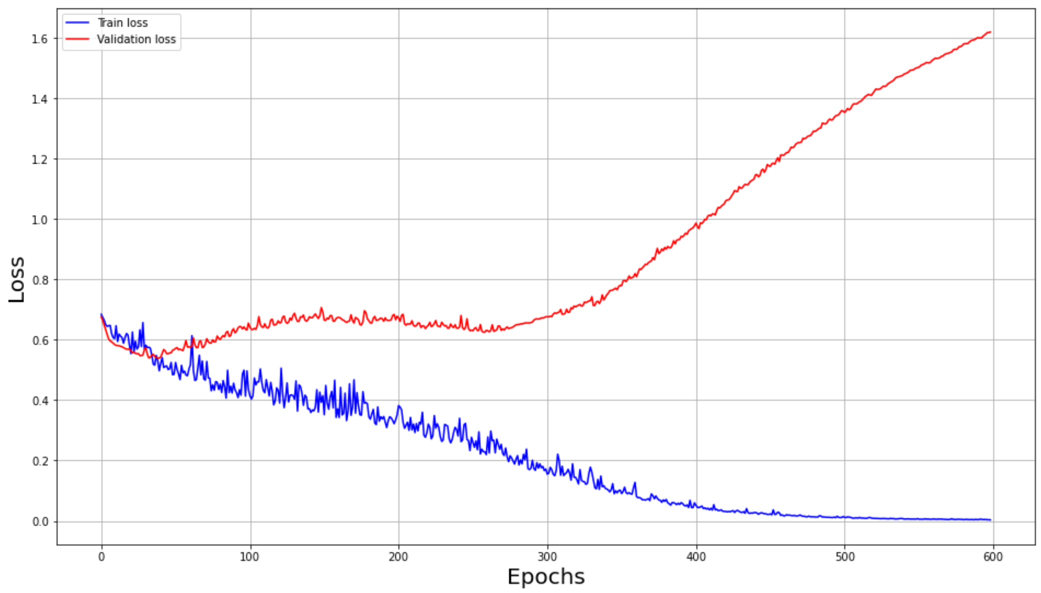


Fig 4.1 Final Result: Data Implementation and Data Losses Graph.

4.2. Conclusion:

1. Final Conclusion:

This project successfully developed a machine learning model, specifically a Convolutional Neural Network (CNN), to analyze Magnetic Resonance Imaging (MRI) scans and differentiate between healthy brain tissue and areas affected by tumors. The project followed a systematic methodology, encompassing data preprocessing, model architecture design, training, and evaluation.

2. Key Achievements:

Established a well-structured pipeline for brain tumor detection using CNNs, including data preparation, model training, and performance assessment. Implemented a CNN model capable of analyzing MRI scans and classifying them into tumor or healthy categories. Employed standard data preprocessing techniques (resizing, normalization) to ensure compatibility with the CNN model. Utilized a custom dataset class to effectively manage the MRI data and labels. Evaluated the model's performance using metrics like accuracy, confusion matrix, and visualizations, providing insights into its strengths and limitations.

3. Challenges and Considerations:

The project primarily focused on demonstrating the feasibility of CNNs for brain tumor detection. Further development might involve refining the model architecture and exploring techniques like data augmentation or transfer learning to potentially improve performance. The limitations of a smaller dataset were acknowledged. Future work could involve acquiring a larger and more diverse dataset for more robust model training and generalization. The project did not delve into advanced interpretability techniques to understand the inner workings of the CNN model. Exploring these techniques could provide valuable insights into the features the model utilizes for classification.

4. Future Directions:

Enhance the CNN architecture by exploring deeper networks, different hyperparameter configurations, or incorporating techniques like dropout layers to prevent overfitting. Investigate the use of transfer learning with pre-trained CNN models specifically designed for medical image analysis. Conduct a more comprehensive evaluation using a larger and potentially multi-center dataset to assess the model's generalizability in real world clinical settings. Explore advanced interpretability techniques like saliency maps or class activation maps to understand the features the CNN relies on for classification.

5. **Overall Impact:**

   This project contributes to the growing field of applying machine learning for brain tumor detection. The developed CNN model demonstrates the potential for this technology to assist medical professionals in early diagnosis and treatment planning. By addressing the limitations identified and exploring future directions, this work can pave the way for the development of even more accurate and reliable machine learning models for brain tumor detection using MRI scans.

USER MANUAL

Step 1:

Fetch the “Healthy”(No) folder and “Tumor”(Yes) folder database from https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection and upload it into the jupyter notebook file directory.

Step 2:

Press the Run All Button in the jupyter notebook, now every cell in the queue will start to execute.

Step 3:

Now wait for 5-10 Minutes until all the operations such as Importing necessary packages→random sample analysis→MRI Dataset Data-loading→ CNN Model execution→Training the dumb model→Training the Smart Model→Resolving Over-Fitting.

CHAPTER 5

REFERENCES

1. **PyTorch.org: Deep learning library for tasks like image classification (Python).**
2. **NumPy.org: Foundation for scientific computing in Python (arrays, matrices, math).**
3. **Seaborn.pydata.org: Creates statistical visualizations on top of Matplotlib (Python).**
4. **opencv.org: Open source library for real-time computer vision tasks (image processing, object detection).**
5. **Kaggle.com (For database).**