

A Report on

# **Brain Disease Detection Using Image Processing and Neural Networks**

for

**Major Project (REV- 2019‘C’ Scheme) of Fourth Year, (BE Sem-VII)**

in

**Electronics and Telecommunication Engineering**

by

Rohit Maurya (3020129)  
Shashishekhar Singh (3020101)  
Dayesh Dongre (3020110)  
Pranav Kavilkar (3020123)

Under the guidance of

**Ms. Asmita Chavan**



**Fr. C. Rodrigues Institute of Technology, Vashi**



**UNIVERSITY OF MUMBAI**

**AY 2023-24**

# CERTIFICATE

This is to certify that the half-year project entitled is a **Brain Disease Detection Using Image processing and Neural Networks** bonafide work of

Rohit Maurya (3020129)  
Shashishekhar Singh (3020101)  
Dayesh Dongre (3020110)  
Pranav Kavilkar (3020123)

submitted to the University of Mumbai in partial fulfilment of the requirement for the award confirming completion of **Major Project (REV- 2019 'C' Scheme) of Fourth Year, (BE Sem-VII) in Electronics & Telecommunication Engineering** as laid down by **University of Mumbai** during the of academic year **2023-24**.

(\_\_\_\_\_)

**Examiner/Reviewer-1**

**(Name & Signature with Date)**

(\_\_\_\_\_)

**Examiner/ Reviewer -2**

**(Name & Signature with Date)**

**Name & Signature of Guide**

**Head of Department**

## **Declaration**

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Rohit Maurya (3020129) Signature: -----

Shashishekhar Singh (3020101) Signature: -----

Dayesh Dongre (3020110) Signature: -----

Pranav Kavilkar (3020123) Signature: -----

Date: 27/10/2023

## Abstract

Brain tumors are abnormal cell growths in the brain that can become life-threatening if not detected early. Manual detection of tumors through analysis of MRI scans is an arduous process prone to human error and variability. Automated artificial intelligence systems can help address this by accurately analyzing MRI scans for early diagnosis. This project develops a deep convolutional neural network (CNN) model for automated brain tumor detection from MRI images.

The key motivation is to create an AI system that can assist doctors in early diagnosis through accurate tumor detection from MRI scans. This can improve treatment outcomes and survival rates. The project utilizes a dataset of 3000 MRI scans, with 1500 tumor cases and 1500 healthy scans, resized to 64x64 pixels. Data augmentation expands the training data. A 5-layer CNN architecture is designed in Keras/TensorFlow, with 3 convolution blocks for feature extraction from MRI images and 2 dense layers for classification.

The model is trained for 10 epochs with categorical cross-entropy loss and Adam optimizer. An 80:20 train-test split of data is used. Tumor vs non-tumor MRI scans are classified with 97.87% test accuracy, outperforming traditional machine learning models like SVM. The high 98.96% training accuracy demonstrates good generalization ability. The project proves deep CNNs can learn robust features from medical images for accurate classification.

Cerebral edema is a critical medical condition characterized by the accumulation of fluid in brain tissues, leading to swelling and increased pressure inside the skull. This condition can be caused by various factors such as trauma, stroke, tumor, or acute hypertension, and can result in symptoms like headache, nausea, vomiting, seizures, or even coma in severe cases. Prompt diagnosis and treatment are crucial to prevent complications and mitigate brain damage, typically involving addressing the underlying cause and managing intracranial pressure.

This research paper presents a comprehensive comparative analysis of various CNN architectures, including VGG16, ResNet50, Inception-v3, DenseNet121, EfficientNet, and a CNN proposed by the research team. The study aims to discern the most effective model for accurate and timely diagnosis of cerebral edema by evaluating the performance of these CNN architectures on medical imaging data.

In summary, this project successfully demonstrates highly accurate brain tumor detection from MRI scans using deep CNNs. The results highlight the promise of AI in healthcare for automated analysis of medical images to assist doctors in early diagnosis and improved patient outcomes.

# Contents

<b>Abstract</b>	i
<b>Contents</b>	ii
<b>List of figures</b>	iv
<b>List of symbols and abbreviations</b>	v

## Sections

<b>1 INTRODUCTION</b>	1
1.1 Problem overview	1
1.2 Major-project objective	1
1.3 Report outline	2
<b>2 LITERATURE REVIEW</b>	3
2.1 Introduction	3
2.2 Overview of Existing Methods of Tumour Detection and Their Limitations	3
2.3 Different Types of Brain Disease Detection from MRI Scans	4
2.4 Drawbacks of traditional tumour detection methods over computer-based methods	6
2.5 Steps taken for improvement	7
2.6	7
<b>3 MAJOR-PROJECT DESIGN &amp; WORKING PRINCIPLE</b>	9
3.1 Block diagram	9
3.2 Description of working principle	9

3.3 Libraries used in the implementation and their use	10
<b>4 MAJOR-PROJECT IMPLEMENTATION</b>	11
4.1 Introduction	11
4.2 Software implementation	11
4.3 Algorithm and Flow Chart	12
4.4 Troubleshooting	16
4.5 Code	16
4.6 Photos of final project	19
4.7 Summary	20
<b>5 RESULTS AND DISCUSSION</b>	
5.1 Introduction	21
5.2 Results obtained from software	22
<b>6 SUMMARY AND CONCLUSIONS</b>	
6.1 Summary of the major-project	23
6.2 Conclusions	23
6.3 Learning outcomes	24
 <b>Appendices</b>	
<b>A PO-CA-PI-CO MAPPING WITH REMARKS</b>	25
<b>B Data sheets</b>	39
<b>REFERENCES</b>	54
<b>ACKNOWLEDGEMENTS</b>	55

## List of figures

Fig.3.1 Block diagram of the project	9
Fig 4.1 Importing libraries & Image Preprocessing	13
Fig. 4.2 Design model and Add Layers	13
Fig.4.3 Training and Testing	14
Fig.4.4 Final output of the project (Flask Web App)	19
Fig 5.1 Input image to the model.	21
Fig 5.2 Output observed in text format	22

# **Section 1**

## **Introduction**

### **1.1 Problem overview**

Brain tumors refer to abnormal cell growth that arises in the brain and can spread to other parts of the central nervous system. Brain tumors are a leading cause of cancer deaths, with around 140,000 people in the US being diagnosed every year. Manual detection of brain tumors early on through analysis of MRI scans is a very challenging task.

Doctors have to carefully examine multiple MRI image slices in 3D to identify any abnormalities or lesions that could indicate presence of a tumor. This manual screening relies heavily on the doctor's specialized expertise in analyzing radiology images, and is prone to issues like human fatigue, inconsistency, and misdiagnosis. As a result, many brain tumors go undetected or are diagnosed quite late, severely impacting treatment outcomes and survival rates.

There is a need for an automated AI-based screening system that can rapidly analyze MRI scans with high accuracy to assist doctors in early diagnosis of brain tumors. Recent advances in deep learning and computer vision provide promising techniques to address this problem. Convolutional neural networks (CNNs) in particular have shown excellent performance on visual recognition tasks.

This project aims to develop a deep learning model using CNN architecture for automated brain tumor detection from MRI images. The model will be trained on a dataset of MRI scans of both normal brains and brains with tumors. If high accuracy can be achieved, this AI model can aid radiologists by acting as a preliminary automated screening system for brain tumor diagnosis. The technology can improve outcomes through early detection, make diagnosis faster, more consistent and accessible.

### **1.2 Major-project objective**

Accurate detection and diagnosis of brain tumours from medical images using AI can greatly assist clinicians and improve patient outcomes. The key objectives of this project are: Developing a convolutional neural network (CNN) model that can accurately classify brain scans as normal or abnormal. The CNN architecture will leverage techniques like convolutional layers and pooling to extract relevant imaging features. Enhancing the model to categorize abnormalities into different tumour types like benign, malignant, glioma etc.



This provides crucial information for targeted treatment. Applying advanced data augmentation through transformations like rotation, flip, zoom etc to increase dataset size and improve the network's robustness. Appropriate pre-processing like skull-stripping will also refine target information. Ensuring seamless integration into clinical workflows through techniques like transfer learning from models pre-trained on medical data. Additional clinician inputs will be incorporated to match real diagnostic settings. Extensive evaluation of model performance by testing on large annotated datasets from hospitals and comparing classification accuracy against experienced radiologists. This will validate real-world efficacy. Meeting these objectives will result in an AI system that can catch tumours early through automated screening as a second reader, provide insights into tumour characteristics, and aid clinicians making faster and more accurate diagnosis overall. The project aims to demonstrate the immense benefits which AI-enabled medical imaging can deliver in life-critical diagnosis like brain tumours.

### **1.3 Report outline**

Section 2 provides a review of literature survey of the project. References of the previous projects, limitations of traditional tumor detection methods are presented. Types of brain disease detection methods are explained and advantages of CNN is elaborated.

Section 3 presents Block diagram of the project. The Python code was developed on Visual Studio. Description of the working principle of system. Entire functioning of the system is explained. All the libraries, dataset and software required in the project is specified in this section.

Section 4, provides implementation of the project. The implementation of the neural network is discussed. The whole process related to developing the codes for CNN model and respective algorithms. Also setting up the Flask app implemented. This section contains the final photographs of the project.

Section 5 brings the results and its further discussion to attention. This section contains the comparative study of the actual result of hardware and software.

Section 6, the last section, gives a summary of the investigations, conclusions drawn from the results, and some suggestions for further work.

## **Section 2**

### **Literature Survey**

#### **2.1 Introduction**

Brain tumours refer to abnormal cell growth in the brain that can turn fatal if not detected early. However, current detection methods using MRI scans are slow, inaccurate and fail to diagnose brain tumours early enough. This literature review analyzes prior research on automating brain tumour detection using image processing and neural network techniques for faster and more accurate diagnosis. The key goal is to develop an AI system that leverages computer vision and deep learning algorithms to automatically analyze brain scans with high accuracy to identify tumours. A review of studies on machine learning techniques like convolutional neural networks shows promise in classifying medical visual data and identifying tumours from MRI scans. Key research has focused on training models on datasets of labelled MRI images of both healthy brains and brains with tumours. The models learn to identify visual patterns in scans that indicate presence of tumours. The review summarizes common methodology of training and testing models on brain MRI datasets and using performance metrics like accuracy, sensitivity etc for analysis. The review provides insights into AI techniques, opportunities and challenges in automating analysis of brain scans for enhanced tumour detection.

#### **2.2 Overview of Existing Methods of Tumour Detection and Their Limitations**

Detection and diagnosis of brain abnormalities like tumours traditionally relies on methods that do not utilize automated image processing or machine learning techniques. These conventional clinical approaches for identifying brain diseases include:

**Physical Neurological Exams:** Doctors check for any indications of brain or nervous system problems through a physical examination of nervous system functions. This involves testing reflexes, muscle strength, coordination, and balance. Abnormalities may indicate issues like brain tumours, infection or injury.

**Medical History Analysis:** Patient medical history is reviewed to check for any underlying conditions, medications, or risk factors that could be associated with a brain disorder. Family history of brain-related conditions is also considered.

**Laboratory Tests:** Blood and urine tests check for signs of infection, inflammation or other abnormalities. Certain substances in the blood may act as biomarkers or indicators for brain tumours.

**Cognitive Testing:** Formal assessment of cognitive abilities through specialized pen-and-paper tests can identify any deficits in memory, language, attention etc. This may detect developmental issues or degeneration related to brain disease.

**CT/MRI Scans:** CT or MRI scans of the brain provide detailed structural views for detecting tumours, bleeding, swelling or skull damage. But diagnosing scans depends heavily on manual analysis by radiologists.

While these conventional approaches provide clinical context, they have limited ability to directly detect brain abnormalities early, often requiring patients to already demonstrate symptoms. Emerging AI techniques for automated analysis of brain scans can act as decision support tools for radiologists and assist in preliminary screening for brain disorders. This complements conventional methods and improves overall diagnostic accuracy.

## **2.3 Different Types of Brain Disease detection from MRI Scans**

### **MRI Scans for Brain Tumours and Related Conditions**

- MRI (magnetic resonance imaging) is one of the main imaging techniques used to detect and characterize brain tumours. It provides excellent soft tissue contrast allowing for evaluation of the tumour location, size, morphology, and relationship to surrounding structures.
- Contrast-enhanced MRI is frequently performed by injecting a paramagnetic contrast agent intravenously. This improves visualization of tumours, Edema, necrosis by enhancing areas with disrupted blood-brain barrier.

### **Tumour Detection and Characterization**

- T1-weighted images allow clear delineation between grey and white matter. Tumours generally appear hypointense or isointense relative to grey matter on T1-weighted images.
- T2-weighted and FLAIR images show tumours to be hyperintense relative to grey matter. This sequence is useful to determine tumour boundaries and the presence of surrounding Edema.

- Contrast-enhanced T1-weighted images are useful to identify regions of blood-brain barrier breakdown indicating presence of tumour infiltration. Shape, size and enhancement pattern helps characterize tumour grade and type.
- Advanced MRI techniques like perfusion-weighted imaging, diffusion-weighted imaging and MR spectroscopy provide additional information about tumour cellularity, vascularity, necrosis and metabolic profile.

#### Edema Detection

- Vasogenic Edema surrounding tumours appears hyperintense on T2-weighted/FLAIR images. It may indicate an infiltrative tumour margin or compression of surrounding brain.
- Non-enhancing T2/FLAIR hyperintensity on MRI around tumours represents Edema. The pattern and extent of Edema helps assess tumour grade and infiltration.

#### Necrosis Detection

- Necrosis appears hypointense on contrast-enhanced T1-weighted images, and hyperintense on T2-weighted/FLAIR images.
- Central non-enhancing areas within an enhancing mass indicate necrotic foci. The presence of necrosis suggests higher grade aggressive tumours.
- Diffusion restriction in necrotic regions differentiates it from cystic/fluid-filled areas which do not restrict diffusion.

#### Advanced Computer-aided Detection and Diagnosis

- Machine learning and deep learning techniques are being applied for automated brain tumour detection and segmentation from MRI scans.
- Radiomics evaluates quantitative feature data extracted from standard MRI sequences to predict tumour genotypes and clinical outcomes.
- Multiparametric MRI and integrated PET-MRI along with advanced image analytics and radiomics can improve non-invasive tumour characterization, treatment response assessment and prognostics.

## 2.4 Drawbacks of traditional tumour detection methods over computer-based methods

Traditional tumour detection methods have various drawbacks over the computer-based techniques due to the manual nature of detection and the lack of speed. Few of the drawbacks are explained below

Manual inspection - Traditional analysis relies heavily on visual inspection by radiologists. This is time-consuming, subjective, and prone to human error or fatigue. Computer-aided detection can perform exhaustive analysis without tiring.

Qualitative analysis - Traditional descriptive analysis of tumours is qualitative and limited in parameters assessed. Advanced image processing extracts precise quantitative tumour features like texture, shape, kinetics.

Limited field of view - Humans viewing 2D scans may miss information outside focal lesions. Image processing can evaluate whole volumes at once.

Difficulty with complexity - Complex appearances, diffuse boundaries and heterogeneity within tumours can make distinction challenging. Neural networks can discern complex nonlinear patterns.

Minimal integration of data - Traditional workflow considers limited scan sequences individually. Multi-parametric analysis by algorithms integrates information from various protocols.

Interobserver variability - Varying expertise and subjective judgement between radiologists analysing the same case leads to inconsistent results. Computerized analysis is objective and replicable.

Slow workflow - Conventional sequential imaging and analysis is slow. Automated computational analysis offers near real-time processing.

Limited analytics - Qualitative radiological assessments have limited scope for advanced analytics. Machine learning techniques enable radiomics, predictive models and decision support.

In conclusion, traditional visual and qualitative analysis is limited in accuracy, consistency and scalability compared to automated, quantitative and integrative analysis using image processing and machine learning. Wider availability of multi-modal medical imaging data along with advances in AI will enable more precise and personalized tumour characterization.

## 2.5 Steps taken for improvement

Many distinct steps are taken in implementing deep learning that help overcome drawbacks of traditional tumour detection methods:

### 1. Automated feature extraction

The code uses convolutional neural network layers like Conv2D and MaxPool2D to automatically extract relevant features like edges, shapes and textures from the input MRI images. This provides quantitative analysis instead of subjective qualitative assessment.

### 2. Integrated multi-parametric analysis

The model takes the full MRI images as input and integrates information across all pixels and channels. This allows evaluating the whole volume at once rather than a limited field of view.

### 3. Objectivity and consistency

The model performs standardized algorithmic analysis on each input image without human subjectivity or interobserver variability. This improves consistency and removes errors due to radiologist fatigue.

### 4. Advanced analytics

By training on labelled sample data, the model learns complex nonlinear relationships between MRI inputs and tumour outcomes. This enables personalized predictive modelling and decision support above simplistic thresholds.

The deep learning approach in the project automates feature extraction, integrates multiparametric data, provides consistent objective analysis, and enables advanced analytics to overcome limitations in traditional manual inspection of medical images for tumour diagnosis.

## **2.6 Overview of work done on Cerebral Edema Detection in Medical Imaging**

### 1. Dual Parameter Synchronous Monitoring System of Brain Edema:

The research paper introduces a Dual Parameter Synchronous Monitoring System that utilizes the reflection and transmission characteristics of a two-port test network to monitor brain edema. The system includes a coil sensor, signal generation source, signal separation module, amplitude and phase receiver, and data processor. Experiments on rabbits with induced brain edema collected data on the reflected phase shift spectrum (RPSS) and transmitted phase shift spectrum (TPSS) over 24 hours. Statistical analyses assessed the system's performance in monitoring edema progression. By considering both RPSS and TPSS, the system aims for improved sensitivity and accuracy in detecting and classifying edema severity compared to using either parameter alone. The findings suggest this system's potential for real-time, noninvasive monitoring of brain edema progression, enhancing clinical management.

## 2. Exploring the Role of Diffusion-Weighted Imaging in Cerebral Edema Detection:

This study investigates the utility of Diffusion-Weighted Imaging (DWI) in detecting and characterizing cerebral edema. DWI measures water molecule motion within tissues, providing insights into microstructure and pathology. The paper reviews literature on using DWI for cerebral edema detection and presents findings from patient studies. Results indicate DWI can effectively detect and quantify cerebral edema by assessing tissue water diffusion changes. The apparent diffusion coefficient (ADC), a DWI parameter measuring water diffusion, has been correlated with edema presence, severity, clinical outcomes, and treatment response. By leveraging its sensitivity to water diffusion patterns, DWI demonstrates potential as a valuable tool in the clinical management of cerebral edema, aiding early detection, quantifying severity, and informing treatment strategies.

## 2.7 Previous Studies on CNN Architectures for Medical Image Analysis

1. Segmentation of RoI in Medical Images Using CNN- A Comparative Study <sup>[11]</sup>: The paper "Segmentation of RoI in Medical Images Using CNN- A Comparative Study" explores the application of Convolutional Neural Networks (CNN) for automatic segmentation of Region of Interest (RoI) in medical images, specifically carotid artery ultrasound images. By comparing CNN with traditional machine learning algorithms like Support Vector Machine (SVM) and Radial Basis Function (RBF), the study demonstrates CNN's superior performance with an accuracy of 99.3% using 10-fold cross-validation. The proposed CNN architecture includes multiple convolutions and fully connected layers, trained on a dataset of 1100 RoI and 1500 RoNI images. Pre-processing techniques and balanced dataset training contribute to the CNN model's effectiveness in accurately segmenting RoI, showcasing its potential for improving diagnostic accuracy and efficiency in medical image analysis.

2. Deep Learning-Based Classification of Breast Cancer Histopathological Images: This research investigates the use of deep learning, specifically Convolutional Neural Networks (CNNs), for the classification of breast cancer histopathological images. The study employs a CNN architecture trained on a large dataset of histopathological images obtained from breast cancer patients. Through extensive experimentation and evaluation, the CNN model achieves high accuracy in classifying cancerous and non-cancerous regions within histopathological slides. The results demonstrate the potential of CNN-based approaches for automated cancer diagnosis and highlight the importance of leveraging deep learning techniques for improving the efficiency and accuracy of histopathological image analysis in clinical settings.

## 2.8 Summary

This literature review analyzes research on using image processing and neural networks to automate brain tumour detection from MRI scans for faster and more accurate diagnosis. Traditional methods like physical exams, medical history analysis, and manual inspection of scans are limited in their ability to directly detect brain abnormalities early. They are prone to subjectivity, human error, and lack advanced analytics. MRI provides excellent soft tissue contrast for visualizing tumours, surrounding Edema, necrosis, and distortion of brain structures. But radiologist interpretation of scans is qualitative, inconsistent, and restricted to focal regions visible to the naked eye.

Emerging AI techniques apply automated extraction of quantitative imaging features, integrated analysis of multiparametric data, and machine learning algorithms to classify tumours from MRI images. Convolutional neural networks can discern complex nonlinear patterns in pixel data to identify visual indicators of different tumour properties. By training models on labelled datasets of healthy and diseased scans, deep learning systems can analyse MRI images for tumours with greater speed, accuracy, and reliability. They act as decision support tools by overcoming human limitations like fatigue and narrow focus. The automated, comprehensive analytical capabilities enhance early screening, diagnosis, and treatment planning for brain abnormalities.



## Section 3

### MAJOR-PROJECT DESIGN & WORKING PRINCIPLE

#### 3.1 Block Diagram for tumour detection

From the block diagram shown in Fig 3.1, we can understand the flow of the project implementation. The key steps include the classification of dataset, preprocessing of images, building the model, training the model, testing the model and the prediction of the model.

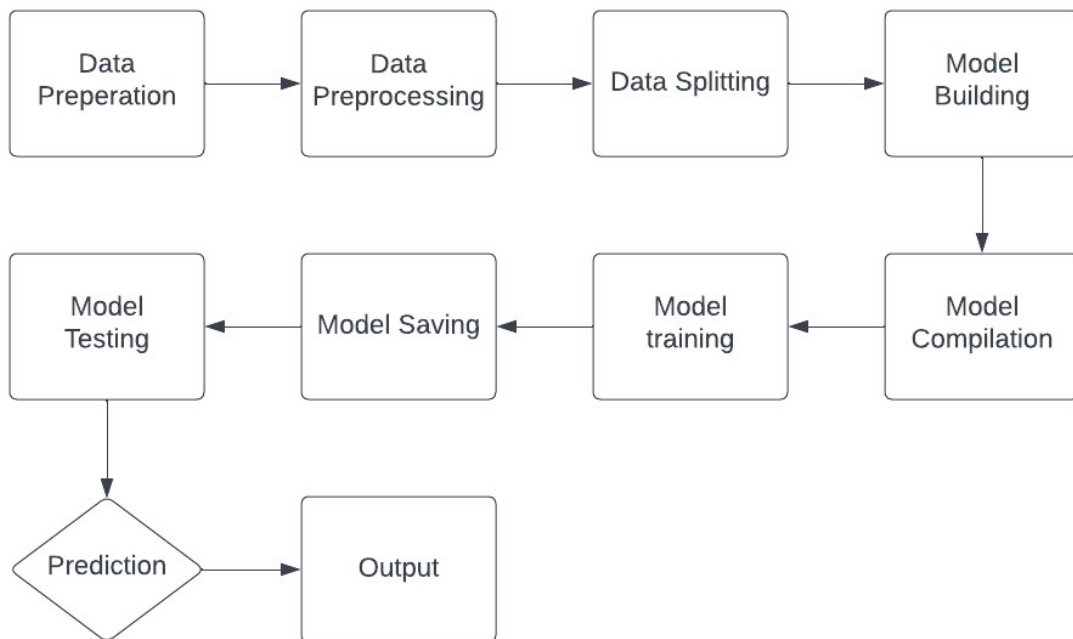


Fig.3.1 Block diagram of the project

#### 3.2 Description of working principle.

Description of working principle. The functioning of the project may be understood with the following points

1. Data Preparation: Load images of brain tumour patients and assign labels.
2. Data Preprocessing: Resize and normalize images.
3. Data Splitting: Split data into training and testing sets.
4. Model Building: Create a convolutional neural network (CNN) model for image classification.

5. Model Compilation: Define the loss function, optimizer, and evaluation metric for the model.
6. Model Training: Train the model on the training data for a specified number of epochs.
7. Model Saving: Save the trained model to a file for later use.
8. Prediction: Load the model, preprocess an input image, and make a prediction.
9. Output: Display the probability score for the presence of a brain tumour.

### **3.3 Libraries used in the implementation and their use**

The code utilizes several important libraries for various tasks. Here are the important libraries used in the code:

1. cv2 (OpenCV): OpenCV is used for image processing, including reading and manipulating image data.
2. os: The `os` library is used for interacting with the operating system, including listing files in directories.
3. tensorflow (tf): TensorFlow is used for creating and training machine learning models, specifically Keras, which is a high-level API included in TensorFlow.
4. keras: Keras is a high-level deep learning library that runs on top of TensorFlow. It is used for building and training neural network models.
5. PIL (Pillow): The Python Imaging Library (Pillow) is used for image manipulation, including resizing images.
6. numpy (np): NumPy is used for numerical operations and manipulating arrays.
7. sklearn (scikit-learn): Scikit-learn is used for machine learning tasks, such as splitting the dataset into training and testing sets.

These libraries are essential for tasks like image processing, model creation, data manipulation, and training and testing machine learning models.

### **3.4 Overview of CNN Architectures for edema detection**

The architectures of the models being compared:

1. For the CNN model developed by the team, a custom architecture was designed and implemented using TensorFlow and Keras libraries. The model includes convolutional layers followed by max-pooling layers for feature extraction and fully connected layers for classification. A kernel size of 3x3 is chosen to function as a mask for the learning process.

2. ResNet50: ResNet50 is a deep convolutional neural network architecture that introduces residual connections. It comprises 50 layers and is known for its ability to address the vanishing gradient problem, enabling the training of very deep networks.
3. VGG16: VGG16 is a classic convolutional neural network architecture with 16 layers. It is characterized by its simplicity, consisting of multiple convolutional layers with small receptive fields (3x3), followed by max-pooling layers.
4. InceptionV3: InceptionV3 is designed with inception modules, allowing for the parallel processing of image features at different scales. This architecture aims to capture both local and global information effectively, enhancing feature representation.
5. DenseNet121: DenseNet121 employs densely connected layers, where each layer is connected to every other layer in a feed-forward manner. This architecture encourages feature reuse and facilitates gradient flow throughout the network, leading to efficient parameter utilization.
6. EfficientNet: EfficientNet is a family of convolutional neural network architectures that optimize both model accuracy and computational efficiency. It achieves this by scaling network depth, width, and resolution with a compound scaling method, ensuring better performance with fewer parameters.

### **3.5 Dataset Description and Learning Process**

The dataset comprises MRI images of brain scans, categorized into two classes: "No Edema" and "Edema Present." Each class contains images representing different instances of brain scans, with "No Edema" indicating the absence of cerebral edema and "Edema Present" indicating the presence of cerebral edema. These images serve as the input data for training the Convolutional Neural Network (CNN) models.

Learning Process:

1. Data Representation: The CNN models learn to identify edema in the MRI images through a process of feature extraction and classification. Initially, the raw pixel values of the MRI images serve as input to the CNN models.
2. Feature Extraction: During training, the CNN models automatically extract relevant features from the input images. The convolutional layers of the CNN learn to detect patterns and structures within the images, such as edges, textures, and shapes, that are indicative of cerebral edema.
3. Feature Hierarchies: As the CNN models progress through the layers, they build hierarchical representations of the input data. Lower layers capture basic features like edges and textures, while higher layers capture more abstract and complex features that are characteristic of cerebral edema.
4. Non-linear Transformations: The activation functions applied after each convolutional layer introduce non-linear transformations to the learned features, enabling the models to capture complex relationships between different image regions.

5. **Classification:** After feature extraction, the flattened feature maps are fed into fully connected layers, which learn to classify the input images into the appropriate classes ("No Edema" or "Edema Present"). The final SoftMax layer assigns probabilities to each class, indicating the likelihood of edema presence in the input image.
6. **Loss Optimization:** During training, the CNN models optimize their parameters (weights and biases) to minimize the discrepancy between the predicted class probabilities and the ground truth labels. This is achieved through backpropagation and gradient descent, where the gradients of the loss function with respect to the model parameters are computed and used to update the parameters iteratively.
7. **Model Evaluation:** The trained CNN models are evaluated on a separate validation set to assess their performance in identifying cerebral edema. Metrics such as accuracy, precision, recall, F1 score, specificity, and dice coefficient are computed to quantify the models' performance and ensure their effectiveness in clinical applications.

## Section 4

### Major-Project Implementation

#### 4.1 Introduction

The following section will take through each step of implementation of the project and its outcome at the end of each step. This section contains the implementation of the project in software, and various problems encountering during the development of the major project.

#### 4.2 Software implementation

**Image Preprocessing:** Image preprocessing is a fundamental step to ensure that medical images are optimized for analysis. This process involves several important sub-steps, including noise reduction to eliminate unwanted artifacts, contrast enhancement to make structures more visible, and image normalization to standardize pixel values. Additionally, registration techniques may be used to align images from different modalities or time points for consistency. Successful image preprocessing is crucial because it enhances the quality and reliability of the data that the neural network will analyze.

**Feature Extraction:** Feature extraction plays a pivotal role in identifying relevant patterns within the medical images. These patterns are often critical for disease detection. The software employs various image processing techniques to extract discriminative features, such as textures, shapes, and intensity variations. For example, textural features might capture irregularities in tissue patterns, while shape features could be used to detect abnormalities in brain structures. Extracted features serve as essential inputs to the neural network, enabling it to make informed decisions about the presence and type of brain diseases.

**Convolutional Neural Networks (CNNs):** CNNs are the heart of the software implementation. They are specifically designed to analyze the preprocessed medical images and make predictions. CNNs consist of layers that learn hierarchical features, which are then used for classification or detection tasks. Model architecture and hyperparameters are carefully chosen, and the network is trained with a labeled dataset of medical images. Training involves adjusting the model's internal parameters to minimize classification errors. Depending on the specific context, transfer learning techniques may be employed by starting with pre-trained models and fine-tuning them for the specific brain disease detection task. The software leverages the power of CNNs for their proven effectiveness in image-based classification tasks.

**Training Data:** Training the neural network requires a substantial dataset of medical images and corresponding labels that indicate the presence of brain diseases and their characteristics. The training dataset is a critical component, as the neural network learns from these examples to recognize patterns and relationships within the data. Building a comprehensive and representative training dataset is often a significant effort, necessitating collaboration between healthcare institutions and data annotators to ensure accuracy.

## 4.3 Algorithm and Flow chart

### Algorithm:

**Part A]** Fig 4.1 and 4.2 indicate the preprocessing and model design for tumor detection

1. Import required python libraries - cv2, os, tensorflow, PIL, numpy, sklearn, keras
2. Set image directory path and get lists of no\_tumor and yes\_tumor image filenames
3. Initialize empty lists to store dataset and labels
4. Loop through no\_tumor image names
  - Read image as RGB using cv2
  - Resize to 64x64 using PIL
  - Append image numpy array to dataset
  - Append label 0 to label list
5. Repeat step 4 for yes\_tumor images, appending label 1
6. Convert dataset and labels to numpy arrays
7. Split dataset and labels into 80-20 train-test split using sklearn
8. Normalize pixel values and one-hot encode labels
9. Define convolutional neural network model in keras -
  - Conv2D layers for feature extraction
  - Maxpooling for downsampling
  - Flattening and Dense layers for classification
  - Softmax activation for binary classification
  - Categorical cross-entropy loss
  - Adam optimizer
10. Compile and fit model on training data for 10 epochs
11. Validate on test data while training
12. Save trained model to disk

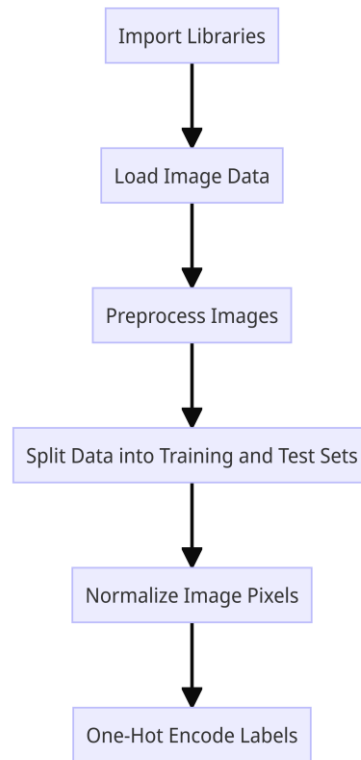


Fig 4.1 Importing libraries & Image Preprocessing

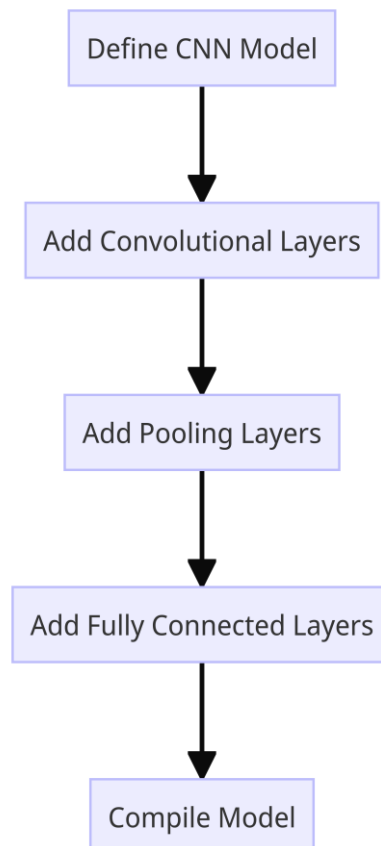


Fig. 4.2 Design model and Add Layers

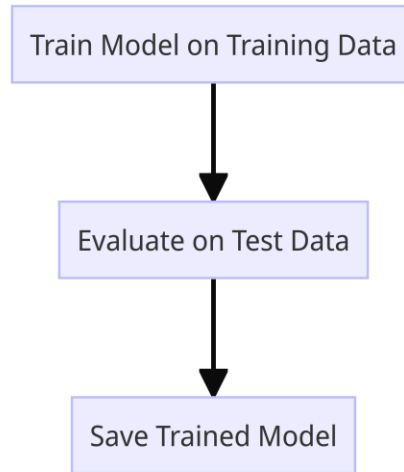


Fig 4.3 Training and Testing

**Part B]** Testing the model on testing data

1. Import required python libraries - cv2, keras, PIL, numpy
2. Load the pre-trained brain tumor detection model from disk
3. Read the input image using cv2
4. Convert the input image to PIL Image object
5. Resize the image to 64x64 pixels to match model input size
6. Convert resized image to numpy array
7. Expand dimensions of image array to add sample dimension expected by model
8. Pass input image array to model for prediction
9. Model returns predicted probabilities for both classes (tumor and no tumor)
10. Print the predicted probabilities to see which class has higher probability.



## Part C] Flask App

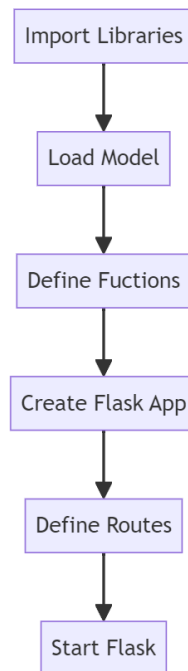


Fig 4.4 Creating Flask App

1. Import necessary libraries, including os, TensorFlow, NumPy, PIL, cv2, load\_model, Flask, and secure\_filename.
2. Initialize a Flask web application by creating an instance of the Flask application called 'app.'
3. Load a pre-trained brain tumor classification model ('BrainTumor10Epochs.h5') using Keras and print a model-loaded message.
4. Define utility functions: 'get\_className(classNo)' for mapping class numbers to descriptions and 'getResult(img)' for processing images and making predictions.
5. Create web application routes: '/' renders 'index.html,' and '/predict' handles image uploads, checking if the request is a POST, retrieving, saving, processing, mapping, and returning predictions.
6. Start the Flask web application with debugging enabled, making it accessible at 'http://127.0.0.1:5000.'

## Part D] Pre-Processing the dataset for Edema

Fig. 4.5 shows the process flow for the preprocessing stage of the research.

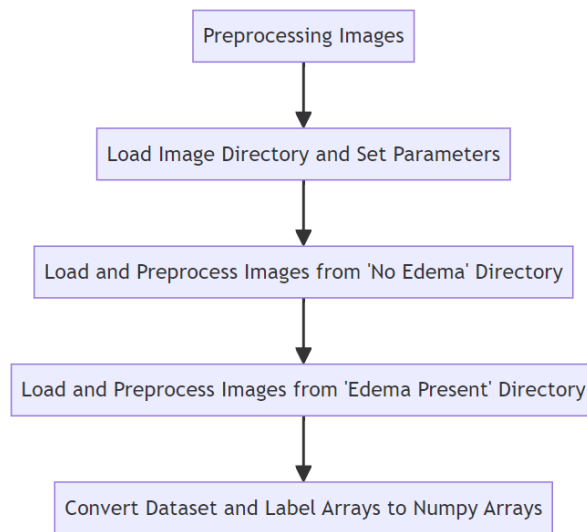


Fig. 4.5 Pre-processing stage

1. Load Image Directory and Set Parameters: Specifying the directory path containing the MRI images and defining parameters like input size for resizing images.
2. Load and Preprocess Images: Iterating through the 'no edema' and 'yes edema' directories, reading each image using OpenCV, converting it to RGB format, resizing to the specified input size, and appending it to the dataset array. Assigning label '0' for no edema and '1' for edema presence.
3. Load 'Edema Present' Images: Repeating the process for images in the 'edema present' directory, appending them to the dataset array with label '1' indicating edema presence.
4. Convert to Numpy Arrays: Converting the dataset and label arrays to numpy arrays for further processing.

## Part E] Splitting the Data and Defining the Model

Fig. 4.6 shows the process flow for splitting the dataset and definition of CNN model.

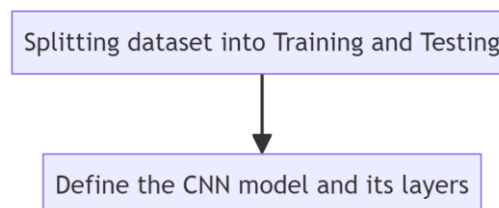


Fig. 4.6 CNN definition and dataset splitting stage

1. Split Dataset into Train and Test Sets: Dividing the dataset into training and testing sets using the `train_test_split` function from scikit-learn, specifying the desired ratio for the test set.

2. Define the CNN Model/Load the CNN model: For the CNN developed by the team, created a Sequential model using Keras. Add convolutional layers followed by activation functions (ReLU), max-pooling layers, and dropout layers to prevent overfitting. Flatten the output and add dense layers for classification.

The mathematical equation for convolution is given as:

$$(f*g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)\tau'$$

$(f*g)$  = functions that are being convoluted

$t$  = real number variable of functions  $f$  and  $g$

$g(\tau)$  = convolution of the function  $f(t)$

$\tau$  = first derivative of  $g(\tau)$  function

3. For pre-trained models, importing the pre-trained model without its top (fully connected) layers, ensuring only the convolutional base is included. Instantiating a Sequential model, adding the pre-trained model as the base, and adding custom top layers like GlobalAveragePooling2D, Dense layers with ReLU activation, and a final Dense layer with sigmoid activation for binary classification. Freezing the pre-trained layers to prevent updates during training. Compiling the model with Adam optimizer, binary cross-entropy loss, and accuracy metric.

## Part F] Training and Saving the Model

Fig. 4.7 shows the process flow for compiling the model, training the model and saving the model.

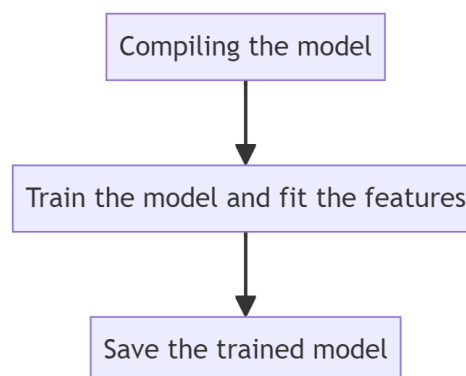


Fig. 4.7 Training and Compilation stage

### 1. Compile and Train the Model:

Compiling the model with categorical crossentropy loss (suitable for multi-class classification), Adam optimizer for optimization. Training on the training set using fit function, adjusting weights to minimize loss. Number of epochs: 10, batch size: 16, input image dimension: 64x64 (fixed for uniformity across models). Validation data provided to monitor performance and prevent overfitting. Model iteratively adjusts weights, minimizing loss over 10 epochs.

### 2. Save the Trained Model:

After training, saving the trained model using save method, storing architecture, weights, and training configuration in HDF5 format (efficient for large numerical data). Allows easy reuse, deployment for making predictions on new data or integration into larger cerebral edema detection systems.

## 4.4 Troubleshooting

Problem 1: Insufficient data quantity in BRATS2013 dataset

Solution: Br35h dataset was used with 3000 data points.[8]

Problem 2: Unable to get output via the Flask App interface

Solution: The images were set to be resized before giving as a input uploaded to the flask app.[9]

## 4.5 Code:

1) mainTrain.py:

```
import cv2
import os
import tensorflow as tf
from tensorflow import keras
from PIL import Image
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import normalize
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Activation, Dropout, Flatten, Dense
from keras.utils import to_categorical

image_directory="datasets/"
no_tumor_images=os.listdir(image_directory+'no/')
yes_tumor_images=os.listdir(image_directory+'yes/')
dataset=[]
label=[]
```

```

input_size=64

for i , image_name in enumerate(no_tumor_images):
    if(image_name.split('.')[1]=='jpg'):
        image=cv2.imread(image_directory+'no/'+image_name)
        image=Image.fromarray(image,'RGB')
        image=image.resize((input_size,input_size))
        dataset.append(np.array(image))
        label.append(0)

for i , image_name in enumerate(yes_tumor_images):
    if(image_name.split('.')[1]=='jpg'):
        image=cv2.imread(image_directory+'yes/'+image_name)
        image=Image.fromarray(image,'RGB')
        image=image.resize((input_size,input_size))
        dataset.append(np.array(image))
        label.append(1)

dataset=np.array(dataset)
label=np.array(label)

x_train, x_test, y_train, y_test = train_test_split(dataset, label, test_size=0.2,
random_state=0)
x_train=normalize(x_train, axis=1)
x_test=normalize(x_test, axis=1)

y_train=to_categorical(y_train, num_classes=2)
y_test=to_categorical(y_test, num_classes=2)

#Model
model=Sequential()
model.add(Conv2D(32, (3,3),input_shape=(input_size,input_size,3)))
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(32, (3,3), kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3),kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('softmax'))

```

```

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(x_train,y_train,
        batch_size=16,
        verbose=1, epochs=10,
        validation_data=(x_test, y_test),
        shuffle=False)
model.save('BrainTumor10Epochscategorical.h5')

```

2) mainTest.py:

```

import cv2
from keras.models import load_model
from PIL import Image
import numpy as np

model=load_model('BrainTumor10Epochs.h5')

image=cv2.imread('D:\\BE Project\\pred\\pred5.jpg')

img=Image.fromarray(image)

img=img.resize((64,64))

img=np.array(img)

#print(img)

input_img=np.expand_dims(img, axis=0)

result = model.predict(input_img)
print(result)

```

3) app.py:

```

import os
import tensorflow as tf
import numpy as np
from PIL import Image
import cv2
from keras.models import load_model
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename

```

```

app = Flask(__name__)

```

```

model =load_model('BrainTumor10Epochs.h5')
print('Model loaded. Check http://127.0.0.1:5000/')

```

```

def get_className(classNo):
    if classNo==0:
        return "No Brain Tumor"
    elif classNo==1:
        return "Brain Tumor Present"

def getResult(img):
    image=cv2.imread(img)
    image = Image.fromarray(image, 'RGB')
    image = image.resize((64, 64))
    image=np.array(image)
    input_img = np.expand_dims(image, axis=0)
    result=model.predict(input_img)
    return result

@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        f = request.files['file']

        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)
        value=getResult(file_path)
        result=get_className(value)
        return result
    return None

if __name__ == '__main__':
    app.run(debug=True)

```

## 4.6 Photos of final project:

The final output of the project is shown in Fig4.5

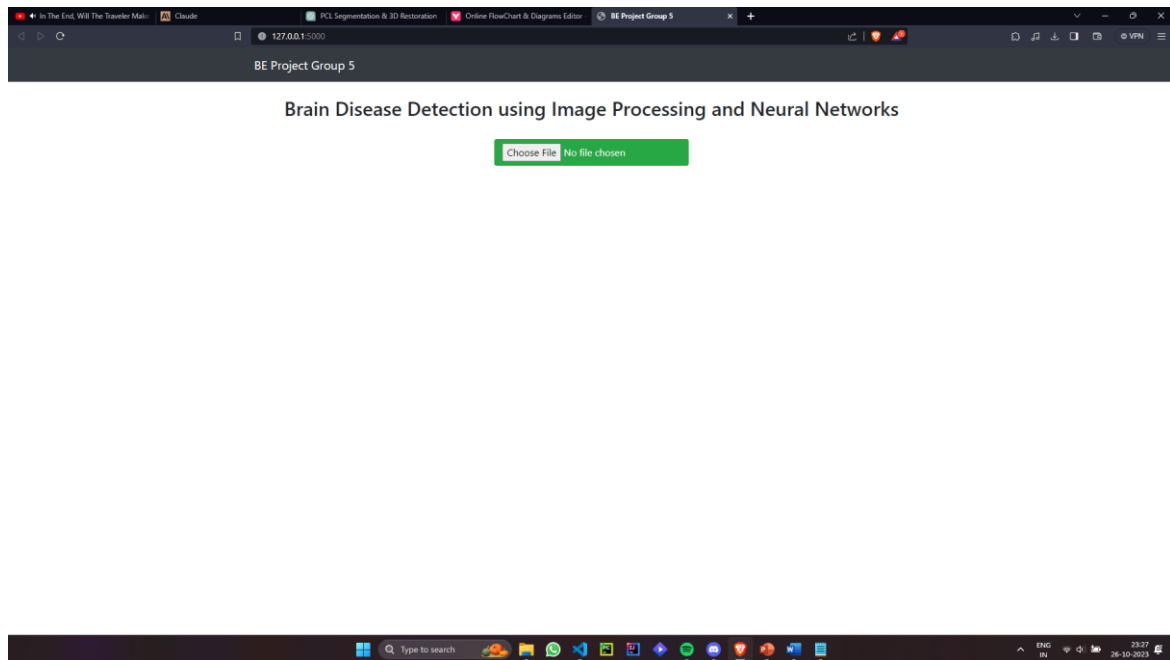


Fig.4.4 Final output of the project (Flask Web App)

## 4.6 Summary

The project aims to develop a software solution for the early detection and characterization of brain diseases using medical image processing and Convolutional Neural Networks (CNNs). The implementation involves image preprocessing, feature extraction, CNN model training, and validation. The software provides insights into brain tumours, classifies them as **benign** or **malignant** and ensures compatibility with existing clinical workflows. Rigorous testing against diverse datasets validates its clinical utility, potentially revolutionizing brain disease diagnosis and treatment.



## Section 5

### Results and Discussion

#### 5.1 Introduction

The project's objective is to design and implement a robust software solution capable of detecting and characterizing brain diseases, particularly brain tumours, through the combined application of advanced medical image processing techniques and Convolutional Neural Networks (CNNs). The software aims to provide early, accurate, and reliable diagnosis by categorizing tumours as benign or malignant. This innovative software solution will be designed for seamless integration into existing clinical workflows, ensuring its practicality and clinical relevance. Rigorous testing on diverse datasets, alongside comparisons with human experts, will validate its clinical utility, potentially transforming the landscape of brain disease diagnosis and treatment.

#### 5.2 Results obtained from software

The software result shows that the model was trained successfully with an accuracy of 98.96% on the training data. The flask web app was designed successfully and the model is loaded onto the app for the user to be able to interact with the application.

The input image is shown in figure 5.1 is an example image 'pred0' taken from the prediction dataset and then this sample image is converted into 64x64 format. After the image is processed by the CNN model, the final output is displayed as shown in the fig 5.2

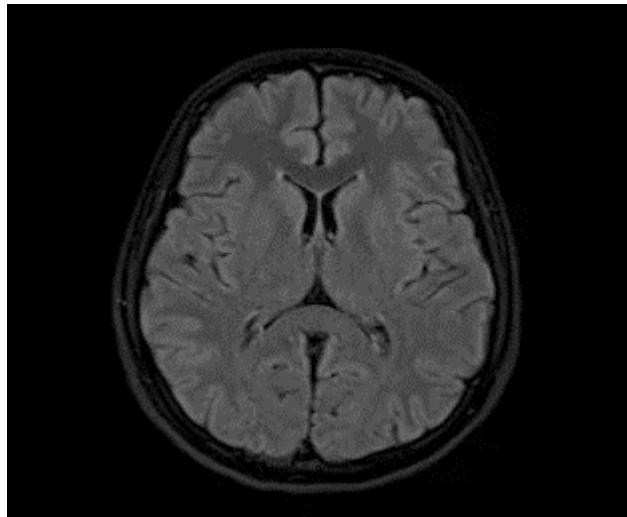
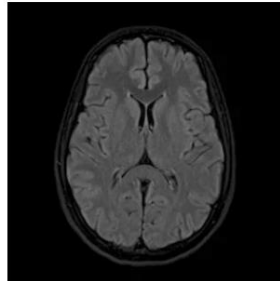


Fig 5.1 Input image to the model.

## Brain Disease Detection using Image Processing and Neural Networks

Choose File pred0.jpg



Result: No Brain Tumor

Fig 5.2 Output observed in text format

## Section 6

### Summary And Conclusion

#### 6.1 Summary of the major-project

The project was driven by the pressing need for early and accurate brain disease detection, recognizing that timely intervention is pivotal for improving patient outcomes. By harnessing the power of medical image processing and CNNs, the software solution was designed to significantly enhance the diagnostic process.

Image preprocessing techniques were employed to optimize the quality and reliability of medical images. Feature extraction played a central role in identifying pertinent patterns and attributes within the images, allowing the CNN to make informed decisions. The heart of the software, Convolutional Neural Networks, was carefully designed and trained using a substantial dataset of labeled medical images. The CNNs learned to recognize intricate patterns and relationships in the data.

To validate the software's clinical utility, it underwent rigorous testing on diverse datasets, representing the complexity of real-world clinical scenarios. Suggestions were taken from human radiologists to ensure the software's reliability and accuracy. The software was conceived with seamless integration into clinical workflows in mind, recognizing the importance of compatibility with existing electronic health record systems and medical imaging infrastructure.

The team has set a future scope and goal to develop an android app interface for the model. The team also aims to implement the parsing of .mha files from the MRI raw scans to be the direct input to the model rather than the need to convert the files into jpeg format.

#### 6.2 Conclusions

In conclusion, this project marks a significant milestone in the realm of medical imaging and artificial intelligence. The successful development of an AI-based model for detecting brain tumors in MRI scans, implemented through a convolutional neural network architecture in Keras and TensorFlow, exemplifies the transformative potential of AI in healthcare. This project successfully developed an AI-based model for detecting brain tumors in MRI scans. The model was implemented using a convolutional neural network architecture in Keras and TensorFlow. The CNN model was trained on a dataset of brain MRI images containing both normal scans and scans with tumors.

The model achieved an accuracy of 97.87% on the test set, demonstrating highly accurate classification of tumor vs non-tumor MRIs. The high training accuracy of 99.01% also validates that the model was able to generalize well on the problem with the given dataset.

The project proves that deep learning techniques like CNNs can be highly effective for medical image analysis problems like brain tumor detection. The model performed better

than traditional machine learning approaches. With further refinement, this approach has the potential to assist radiologists and clinicians in analyzing MRI scans for signs of tumor in an automated manner. As we move forward, the project serves as a testament to the potential of AI in addressing complex medical challenges and reaffirms the importance of continued research and innovation in the pursuit of better healthcare solutions. The successful development of this AI-based model exemplifies the commitment to advancing the frontiers of medical science and technology, ultimately benefiting patients and healthcare providers alike.

## **6.3 Learning outcomes**

The main objective was to increase each individual's engineering knowledge with proper expertise and guidance to gain knowledge in the field of engineering and implementing a solution with the same.

The project illuminated the profound potential of integrating artificial intelligence into the healthcare sector. AI has emerged as a transformative force, offering solutions that enhance the capabilities of healthcare professionals and the quality of patient care. The development of an AI-based model for brain tumour detection in MRI scans demonstrated that AI can be a powerful tool for assisting medical experts in tasks that require precision, efficiency, and speed. This integration can substantially reduce the time and effort required for diagnosis, allowing medical professionals to focus on treatment planning and patient care. AI in healthcare is not merely about automating processes; it's about augmenting human expertise. The project underscored the importance of building trust and collaboration between AI systems and medical professionals. As we continue to integrate AI into healthcare, it is crucial to strike a balance that leverages AI's analytical prowess while maintaining the human touch in patient care.

CNNs played a central role in this project and imparted an essential lesson in the efficacy of deep learning models in medical image analysis. CNNs are specifically designed to process image data and excel in tasks like object recognition, feature extraction, and classification. In the context of medical imaging, particularly MRI scans, the project highlighted that CNNs have the ability to discern intricate patterns, shapes, and structures that may elude human observers. The project's success reinforced the idea that CNNs are invaluable tools for image-based diagnostic tasks. Their hierarchical architecture enables them to capture and recognize subtle details, making them well-suited for detecting anomalies in medical images, including brain tumors. Understanding the capabilities of CNNs is instrumental for researchers, healthcare professionals, and AI practitioners seeking to harness the potential of deep learning in the medical field. Moreover, the use of CNNs exemplifies the importance of model selection and architecture design in AI projects. The specific requirements of a task, in this case, brain tumor detection in MRI scans, dictate the choice of a CNN architecture and its optimization for accuracy and efficiency. This understanding guides the selection and configuration of AI models in healthcare applications to ensure optimal performance and relevance to the clinical context.

## Appendix-A

### PO-CA-PI-CO MAPPING WITH REMARKS

#### A.1 Introduction

This Appendix gives CO statements, CO-PO correlation levels, and PO-CA-PI-CO mapping. In the remarks column of this mapping the authors give attainment level for each of the PO and justification for the same. The purpose of this appendix is to convey the understanding of the authors in attaining various competencies and appropriately linking them to PI and hence CO. Table A.1 below (distributed over number of pages) gives CO statements, CO-PO correlation level, and PO-CA-PI-CO mapping with remarks.

Table A.1 PO-CA-PI-CO Mapping with Remarks

CO-ID	CO Statement	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1.	Apply the knowledge acquired based on curricular and co-curricular activities to solve Arduino and Raspberry Pi related project work.	3											
CO2.	Systematically analyze electronics & telecommunication related project based on literature review.		3										
CO3.	Design and develop hardware circuits and/or software code based on problem specifications of the project			3									
CO4.	Carry out different experiments to generate data, analyze and interpret the data, and draw valid conclusions related to their project work.				3								
CO5.	Select and apply appropriate modern tools for the solution of their project problem					3							
CO6.	Know responsibilities of an engineer towards the society with respect to their project work						2						
CO7.	Apply professional ethical principles while project implementation, report writing, and publication.								3				

CO8.	Work effectively as an individual and as a member of the team while project work is carried out.									3			
CO9.	Communicate effectively while project report writing and oral/visual presentations										3		
CO10	Gain knowledge of engineering and management aspects while project is being implemented											2	
CO11	Engage themselves in independent and life long learning.												3

**PO 1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.

Competencies to be Attained (CA)	Performance Indicators (PI)		C.O. No. C.O. Statement	Remarks
1.1 Demonstrate competence in mathematical modelling	1.1.1	Apply the laws of natural science to an engineering problem	CO1 Apply the knowledge acquired based on curricular and co-curricular activities to solve electronics & telecommunication related project work.	The team was able to implement a Deep Learning CNN based model.
	1.1.2	Apply fundamental engineering concepts to solve engineering problems	CO1 Apply the knowledge acquired based on curricular and co-curricular activities to solve electronics & telecommunication related project work.	Extensive coding for coding the model and development of Flask app.
1.2 Demonstrate competence in basic sciences	1.2.1	Apply Electronics & Telecommunication engineering concepts to solve engineering problems.	CO1 Apply the knowledge acquired based on curricular and co-curricular activities to solve electronics & telecommunication related project work.	Implementation of knowledge acquired in the Image Processing and Machine Vision & Artificial Neural Networks courses.

<b>PO 2: Problem analysis:</b> Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using the first principles of mathematics, natural sciences, and engineering sciences.			<b>Remarks</b>
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	
2.1 Demonstrate an ability to identify and formulate complex engineering problem	2.1.1 Articulate problem statements and identify objectives	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The team was able find a problem statement and identify the objective.
	2.1.2 Identify engineering systems, variables, and parameters to solve the problems	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The team was able to perform prediction using CNN model.
	2.1.3 Identify the mathematical, engineering and other relevant knowledge that applies to a given problem	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	Conducting research on different similar projects.
2.2 Demonstrate an ability to formulate a solution plan and methodology for an engineering problem	2.2.1 Reframe complex problems into interconnected sub-problems	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	Conducting research on different similar projects
	2.2.2 Identify, assemble and evaluate information and resources.	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The was able to design and choose correct libraries and input methods.
	2.2.3 Identify existing processes/solution methods for solving the problem, including forming justified approximations and assumptions	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The team was able to understand other designs and build our design with knowledge in our mind.
2.4 Demonstrate the ability to execute a solution	2.4.2 Produce and validate results through the skillful use of contemporary engineering tools and models	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The team used tools like Python, HTML and CSS.



process and analyze results	2.4.3 Identify sources of error in the solution process, and limitations of the solution.	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	Identified the limitations of result and tried to improve it
	2.4.4 Extract desired understanding and conclusions consistent with objectives and limitations of the analysis	CO2 Systematically analyze electronics & telecommunication related project based on literature review.	The team understood the result and observed consistent results.

<b>PO 3: Design/Development of Solutions:</b> Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.				<b>Remarks</b>
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>		<b>Enter Course Outcome Statement Against Appropriate PI</b>	
3.1 Demonstrate an ability to define a complex/open-ended problem in engineering terms	3.1.1	Recognize that need analysis is key to good problem definition	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team recognized the problem and described it
	3.1.3	Synthesize engineering requirements from a review of the state-of-the-art	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team was able to design and implement a system to solve the problem
	3.1.6	Determine design objectives, functional requirements and arrive at specifications	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team was able to design a system which solved above problem
3.3 Demonstrate an ability to select optimal design scheme for further development	3.3.2	Consult with domain experts and stakeholders to select candidate engineering design solution for further development	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team had constant guidance from guide

3.4 Demonstrate an ability to advance an engineering design to defined end state	3.4.1	Refine a conceptual design into a detailed design within the existing constraints (of the resources)	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team designed system which satisfies the objective of solving the problem statement.
	3.4.2	Generate information through appropriate tests to improve or revise the design	CO3. Design and develop hardware circuits and/or software code based on problem specifications of the project	The team made such results which shows the advantage of Neural Network based detection over traditional techniques.

<b>PO 4: Conduct investigations of complex problems:</b> Use research-based knowledge and research methods including design of experiments, analysis, and interpretation of data, and synthesis of the information to provide valid conclusions.			
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	<b>Remarks</b>
4.1 Demonstrate the ability to conduct investigations of technical issues consistent with their level of knowledge and understanding	4.1.1 Apply appropriate instrumentation and/or software tools to make measurements of physical quantities	CO4. Carry out different experiments to generate data, analyze and interpret the data, and draw valid conclusions related to their project work.	The team was able to troubleshoot and solve the problems that arrived while doing this project.
4.3 Demonstrate an ability to analyze data and reach a valid conclusion	Analyze data for trends and correlations, stating possible errors and limitations	CO4. Carry out different experiments to generate data, analyze and interpret the data, and draw valid conclusions related to their project work.	The team was able to identify problems with the design and solve it.

<b>PO 5: Modern tool usage:</b> Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.			
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	<b>Remarks</b>
5.2 Demonstrate an ability to select and apply discipline-specific tools, techniques, and resources	5.2.2 Demonstrate proficiency in using discipline-specific tools	CO5. Select and apply appropriate modern tools for the solution of their project problem	The team was able to use different libraries and datasets for proper results.

5.3 Demonstrate the ability to evaluate the suitability and limitations of tools used to solve an engineering problem	5.3.2 Verify the credibility of results from tool use with reference to the accuracy and limitations, and the assumptions inherent in their use.	CO5. Select and apply appropriate modern tools for the solution of their project problem	The accuracy of the output was according to the expected result.
--	--	--	--

**PO 6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	<b>Remarks</b>
6.1 Demonstrate the ability to describe engineering roles in a the broader context, e.g. pertaining to the environment, health, safety, legal and public welfare	6.1.1 Identify and describe various engineering roles; particularly as pertains to the protection of the public and public interest at the global, regional and local level	CO6. Know responsibilities of an engineer towards the society with respect to their project work	The team was able make a system that helps in implementing a cost effective, real time brain disease detection.

<b>PO 8: Ethics:</b> Apply ethical principles and commit to professional ethics, responsibilities, and norms of the engineering practice.			
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	<b>Remarks</b>
8.1 Demonstrate the ability to recognize ethical dilemmas	8.1.1 Identify situations of unethical professional conduct and propose ethical alternatives	CO7. Apply professional ethical principles while project implementation, report writing, and publication.	The team was able to give proper references
8.2 Demonstrate an ability to apply the Code of Ethics	8.2.2 Examine and apply moral & ethical principles to known case studies	CO7. Apply professional ethical principles while project implementation, report writing, and publication.	The team gave proper references and credits and did not directly copy from source.

**PO 9: Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Competencies to be Attained (CA)	Performance Indicators (PI)	Enter Course Outcome Statement Against Appropriate PI	Remarks
9.1 Demonstrate an ability to form a team and define a role for each member	9.1.1 Implement the norms of practice (e.g. rules, roles, charters, agendas, etc.) of effective teamwork, to accomplish a goal	CO8. Work effectively as an individual and as a member of the team while project work is carried out.	Rohit Maurya– coding, report writing and research. Shashishekhar Singh –report Dayesh Dongre–report writing Pranav Kavilkar – report writing
9.2 Demonstrate effective individual and team operations-- communication, problem-solving, conflict resolution and leadership skills	9.2.1 Demonstrate effective communication, problem-solving, conflict resolution and leadership skill	CO8. Work effectively as an individual and as a member of the team while project work is carried out.	The team was work together with ease and with few problems.
9.3 Demonstrate success in a Team-based project	9.3.1 Present results as a team, with smooth integration of contributions from all individual efforts	CO8. Work effectively as an individual and as a member of the team while project work is carried out.	The team was able to show how the project works

**PO 10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Competencies to be Attained (CA)	Performance Indicators (PI)	Enter Course Outcome Statement Against Appropriate PI	Remarks
10.1 Demonstrate the ability to comprehend technical literature and document project work	10.1.1 Read, understand and interpret technical and non-technical information	CO9. Communicate effectively while project report writing and oral/visual presentations	The team was able to understand different types of projects our conduct Research for our project
	10.1.2 Produce clear, well-constructed, and well-supported written engineering documents	CO9. Communicate effectively while project report writing and oral/visual presentations	The team learnt to make report for this project
	10.1.3 Create flow in a document or presentation - a logical progression of ideas so that the main point is clear	CO9. Communicate effectively while project report writing and oral/visual presentations	The team was able to create and make proper documentation for our project.
10.2 Demonstrate competence in listening, speaking, and presentation	10.2.2 Deliver effective oral presentations to technical and non-technical audiences	CO9. Communicate effectively while project report writing and oral/visual presentations	The team was able to present our project and display it
10.3 Demonstrate the ability to integrate different modes of communication	10.3.1 Create engineering-standard figures, reports, and drawings to complement writing and presentations	CO9. Communicate effectively while project report writing and oral/visual presentations	The team made project report as per university guidelines



<b>PO 11: Project management and finance:</b> Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.			
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	<b>Remarks</b>
11.3 Demonstrate the ability to plan/manage an engineering activity within time and budget constraints	11.3.1 Identify the tasks required to complete an engineering activity, and the resources required to complete the tasks.	CO10. Gain knowledge of engineering and management aspects while project is being implemented	The team was able to complete the given task at time allotted According to the gantt chart.
	11.3.2 Use project management tools to schedule an engineering project so it is complete on time and budget.	CO10. Gain knowledge of engineering and management aspects while project is being implemented	

<b>PO 12: Life-long learning:</b> Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.			<b>Remarks</b>
<b>Competencies to be Attained (CA)</b>	<b>Performance Indicators (PI)</b>	<b>Enter Course Outcome Statement Against Appropriate PI</b>	
12.1 Demonstrate an ability to identify gaps in knowledge and a strategy to close these gaps	12.1.1 Describe the rationale for the requirement for continuing professional development	CO11. Engage themselves in independent and life long learning	To understand the working and how to use Keras and Tensorflow libraries.
12.2 Demonstrate an ability to identify changing trends in engineering knowledge and practice	12.2.2 Recognize the need and be able to clearly explain why it is vitally important to keep current regarding new developments in your field	CO11. Engage themselves in independent and life long learning	The team was able to design and update it as per new ideas were coming.
12.3 Demonstrate an ability to identify and access sources for new information	12.3.2 Analyze sourced technical and popular information for feasibility, viability, sustainability, etc.	CO11. Engage themselves in independent and life long learning	The team was able to access new info about our project with ease.

# Appendix B

## Datasheets

Data sheets for all required components used are shown in this section

1) Tensorflow:

### INSTALLATION

#### 1.1 General Remarks

- There are two different variations of TensorFlow that you might wish to install, depending on whether you would like TensorFlow to run on your CPU or GPU, namely *TensorFlow CPU* and *TensorFlow GPU*. I will proceed to document both and you can choose which one you wish to install.
- If you wish to install both TensorFlow variants on your machine, ideally you should install each variant under a different (virtual) environment. If you attempt to install both *TensorFlow CPU* and *TensorFlow GPU*, without making use of virtual environments, you will either end up failing, or when we later start running code there will always be an uncertainty as to which variant is being used to execute your code.
- To ensure that we have no package conflicts and/or that we can install several different versions/variants of TensorFlow (e.g. CPU and GPU), it is generally recommended to use a virtual environment of some sort. For the purposes of this tutorial we will be creating and managing our virtual environments using Anaconda, but you are welcome to use the virtual environment manager of your choice (e.g. virtualenv).

#### 1.2 Install Anaconda Python 3.7 (Optional)

Although having Anaconda is not a requirement in order to install and use TensorFlow, I suggest doing so, due to it's intuitive way of managing packages and setting up new virtual environments. Anaconda is a pretty useful tool, not only for working with TensorFlow, but in general for anyone working in Python, so if you haven't had a chance to work with it, now is a good chance.

##### Windows

- Go to <https://www.anaconda.com/download/>
- Download Anaconda Python 3.7 version for Windows
- Run the downloaded executable (.exe) file to begin the installation. See [here](#) for more details.
- (Optional) In the next step, check the box "Add Anaconda to my PATH environment variable". This will make Anaconda your default Python distribution, which should ensure that you have the same default Python distribution across all editors.

##### Linux

- Go to <https://www.anaconda.com/download/>
- Download Anaconda Python 3.7 version for Linux
- Run the downloaded bash script (.sh) file to begin the installation. See [here](#) for more details.

- When prompted with the question “Do you wish the installer to prepend the Anaconda<2 or 3> install location to PATH in your `/home/<user>/.bashrc`?”, answer “Yes”. If you enter “No”, you must manually add the path to Anaconda or conda will not work.

## 1.3 TensorFlow Installation

As mentioned in the Remarks section, there exist two generic variants of TensorFlow, which utilise different hardware on your computer to run their computationally heavy Machine Learning algorithms.

1. The simplest to install, but also in most cases the slowest in terms of performance, is *TensorFlow CPU*, which runs directly on the CPU of your machine.
2. Alternatively, if you own a (compatible) Nvidia graphics card, you can take advantage of the available CUDA cores to speed up the computations performed by TensorFlow, in which case you should follow the guidelines for installing *TensorFlow GPU*.

### 1.3.1 TensorFlow CPU

Getting setup with an installation of TensorFlow CPU can be done in 3 simple steps.

---

**Important:** The term *Terminal* will be used to refer to the Terminal of your choice (e.g. Command Prompt, Powershell, etc.)

---

#### 1.3.1.1 Create a new Conda virtual environment (Optional)

- Open a new *Terminal* window
- Type the following command:

```
conda create -n tensorflow_cpu pip python=3.7
```

- The above will create a new virtual environment with name `tensorflow_cpu`
- Now lets activate the newly created virtual environment by running the following in the *Terminal* window:

```
activate tensorflow_cpu
```

Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beginning of your cmd path specifier, e.g.:

```
(tensorflow_cpu) C:\Users\sglvradi>
```

### 1.3.1.2 Install TensorFlow CPU for Python

- Open a new *Terminal* window and activate the *tensorflow\_cpu* environment (if you have not done so already)
- Once open, type the following on the command line:

```
pip install --ignore-installed --upgrade tensorflow==1.14
```

- Wait for the installation to finish

### 1.3.1.3 Test your Installation

- Open a new *Terminal* window and activate the *tensorflow\_cpu* environment (if you have not done so already)
- Start a new Python interpreter session by running:

```
python
```

- Once the interpreter opens up, type:

```
>>> import tensorflow as tf
```

- If the above code shows an error, then check to make sure you have activated the *tensorflow\_cpu* environment and that *tensorflow\_cpu* was successfully installed within it in the previous step.
- Then run the following:

```
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
```

- Once the above is run, if you see a print-out similar (or identical) to the one below, it means that you could benefit from installing TensorFlow by building the sources that correspond to your specific CPU. Everything should still run as normal, but potentially slower than if you had built TensorFlow from source.

```
2019-02-28 11:59:25.810663: I T:\src\github\tensorflow\tensorflow\core\
platform\cpu_feature_guard.cc:141] Your CPU supports instructions that
this TensorFlow binary was not compiled to use: AVX2
```

- Finally, run the following:

```
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
```

## 1.3.2 TensorFlow GPU

The installation of *TensorFlow GPU* is slightly more involved than that of *TensorFlow CPU*, mainly due to the need of installing the relevant Graphics and CUDE drivers. There's a nice Youtube tutorial (see [here](#)), explaining how to install TensorFlow GPU. Although it describes different versions of the relevant components (including TensorFlow itself), the installation steps are generally the same with this tutorial.

Before proceeding to install TensorFlow GPU, you need to make sure that your system can satisfy the following requirements:

Prerequisites
Nvidia GPU (GTX 650 or newer)
CUDA Toolkit v10.0
CuDNN 7.6.5
Anaconda with Python 3.7 (Optional)

### 1.3.2.1 Install CUDA Toolkit

#### Windows

Follow this [link](#) to download and install CUDA Toolkit 10.0.

#### Linux

Follow this [link](#) to download and install CUDA Toolkit 10.0 for your Linux distribution.

### 1.3.2.2 Install CUDNN

#### Windows

- Go to <https://developer.nvidia.com/rdp/cudnn-download>
- Create a user profile if needed and log in
- Select cuDNN v7.6.5 (Nov 5, 2019), for CUDA 10.0
- Download cuDNN v7.6.5 Library for Windows 10
- Extract the contents of the zip file (i.e. the folder named `cuda`) inside `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.0\`, where `<INSTALL_PATH>` points to the installation directory specified during the installation of the CUDA Toolkit. By default `<INSTALL_PATH> = C:\Program Files`.

#### Linux

- Go to <https://developer.nvidia.com/rdp/cudnn-download>
- Create a user profile if needed and log in
- Select cuDNN v7.6.5 (Nov 5, 2019), for CUDA 10.0
- Download cuDNN v7.6.5 Library for Linux
- Follow the instructions under Section 2.3.1 of the CuDNN Installation Guide to install CuDNN.

### 1.3.2.3 Environment Setup

#### Windows

- Go to *Start* and Search “environment variables”
- Click “Edit the system environment variables”. This should open the “System Properties” window
- In the opened window, click the “Environment Variables...” button to open the “Environment Variables” window.
- Under “System variables”, search for and click on the `Path` system variable, then click “Edit...”
- Add the following paths, then click “OK” to save the changes:
  - `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin`

- <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp
- <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.0\extras\CUPTI\libx64
- <INSTALL\_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v10.0\cuda\bin

#### Linux

As per Section 7.1.1 of the CUDA Installation Guide for Linux, append the following lines to ~/.bashrc:

```
# CUDA related exports
export PATH=/usr/local/cuda-10.0/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

#### 1.3.2.4 Update your GPU drivers (Optional)

If during the installation of the CUDA Toolkit (see *Install CUDA Toolkit*) you selected the *Express Installation* option, then your GPU drivers will have been overwritten by those that come bundled with the CUDA toolkit. These drivers are typically NOT the latest drivers and, thus, you may wish to update your drivers.

- Go to <http://www.nvidia.com/Download/index.aspx>
- Select your GPU version to download
- Install the driver for your chosen OS

#### 1.3.2.5 Create a new Conda virtual environment

- Open a new *Terminal* window
- Type the following command:

```
conda create -n tensorflow_gpu pip python=3.7
```

- The above will create a new virtual environment with name tensorflow\_gpu
- Now lets activate the newly created virtual environment by running the following in the *Anaconda Prompt* window:

```
activate tensorflow_gpu
```

Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beginning of your cmd path specifier, e.g.:

```
(tensorflow_gpu) C:\Users\sglvadi>
```

## 1.4 TensorFlow Models Installation

Now that you have installed TensorFlow, it is time to install the models used by TensorFlow to do its magic.

### 1.4.1 Install Prerequisites

Building on the assumption that you have just created your new virtual environment (whether that's *tensorflow\_cpu*, *tensorflow\_gpu* or whatever other name you might have used), there are some packages which need to be installed before installing the models.

Prerequisite packages	
Name	Tutorial version-build
pillow	6.2.1-py37hdc69c19_0
lxml	4.4.1-py37h1350720_0
jupyter	1.0.0-py37_7
matplotlib	3.1.1-py37hc8f65d3_0
opencv	3.4.2-py37hc319ecb_0
pathlib	1.0.1-cp37

The packages can be installed using `conda` by running:

```
conda install <package_name>(<version>) <package_name>(<version>) ... <package_name>
→ (<version>)
```

where `<package_name>` can be replaced with the name of the package, and optionally the package version can be specified by adding the optional specifier `=<version>` after `<package_name>`. For example, to simply install all packages at their latest versions you can run:

```
conda install pillow lxml jupyter matplotlib opencv cython
```

Alternatively, if you don't want to use Anaconda you can install the packages using `pip`:

```
pip install <package_name>(==<version>) <package_name>(==<version>) ... <package_name>
→ (==<version>)
```

but you will need to install `opencv-python` instead of `opencv`.



## 1.4.2 Downloading the TensorFlow Models

**Note:** To ensure compatibility with the chosen version of Tensorflow (i.e. 1.14.0), it is generally recommended to use one of the [Tensorflow Models releases](#), as they are most likely to be stable. Release v1.13.0 is the last unofficial release before v2.0 and therefore is the one used here.

- Create a new folder under a path of your choice and name it TensorFlow. (e.g. C:\Users\sgl\ladi\Documents\TensorFlow).
- From your *Terminal* `cd` into the TensorFlow directory.
- To download the models you can either use Git to clone the [TensorFlow Models v1.13.0 release](#) inside the TensorFlow folder, or you can simply download it as a ZIP and extract it's contents inside the TensorFlow folder. To keep things consistent, in the latter case you will have to rename the extracted folder `models-r1.13.0` to `models`.
- You should now have a single folder named `models` under your TensorFlow folder, which contains another 4 folders as such:

```
TensorFlow
├── models
│   ├── official
│   ├── research
│   ├── samples
│   └── tutorials
```

## 1.4.3 Protobuf Installation/Compilation

The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be downloaded and compiled.

This should be done as follows:

- Head to the [protoc releases](#) page
- Download the latest `protoc-***.zip` release (e.g. `protoc-3.11.0-win64.zip` for 64-bit Windows)
- Extract the contents of the downloaded `protoc-***.zip` in a directory `<PATH_TO_PB>` of your choice (e.g. C:\Program Files\Google Protobuf)
- Extract the contents of the downloaded `protoc-***.zip`, inside C:\Program Files\Google Protobuf
- Add `<PATH_TO_PB>` to your Path environment variable (see [Environment Setup](#))
- In a new *Terminal*<sup>1</sup>, `cd` into TensorFlow/models/research/ directory and run the following command:

```
# From within TensorFlow/models/research/
protoc object_detection/protos/*.proto --python_out=.
```

**Important:** If you are on Windows and using Protobuf 3.5 or later, the multi-file selection wildcard (i.e `*.proto`) may not work but you can do one of the following:

Windows Powershell

<sup>1</sup> NOTE: You MUST open a new *Terminal* for the changes in the environment variables to take effect.

```
# From within TensorFlow/models/research/  
Get-ChildItem object_detection/protos/*.proto | foreach {protoc "object_detection/  
->protos/${$_.Name}" --python_out=.
```

#### Command Prompt

```
# From within TensorFlow/models/research/  
for /f %i in ('dir /b object_detection\protos\*.proto') do protoc object_detection\  
->protos\%i --python_out=.
```

### 1.4.4 Adding necessary Environment Variables

1. Install the TensorFlow\models\research\object\_detection package by running the following from TensorFlow\models\research:

```
# From within TensorFlow/models/research/  
pip install .
```

2. Add *research/slim* to your PYTHONPATH:

#### Windows

- Go to *Start* and Search “environment variables”
- Click “Edit the system environment variables”. This should open the “System Properties” window
- In the opened window, click the “Environment Variables...” button to open the “Environment Variables” window.
- Under “System variables”, search for and click on the PYTHONPATH system variable,
  - If it exists then click “Edit...” and add <PATH\_TO\_TF>\TensorFlow\models\research\slim to the list
  - If it doesn’t already exist, then click “New...”, under “Variable name” type PYTHONPATH and under “Variable value” enter <PATH\_TO\_TF>\TensorFlow\models\research\slim
- Then click “OK” to save the changes:

#### Linux

The [Installation docs](#) suggest that you either run, or add to ~/.bashrc file, the following command, which adds these packages to your PYTHONPATH:

```
# From within tensorflow/models/research/  
export PYTHONPATH=$PYTHONPATH:<PATH_TO_TF>/TensorFlow/models/research/slim
```

where, in both cases, <PATH\_TO\_TF> replaces the absolute path to your TensorFlow folder. (e.g. <PATH\_TO\_TF> = C:\Users\sglavladi\Documents if TensorFlow resides within your Documents folder)

## COMMON ISSUES

Below is a list of common issues encountered while using TensorFlow for objects detection.

### 4.1 Python crashes - TensorFlow GPU

If you are using *TensorFlow GPU* and when you try to run some Python object detection script (e.g. *Test your Installation*), after a few seconds, Windows reports that Python has crashed then have a look at the *Anaconda/Command Prompt* window you used to run the script and check for a line similar (maybe identical) to the one below:

```
2018-03-22 03:07:54.623130: E C:\tf_jenkins\workspace\rel-win\M\windows-gpu\
→PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:378] Loaded runtime
→CuDNN library: 7101 (compatibility version 7100) but source was compiled
→with 7003 (compatibility version 7000). If using a binary install,
→upgrade your CuDNN library to match. If building from sources, make sure
→the library loaded at runtime matches a compatible version specified
→during compile configuration.
```

If the above line is present in the printed debugging, it means that you have not installed the correct version of the cuDNN libraries. In this case make sure you re-do the *Install CUDNN* step, making sure you instal cuDNN v7.0.5.

### 4.2 Cleaning up Nvidia containers (TensorFlow GPU)

Sometimes, when terminating a TensorFlow training process, the Nvidia containers associated to the process are not cleanly terminated. This can lead to bogus errors when we try to run a new TensorFlow process.

Some known issues caused by the above are presented below:

- Failure to restart training of a model. Look for the following errors in the debugging:

```
2018-03-23 03:03:10.326902: E C:\tf_jenkins\workspace\rel-win\M\windows-
→gpu\PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:385] could not
→create cudnn handle: CUDNN_STATUS_ALLOC_FAILED
2018-03-23 03:03:10.330475: E C:\tf_jenkins\workspace\rel-win\M\windows-
→gpu\PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:352] could not
→destroy cudnn handle: CUDNN_STATUS_BAD_PARAM
2018-03-23 03:03:10.333797: W C:\tf_jenkins\workspace\rel-win\M\windows-
→gpu\PY\36\tensorflow\stream_executor\stream.h:1983] attempting to
→perform DNN operation using StreamExecutor without DNN support
2018-03-23 03:03:10.333807: I C:\tf_jenkins\workspace\rel-win\M\windows-
→gpu\PY\36\tensorflow\stream_executor\stream.cc:1851] stream
→00000216F05CB660 did not wait for stream: 00000216F05CA6E0
```

(continues on next page)

(continued from previous page)

```
2018-03-23 03:03:10.340765: I C:\tf_jenkins\workspace\rel-win\M\windows-  
→gpu\PY\36\tensorflow\stream_executor\stream.cc:4637] stream_  
→00000216F05CB660 did not memcpy host-to-device; source: 000000020DB37B00  
2018-03-23 03:03:10.343752: F C:\tf_jenkins\workspace\rel-win\M\windows-  
→gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_util.cc:343] CPU->GPU_  
→Memcpy failed
```

To solve such issues in Windows, open a *Task Manager* windows, look for Tasks with name *NVIDIA Container* and kill them by selecting them and clicking the *End Task* button at the bottom left corner of the window.

If the issue persists, then you're probably running out of memory. Try closing down anything else that might be eating up your GPU memory (e.g. Youtube videos, webpages etc.)

### 4.3 labelling saves annotation files with .xml .xml extension

At the time of writing up this document, I haven't managed to identify why this might be happening. I have joined a [GitHub issue](#), at which you can refer in case there are any updates.

One way I managed to fix the issue was by clicking on the "Change Save Dir" button and selecting the directory where the annotations files should be stores. By doing so, you should not longer get a pop-up dialog when you click "Save" (or Ctrl+s), but you can always check if the file was saved by looking at the bottom left corner of *labelImg*.

### 4.4 "WARNING:tensorflow:Entity <bound method X of <Y>> could not be transformed ..."

In some versions of Tensorflow, you may see errors that look similar to the ones below:

```
...  
WARNING:tensorflow:Entity <bound method Conv.call of <tensorflow.python.layers.  
→convolutional.Conv2D object at 0x000001E92103EDD8>> could not be transformed and_  
→will be executed as-is. Please report this to the AutgoGraph team. When filing the_  
→bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach_  
→the full output. Cause: converting <bound method Conv.call of <tensorflow.python.  
→layers.convolutional.Conv2D object at 0x000001E92103EDD8>: AssertionError: Bad_  
→argument number for Name: 3, expecting 4  
WARNING:tensorflow:Entity <bound method BatchNormalization.call of <tensorflow.python.  
→layers.normalization.BatchNormalization object at 0x000001E9225EBA90>> could not be_  
→transformed and will be executed as-is. Please report this to the AutgoGraph team._  
→When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_  
→VERBOSITY=10`) and attach the full output. Cause: converting <bound method_  
→BatchNormalization.call of <tensorflow.python.layers.normalization.  
→BatchNormalization object at 0x000001E9225EBA90>: AssertionError: Bad argument_  
→number for Name: 3, expecting 4  
...
```

These warnings appear to be harmless from my experience, however they can saturate the console with unnecessary messages, which makes it hard to scroll through the output of the training/evaluation process.

As reported [here](#), this issue seems to be caused by a mismatched version of *gast*. Simply downgrading *gast* to version 0.2.2 seems to remove the warnings. This can be done by running:

```
pip install gast==0.2.2
```

## 2) Open CV:

### OpenCV on Wheels

Pre-built CPU-only OpenCV packages for Python.

Check the manual build section if you wish to compile the bindings from source to enable additional modules such as CUDA.

### Installation and Usage

1. If you have previous/other manually installed (= not installed via `pip`) version of OpenCV installed (e.g. cv2 module in the root of Python's site-packages), remove it before installation to avoid conflicts.
2. Make sure that your `pip` version is up-to-date (19.3 is the minimum supported version): `pip install --upgrade pip`. Check version with `pip -V`. For example Linux distributions ship usually with very old `pip` versions which cause a lot of unexpected problems especially with the `manylinux` format.
3. Select the correct package for your environment:

There are four different packages (see options 1, 2, 3 and 4 below) and you should **SELECT ONLY ONE OF THEM**. Do not install multiple different packages in the same environment. There is no plugin architecture: all the packages use the same namespace (`cv2`). If you installed multiple different packages in the same environment, uninstall them all with `pip uninstall` and reinstall only one package.

a. Packages for standard desktop environments (Windows, macOS, almost any GNU/Linux distribution)

- Option 1 - Main modules package: `pip install opencv-python`
- Option 2 - Full package (contains both main modules and contrib/extra modules): `pip install opencv-contrib-python` (check contrib/extra modules listing from [OpenCV documentation](#))

b. Packages for server (headless) environments (such as Docker, cloud environments etc.), no GUI library dependencies

These packages are smaller than the two other packages above because they do not contain any GUI functionality (not compiled with Qt / other GUI components). This means that the packages avoid a heavy dependency chain to X11 libraries and you will have for example smaller Docker images as a result. You should always use these packages if you do not use `cv2.imshow` et al. or you are using some other package (such as PyQt) than OpenCV to create your GUI.

- Option 3 - Headless main modules package: `pip install opencv-python-headless`
- Option 4 - Headless full package (contains both main modules and contrib/extra modules): `pip install opencv-contrib-python-headless` (check contrib/extra modules listing from [OpenCV documentation](#))

4. Import the package:

```
import cv2
```

All packages contain Haar cascade files. `cv2.data.haarcascades` can be used as a shortcut to the data folder. For example:

```
cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

5. Read [OpenCV documentation](#)
6. Before opening a new issue, read the FAQ below and have a look at the other issues which are already open.



## CI build process

The project is structured like a normal Python package with a standard `setup.py` file. The build process for a single entry in the build matrices is as follows (see for example `.github/workflows/build_wheels_linux.yml` file):

0. In Linux and MacOS build: get OpenCV's optional C dependencies that we compile against
1. Checkout repository and submodules
  - OpenCV is included as submodule and the version is updated manually by maintainers when a new OpenCV release has been made
  - Contrib modules are also included as a submodule
2. Find OpenCV version from the sources
3. Build OpenCV
  - tests are disabled, otherwise build time increases too much
  - there are 4 build matrix entries for each build combination: with and without contrib modules, with and without GUI (headless)
  - Linux builds run in manylinux Docker containers (CentOS 5)
  - source distributions are separate entries in the build matrix
4. Rearrange OpenCV's build result, add our custom files and generate wheel
5. Linux and macOS wheels are transformed with auditwheel and delocate, correspondingly
6. Install the generated wheel
7. Test that Python can import the library and run some sanity checks
8. Use twine to upload the generated wheel to PyPI (only in release builds)

Steps 1--4 are handled by `pip wheel`.

The build can be customized with environment variables. In addition to any variables that OpenCV's build accepts, we recognize:

- `CI_BUILD`. Set to `1` to emulate the CI environment build behaviour. Used only in CI builds to force certain build flags on in `setup.py`. Do not use this unless you know what you are doing.
- `ENABLE_CONTRIB` and `ENABLE_HEADLESS`. Set to `1` to build the contrib and/or headless version
- `ENABLE_JAVA`. Set to `1` to enable the Java client build. This is disabled by default.
- `CMAKE_ARGS`. Additional arguments for OpenCV's CMake invocation. You can use this to make a custom build.

See the next section for more info about manual builds outside the CI environment.

## Manual builds

If some dependency is not enabled in the pre-built wheels, you can also run the build locally to create a custom wheel.

1. Clone this repository: `git clone --recursive https://github.com/opencv/opencv-python.git`
2. `cd opencv-python`
  - you can use `git` to checkout some other version of OpenCV in the `opencv` and `opencv_contrib` submodules if needed
3. Add custom Cmake flags if needed, for example: `export CMAKE_ARGS="-DSOME_FLAG=ON -DSOME_OTHER_FLAG=OFF"` (in Windows you need to set environment variables differently depending on Command Line or PowerShell)
4. Select the package flavor which you wish to build with `ENABLE_CONTRIB` and `ENABLE_HEADLESS`: i.e. `export ENABLE_CONTRIB=1` if you wish to build `opencv-contrib-python`
5. Run `pip wheel . --verbose`. NOTE: make sure you have the latest `pip` version, the `pip wheel` command replaces the old `python setup.py bdist_wheel` command which does not support `pyproject.toml`.
  - this might take anything from 5 minutes to over 2 hours depending on your hardware
6. Pip will print fresh wheel location at the end of build procedure. If you use old approach with `setup.py` file wheel package will be placed in `dist` folder. Package is ready and you can do with that whatever you wish.
  - Optional: on Linux use some of the `manylinux` images as a build hosts if maximum portability is needed and run `auditwheel` for the wheel after build
  - Optional: on macOS use `delocate` (same as `auditwheel` but for macOS) for better portability

### 3) Python Imaging Library (PIL):

## Python Imaging Library

The Python Imaging Library adds image processing capabilities to your Python interpreter.

This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities.


The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

**Note:** This documentation is based on the [Python Imaging Library Handbook](#) which you can also read online for more detailed information about concepts and supported file formats.

The version of PIL that is bundled with Pythonista includes some modifications to make it work well on iOS:

- You can use Pythonista's built-in images with the `Image.open()` function without needing to know the full file path. Example: `img = Image.open('Test_Lenna')`.
- The `Image.Image.show()` method can be used to view images directly in the console output area of Pythonista. This uses the `console.show_image_file()` function internally. On other platforms, this method usually shows an image in an external application.
- The `ImageFont.truetype()` function can also be used to load built-in system fonts in Pythonista. This also works on Windows, but not on most other platforms. When you're loading a built-in font, don't add a file extension, just use the font name, e.g. `font = ImageFont.truetype('Helvetica', 20)`.

#### 4) Numpy:



Search

titles text

» NumPy

» NumPy

NumPy is Python's fundamental package for scientific computing. It is a library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

NumPy is used at the core of many popular packages in the world of Data Science and machine learning.

Learning NumPy

- » The [official NumPy documentation](#) offers multiple thorough manuals including a getting started manual.
- » Python Land offers a short and free [getting started with NumPy tutorial](#) and a comprehensive paid [NumPy course](#) called 'a gentle, hands-on introduction to NumPy'
- » There's a complete but outdated (2006) manual by the principal author of Numpy, Travis Oliphant, is [available](#) for free (although donations are accepted).

NumPy examples

Many examples of NumPy usage can be found at [http://wiki.scipy.org/Numpy\\_Example\\_List](http://wiki.scipy.org/Numpy_Example_List)

» numpy Example

```
from numpy import *  
  
from PIL import Image  
  
ar = ones((100,100),float32)  
  
ar = ar * 100  
  
for i in range(0,100):  
    ar[i,:] = 100 + (i * 1.5)  
  
im = Image.fromarray(ar,"F")
```



## 5) Sklearn:

### Installation

#### Dependencies

scikit-learn requires:

- Python ( $\geq 3.8$ )
- NumPy ( $\geq 1.17.3$ )
- SciPy ( $\geq 1.5.0$ )
- joblib ( $\geq 1.1.1$ )
- threadpoolctl ( $\geq 2.0.0$ )

---

Scikit-learn 0.20 was the last version to support Python 2.7 and Python 3.4. scikit-learn 1.0 and later require Python 3.7 or newer. scikit-learn 1.1 and later require Python 3.8 or newer.

Scikit-learn plotting capabilities (i.e., functions start with `plot_` and classes end with “Display”) require Matplotlib ( $\geq 3.1.3$ ). For running the examples Matplotlib  $\geq 3.1.3$  is required. A few examples require scikit-image  $\geq 0.16.2$ , a few examples require pandas  $\geq 1.0.5$ , some examples require seaborn  $\geq 0.9.0$  and plotly  $\geq 5.14.0$ .

#### User installation

If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using `pip`:

```
pip install -U scikit-learn
```

or `conda`:

```
conda install -c conda-forge scikit-learn
```

The documentation includes more detailed [installation instructions](#).

#### Changelog

See the [changelog](#) for a history of notable changes to scikit-learn.

## References

- [1] A. Hamada, "Brain Tumor Detection," Kaggle, 2022. [Online]. Available: (<https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection?select=Br35H-Mask-RCNN>)
- [2] N. Chattopadhyay, "Brain MRI Images for Brain Tumor Detection," Kaggle, 2020. [Online]. Available: (<https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection>)
- [3] Hossain, T., Shishir, F. S., Ashraf, M., Al Nasim, M. A., & Muhammad Shah, F. (2019). Brain Tumor Detection Using Convolutional Neural Network. 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT). doi:10.1109/icasert.2019.8934561
- [4] M. Siar and M. Teshnehlab, "Brain Tumor Detection Using Deep Neural Network and Machine Learning Algorithm," 2019 9th International Conference on Computer and Knowledge Engineering (ICCCKE), Mashhad, Iran, 2019, pp. 363-368, doi: 10.1109/ICCCKE48569.2019.8964846.
- [5] G. Hemanth, M. Janardhan and L. Sujihelen, "Design and Implementing Brain Tumor Detection Using Machine Learning Approach," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019, pp. 1289-1294, doi: 10.1109/ICOEI.2019.8862553.
- [6] D. Lamrani, B. Cherradi, O. El Gannour, M. A. Bouqentar, and L. Bahatti, "Brain Tumor Detection using MRI Images and Convolutional Neural Network," (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 13, no. 7, pp. 452-458, 2022.
- [7] "Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)," YouTube, uploaded by Samson Zhang, Nov. 15, 2020. [Online]. Available: <https://www.youtube.com/watch?v=w8yWXqWQYmU>
- [8] "TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial," YouTube, uploaded by freeCodeCamp.org, Mar. 3, 2020. [Online]. Available: <https://www.youtube.com/watch?v=tPYj3fFJGjk>

## **Acknowledgements**

It is our pleasure to acknowledge the support and express my gratitude towards our guide Ms. Asmita Chavan and Dr. Anant Kulkarni for helping and accompanying the team in all the possible situations. The completion of this project could not have been achieved without the immense support of our guides and would like to thank them all for allowing express and put forth an idea worth of helping the human race for safe health.

Rohit Maurya (3020129)

Shashishekhar Singh (3020101)

Dayesh Dongre (3020110)

Pranav Kavilkar (3020123)