A PROJECT REPORT

ON

**GENERATIVE NEUROLOGIST**

A dissertation submitted to UDAAN in partial fulfillment of the requirement for the award of

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**SUBMITTED BY**

**VADNALA NARENDRA VARDHAN 23BD5A0506 /CSE-A/II-II**

Under the esteemed guidance **Mrs. Dr. Devika Rubi**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY**

**HYDERABAD, TELANGANA**

**02/04/2024-23/07/2024**

**SIGNATURE OF THE STUDENT** **SIGNATURE OF THE MENTOR**

**TABLE OF CONTENTS**

Objective…………………………………………………….

Technology Stack……………………………………………

Project Workflow……………………………………………

Dataset Description…………………………………………

Model used and visualization………………………………

Front-end design……………………………………………

Flask Framework………………………………………….

GPU Training (GPU vs CPU)

Platforms Used

References

Summary

**PART-1:** **Brain Tumour Segmentation Using UNETR**

**Objective**: The objective of the project "Generative Neurologist for Brain Tumour Detection" is to develop an advanced Large language model that accurately identifies and segments brain tumours from MRI images. By leveraging cutting-edge techniques like the UNETR (UNET Transformer) architecture, the project aims to improve diagnostic precision, enabling early detection and treatment planning.

**Technology Stack:**

|  |  |
| --- | --- |
| Frontend | React |
| Backend | MongoDB Atlas |
| Authentication |  |
| Model | UNETR |
| Backend API | Flask |
| Deployment |  |

**Dataset Description:**

**Name:** Brain tumour Segmentation

**Description:** Dataset contains MRI images and their corresponding mask images for brain tumour segmentation tasks. The dataset includes 3064 sample collection of T1-weighted contrast-enhanced MRI images along with manually segmented binary masks indicating the presence of brain tumours. This dataset is designed for training and evaluating deep learning models for the segmentation of brain tumours from MRI scans.

**Download Link:** <https://www.kaggle.com/datasets/nikhilroxtomar/brain-tumor-segmentation>

**Preprocessing:**

1. **Resize Image:** Convert the original (516 x516) image to(256 x 256) for standardization.

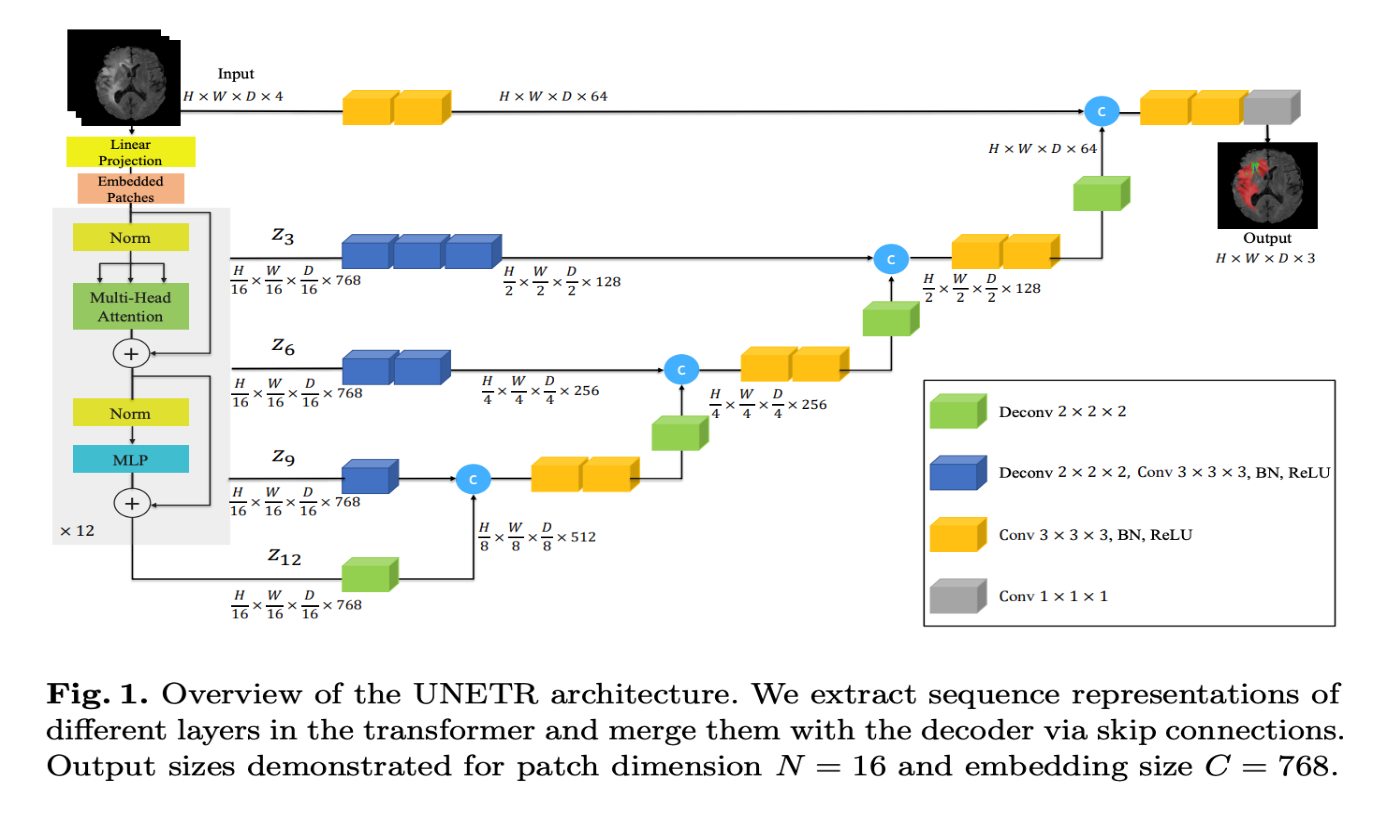
2. **Divide into Patches:** Split the resized (256 x 256) image into (16 x16) patches, resulting in 256 patches.

3. **Flatten Patches**: Each (16 x 16) patch is flattened into a (256 x 3 = 768)-dimensional vector.

4. **Prepare for Model**: Flattened patches are linearly embedded into vectors of a fixed size dmodel and combined with position embeddings before being input into the transformer encoder.

**Model used:**

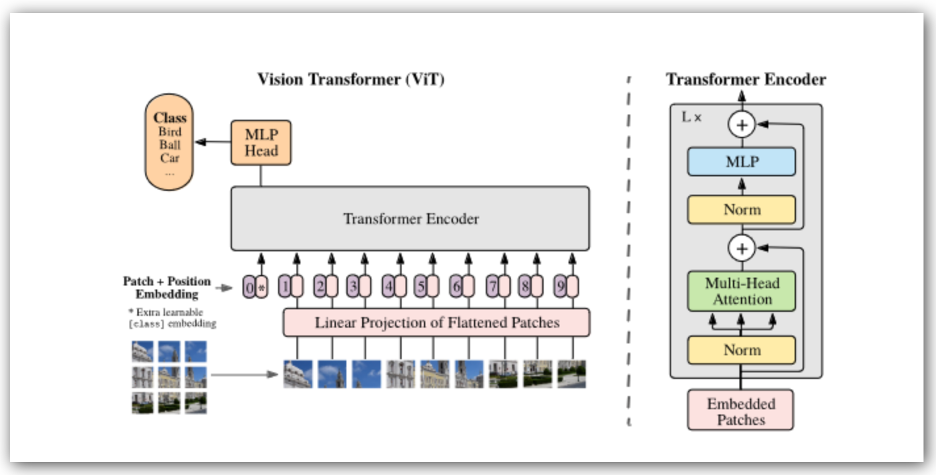
The UNETR (UNET Transformer) is a novel architecture for medical image segmentation that combines the strengths of the U-Net and Transformer models. It leverages the Transformer encoder to capture long-range dependencies in the input data and the U-Net decoder to reconstruct high-resolution segmentation maps. This combination allows the model to achieve high accuracy in segmenting complex structures in medical images, making it particularly effective for tasks like brain tumor segmentation. UNETR outperforms traditional convolutional neural networks by utilizing global context and detailed spatial information.



**Fig: UNETR ARCHITECTURE**

**Transformer architecture:**

The Vision Transformer (ViT) is a novel architecture introduced for image classification, leveraging the transformer model traditionally used in natural language processing. It divides an image into fixed-size patches, linearly embeds each patch, and adds position embeddings to maintain spatial information. These embeddings are then processed by a standard transformer encoder. ViT's key advantage lies in its ability to capture long-range dependencies and global context more effectively than convolutional neural networks (CNNs). It has demonstrated competitive performance on image classification tasks, particularly with large datasets, highlighting its potential as a robust alternative to CNNs.

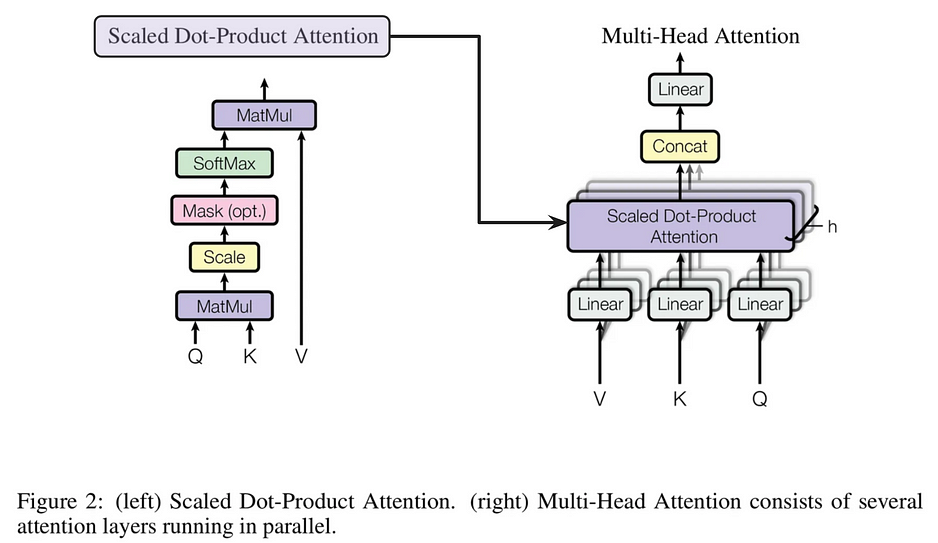


**Multi-Head Attention:**

Multi-head attention extends this idea by using multiple attention heads, each learning different aspects of the input data. Here's how it works:

1. **Linear Projections**: The input embeddings are linearly projected into queries (Q), keys (K), and values (V) for multiple attention heads. This results in multiple sets of Q, K, and V.
2. **Scaled Dot-Product Attention**: Each set of Q, K, and V is fed into the attention mechanism. The attention scores are computed using the dot product of Q and K, scaled by the square root of the dimension of K, and passed through a softmax function to get the attention weights. These weights are then used to weight the values (V).
3. **Concatenation and Linear Transformation**: The outputs of all attention heads are concatenated and linearly transformed to produce the final output.

Attention (Q, K, V) = softmax (​) V



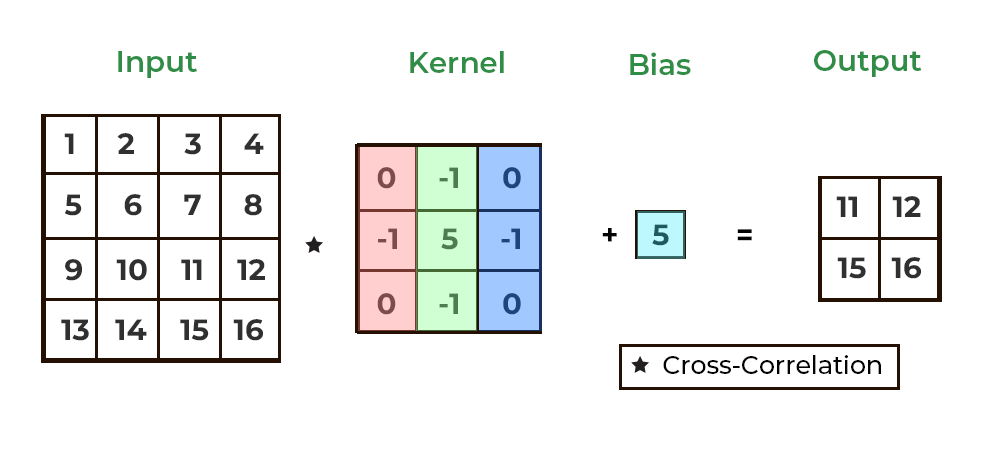
**Convolution:**

Convolution is a fundamental operation in signal processing and is widely used in machine learning, particularly in Convolutional Neural Networks (CNNs). It involves applying a filter (or kernel) to an input to produce an output that emphasizes specific features of the input.

· **Filter (Kernel)**: A small matrix or tensor used to process the input data. For instance, in image processing, this could be a 3x3 or 5x5 matrix that detects edges, textures, or other features.

· **Sliding Window**: The filter slides over the input data (such as an image) in a process called "sliding window" or "striding." At each position, the filter performs element-wise multiplication with the part of the input it covers, followed by summation to produce a single output value.

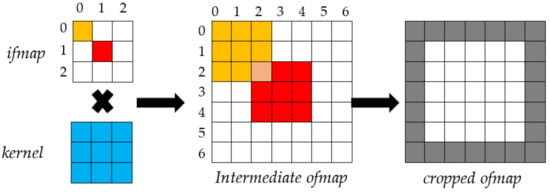
· **Feature Map**: The result of applying the filter across the entire input is called the feature map or activation map. It highlights the presence of specific features in the input.



**Deconvolution:**

Deconvolution, also known as transposed convolution or upsampling, is an operation often used in neural networks to increase the spatial dimensions of the input. This is the reverse process of convolution, aiming to reconstruct or generate higher-resolution data from lower-resolution inputs.

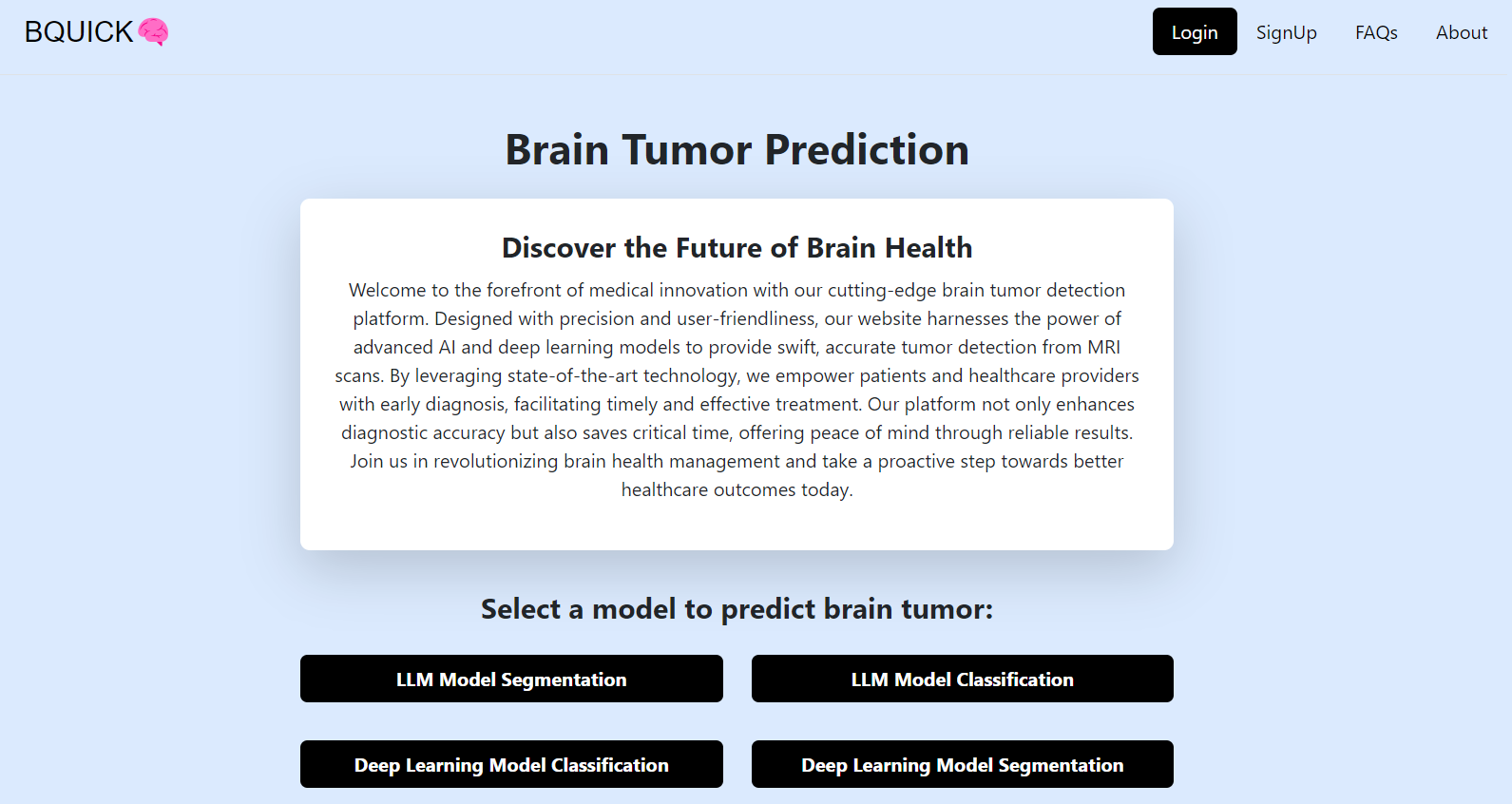
1. **Filter (Kernel)**: Similar to convolution, deconvolution uses a filter (or kernel), but instead of sliding over the input to reduce its size, the filter is applied to upsample the input.
2. **Striding**: In convolution, striding refers to the steps the filter takes as it moves across the input. In deconvolution, striding determines how the input values are spread out to form a larger output.
3. **Output**: The result of the deconvolution is a larger output matrix or feature map, which is typically used to reconstruct the original spatial dimensions of the input before convolution or to generate new high-resolution data.



**FRONTEND DESIGN**

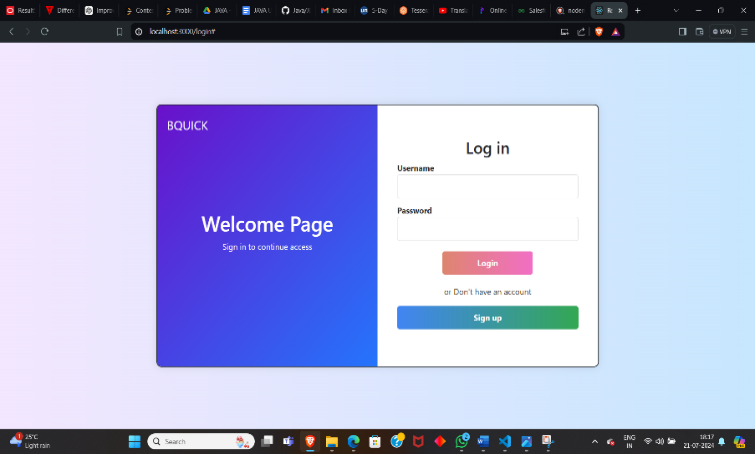
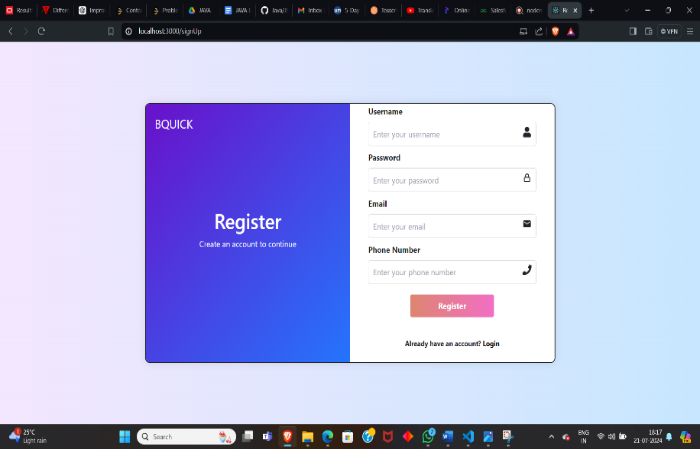
The frontend of this application is built using the **React.js** framework, which is chosen for its modularity, reusability of components, and efficient handling of the user interface. Here’s a comprehensive overview of how the frontend workflow is structured and implemented.

**Component-Based Architecture -** React’s component-based architecture is utilized to create modular and reusable UI components. Each component handles a specific part of the user interface and functionality. The main components include Login, Register, OTP Verification, Dashboard, and Prediction Form.

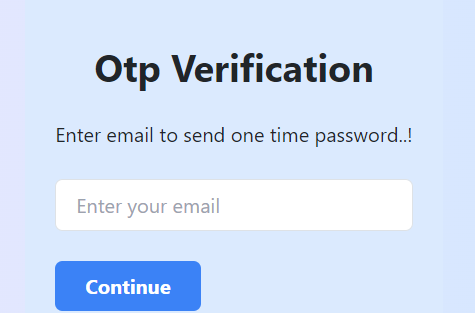
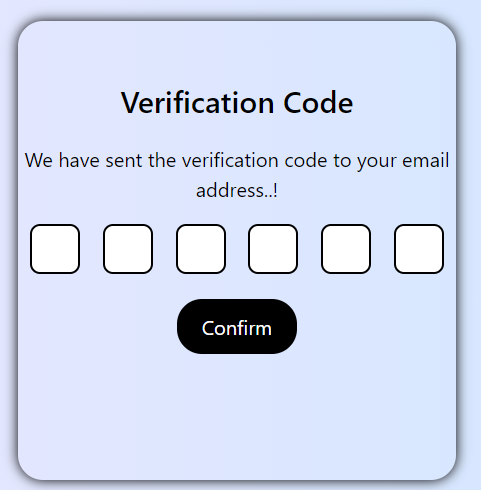


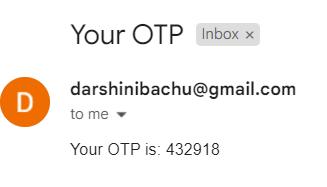
**Fig : HOME PAGE**

**Fig: LOGIN PAGE Fig: SIGN UP PAGE**



**AUTHENTICATION**

** **

** **

**PREDICTION AND RESULT DISPLAY**

**1.UNETR MODEL**

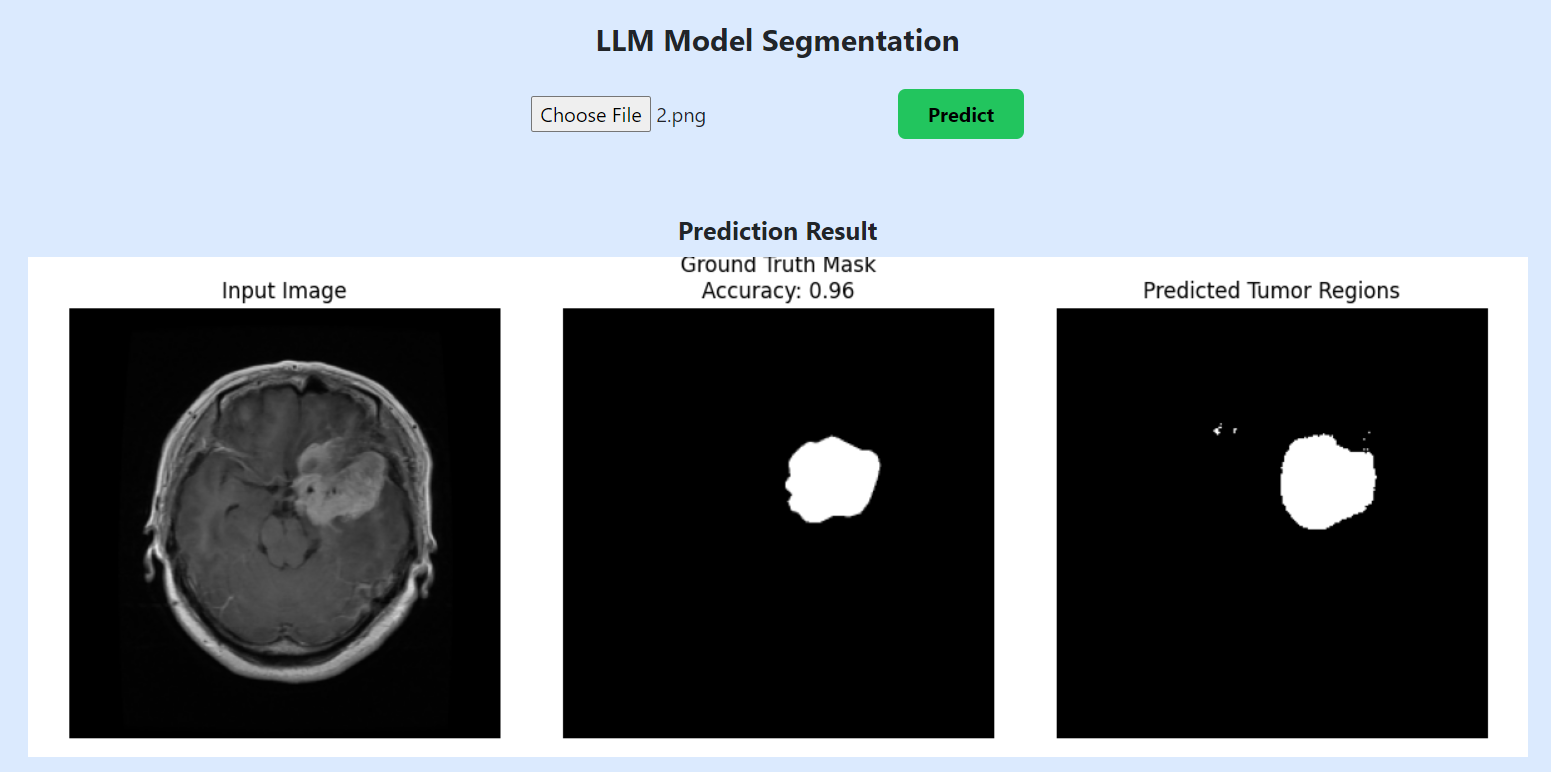


Fig : Prediction result on frontend

**Flask framework:**

Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(\_\_name\_\_)**

This line initializes a new Flask application.

1. **Load the UNETR Model**

Save your pre-trained model using a library like keras. Here, we use keras.models.load\_model() to load the model



1. **Define the Prediction Endpoint**

Flask routes are used to define the endpoints of your web service. In this case, we will create an

endpoint to handle predictions

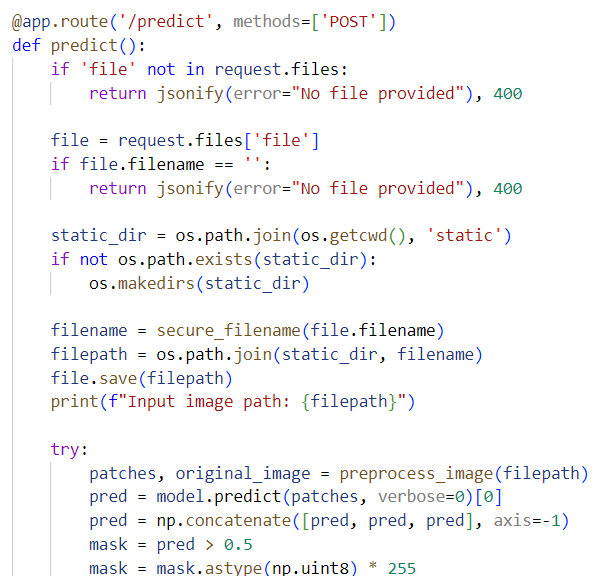
****

Fig :Example of route

1. **Run the Flask Application**



**PART-2:** **Brain Tumour Segmentation Using UNET**

**Objective**: The objective of the project "Generative Neurologist for Brain Tumour Detection" is to develop a Deep Learning model that accurately identifies and segments brain tumours from MRI images. By leveraging cutting-edge techniques like the UNET architecture, the project aims to improve diagnostic precision, enabling early detection and treatment planning.

**Technology Stack:**

|  |  |
| --- | --- |
| Frontend | React |
| Backend | MongoDB Atlas |
| Authentication | Nodemailer |
| Model | UNET |
| Backend API | Flask |

**Dataset Description:**

**Name:** Brain tumour Segmentation

**Description:** Dataset contains MRI images and their corresponding mask images for brain tumour segmentation tasks. The dataset includes 3064 sample collection of T1-weighted contrast-enhanced MRI images along with manually segmented binary masks indicating the presence of brain tumours. This dataset is designed for training and evaluating deep learning models for the segmentation of brain tumours from MRI scans.

**Download Link:** [**https://www.kaggle.com/datasets/nikhilroxtomar/brain-tumor-segmentation**](https://www.kaggle.com/datasets/nikhilroxtomar/brain-tumor-segmentation)

**Preprocessing:**

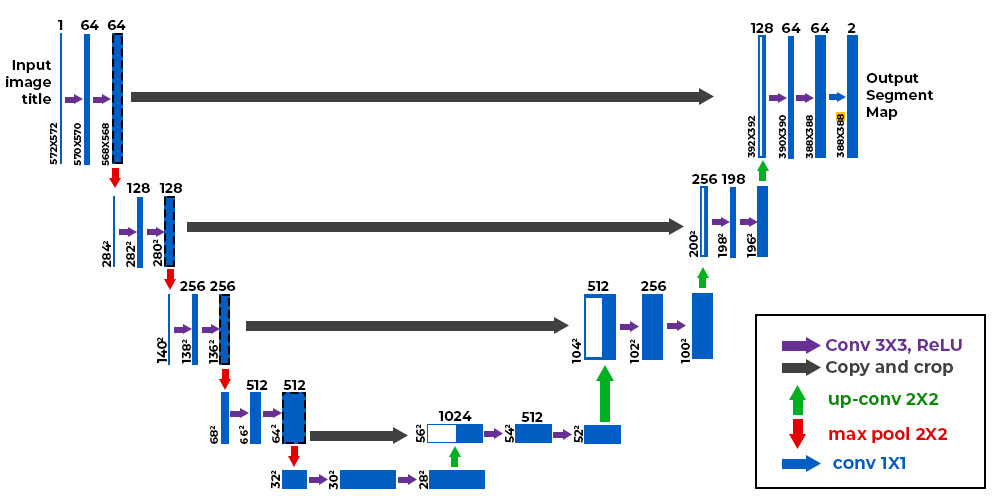
Data preprocessing for image segmentation involves several critical steps to prepare raw image data for effective segmentation tasks. One essential technique is data augmentation, which artificially increases the size of the training dataset by applying various transformations. This includes rotations, horizontal and vertical flips ,scaling, cropping, translation, color jittering, and adding noise. These transformations enhance the model's robustness and generalization ability by exposing it to a broader variety of image scenarios, improving its performance on unseen data.



**Fig:Data Augmentation**

**Model used:**

The U-Net is a convolutional neural network designed for biomedical image segmentation. It has a U-shaped architecture with an encoder that captures features at multiple scales and a decoder that reconstructs high-resolution segmentation maps. Skip connections between corresponding encoder and decoder layers preserve spatial information, enabling precise localization of structures. This makes the U-Net highly effective for tasks like brain tumor segmentation, achieving high performance even with limited training data. Its ability to accurately segment complex medical images has made it a widely used model in medical image analysis.



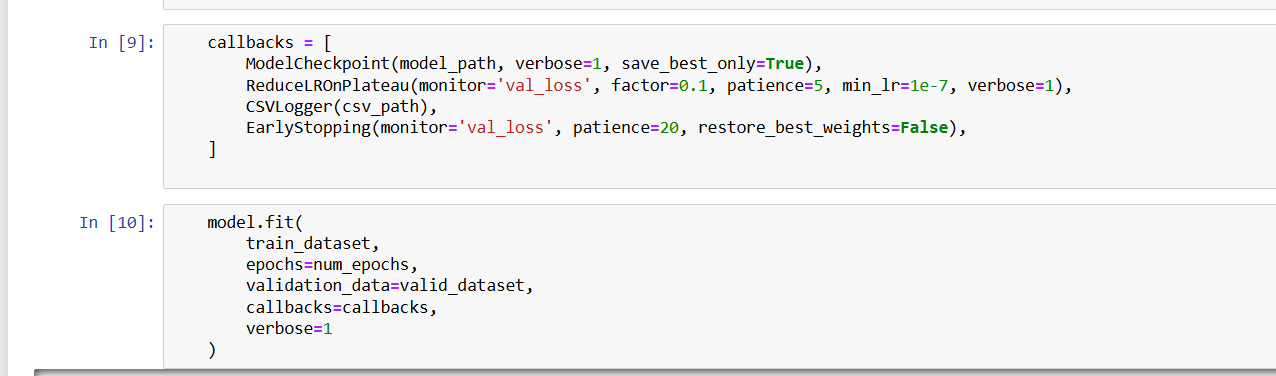
**Fig: UNET ARCHITECTURE**



**Fig: UNet Model**

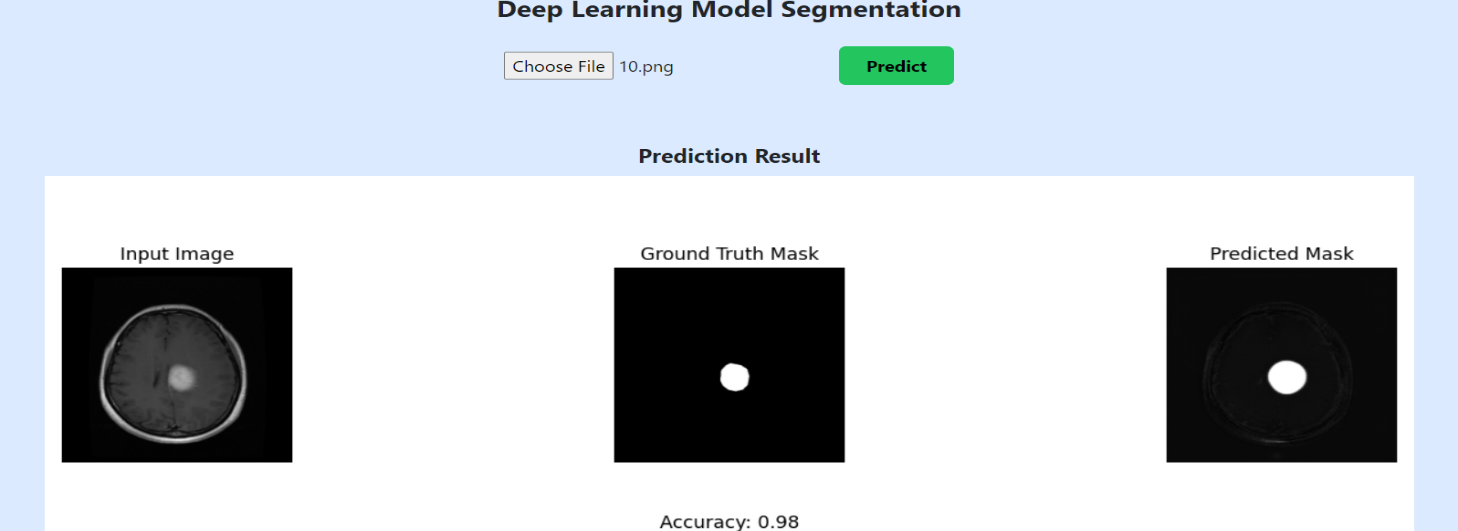
**Model Checkpoint:**

A model checkpoint is a saved state of a machine learning model at a specific point during the training process. It includes the model's weights, optimizer state, and sometimes other relevant information like the current epoch number and the training state.



**Fig: ModelCheckpoint**

**Few Model Predictions:**



**Flask framework:**

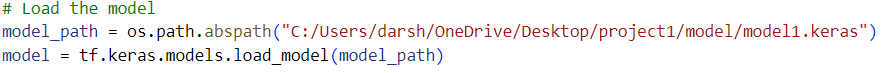
Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(\_\_name\_\_)**

This line initializes a new Flask application.

1. **Load the UNET Model**

Save your pre-trained model using a library like keras. Here, we use keras.models.load\_model() to load the model

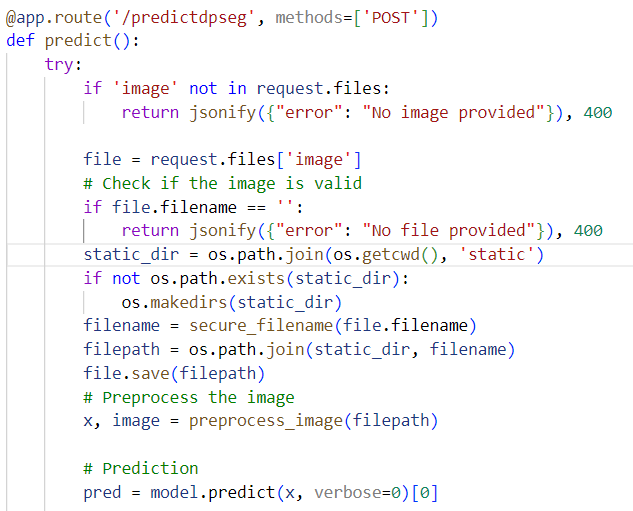


1. **Define the Prediction Endpoint**

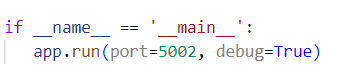
Flask routes are used to define the endpoints of your web service. In this case, we will create an

endpoint to handle predictions

Example of route

****

1. **Run the Flask Application**



**PART-3:** **Brain Tumour Classification Using VGG-16 Architecture**

**Objective**: The objective of the project "Generative Neurologist for Brain Tumour Detection" is to develop a Deep Learning model that accurately identifies and classify brain tumours from MRI images. By leveraging cutting-edge techniques like the VGG-16 architecture, the project aims to improve diagnostic precision, enabling early detection and treatment planning.

**Technology Stack:**

|  |  |
| --- | --- |
| Frontend | React |
| Backend | MongoDB Atlas |
| Authentication | Node mailer |
| Model | UNET |
| Backend API | Flask |

**Dataset Description:**

**Name:** Brain tumour Classification

**Description:** Dataset contains MRI images and their corresponding images for brain tumour classification tasks. The dataset includes 9000 sample collection contrast-enhanced MRI images along with manually segmented binary masks indicating the presence of brain tumours. This dataset is designed for training and evaluating deep learning models for the classification of brain tumours from MRI scans.

**Download Link:** '/kaggle/input/brain-tumor/Neuro\_9k'

**Preprocessing:**

Data preprocessing for image classification involves several critical steps to prepare raw image data for effective classification tasks. One essential technique is data augmentation, which artificially increases the size of the training dataset by applying various transformations. This includes rotations, horizontal and vertical flips ,scaling, cropping, translation, color jittering, and adding noise. These transformations enhance the model's robustness and generalization ability by exposing it to a broader variety of image scenarios, improving its performance on unseen data.

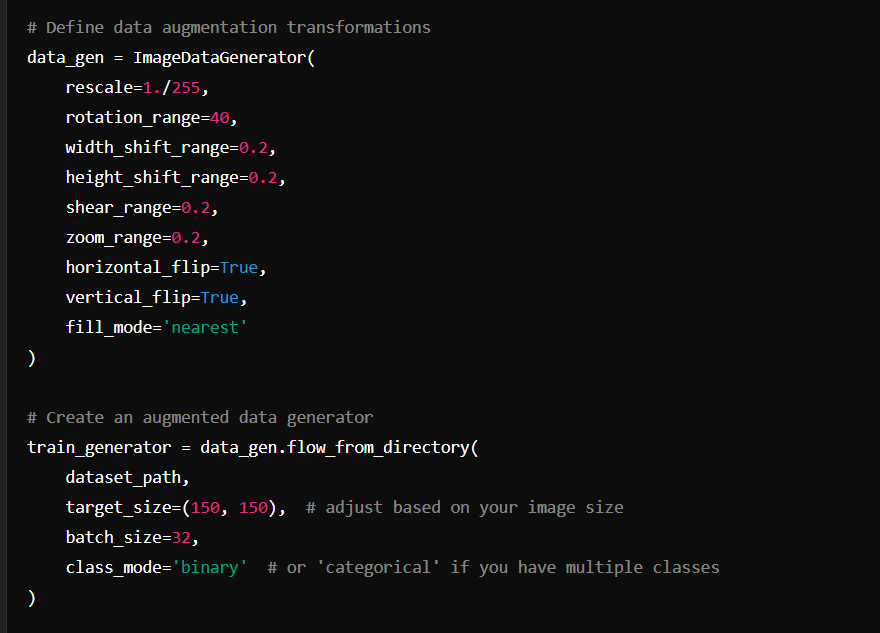
****

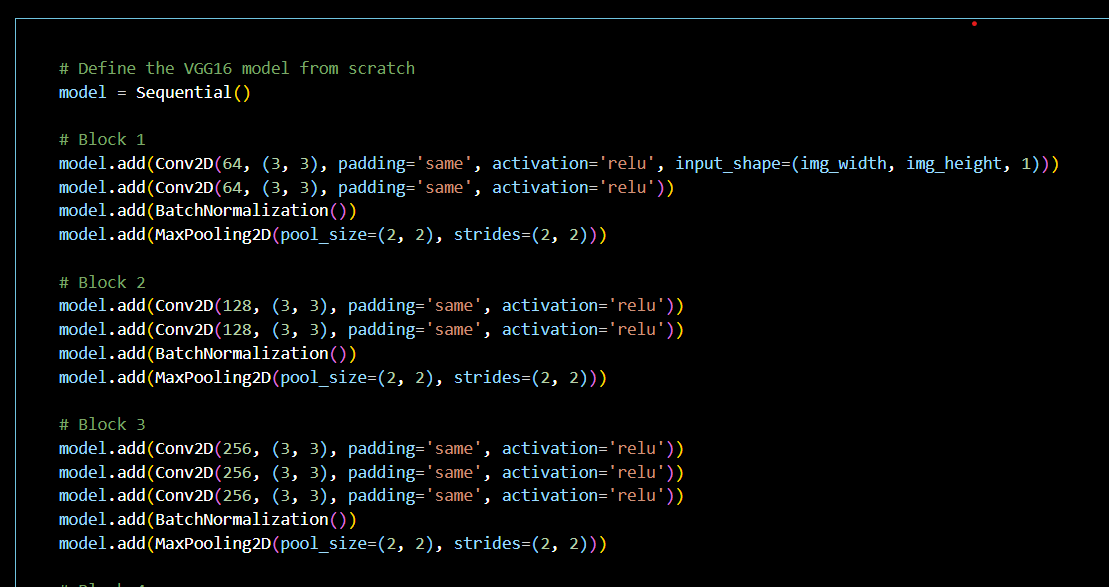
Fig: Data Augmentation Code

**MODEL Used:**

VGG-16 is a convolutional neural network architecture consisting of 16 weight layers: 13 convolutional layers and 3 fully connected layers. The network uses small 3x3 convolutional filters stacked together, which enables it to capture intricate patterns in the input images. It follows a pattern of convolutional layers followed by max-pooling layers, which progressively reduce the spatial dimensions while increasing the depth of feature maps. The final layers are fully connected, ending with a sigmoid layer for classification. VGG-16's design allows it to achieve high performance in image recognition tasks while maintaining a straightforward and uniform structure.



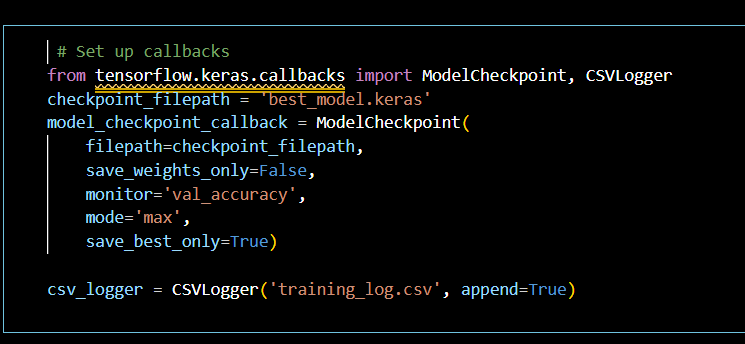
**Fig: VGG-16 Architect**

****

**Fig: VGG-16 Architecture Code Snippet**

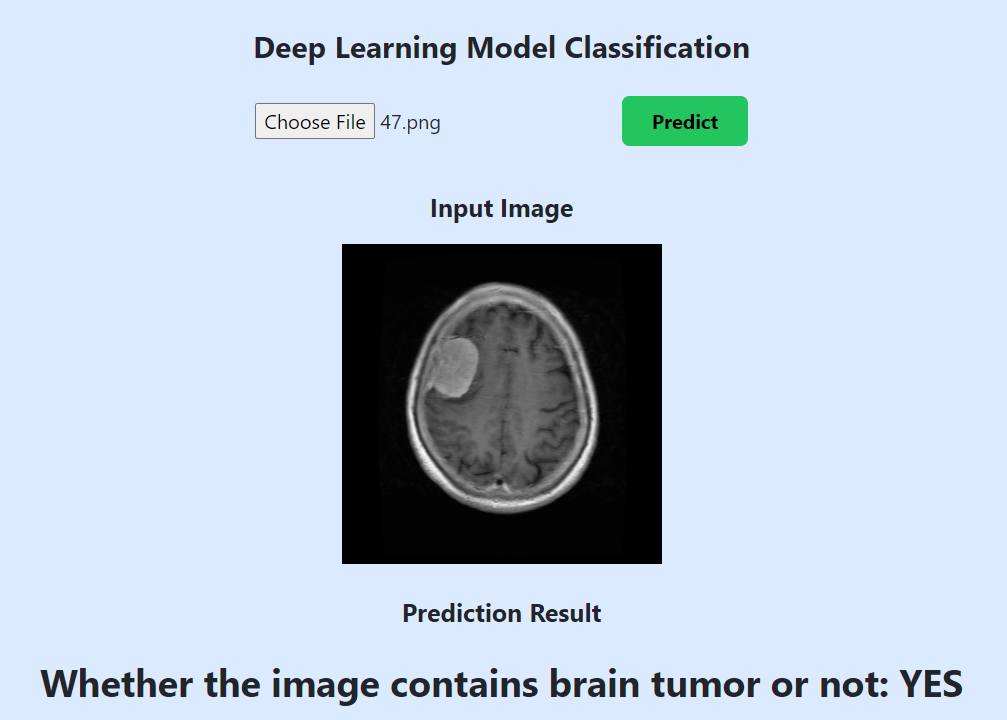
**Model Checkpoint:**

A model checkpoint is a saved state of a machine learning model at a specific point during the training process. It includes the model's weights, optimizer state, and sometimes other relevant information like the current epoch number and the training state. Checkpoints allow the training process to be paused and resumed without loss of progress, providing a safeguard against interruptions such as hardware failures or the need to fine-tune training parameters. They are also used to save the best-performing models based on validation metrics, ensuring that the final model is optimized for the task at hand.



**Fig: ModelCheckpoint**

**FEW MODEL PREDICTIONS:**



**Flask framework:**

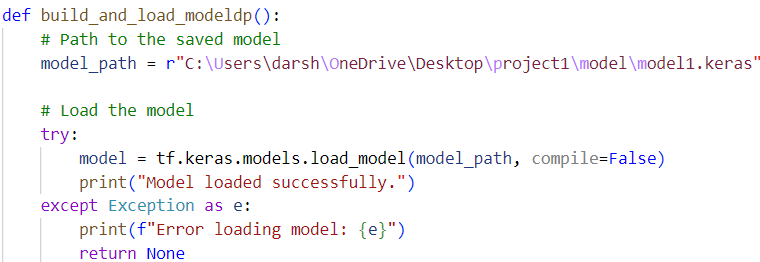
Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(\_\_name\_\_)**

This line initializes a new Flask application.

1. **Load the VGG-16 Model**

Save your pre-trained model using a library like keras. Here, we use keras.models.load\_model() to load the model

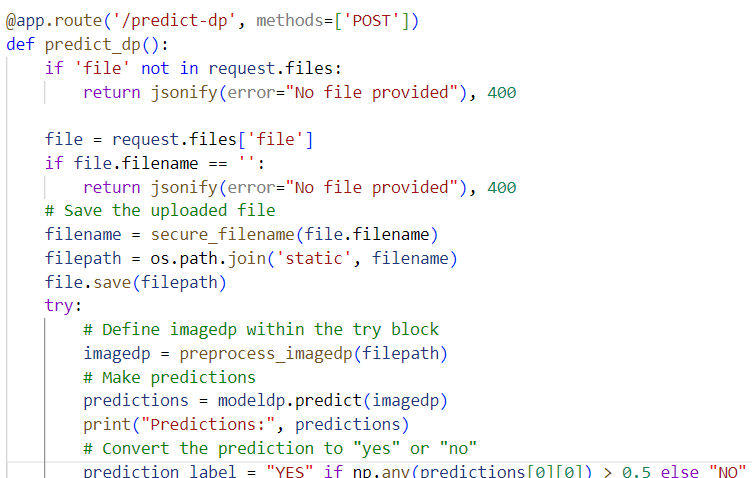


1. **Define the Prediction Endpoint**

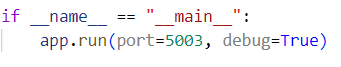
Flask routes are used to define the endpoints of your web service. In this case, we will create an

endpoint to handle predictions

Example of route

****

1. **Run the Flask Application**



**Flask framework:**

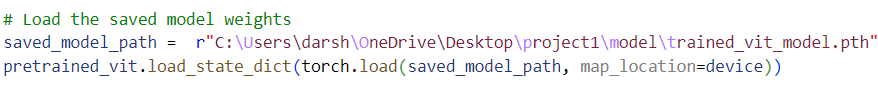
Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(\_\_name\_\_)**

This line initializes a new Flask application.

1. **Load the VIT TRANSFORMER Model**

Save your pre-trained model using a library like torch. Here, we use torch.load to load the model.

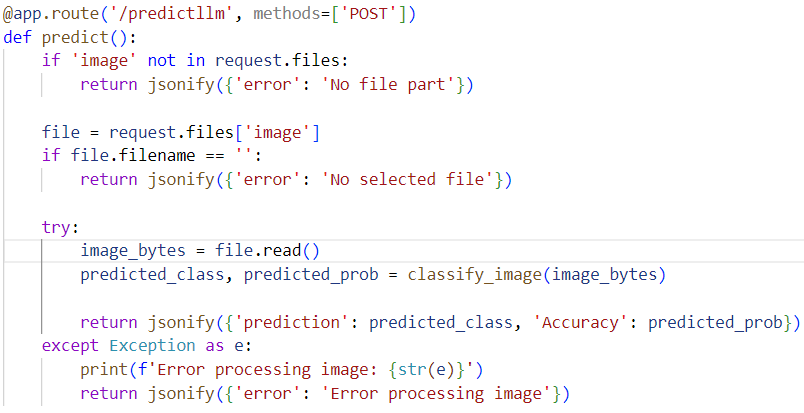


1. **Define the Prediction Endpoint**

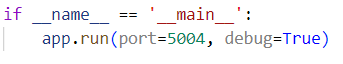
Flask routes are used to define the endpoints of your web service. In this case, we will create an

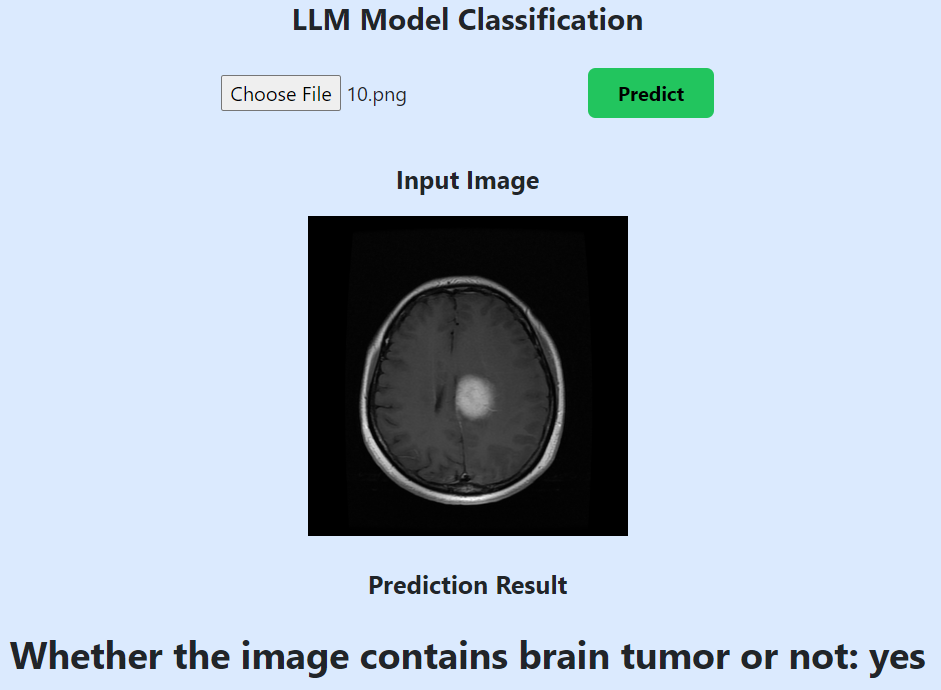
endpoint to handle predictions.

Example of route

****

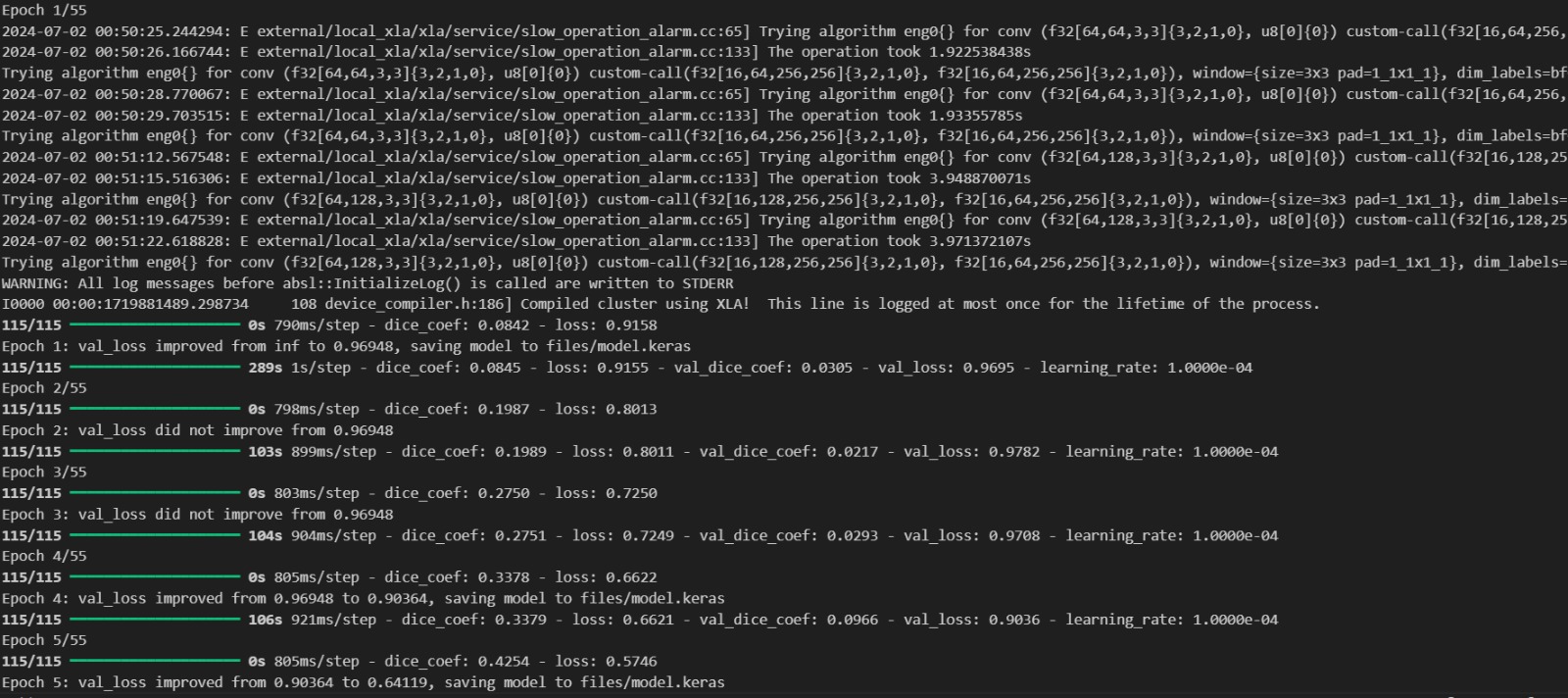
1. **Run the Flask Application**



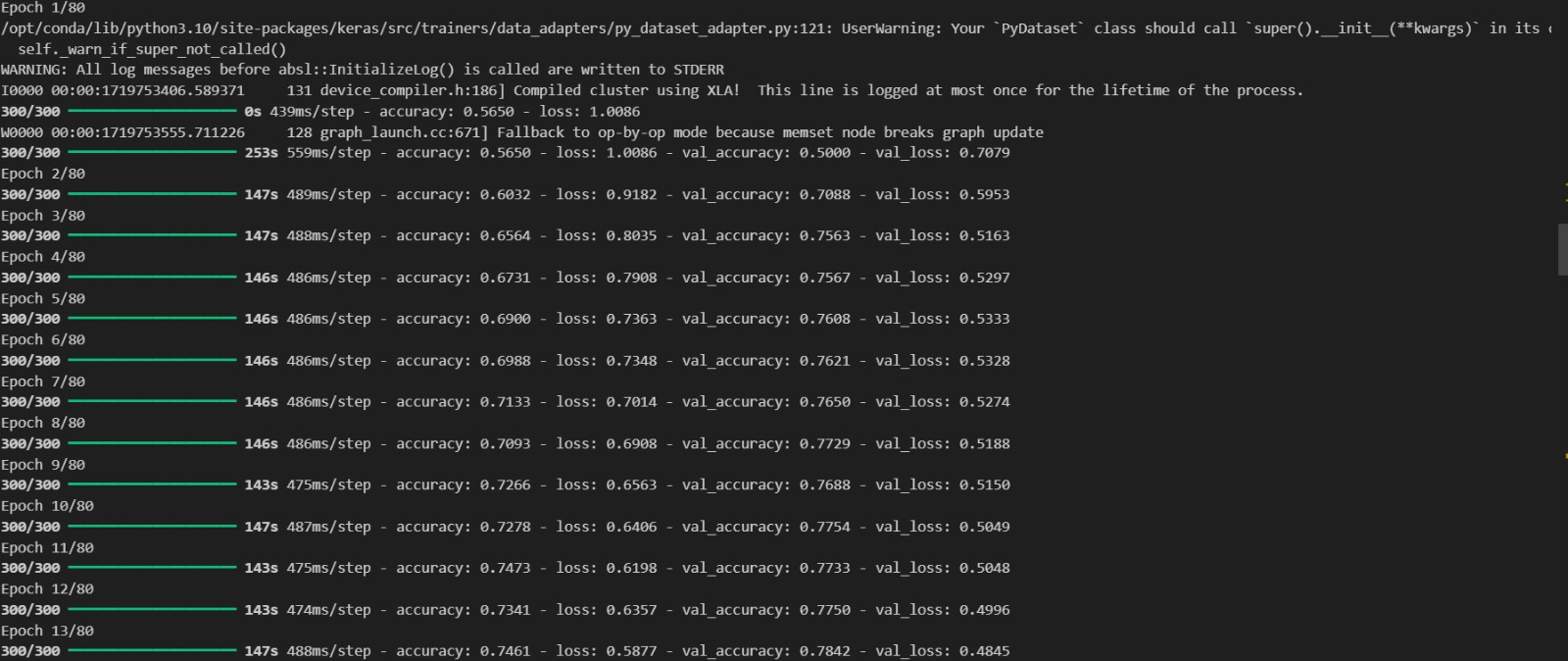


**CPU VS GPU**

**CPU:**



**GPU:**



**PLATFORMS USED:**

1.Vs code , jupyter and pycharm for code implementation.

2.Kaggle for dataset and GPU training.

**REFERENCES:**

1.LLM SEGMENTATION - <https://www.youtube.com/watch?v=hOnX9KPimsM>

2.LLM CLASSIFICATION – <https://www.youtube.com/watch?v=awyWND506NY>

3.DEEP LEARNING SEGMENTATION – <https://www.youtube.com/watch?v=mgdB7WezqbU&list=PLHYn9gDxQOpixfTc_9GdE9H4mbkExEziB>

4. DEEP LEARNING CLASSIFICATION – <https://www.youtube.com/watch?v=awyWND506NY>

**SUMMARY:**

This project focuses on developing a system for detecting and analysing brain tumours using deep learning and large language models (LLMs). The UNETR model is employed for accurate tumour segmentation from MRI scans. The LLM interprets these results to generate detailed medical reports, assisting clinicians in diagnosis and treatment planning. The system integrates a user-friendly frontend, deep learning, and LLM components to enhance the accuracy and efficiency of brain tumour detection.