THESIS TITLE: TUMOR IDENTIFICATION IN CT AND MRI IMAGING USING DEEP LEARNING FOR ACCURATE DIAGNOSIS

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

ANJEZA KANXHA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR BACHELOR DEGREE
IN COMPUTER ENGINEERING

MAY, 2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Anjeza Kanxha

Signature:

ABSTRACT

TUMOR IDENTIFICATION IN CT AND MRI IMAGING USING DEEP LEARNING FOR ACCURATE DIAGNOSIS

Kanxha, Anjeza
BA. E. Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Dimitros Karras

Deep Learning with Convolutional Neural Networks has become a powerful tool for solving a wide range of real-life problems. Image classification and medical imaging are cases of implementation of deep learning, a subset of Machine Learning. Deep learning has a high success rate, due to its variety of algorithms and the ability to learn complex patterns directly from data. This thesis aims to test the effectiveness of the Convolutional Neural Network, in detecting tumors and classifying different medical images in tumoral or healthy. This model will be trained and tested on our chosen datasets for its accuracy and other performance metrics. The performance was satisfying with high accuracy, during training and testing on unseen data.

Keywords: *Deep Learning, Convolutional Neural Network, Medical Images, Classification, Accuracy.*

ABSTRAKT

IDENTIFIKIMI I TUMOREVE NË IMAZHERINË CT DHE ATYRE ME REZONANCË MAGNETIKE DUKE PËRDORUR DEEP LEARNING PËR DIAGNOZË TË SAKTË

Kanxha, Anjeza

Bachelor, Departamenti i Inxhinierise Kompjuterike
Udhëheqësi: Assoc. Prof. Dr. Dimitros Karras

Deep Learning me Convolutional Neural Network është bërë një mjet i fuqishëm për zgjidhjen e një gamë të gjerë të problemeve të jetës reale. Njohja e imazhit dhe imazheneria mjekesore jane rastet e zbatimit të Deep Learning, nëngrup i Machine Learning. Deep Learning ka një nivel të lartë të suksesit, për shkak të varietetve të algoritmeve dhe të aftësisë së tij për të mësuar modele komplekse direkt nga të dhënat. Kjo temë synon të testojë efektshmërinë e Convolutional Neural Network në njohjen e tumorit dhe klasifikimin e imazheve mjekësore në tumoriale ose të shëndetshme. Ky model do të trajnohet dhe do të testohet në koleksione të dhënash të zgjedhura nga ne, për saktësinë e tij dhe metrika të tjera të performancës. Performanca ishte shumë e mirë dhe e kënaqshme, me saktësi të lartë, gjatë trajnimit dhe testimit të imazheve që nuk janë trajnuar më parë.

Fjalët kyçe: *Deep Learning, Convolutional Neural Network, Imazhe mjekësore, Klasifikimi, Saktësia.*

Dedicated to……..…

ACKNOWLEDGEMENTS

I would like to express my special thanks to my supervisor Assoc. Prof. Dr. Dimitros Karras for his continuous guidance, encouragement, motivation, and support during all the stages of my thesis. I sincerely appreciate the time and effort he has spent to improve my experience during my graduate years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF Tables

LIST OF Tables

LIST OF ABBREVIATIONS

NN – Neural networks

ANN – Artificial neural networks

CNN – Convolutional neural networks

DNN – Deep Neural Network

CT – Computed Tomography

FC – Fully connected

RGB – Red green blue

IQ-OTH/NCCD - Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases

SOMATOM – Brand name for a type of CT scanner made by Siemens

RNN – Recurrent Neural Network

ML – Machine Learning

ReLU - Rectified linear unit

TanH – Hyperbolic Tangent

CHAPTER 1

INTRODUCTION

In recent years, lung cancer remains the second most lethal cancer type worldwide. The American Cancer Society's estimates for lung cancer in the US for 2024 are about 234,580 new cases of lung cancer (116,310 in men and 118,270 in women) [1]. Advancements in technology have improved the field of healthcare, enabling more accurate and efficient diagnosis. The best medical imaging used for detecting lung cancer is CT scans.

Similarly, brain tumors are also a big challenge in detecting and diagnosing. That's due to the wide variety of types and grades. As with lung cancer, advancements in imaging technology have proved that MRI plays a significant role in diagnosing brain tumors.

Despite this big evolution in medical imaging, the accuracy and diagnosis of cancer or tumors is a challenging task. Even for healthcare professionals such as radiologists, is tough to analyze CT scans and MRIs. That's due to the complex patterns that are difficult to interpret for the human eye. The number of cases around the world has increased in recent years. This fact claims the urgent need for automated and efficient solutions. So, we aim to help in the early detection of these lethal diseases and treatment planning.

Deep Learning is a subfield of Machine Learning. It has proven highly successful results in medical image analysis. One of the most applied subsets of Machine Learning is Convolutional Neural Networks (CNNs). They have shown very good performance in detecting features from complex images and classifying them. The predictions and results with high accuracy, are essential proof of successful detection. Using the power of CNNs, it is possible to develop automated systems for the detection of lung cancer from CT scans and tumor brains from MRIs. In this way supporting healthcare professionals in making swifter and better decisions for the patient's diagnosis and treatment.

1.1. AI, Machine Learning and Deep Learning

AI is a boarded field, that has created successful systems necessary to perform tasks that would normally require human intelligence. Machine learning is a branch of artificial intelligence (AI) that focuses on building systems capable of learning from data, identifying patterns, and making decisions. Deep learning is a subset of Machine learning and consequently of Artificial Intelligence too.

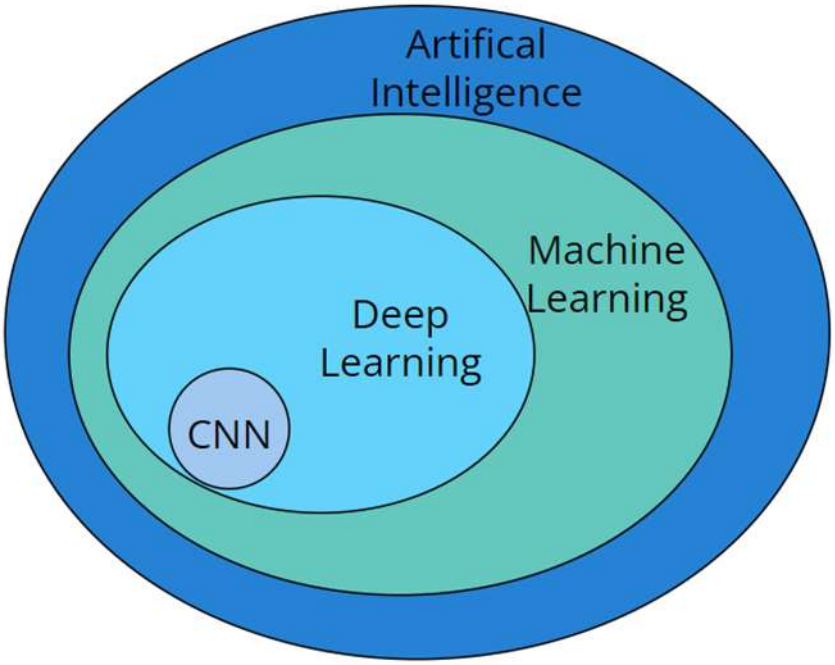

Figure 1 AI, Machine Learning, and Deep Learning

"Deep" word stands for the fact that in deep learning there is more than one hidden layer. Using many hidden layers, we give the model the ability to extract different features like edges, shapes, and patterns. These features are too complex patterns to be recognized by the human eye

1.1.1. Artificial Neural Network

ANN (Artificial Neural Network) is a model inspired by a human's brain. A system, where the key component is the neuron. Similarly, as the brain where neurons are responsible for receiving input, processing it, and returning an output. The data structure

of ANN represents an abstract model of the human brain, including neurons, dendrites, axons, and synapses.

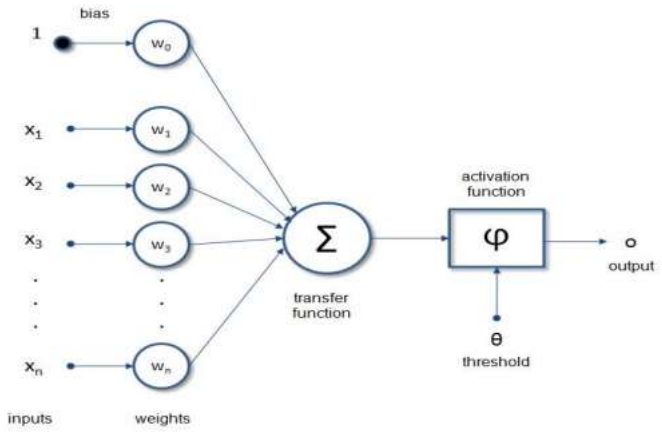

Figure 2 Artificial Neural Network [16]

The most widely used and simplest model of ANN is the perceptron, an artificial neuron. The above image is the artificial neural network structure. In the Perceptron model, the first layer is X_n inputs, for example in our case each pixel value of a CT scan or MRI image of our datasets. Each X_n input has "weight", in other words, weight signifies the importance and strength of the input(pixels). Inputs are multiplied by these W_n weights. This multiplication is done for each "neuron" and the result of the sum function: $\sum Xn*Wn$, which is summed up with a bias. The bias is a value that allows the activation function to be shifted to the left or right.

The next step is applying the activation function, which is a mathematical function applied to the output content of a neuron or a layer of neurons. In this way, the input "neuron" content is transformed into output. Its task is to shape the output of the neural network. The output generated from the perceptron model, results are compared with the true result (initial labeled ones) to calculate the error. Error is calculated to measure the

performance of the model. The lower the error, the higher the accuracy between the predicted results and actual ones. In case the error is large, weights must be adjusted until the error becomes small.

1.1.2. Deep Neural Network

Deep Neural Network is a type of ANN (Artificial Neural Network) that has a feature called depth. This feature allows deep neural networks, DNN to learn deeply hierarchical patterns in data. Since Deep Neural Networks are a subset of Artificial Neural Networks, that means DNN inherits the structure of ANN. Intermediate hidden layers and the learning is done deeply in these layers. Different from ANN, DNN is just an extension of ANN, adding layers and complexity.

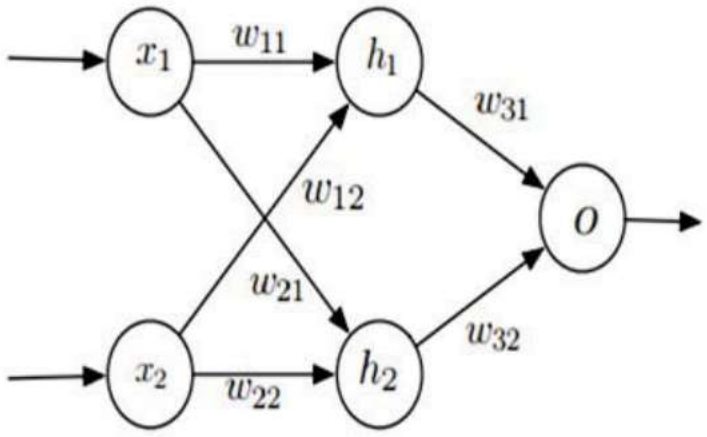

Figure 3. Deep Neural Network [17]

In a deep neural network, each layer added will transform the input into a more abstract representation. The first layer is the input layer and its function is needed to pass the features to the next layers. No processing or transformation action occurs here. It is a

layer where data is entered and passed. The other upcoming part is called the “hidden layers” part since all the layers in this section do computation functionalities. The keyword “hidden” stands for the fact that these layers do not directly communicate with the training set. Also, their inputs, and outputs are not observed in the training set.

1.1.3 Convolutional Neural Network

Convolutional Neural Network is one of the well-known fields used in image recognition, image classification, and medical image analysis. The main advantage why we are using it is due to the efficiency in cases with high-dimensional data. Different from other deep neural network models, CNN has some convolution layers before the Neural Network layers. They are useful to filter the images provided. Each convolutional layer contains a set of learnable filters (also known as kernels). This layered architecture allows CNNs to excel in tasks like image and video recognition.

During image processing the convolution formula denoted by (*), between two matrices K and I:

$$S(i,j) = (I \star K)(i,j) = \sum_{m}\sum_{n} K(i+m, j+n) I(m,n)$$

In formula:

S(i,j) is the output of the convolution operation at position (i, j)

"I" represents the input image or input feature map.

"K" represents the kernel or filter.

"$\star$" denotes the convolution operation.

"(i, j)" indicates the position in the output feature map where the result of this convolution is stored.

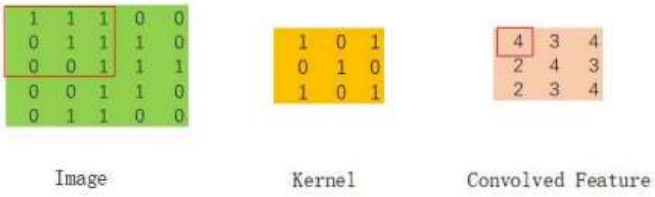

Figure 4 Convolution of the image [18]

Convolution between two matrices is the task of image processing where one matrix(kernel) slides into another larger matrix(input or image). The dot product of positions generates a new matrix. This new matrix represents how the filter interacts with different parts of the input image.

1.2 Sequential model

Sequential Model is a linear architecture where each layer is stacked one after another in sequence. The output of the layer serves as input for the next layer. This type is characterized by 4 properties, where the first one is Linearity. Linearity because there is no loop, branch, or skip. The second characteristic is the ease of construction because it can easily be constructed and visualized. The last two characteristics of the Sequential model are flexibility and modularity. Flexibility stands for the ability to build anything from simple linear regression to a deep convolutional neural network. Modularity means that each layer is treated as a module, which is easily added or removed. Modularity allows rapid experimentation and iteration.

1.3. Datasets

1.3.1 Introduction to CT scans

For our model, the dataset consists of medical images. One of our datasets consists of CT scans. CT scans (Computed Tomography scans) are a type of medical imaging technique that uses X-rays and computer processing. They are used to create detailed cross-sectional images of the body, in our case, the lung part.

The three main components of CT scans are slice thickness, contrast, and resolution. These components make the difference from one image to another. Thickness stands for the depth of each image slice. This might impact the detail visible in each slice. Contrast helps in making a tissue more visible, and easier to be recognized. High resolution, means a detailed look at the structure, crucial for diagnosing and monitoring medical conditions.

Examples of the CT scans from our datasets are below:

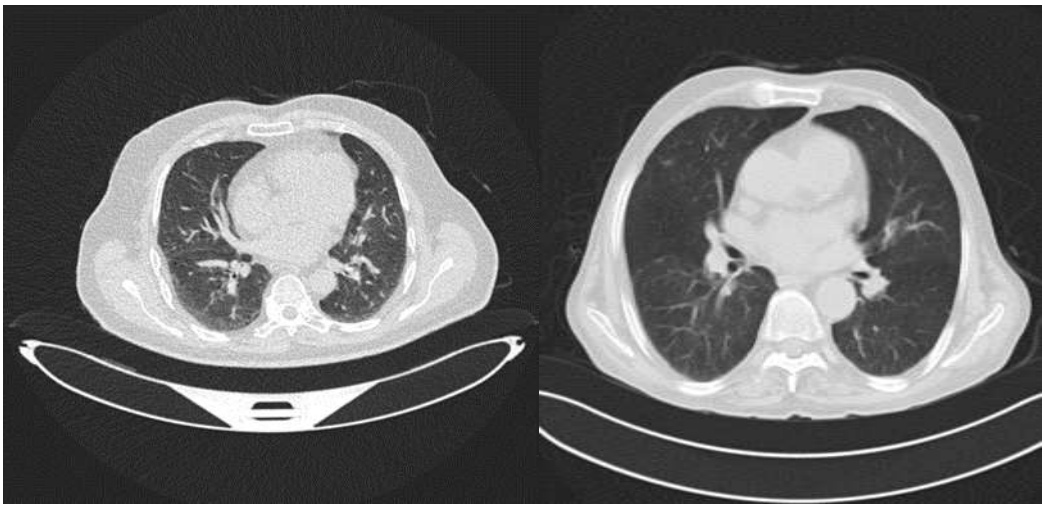

Figure 5 Visualization of CT scans[2]

For our system, we are using one dataset for Lung with only grayscale images.

The first dataset is from the Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) and this lung cancer dataset was collected in the above-mentioned specialist hospitals over a period of three months in the fall of 2019. It includes CT scans of patients diagnosed with lung cancer in different stages, as well as healthy subjects. IQ-OTH/NCCD slides were marked by oncologists and radiologists in these two centers. The dataset contains a total of 1190 images representing CT scan slices of 110 cases[2]. In this dataset of grayscale images, the structure of images is represented in matrix form. Based on how they are categorized where each pixel represents a shade of gray. The matrix values range from 0 to 255, where 0 represents black, and 255 represents white. The values in between represent various shades of gray.

1.3.2 Introduction to MRIs

MRIs or Magnetic Resonance Imaging is a medical imaging technique. It is used in radiology, which forms pictures and physiological processes of the body. Different from CT scans, MRI uses powerful magnets and radio waves to create detailed images of organs. It includes 4 steps. The part of the body being examined is put inside a large tube-shaped magnet to realign water molecules in the body. Then a radio frequency is turned on to produce faint signals. These signals are detected by a receiver in the MRI machine. They are of different types, and divided for example normal ones and tumor ones. These signals are converted to images by a computer attached to the MRI scanner. There are two brain MRIs from the dataset, one healthy case and one with tumor case:

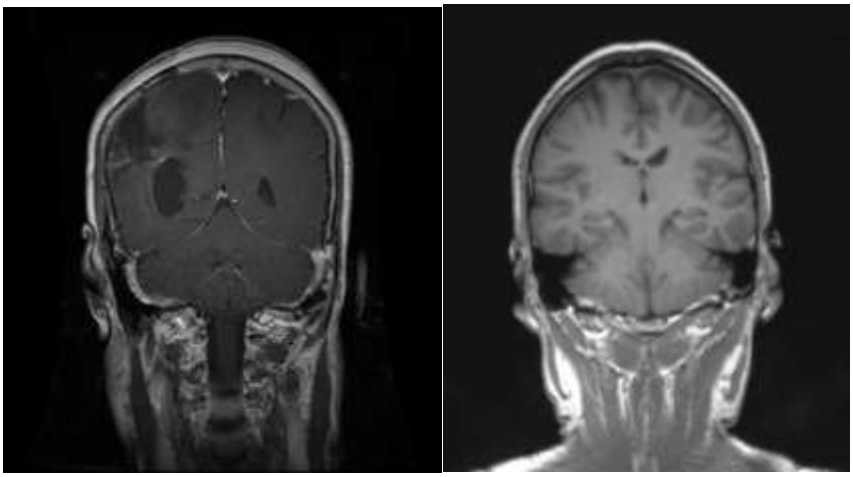

Figure 6 Visualization of Brain MRIs [3]

This dataset contains 7023 images of human brain MRI images which are classified into 4 classes: glioma - meningioma - no tumor and pituitary. Also, there is a healthy one's class. This dataset is a combination of three datasets: figshare[11], Sartaj Dataset[12], Br35H[13].

CHAPTER 2

OBJECTIVES OF THIS STUDY

1. Collecting datasets of lung and brain radiographic images to train and test our model, making sure datasets provide diverse cases and sizes.

2. Enhancing the data making it suitable for our detection system using preprocess. This avoids imbalances in the dataset.

3. Select the appropriate deep-learning algorithm that proves high performance for the Tumor Detection System for medical images.

4. Achieving high accuracy and overall performance using our model to identify the tumor's presence.

5. Creating a Tumor Detecting System, which is reliable and useable in real-life scenarios for medical professionals.

6. Creating a multipurpose model tested in several datasets.

CHAPTER 3

LITERATURE REVIEW

3.1 History background of CNN

CNN is named an improvement of the backpropagation neural network. They use forward propagation to output calculated values. Meanwhile backpropagation is used to adjust weights and biases. CNN uses the convolution operation and pooling operation. This is applied in the original input to obtain increasingly complex feature graphs. Then, it directly outputs the results through the fully connected layer. It consists of five parts which are: (i) input layer, (ii) convolution layer, (iii) pooling layer, (iv) fully connected (FC) layer, and (v) output layer [4].

The concept of connecting units to local receptive fields on the input is firstly seen in the early 1960s, perceptron. It was almost simultaneous with Hubel and Wiesel's discovery of locally sensitive, orientation-selective neurons in the cat's visual system. Local

connections have been used in neural models of visual learning. With local receptive fields neurons can extract elementary visual features such as oriented edges, endpoints, and corners. These features are then combined by the subsequent layers to detect higher-order features [9].

3.1.1. Application of CNN in CT scans

Studies have shown that the application of deep learning, particularly Convolutional Neural Networks, has been effective in diagnosing COVID-19 from CT scans. During the pandemic, CNN had a high potential to enhance diagnostic processes in healthcare and in managing pandemic-related challenges. For instance, one study was done in 2020 where the application of CNN in chest CT scans is considered a good achievement. The model had an accuracy of 97.59%. The model was tested in a dataset of 1119 COVID-19 chest CT images and 446 normal images[14].

3.1.2. Application of CNN in MRIs

CNN is widely used as a deep-learning algorithm for different reasons, mostly in cases of images. Studies have proved that CNN is efficiently used, especially with medical imaging. One type of medical image is MRI. It has differences compared to CT scans, in the aspect of how they work. Among several types of research, one very good to be considered is the “Brain Tumor Detection and Classification Using CNN Algorithm and Deep Learning Techniques”. The aim was the classification of brain MRIs into tumoral

or not. Using CNN, the accuracy achieved was 98.029%, which demonstrates the success of applying CNN in MRI images [15].

3.2 Types of Layers in CNN

3.2.1 Input Layer

The input layer is the initial layer of a neural network with the responsibility of taking raw data. Data might be images, text, audio, etc. In this study, the input layer has the image fed into the network. It is a three-dimensional array of pixels [7]. This layer has no computation role or transformations on the data, just to pass the structured data to the next layers.

In the case of the image as input data, the 32x32 pixel color image, in the input layer will be accepted as an array shape 32x32x3.

3.2.2 Convolution Layer

The convolution layer is the core building of CNN. It applies several filters to the input and each filter detects different features. This is done by performing a convolution operation between the filter and input, producing a feature map.

The feature map is created in the convolution layer by applying dot product numerical operation of grids of weights. It is done all through each example of input information like pictures, videos, and many others[5].

The main purpose of the convolution layer is to transform the segmented images into a set of feature maps. This is done by performing convolution operation i.e. defined as the sum of the dot product of two functions after one function is reversed and shifted. A kernel filter is used to filter the input segmented images and the resultant convolved output produces the feature maps[6].

$$F(i,j)=(I*K)(i,j)$$

Where, F(i,j) = set of feature maps, I(i,j) = input (segmented) image, and K = kernel filter.

3.2.3 Pooling Layer

Pooling Layer is essential for computational reasons and in the improvement of detection capability. Its main function is reducing width and height but not depth. The reduction of the input feature is done to decrease the amount of computation and weights. Two common methods: Max Pooling and Average Pooling. Overall, this layer aims to simplify the network. Two well-known operations of the Pooling layer are upsampling and downsampling. The current model consists of average pooling layers.

3.2.3.1 Average Pooling

Average Pooling or mean pooling, is one of the methods used in the Pooling layer. This method or technique is used to downsample an input representation, reducing its dimensionality.

This method operates by sliding a window(pool size) over the input matrix. For each position of the window, it calculates the average of elements within that window. This average value then becomes a single element in the output matrix. Parameters are pool size, strides, and padding. Pool size is the size of the window over which the average is computed. Stride shows the step size as the window moves over the input matrix. Padding is a parameter, especially if the input size is not perfectly divisible, so can be used to manage the dimensions.

In our model, each convolutional layer is followed by an average pooling layer, implemented as nn.AvgPool2d. This helps in reducing the spatial dimension of the output from the convolutional layers. Also, reducing the feature maps and ensuring the most essential information.

3.2.3.2 Down-Sampling

In this layer, down-sampling of the feature maps is done by taking the maximum value from a small window length map.

$F=Max\{F(i,j)\}$

Where, F(i,j) = set of feature maps produced by the convolutional layer, F = reduced feature maps[6].

3.2.4 Activation Layer

The activation Layer applies a nonlinear function to the input it receives. Activation Layer applies a nonlinear function to each pixel in the feature. Without the activation layer, the model would remain a linear regression model. This limits solving complex problems that require non-linear solutions. This layer introduces non-linearity into the model which facilitates learning complex data. The normally used functions for this model are:

$f(x) = tanh(x)$,

$f(x) = sigmoid(x)$

$ReLu(x) = Max(0,x)$ [7]

In this model, I have chosen the tanH activation function. The input data is dataset type, which means there are MRIs and CT images. They involve different intensities and contrasts, reflecting different types of abnormalities. One advantage of TanH is the output values range [-1;1], fitting data that might be centered around 0. In cases, when the means intensity of images is adjusted to be around 0, here TanH helps us in transitioning these features.

3.2.4.1 TanH

The output values of this function range (1,-1). Its function is S-shaped, which spans from -1 to 1, and this symmetry leads to better performance of deep neural networks. The derivative of tanh functions indicates a strong gradient when the input is near 0. The disadvantage of this function is the vanishing gradient problem for large absolute inputs. In practice, the Tanh function is mostly used in cases where normalization is significant.

3.2.4.2 ReLU

ReLu function or Rectified Linear Unit function is used, due to its efficiency computational ability, good performance, and its simplicity.

$$ReLu(x) = Max(0,x)$$

Its formula states that each input is the output itself if the input is greater than 0, otherwise the output is 0.

The function's characteristics are linearity(partially), computational efficiency, and sparse activation. Linearity stands partially since allows both linear and non-linear properties for learning complex patterns. Computational Efficiency stands for the reason that this function does not require expensive operations or calculations. In both calculations and backpropagation has efficiency. Sparse activation characteristics occur from the fact that when input is smaller than 0, output remains 0. In practice, only a subset of neurons remains active at any time.

Many times ReLu causes the death of neurons during training if they fall into the region where their only output is 0. This problem does not exist in the case of TanH, due to its non-zero gradient across all input values.

3.2.4.3 Sigmoid Function

The sigmoid function, known as the logistic function is primarily used in the output layer of binary classifiers. The range of output values is (0,1), which is suitable for interpreting output as probability. The function is S-shaped, smooth, and differentiable, so it is efficiently used for backpropagation in calculating gradients. Sigmoid has the same problem as the ReLu function. It pushes output values towards extremes of 0 and 1(in hidden layers), leading to the "dying" of neurons.

3.2.5 Fully Connected Layer

A fully connected Layer takes all neurons in the previous layer and connects it to every neuron it feeds into. Final decisions based on the features detected by convolutional and pooling layers are made by Fully connected layers.

A fully Connected Layer is applied to the sequence of Convolution, TanH, and AvgPooling layers. The input to a fully connected layer is a meaningful, low-dimensional feature space. The fully connected layer is used to get a non-linear combination of the features. It holds a feature vector for the input, which is needed for classification or regression and categorization. A transfer function 'tf' shows the input-output relation in the layer. A fully Connected layer gives end-to-end training to the network. The final layer outputs 1 feature, representing a binary classification output given the subsequent sigmoid activation.[8].

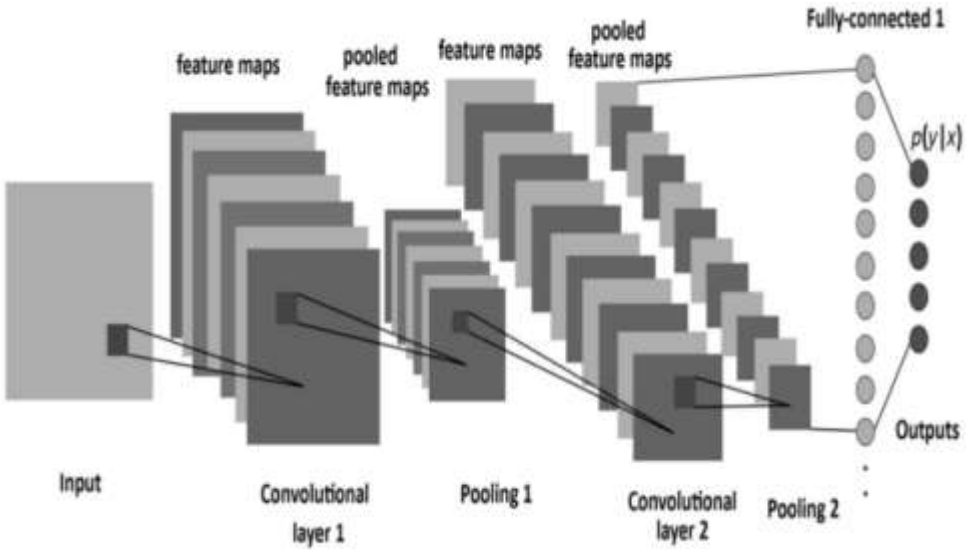

Figure 7 Convolution Neural Network with Pooling and Fully Connected Layers[5]

3.2.5.1 Flatten Operation

Flatten operation occurs before feeding into the fully connected layers. The output of convolutional is flattened using x=x.view(x.size(0),-1). This operation reshapes the multi-dimensional output of convolutional layers into a single vector for each image. This is suitable for processing by the fully connected layers.

3.2.6 Sigmoid Activation

In the forward pass, after processing through the fully connected layers, there is the application of the sigmoid function (F.sigmoid(x)) at the output. So, in this way, this activation function is used to map the final output to a probability of 0 or 1. In this study of binary classification, tumoral or not.

3.3 Summary of literature review

References stated are gathered from different sources and cited in the References section.

Studies [4,5,6,7,8] focus on the development of automated systems for detecting Lung cancer from CT scans using Convolutional Neural Networks. These aim to identify lung nodules and classify them as tumorial or not. These references are cited in our study because a part of our study is related to CT scans and cancer cases in the lung. Also [14], and [15] focus on the application of CNN architecture in medical imaging such as MRI and CT scans. We have cited [14] and [15] to use them as a foundation base where other previous studies have stated the application of CNN in different medical images.

Data sources and annotations

Data sources, public datasets, [2] IQ-OTH/NCCD lung cancer dataset, [3] data from platform Mendeley, [12][13][14] Brain Tumor datasets, which make the compound of the dataset we have used of Brain MRIs.

Technological and Methodological

Papers [4,7,8] demonstrate the use of sophisticated neural network models for image classification with medical imaging. We have cited them in the section of CNN introduction, input layers, fully connected layers and activation layers.

Theoretical Contribution

Gradient-based learning [9]: The work of Y. Lecun et al. laid to groundwork for many advancements in using convolutional neural networks for image data. We have cited this because our model uses gradient-based learning.

CHAPTER 4

METHODOLOGY

4.1 Overview

This section details the methodology employed in the development of a convolutional neural network(CNN) model to classify medical imaging data. The approach integrates libraries and frameworks for handling data, processing, training, and evaluation.

4.2. Libraries

4.2.1 Python Libraries

Libraries used for the code are Pytorch, NumPy, SciKit-learn, OpenCV, and Matplotlib. Pytorch assists in creating complex architectures, in this case, CNN.

TorchVision in this project uses a transforms module for data processing and augmentation such as random rotations and flips.

Torch. nn provides the building blocks used for designing neural networks, layers, activation functions, and loss functions.

Torch.optim: includes optimization algorithms like Adam, for updating network weights.

NumPy is used mostly for data manipulation and preprocessing before feeding data into a neural network.

SciKit-Learn aims to feature tools for machine learning. This library is used for train-test split functions, which divide the model into train and test parts.

OpenCV or Open Source Computer Vision Library, consists of image-processing functions, concretely used in image augmentation, resizing, and color transformation from BGR to RGB format.

Matplotlib is a plotting library used for visualizations in Python. In this thesis, this library is used for displaying healthy and tumoral cases of both lung scans and brain MRIs. In a general term visualizing the dataset.

4.3. Data Visualization

For the thesis, are collected two datasets, one with CT scans for lung and one with brain MRIs.

4.3.1 IQ-OTH/NCCD Dataset

This dataset is from the Iraq-Oncology Teaching Hospital/National Center for Cancer Diseases (IQ-OTH/NCCD) and this lung cancer dataset was collected in the above-mentioned specialist hospitals over a period of three months in the fall of 2019. The dataset has 1190 images representing CT scan slices of 110 cases. These cases are grouped into three categories: normal, benign, and malignant. The CT scans were originally collected in DICOM format. The scanner used is SOMATOM from Siemens. CT protocol includes: 120 kV, slice thickness of 1 mm, with window width ranging from 350 to 1200 HU, and window center from 50 to 600 were used for reading. Cases

vary in gender, age, area of residence, and living status. Most of them come from places in the middle region of Iraq, particularly, the provinces of Baghdad, Wasit, Diyala, Salahuddin, and Babylon.[2] This dataset for our study is organized only into two classes, one with normal cases and another with tumorial cases(merged Malignant and Benign). Results will be given only if a CT scan is the normal case or tumorial case.

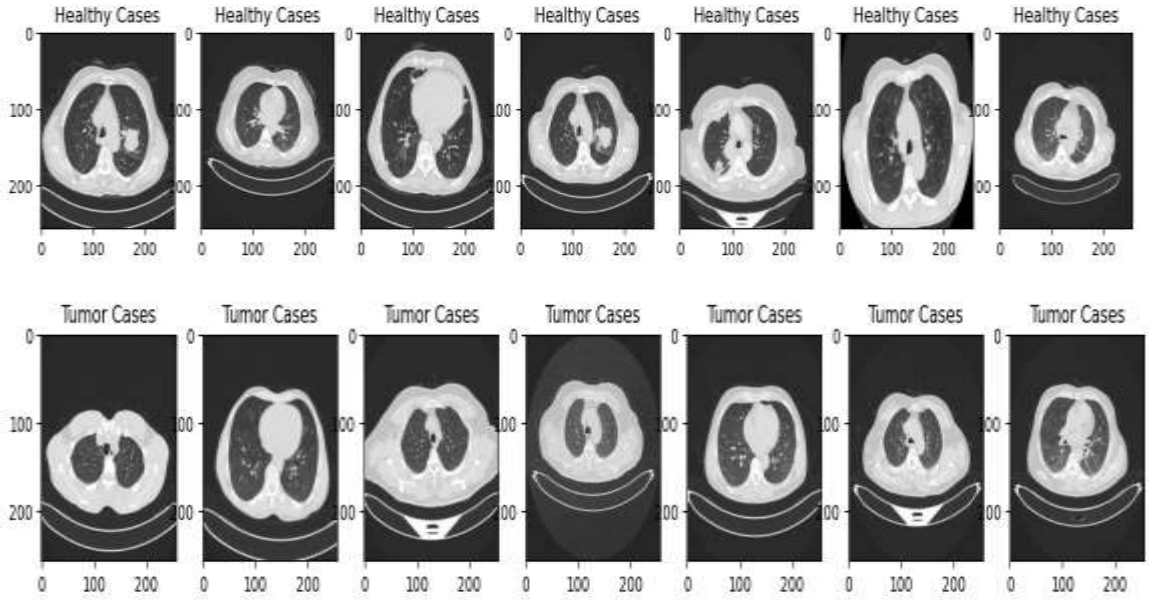

Figure 8 CT scans images

4.3.2 Brain MRIs Dataset

The second dataset consists of brain MRIs. This dataset contains 7023 images of human brain MRI images which are classified into 4 classes: glioma - meningioma - no

tumor and pituitary. Also, there is a healthy one's class. This dataset is a combination of three datasets : figshare[11], Sartaj Dataset[12], Br35H[13].

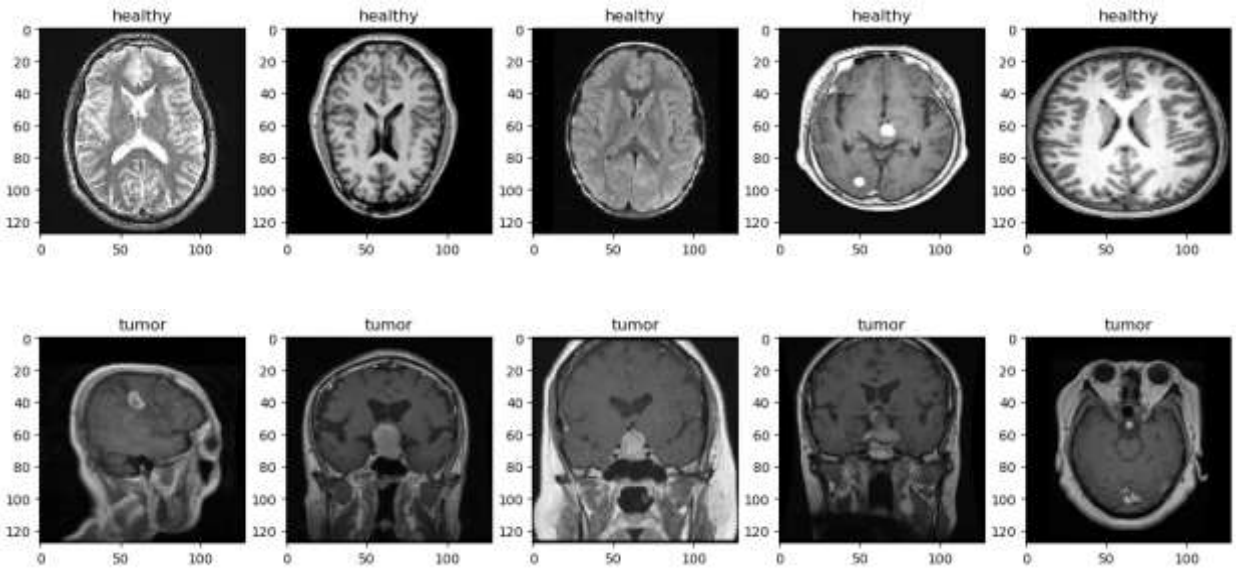

Figure 9 MRIs images

4.4. Code analyze

4.4.1 Image Collection

Images are sourced from local directories using the glob module.

Brain Tumor Dataset

Tumor Images: Files loaded from the directory labeled ‘glioma’ for tumor cases.

Healthy Images: Files are loaded from the directory labeled ‘notumor’ for non-tumor cases.

Lung CT scans Dataset

Tumor Images: Files loaded from the directory labeled ‘CancerCases’ for non-healthy cases.

Healthy Images: Files are loaded from the directory labeled ‘NoCancerCase’ for healthy cases.

Each file path returned by ‘glob.iglob’ is used to load images using OpenCV’s cv2.imread. This ensures that images are read into memory for processing.

4.4.2 Image Preprocessing

Image Reading: Use of the cv2.imread function to load images in the BGR color space.

Resizing: Standardizing the images to a uniform size of 128x128 pixels. This ensures consistency in input data dimensions.

Color Chanel Reordering: Converting images from BGR to RGB format using split() and merge() methods for each image.

Image Normalization: Dividing pixel values by 255 to scale them between 0 and 1. This helps in speeding up the convergence during training.

4.4.3 Dataset Class Creation

The class called MRI (library torch.utlis.data) inherits from PyTorch’s Dataset (abstract class representing a dataset) class. MRI class is used to handle MRI image data specifically and includes the following overridden methods:

__init__: Initializes the dataset object, loading and preprocessing the images, and storing them along with their labels(1-tumor, 0-healthy).

__getitem__: allows indexing into the dataset such that each call returns a specific image and its label. It retrieves an image-label pair based on the index provided. The conversion of the image data into a tensor format is done to make it suitable for model input.

__len__: returns the total number of images in the dataset.

Healthy images are stored in the list named ‘healthy’, while tumor images are in the list named ‘tumor’. After all the preprocessing steps mentioned, the lists are converted to arrays with type ‘float32’ for compatibility with PyTorch. The labels are assigned for each array. Then we have concatenated tumor and healthy images in one single array called ‘self. images’ and labels in one single array called ‘self. labels’.

Here are 5 random MRIs for each category after the preprocessing step. After creating an array called ‘healthy_imgs’ for 5 random normal cases and an array of 5 random cases called ‘tumor_imgs’ for tumor cases (using NumPy library). For generating 5 random images for each array, we have used ‘random. choice’ from the Random library. To visualize we have used plt. imshow() from library matplotlib.pyplot and its function is visualization.

4.4.4 Train/Test Split of Dataset

After the preprocessing step, we split the dataset into training and testing parts with the help of the SciKit-Learn library importing train_test_split.

The current name of the dataset is mri_dataset, so mri_dataset.images and mri_dataset.labels are used as inputs to access the dataset. As parameters are used test_size=0.2 and random state=42. Test_size parameter specifies that 20% of the dataset is reserved for testing, while random_state is a random number generated.

As output, the split will divide the dataset into training sets (X_train, Y_train) and testing sets (X_test, Y_test).

We have created two instances of the MRI dataset, one for training (train_dataset), and one for testing (test_dataset). For train_dataset, are assigned X_train and Y_train, while for test_dataset are assigned X_test and Y_test.

4.4.5 Integration with PyToch's DataLoader

For each instance we created from the class MRI dataset, instances of this class are directly passed to a DataLoader object. This supports efficient batch processing, shuffling, and parallel processing of the data during the model training phase.

Each instance created during the Train/Test split of the dataset is passed to a data loader, train_dataloader, and test_dataloader. For the train_dataloader parameter(shuffle=True), means the order is shuffled at the beginning of each training epoch to reduce model overfitting. For the test_dataloader parameter(shuffle=False) means no need to shuffle the testing data.

We have created a dictionary to map numeric labels to readable strings. Images with numeric value 0 are Healthy, while numeric value 1 are Tumor. To display the images from the training set created, we use a sample that contains a batch of images. During iteration with for-loop, each image in the batch is iterated. Since the batch of images is not specified, by default is 1. Inside the for-loop, is the squeeze() method. This method removes single-dimensional entries from the shape of the array: [1,128,128,3]→[128,128,3]. Then, images are reshaped(reshape()), ensuring that each image has the shape (height, width, channels). Reshape is done because the image must fit expected format by matplotlib(imshow() function) library.

First six shuffled images from the training dataset:

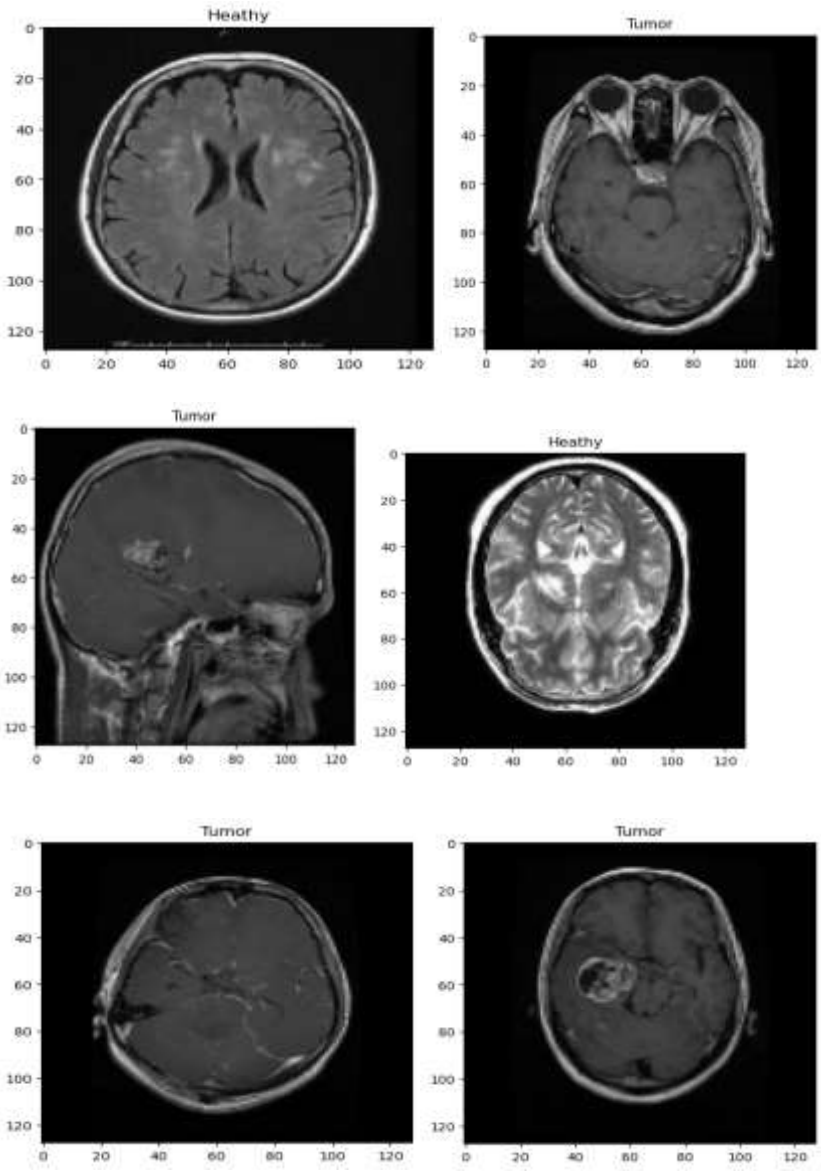

Figure 10 Six shuffled images during training

4.4.6 Model

For this study we have used CNN, so the model will be a Convolutional Neural Network model. Firstly is created a new class CNN(nn.Module). nn.Module is the base class for all neural network modules.

Self.cnn_model: stores the sequential container(nn.Sequential), which executes operations in a sequential order.

The Conv2d Layer is the first convolutional layer with 3 input channels(RGB channels), 6 output channels, and a kernel size of 5x5. In this layer are 6 different filters

Tanh Activation(nn.Tanh()) is the activation function that scales the output of the previous layer from -1 to 1. It aims to introduce non-linearity into the model. This helps in learning complex patterns.

Average Pooling Layer: (nn.AvgPool2d) is put to perform average pooling with parameters(2x2 window,stride=5). It aims to reduce the spatial dimensions for the next convolutional layer.

Second Conv2d Layer: second convolutional layer with parameters(in_channels=6, out_channels=6, kernel_size=6). It increases the depth by transforming from 6 to 16 output channels with the same kernel size, to learn complex features.

Second Tanh Activation: used to maintain non-linearity in deeper parts of the network.

Second Average Pooling Layer: The second pooling layer reduces deeper the dimensionality, simplifying the network.

Fully Connected Layers: **self.fc_model** is a container(sequential) that holds the dense layers.

Linear Layer: a fully connected layer that takes the flattened output of the previous layer(256) and outputs 120 features.

Third Tanh Activation: non-linearity is introduced in this level of depth.

Second Linear Layer: reduces dimensionality from 120 features to 84 features.

Fourth Tanh Activation: continues introducing non-linearity.

Output Linear Layer: maps 84 features to a single output feature, for binary classification.

Forward Pass: this method defines the data flow.

X=self.cnn_model(x): passes the input through convolutional layers.

X=x.view(x.size(0), -1): flattens the output of the convolutional neural network, so it can be fed into the fully connected layers.

X=self.fc_model(x): passes the flattened output through the dense layers.

In the end, the sigmoid function is applied to the final output, so the output ranges between 0 and 1. This is useful for this study, of binary classification.

4.4.7 Experimental Setup

In the hardware implementation, the main task is the Configuration of Device: if GPU is available and set the device. Tensor Operations on GPU: moving tensors to the chosen device(GPU/CPU), ensuring the full hardware capabilities.

In our case, we have used a CPU which needs more time to be executed compared to a GPU. Also, the time needed for running depends on the size of the dataset. The larger the dataset, the longer the execution time. With CPU support, the needed time for all execution(including testing, training, and validation) for a dataset of around 2000 images, is around 35 minutes. Meanwhile, the dataset of MRIs of 10000 images needs about 3 hours, to be fully executed.

4.4.8 DataLoader for Batch Processing

Train_dataloader is modified, in which is added the parameter of batch_size=32. This parameter means the dataset will be divided into batches of 32 images each during

training. Also, the shuffle is changed from TRUE to FALSE, to maintain consistency of data order during training.

4.4.9 Model Evaluation

Model Evaluation Mode: model.eval() sets the model to evaluation mode.

No Gradient Calculation: torch.no_grad() is used to disable gradient calculation. It aims to reduce memory consumption and speed up computations.

Batch processing: Iterating over the train_dataloader, processing images and labels among each batch. Both of them are moved to the configured device, GPU or CPU. .to(device): moves data to GPU or CPU, where the model is located.

Model prediction and Storage: Predictions(y_hat) for a given batch of images are made using the model, and both predictions and labels are detached from the GPU, moved to the CPU, and converted to NumPy arrays.

Then we check for dimensionality, if the output is 0 dimensional, extra dimensions are added.

All individual output arrays and labels are combined in a single array. The axis=0 specifies that concatenation happens along the first axis. Squeeze(): removes any single-dimensional entries from the shape of the arrays. This makes it suitable for metrics calculations.

4.4.10 Threshold Function

The threshold function is a function that applies a threshold to the raw scores output by the model. In this study, the threshold is set at 0.50. If a score is higher or equal to 0.50, it is set to 1 otherwise 0.

Accuracy_score(scikit-learn) is used to compute the accuracy of predictions against the actual labels.

4.4.11 Optimization and Loss Function

Our model is optimized using Adam optimizer with a very small learning rate (0.0001) for smooth convergence. The BCELoss(binary cross-entropy loss) is used as a loss function, one of the best approaches for binary classification tasks. Between the target and the output prediction, the loss function is used to calculate the difference.

4.4.12 Training Procedure

Training occurs over 400 epochs. Gradient clipping is used during training by setting a maximum gradient norm of 1.0 to prevent the exploding gradient problem. The learning rate scheduler(ReduceLROnPlateau) adjusts the learning rate based on training progress. The scheduler reduces the learning rate by a factor of ten if the training loss does not decrease for five consecutive epochs.

MRIs case:

Train Epoch: 10	Loss: 0.301650
Train Epoch: 20	Loss: 0.235083
Train Epoch: 30	Loss: 0.200398
Train Epoch: 40	Loss: 0.172511
Train Epoch: 50	Loss: 0.151849
Train Epoch: 60	Loss: 0.136373
Train Epoch: 70	Loss: 0.120748
Train Epoch: 80	Loss: 0.111547
Train Epoch: 90	Loss: 0.094803
Train Epoch: 100	Loss: 0.086058
Train Epoch: 110	Loss: 0.076830

Train Epoch: 120	Loss: 0.069612
Train Epoch: 130	Loss: 0.055980
Train Epoch: 140	Loss: 0.046692
Train Epoch: 150	Loss: 0.037034
Train Epoch: 160	Loss: 0.026720
Train Epoch: 170	Loss: 0.020285
Train Epoch: 180	Loss: 0.014097
Train Epoch: 190	Loss: 0.012156
Train Epoch: 200	Loss: 0.007070
Train Epoch: 210	Loss: 0.004511
Train Epoch: 220	Loss: 0.003067
Train Epoch: 230	Loss: 0.002834
Train Epoch: 240	Loss: 0.001725
Train Epoch: 250	Loss: 0.001003
Train Epoch: 260	Loss: 0.000057
Train Epoch: 270	Loss: 0.000032
Train Epoch: 280	Loss: 0.000031
Train Epoch: 290	Loss: 0.000027
Train Epoch: 300	Loss: 0.000023
Train Epoch: 310	Loss: 0.000019
Train Epoch: 320	Loss: 0.000015
Train Epoch: 330	Loss: 0.000012
Train Epoch: 340	Loss: 0.000010
Train Epoch: 350	Loss: 0.000008
Train Epoch: 360	Loss: 0.000007
Train Epoch: 370	Loss: 0.000006
Train Epoch: 380	Loss: 0.000005
Train Epoch: 390	Loss: 0.000004

Train Epoch: 400	Loss: 0.000003

Table 1. Training results with ReduceLROnPlateau in MRI dataset

CT scans case:

Train Epoch: 10	Loss: 0.562833
Train Epoch: 20	Loss: 0.460065
Train Epoch: 30	Loss: 0.339362
Train Epoch: 40	Loss: 0.241436
Train Epoch: 50	Loss: 0.170281
Train Epoch: 60	Loss: 0.129412
Train Epoch: 70	Loss: 0.098642
Train Epoch: 80	Loss: 0.064749
Train Epoch: 90	Loss: 0.053960
Train Epoch: 100	Loss: 0.036211
Train Epoch: 110	Loss: 0.029523
Train Epoch: 120	Loss: 0.017139
Train Epoch: 130	Loss: 0.019063
Train Epoch: 140	Loss: 0.010672
Train Epoch: 150	Loss: 0.010659
Train Epoch: 160	Loss: 0.010631
Train Epoch: 170	Loss: 0.010609
Train Epoch: 180	Loss: 0.010618
Train Epoch: 190	Loss: 0.010799
Train Epoch: 200	Loss: 0.010612
Train Epoch: 210	Loss: 0.010719
Train Epoch: 220	Loss: 0.010676
Train Epoch: 230	Loss: 0.010640
Train Epoch: 240	Loss: 0.010612

Train Epoch: 250	Loss: 0.010689
Train Epoch: 260	Loss: 0.010634
Train Epoch: 270	Loss: 0.010609
Train Epoch: 280	Loss: 0.010611
Train Epoch: 290	Loss: 0.010631
Train Epoch: 300	Loss: 0.010756
Train Epoch: 310	Loss: 0.010684
Train Epoch: 320	Loss: 0.010689
Train Epoch: 330	Loss: 0.010617
Train Epoch: 340	Loss: 0.010606
Train Epoch: 350	Loss: 0.010615
Train Epoch: 360	Loss: 0.010596
Train Epoch: 370	Loss: 0.010699
Train Epoch: 380	Loss: 0.010616
Train Epoch: 390	Loss: 0.010622
Train Epoch: 400	Loss: 0.010614

Table 2. Training results with ReduceLROnPlateau in CT scans dataset

4.4.13 Optimized Model Evaluation

After training using Adam optimizer with learning rate and loss function(BCELoss) use, the model is again evaluated for better results. Visualizations are done with heatmaps and the results are analyzed using a confusion matrix.

4.4.14 Visualization of Convolutional Layers

Extraction: The model has several layers, among which convolutional layers are targeted for visualization due to their role in feature extraction from images. Decomposition of network architecture in individual layers is needed so we extract only

convolutional(nn.Conv2d) layers. Convolutional layers are found by iterating through the model's children and we collect them into a list.

Data Preparation: Selection of one image from the dataset(100th image) for visualization. Initially is displayed as an input being processed by the network.

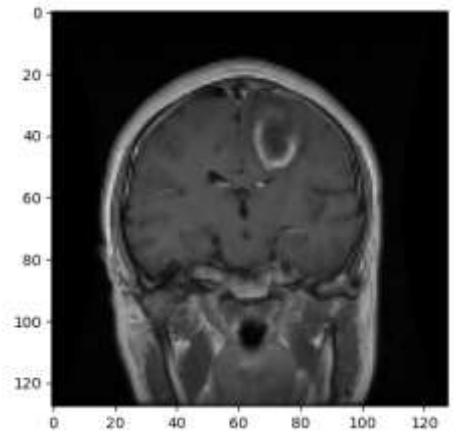

Figure 11 MRI image selected

Then this image is converted into a PyTorch tensor. This format is required for processing through the model and additional dimensions to accommodate the batch size.

Forward Pass Through Selected Layers: The processed image is passed through each extracted convolutional layer. The output of each layer is called a feature map, which represents the activation of the layer. During training these feature maps are indicative of the patterns the network learns to recognize.

Visualization of Feature Maps: Each feature mapped as output from the convolutional layers, is visualized displaying multiple channels within each layer. Each subplot corresponds to a single channel of the feature map. This is important for understanding the layer-wise transformations that occur within the network. Also, significant during this step is understanding the insight into what features of the data are considered important at different stages of processing.

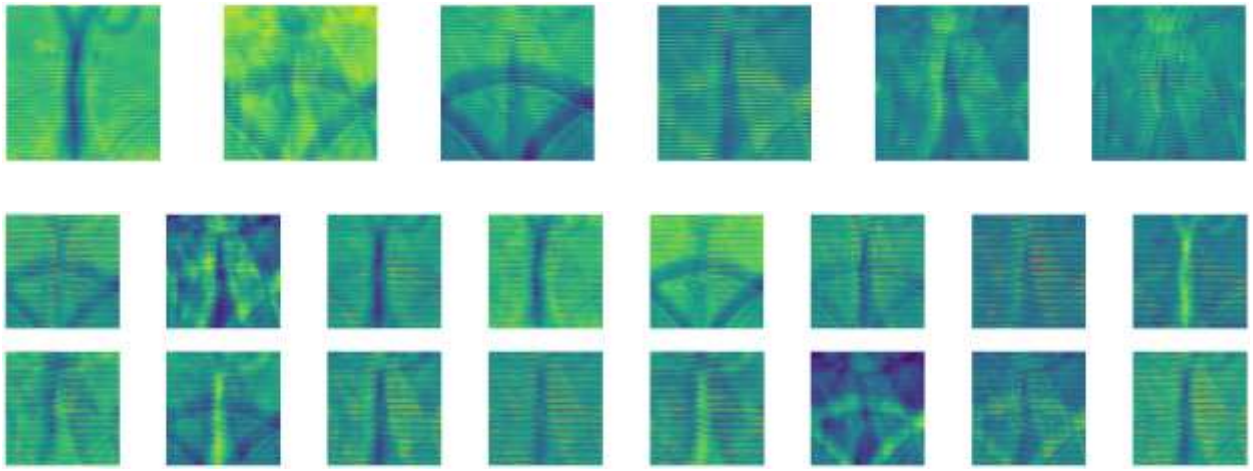

Figure 12 Visualization of features from Convolution layer

4.4.15 Training and Validation Process

MRI dataset is instantiated for training using PyTorch's dataloader. It provides a function to iterate over the dataset in batches. Two dataloaders are train_dataloader for training the data with shuffling and val_dataloader for validation data without shuffling. In this way does not affect performance.

Adam optimizer with a learning rate(eta) of 0.0001. The training occurs over 400 epochs. Each epoch consists of a training phase followed by a validation phase.

Train Phase:

model mode is(model. train()), enabling dropout layers if exist in our model into training mode.

batch processing: each batch of images and labels is loaded from the train_dataloader with the help of a for-loop.

Forward pass: used for the computation of y_hat for the given data.

Loss Computation: use of BCELoss function between predicted outputs and actual labels.

Backpropagation: the gradient of the loss concerning the model parameters computed.

Gradient Clipping: Gradients are clipped to a maximum norm of 1.0 to prevent exploding gradients, from destabilizing the training.

Optimization Step: The optimizer updates the model parameters.

Loss Tracking: The loss for each batch and the average loss per epoch is computed.

Validation Phase:

Model mode: model set to evaluation mode(model.eval()), to disable layers such dropout if exists.

No Gradient Computation: Gradients are not computed, because of(torch.no_grad()).

Batch Processing and Loss Computation: The computed losses are used only for evaluation and not for backpropagation.

Loss Tracking: The average validation loss for the epoch is computed.

MRIs training case:

Train Epoch: 10	Train Loss: 0.303000	Val Loss: 0.292715
Train Epoch: 20	Train Loss: 0.236320	Val Loss: 0.228751
Train Epoch: 30	Train Loss: 0.193609	Val Loss: 0.184619
Train Epoch: 40	Train Loss: 0.159293	Val Loss: 0.184990
Train Epoch: 50	Train Loss: 0.141757	Val Loss: 0.136623
Train Epoch: 60	Train Loss: 0.127096	Val Loss: 0.135775
Train Epoch: 70	Train Loss: 0.115164	Val Loss: 0.104141
Train Epoch: 80	Train Loss: 0.102495	Val Loss: 0.092318

Train Epoch: 90	Train Loss: 0.090556	Val Loss: 0.095216
Train Epoch: 100	Train Loss: 0.076802	Val Loss: 0.078608
Train Epoch: 110	Train Loss: 0.072944	Val Loss: 0.060831
Train Epoch: 120	Train Loss: 0.057174	Val Loss: 0.055813
Train Epoch: 130	Train Loss: 0.070401	Val Loss: 0.071685
Train Epoch: 140	Train Loss: 0.069931	Val Loss: 0.069946
Train Epoch: 150	Train Loss: 0.069924	Val Loss: 0.069946
Train Epoch: 160	Train Loss: 0.069931	Val Loss: 0.069941
Train Epoch: 170	Train Loss: 0.069936	Val Loss: 0.069940
Train Epoch: 180	Train Loss: 0.070772	Val Loss: 0.069936
Train Epoch: 190	Train Loss: 0.070056	Val Loss: 0.069923
Train Epoch: 200	Train Loss: 0.069925	Val Loss: 0.069919
Train Epoch: 210	Train Loss: 0.069928	Val Loss: 0.069909
Train Epoch: 220	Train Loss: 0.069904	Val Loss: 0.069906
Train Epoch: 230	Train Loss: 0.069884	Val Loss: 0.069905
Train Epoch: 240	Train Loss: 0.069885	Val Loss: 0.069896
Train Epoch: 250	Train Loss: 0.069910	Val Loss: 0.069897
Train Epoch: 260	Train Loss: 0.069898	Val Loss: 0.069899
Train Epoch: 270	Train Loss: 0.069876	Val Loss: 0.069901
Train Epoch: 280	Train Loss: 0.069872	Val Loss: 0.069891
Train Epoch: 290	Train Loss: 0.070277	Val Loss: 0.069893
Train Epoch: 300	Train Loss: 0.069867	Val Loss: 0.069892
Train Epoch: 310	Train Loss: 0.069882	Val Loss: 0.069899
Train Epoch: 320	Train Loss: 0.069882	Val Loss: 0.069896
Train Epoch: 330	Train Loss: 0.069994	Val Loss: 0.069896
Train Epoch: 340	Train Loss: 0.069872	Val Loss: 0.069899
Train Epoch: 350	Train Loss: 0.069885	Val Loss: 0.069887
Train Epoch: 360	Train Loss: 0.069921	Val Loss: 0.069887

Train Epoch: 370	Train Loss: 0.069880	Val Loss: 0.069880
Train Epoch: 380	Train Loss: 0.069868	Val Loss: 0.069888
Train Epoch: 390	Train Loss: 0.069910	Val Loss: 0.069886
Train Epoch: 400	Train Loss: 0.070712	Val Loss: 0.069876

Table 3. Training and validation results in MRI dataset

CT scans training case:

Train Epoch: 10	Train Loss: 0.572070	Val Loss: 0.568871
Train Epoch: 20	Train Loss: 0.435668	Val Loss: 0.428367
Train Epoch: 30	Train Loss: 0.341289	Val Loss: 0.325231
Train Epoch: 40	Train Loss: 0.256313	Val Loss: 0.252201
Train Epoch: 50	Train Loss: 0.199934	Val Loss: 0.222654
Train Epoch: 60	Train Loss: 0.157346	Val Loss: 0.160207
Train Epoch: 70	Train Loss: 0.121625	Val Loss: 0.114409
Train Epoch: 80	Train Loss: 0.091725	Val Loss: 0.083656
Train Epoch: 90	Train Loss: 0.066635	Val Loss: 0.068710
Train Epoch: 100	Train Loss: 0.048773	Val Loss: 0.078657
Train Epoch: 110	Train Loss: 0.030153	Val Loss: 0.045643
Train Epoch: 120	Train Loss: 0.025610	Val Loss: 0.024685
Train Epoch: 130	Train Loss: 0.026984	Val Loss: 0.023770
Train Epoch: 140	Train Loss: 0.024970	Val Loss: 0.021994

Train Epoch: 150	Train Loss: 0.023636	Val Loss: 0.021074
Train Epoch: 160	Train Loss: 0.021972	Val Loss: 0.021581
Train Epoch: 170	Train Loss: 0.019504	Val Loss: 0.018281
Train Epoch: 180	Train Loss: 0.021210	Val Loss: 0.017240
Train Epoch: 190	Train Loss: 0.017442	Val Loss: 0.016226
Train Epoch: 200	Train Loss: 0.015930	Val Loss: 0.015202
Train Epoch: 210	Train Loss: 0.015096	Val Loss: 0.014266
Train Epoch: 220	Train Loss: 0.013987	Val Loss: 0.013862
Train Epoch: 230	Train Loss: 0.013405	Val Loss: 0.012737
Train Epoch: 240	Train Loss: 0.011852	Val Loss: 0.011655
Train Epoch: 250	Train Loss: 0.012133	Val Loss: 0.011558
Train Epoch: 260	Train Loss: 0.010904	Val Loss: 0.010508
Train Epoch: 270	Train Loss: 0.010212	Val Loss: 0.010178
Train Epoch: 280	Train Loss: 0.010260	Val Loss: 0.010015
Train Epoch: 290	Train Loss: 0.010065	Val Loss: 0.010053
Train Epoch: 300	Train Loss: 0.010098	Val Loss: 0.010055
Train Epoch: 310	Train Loss: 0.010164	Val Loss: 0.010055
Train Epoch: 320	Train Loss: 0.010075	Val Loss: 0.010054
Train Epoch: 330	Train Loss: 0.010067	Val Loss: 0.010053

Train Epoch: 340	Train Loss: 0.010053	Val Loss: 0.010052
Train Epoch: 350	Train Loss: 0.010111	Val Loss: 0.010052
Train Epoch: 360	Train Loss: 0.010166	Val Loss: 0.010052
Train Epoch: 370	Train Loss: 0.010050	Val Loss: 0.010049
Train Epoch: 380	Train Loss: 0.010154	Val Loss: 0.010047
Train Epoch: 390	Train Loss: 0.010097	Val Loss: 0.010046
Train Epoch: 400	Train Loss: 0.010101	Val Loss: 0.010044

Table 4. Training and validation results in CT scans dataset

Learning Rate Adjustment:

The ReduceLROnPlateau scheduler is used to decrease the learning rate when the validation loss does not improve for a specified number of epochs. In our case is (patience=5).

At every 10th epoch, the training and validation losses are printed insights for the model's performance and convergence.

4.4.16 Testing

We have split the dataset into 20% test part and 80% training part. The testing part is done to evaluate the accuracy of the classification of unseen data. That means it is a crucial part of our study.

For testing, we have used libraries for the adjustments in the learning rate and to prevent the exploding gradient problem. The learning rate scheduler reduces the learning rate when the validation loss has stopped decreasing. It will wait for 5 consecutive epochs, and if there is no improvement, the learning rate is reduced.

Testing is done over 400 epochs and model. eval() is set. This means the model is set to evaluation mode. No gradient calculation(torch.no_grad) is used to disable gradient calculation. In this way, we reduce memory consumption and increase the speed. Data is moved to the device, which might be CPU or GPU.

Test Loss

Loss calculation is done using BCELoss(Binary Cross Entropy) between y_hat and label. Then we add the result for each batch in the test_epoch list. After loss calculation is done for each batch, we calculate the average test loss.

Validation and Learning Rate Adjustment

The model is set to eval() mode and we calculate validation loss. This helps in detecting overfitting and underfitting. At the end of the for loop, we compute the average validation loss. If validation loss has not improved within several epochs(patience), then the learning rate is reduced.After every 10 epochs, we print test loss, validation loss, and learning rate.

Overall performance Evaluation

To provide an overall model's performance, we created a script. This script calculates the average accuracy, precision, recall, and F1 score. Then, the overall performance is printed.

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1. Performance Comparison of Datasets

The model in our project demonstrates high performance in both types of datasets used. CT and MRI are two different medical imaging techniques, but there might be some factors applied in our model, which have a high impact.

Use of CNN(Convolutional Neural Network), which is good for capturing features of images. In both CT scans and MRIs, edges, shapes, and textures are important, so the use of CNN has a high impact on results achieved.

Also, preprocessing techniques such as resizing, normalization, and augmentation have helped in minimizing differences between CT scans and MRIs. This provides high performance in both types.

5.2. Statistical Illustrations

5.2.1 MRIs Dataset

For graphical illustrations use Accuracy across datasets and Sensitivity and specificity comparison. Here are illustrated MRI cases:

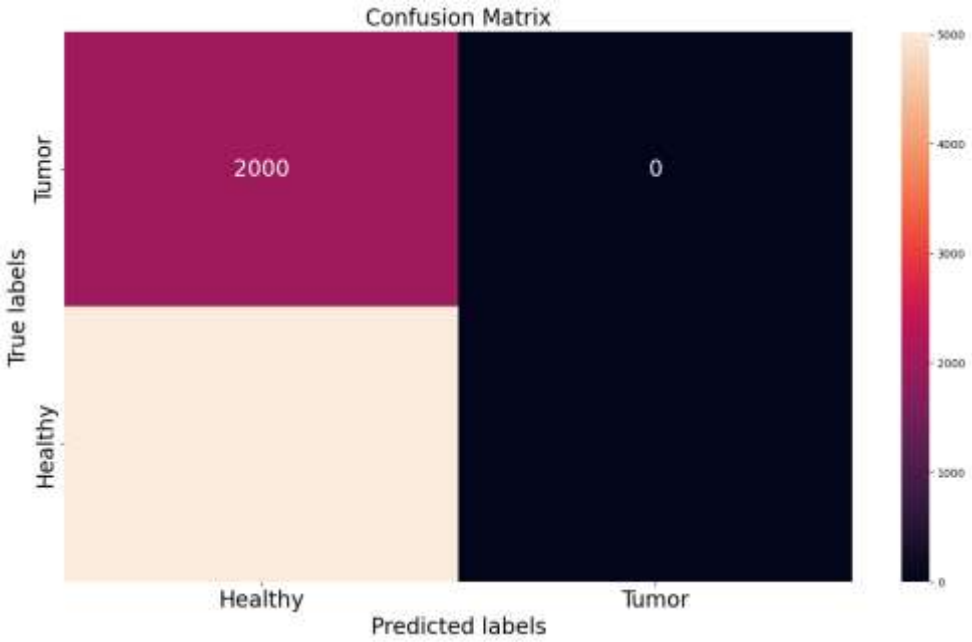

Figure 13 Confusion matrix before Training in MRI

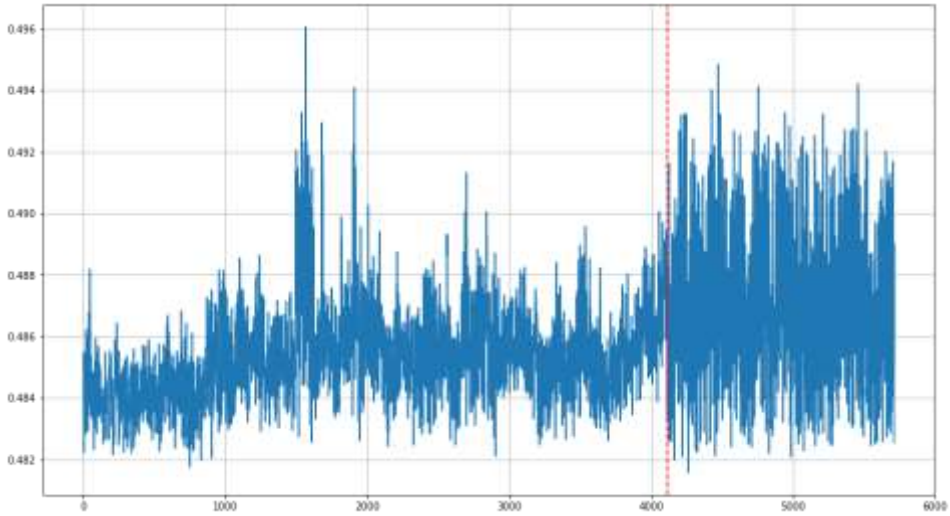

Figure 14 Sensitivity illustration before Training in MRI

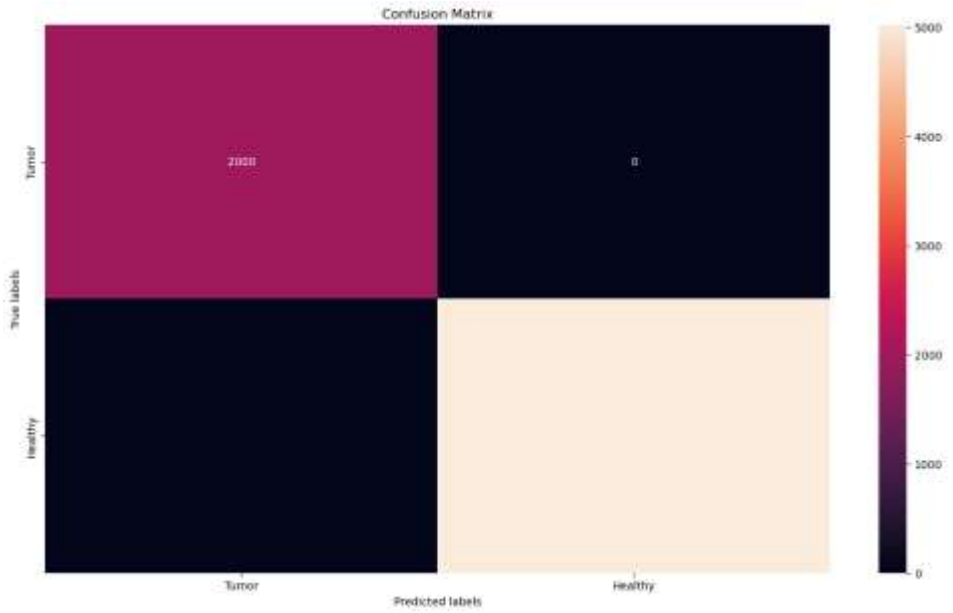

Figure 15 Confusion Matrix Post-Training in MRI

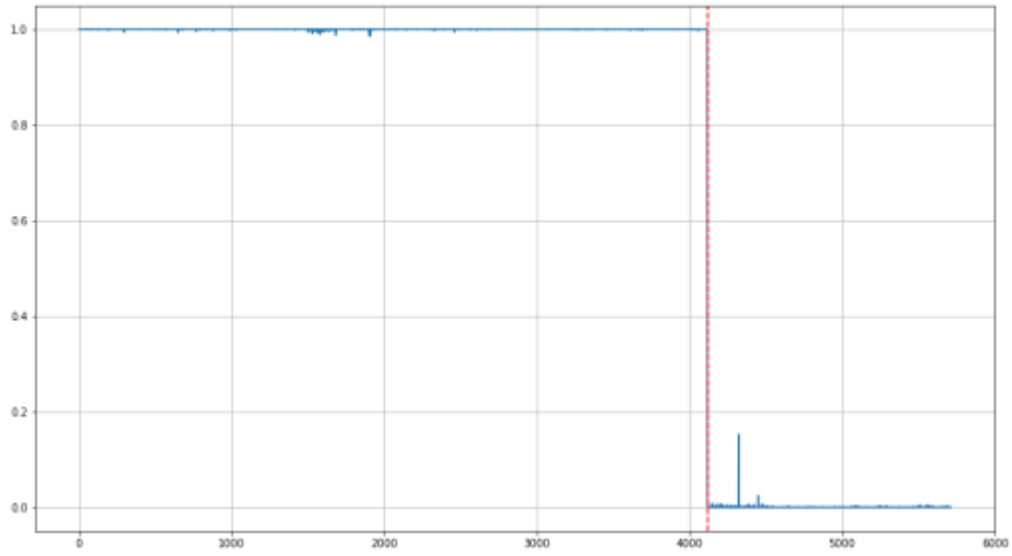

Figure 16 Sensitivity Illustration Post-Training in MRI

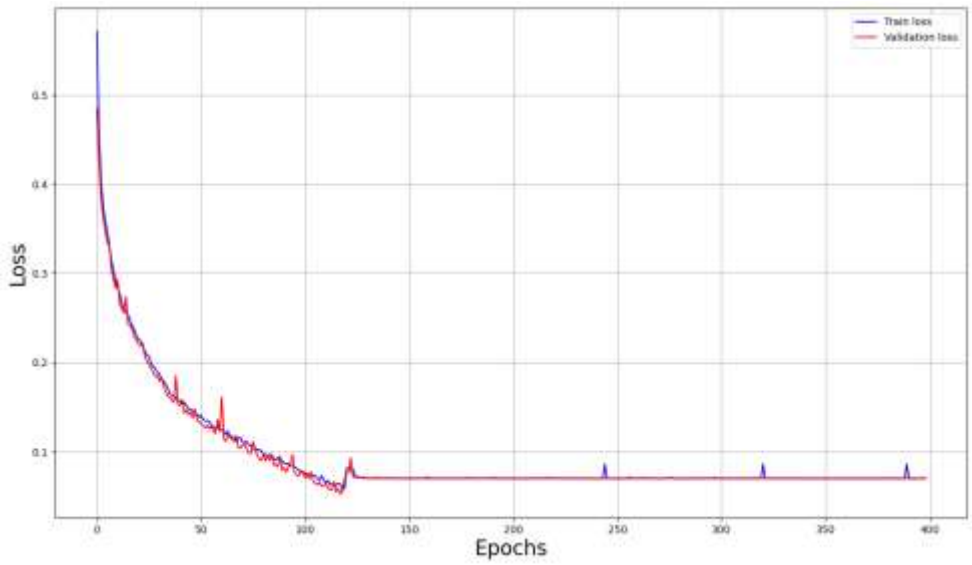

Figure 17 Graph of Model Performance in MRI

This graph provides insight into how well the model performs. The red line stands for validation loss, while the blue for training loss.

The training loss sharply decreases. After epoch 110, it flattens. This means that the model quickly learns to fit the training data.

The validation loss decreases by following training loss, indicating that there is no overfitting. No overfitting provides a well generalizing to the new data.

5.2.2 CT Scans Dataset

These are the visualizations of the results of CT scans of the lung:

CT-scans case: Before training results:

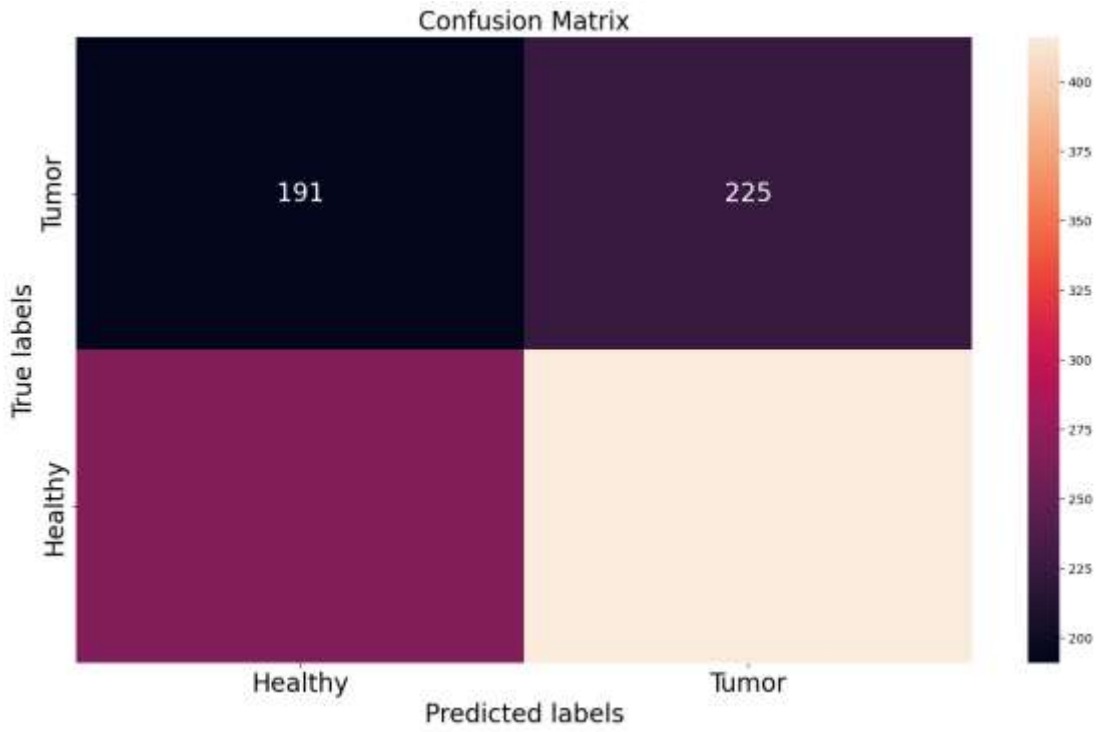

Figure 18 Confusion Matrix before Training in CT Scans

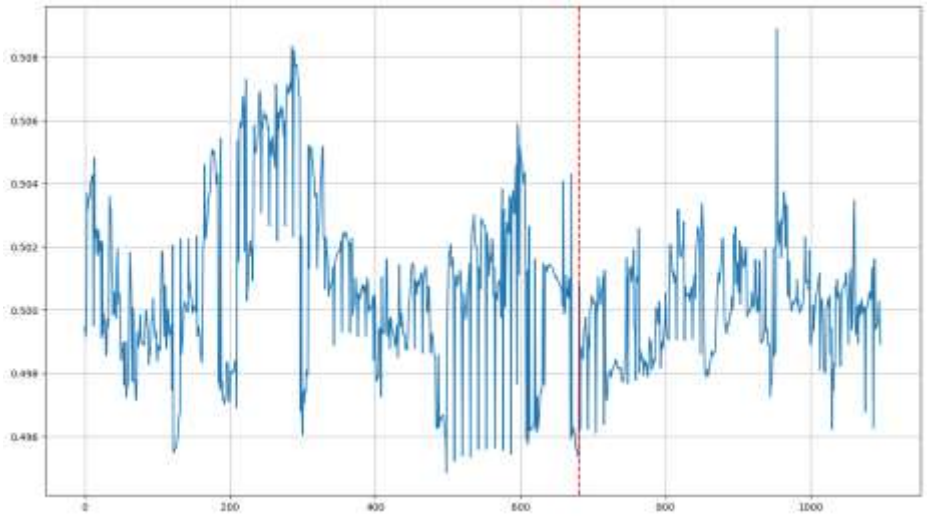

Figure 19 Sensitivity Illustration Post-Training in CT Scans

CT-scans case: Post-Training Results:

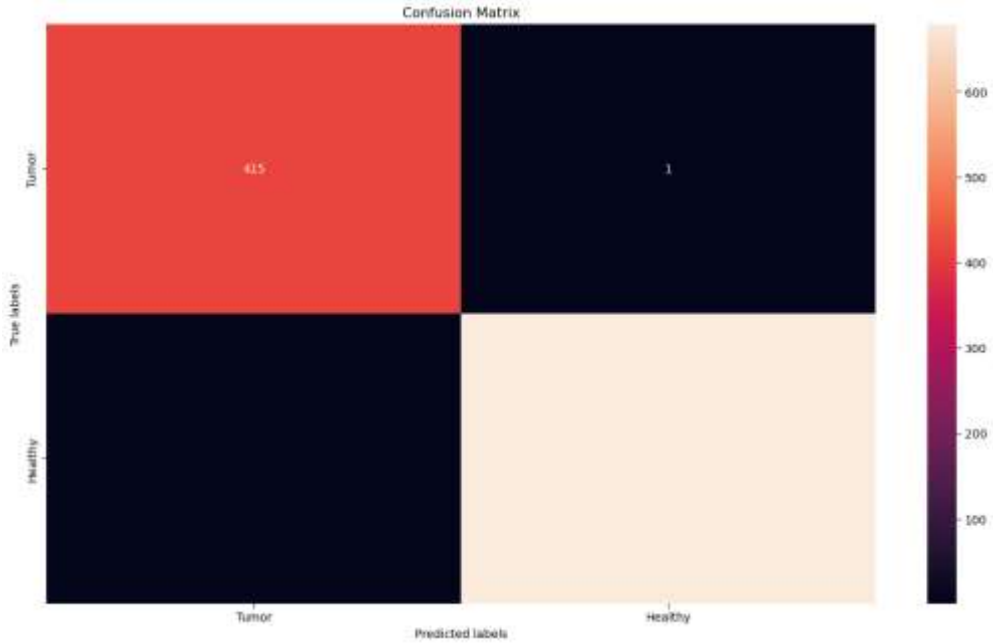

Figure 20 Confusion Matrix Post-Training in CT Scans

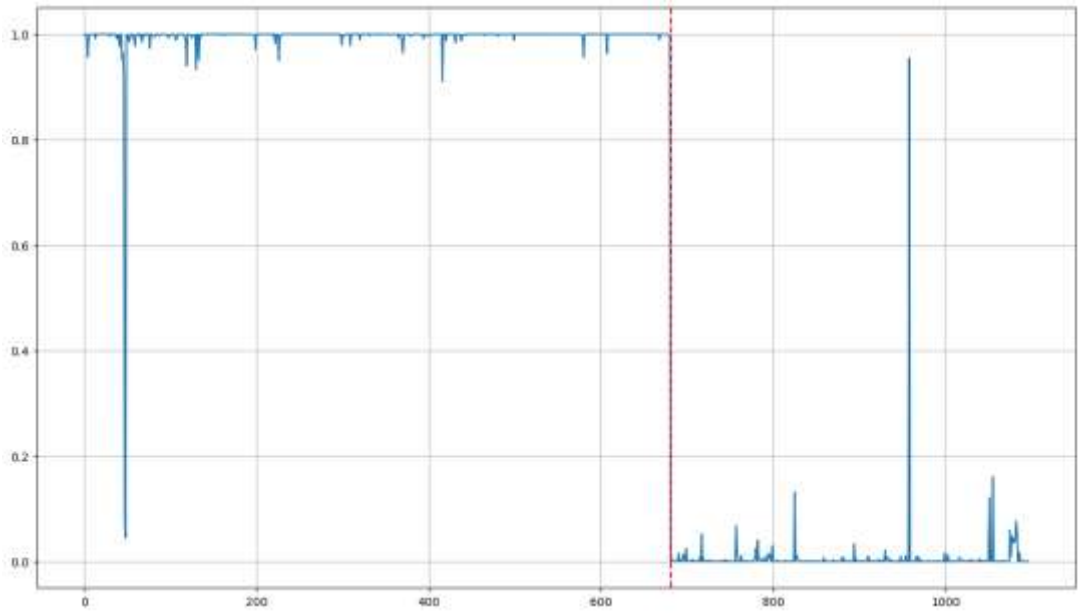

Figure 21 Sensitivity Illustration Post-Training in CT Scans

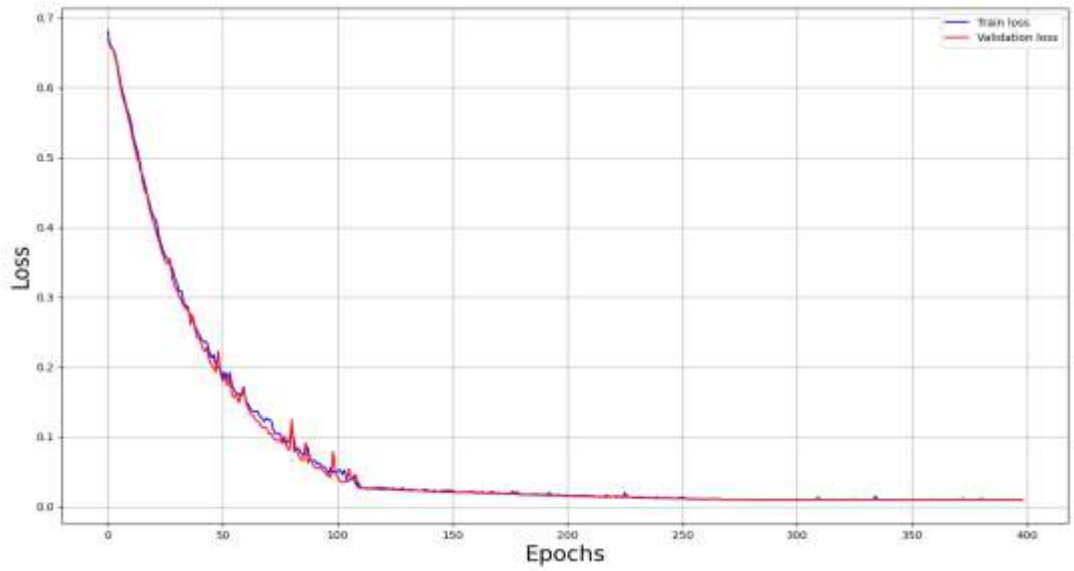

Figure 22 Graph of Model Performance in CT Scans

This graph provides a clear indicator of how well the model performs.

Convergence of Losses: with the increasing of epochs number, blue and red lines decrease significantly. This drastic decrease shows that the model has effectively learned.

Stability and Overfitting: lines remain close to each other, which means the model is not just memorizing. It is also generalizing well to unseen data. No signs of underfitting because loss is low. Also, there is no sign of significant overfitting, since the training loss decreases while the validation loss decreases too.

5.3. Performance Analysis

5.3.1 CT scans

Performance metric	Training	Testing
Accuracy	0.996767	0.995455
Precision	0.998528	0.992806
Recall	0.996261	1.000000
F1-score	0.997393	0.996390

Table 5. Performance metrics results of training and testing in CT scans

Accuracy shows the proportion of true results among all the total numbers examined. For the training phase, the accuracy achieved a result of 99.6767%. For the testing phase, it achieved an accuracy of 99.5455%. In both phases, it proves that the correctness of identifying classes is high.

Precision shows the ratio of only correctly positive predicted over the total predicted positives. That means, that the result of 99.85% in training and 99.28% in testing, is very reliable in predicting. It shows a low false positive rate.

Recall measures the ability of the model to recognize and find cases within the dataset. Our result of 1.00000 in testing, means the model successfully identified all actual positives. Also, the result of 99.6261% in the testing phase, is very high and is a success.

F1-score or the weighted average of precision and recall, shows a result of 99.63% in testing and 99.73% in training. These values indicate a robust balance between precision and recall.

5.3.2. MRIs

Performance metric	Training	Testing
Accuracy	0.989777	0.995018
Precision	0.992681	0.997021
Recall	0.993028	0.996032
F1-score	0.992854	0.996526

Table 6. Performace metrics results of training and testing in MRIs

Accuracy for the training phase achieved a result of 98.9777%. For the testing phase, it achieved an accuracy of 99.5018%. In both phases, it proves that the correctness of identifying classes is high.

Precision shows the ratio of only correctly positive predicted over the total predicted positives. That means, that the result of 99.26% in training and 99.7% in testing, is very reliable in predicting. It shows a low false positive rate.

Recall measures the ability of the model to recognize and find cases within the dataset. Our result of 99.3% in the training phase, means the model successfully identified all actual positives. Also, the result of 99.6% in the testing phase, is very high and is a success.

F1-score or the weighted average of precision and recall, shows a result of 99.65% in testing and 99.28% in training. These values indicate a robust balance between precision and recall.

5.4. Summary of Results

In our study, CNN is used to identify tumors in two different medical imaging, CT and MRI. The model is trained and tested in both CT scans of the lung and MRIs of the brain. This study proved successful in identifying and classifying medical images in tumoral or not, with high accuracy. The accuracy for the detection of brain tumors in MRIs was 99.6767%. For CT lung scans the accuracy for training was 98.97%, while for testing it was 99.55%.

	MRI Dataset	CT scans Dataset
Accuracy(Train Phase)	0.9897	0.996767
Accuracy(Test Phase)	0.995018	0.995455

Table 7. Comparison of accuracy between CT scans and MRIs

CHAPTER 6

CONCLUSIONS

This thesis uses deep learning, specifically the use of Convolutional Neural Networks in medical images, aiming at tumor identification. We have created a model which aids in early diagnosis, for both types of medical images, CT scans, and MRIs. The results demonstrated the high accuracy and effectiveness of deep learning in learning complex images. The model is effective in both MRIs and CT scans. The accuracy for the detection of brain tumors in MRIs was 98.97%. For CT lung scans the accuracy for training was 99.6767%. Also, the model achieved satisfying results in unseen data.

CHAPTER 7

FUTURE WORK

For future work, this study can be extended by incorporating larger and more diverse datasets. Also, exploring datasets collected from demographic and different geographic backgrounds.

REFERENCES

[1] Key Statistics for Lung Cancer \ 2024 \ American Cancer Society

[2] The IQ-OTH/NCCD lung cancer dataset \ HAMDALLA F. AL-YASRIY \ 2020 \ Kaggle

[3] Brain Tumor MRI Dataset \ Msoud Nickparvar \ 2021 \ Kaggle

[4] An Automatic Lung Cancer Detection and Classification (ALCDC) System Using Convolutional Neural Network \ Wadood Abdul\ 2021 \ IEEE

[5] Detection of Lung Nodules using Convolution Neural Network: A Review \ Babu Kumar S, M Vinoth Kumar \ 2020 \ IEEE

[6] Deep Learning Algorithm for Classification and Prediction of Lung Cancer using CT Scan Images \ Diksha Mhaske, Kannan Rajeswari, Ruchita Tekade \ 2020 \ IEEE

[7] Lung Cancer Detection from Computed Tomography (CT) Scans using Convolutional Neural Network \ M.Bikromjit Khumancha, Aarti Barai, C.B Rama Rao \ 2019 \ IEEE

[8]A novel approach for detection of Lung Cancer using Digital Image Processing and Convolution Neural Networks \ Rohit Y. Bhalerao, Harsh P. Jani, Rachana K. Gaitonde, Vinit Raut \ 2019 \ IEEE

[9] Gradient-based learning applied to document recognition \ Y. Lecun, L. Bottou, Y. Bengio, P. Haffner \ 1998 \ IEEE

[11] Brain tumor dataset \ Cheng, Jun \ 2017 \figshare

[12] Brain Tumor Classification \ Sartaj Bhuvaji, Ankita Kadam, Prajakta Bhumkar, Sameer Dedge, and Swati Kanchan \ 2020 \ Kaggle.

[13]Brain Tumor Detec \ Ahmed Hamada \ 2020 \ Kaggle

[14] A Basic Concept of Image Classification for Covid-19 Patients Using Chest CT Scan and Convolutional Neural Network \ Irma Permata Sari, Widodo, Murien Nugraheni, Putra Wanda\2020\ IEEE

[15] Brain Tumor Detection and Classification Using CNN Algorithm and Deep Learning Techniques \ Sultan B. Fayyadh, Abdullahi A.Ibrahim \ 2020 \ IEEE

[16] Highly nonlinear process model using optimal artificial neural network \ Peter Karas, Stefan Kozak \ 2018 \ IEEE

[17] Design and Development of Integrated Deep Convolution Neural Network Approach for Handling Heterogeneous Medical Data\ Arifa J.Shikalgar, Shefali Sonvane\ 2019 \ IEEE

[18] Folding Paper Currency Recognition and Research Based on Convolution Neural Network\ Mengshu JIAO, Jianbiao HE, Baojiang Zhang \2018 \ IEEE