KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

PROJECT SCHOOL CERTIFICATE

Title: Generative Neurologist

Mentor: Dr. Devika Rubi

Duration: 02/04/2024 - 23/07/2024

Name: LINGAMPELLI SRINIVAS

Class: CSE-A

Roll Number: 23BD5A0504

Year: II-II

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

TABLE OF CONTENTS

PART-1: <u>Brain Tumour Segmentation Using UNETR</u>

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	Node Mailer
Model	UNETR
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

Preprocessing:

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

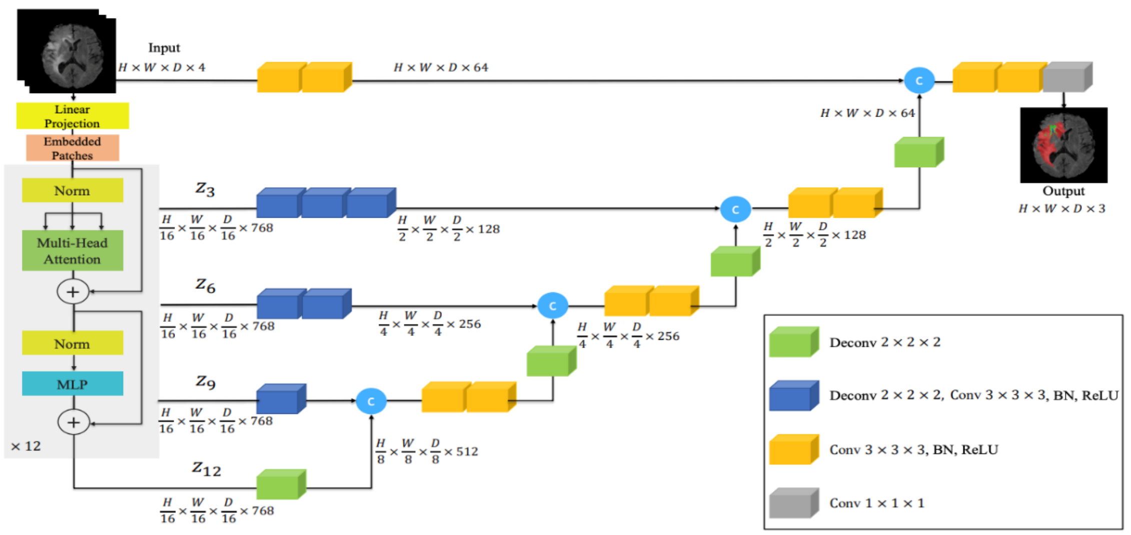
2. **Divide into Patches:** Split the resized (256 x 256) image into (16 x16) patches, resulting in 256 patches.

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.



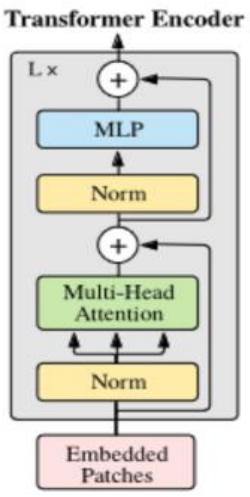
Input
$H \times W \times D \times 4$
Linear Projection
Embedded Patches
Norm
Multi-Head Attention
Norm
MLP
× 12
z_3
z_6
z_9
z_{12}
$\frac{H}{16} \times \frac{W}{16} \times \frac{D}{16} \times 768$
$H \times W \times D \times 64$
$\frac{H}{2} \times \frac{W}{2} \times \frac{D}{2} \times 128$
$\frac{H}{4} \times \frac{W}{4} \times \frac{D}{4} \times 256$
$\frac{H}{8} \times \frac{W}{8} \times \frac{D}{8} \times 512$
c
Output
$H \times W \times D \times 3$
Deconv 2 × 2 × 2
Deconv 2 × 2 × 2, Conv 3 × 3 × 3, BN, ReLU
Conv 3 × 3 × 3, BN, ReLU
Conv 1 × 1 × 1

Fig. 1. Overview of the UNETR architecture. We extract sequence representations of different layers in the transformer and merge them with the decoder via skip connections. Output sizes demonstrated for patch dimension $N = 16$ and embedding size $C = 768$.

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.

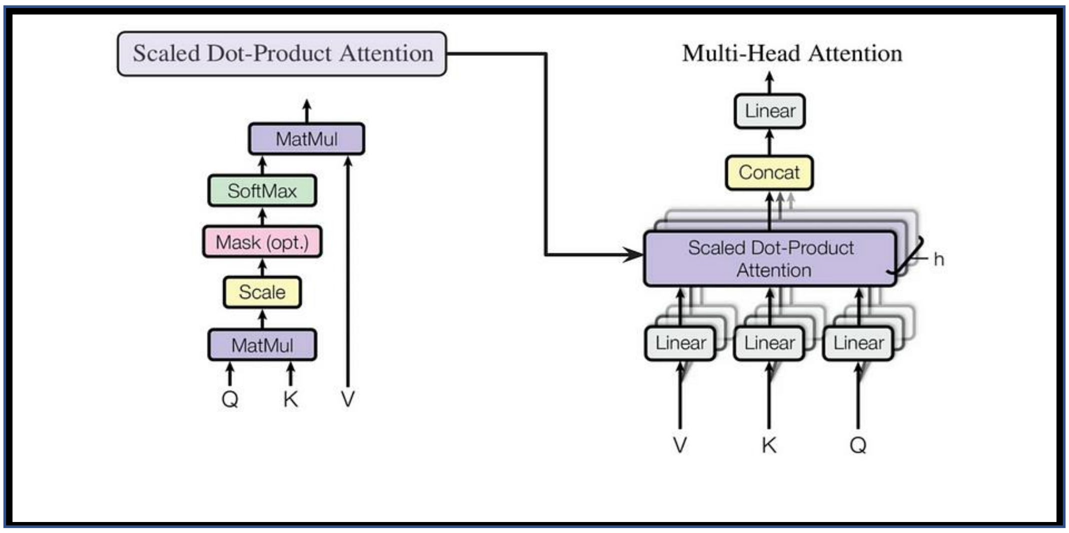


Multi-Head Attention:

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.

$$\text{Attention (Q, K, V)} = \text{softmax}\ (\frac{QK^T}{\sqrt{d_k}})\ V$$



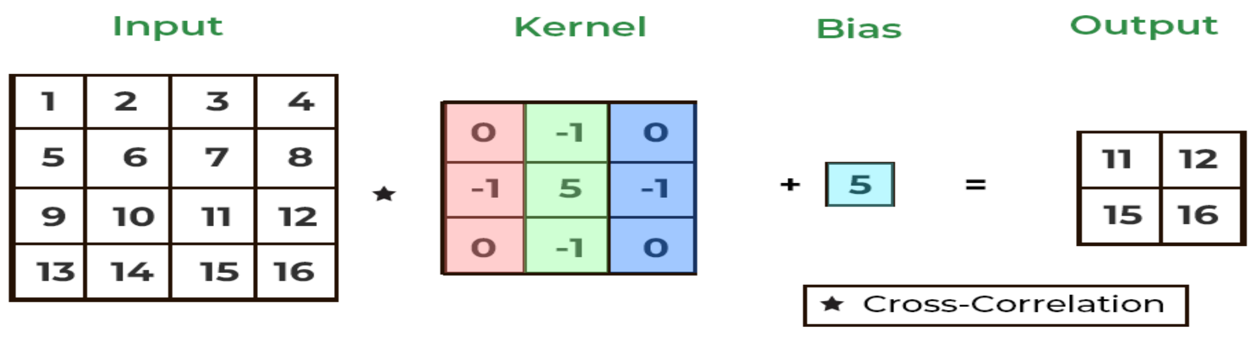
Convolution:

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.

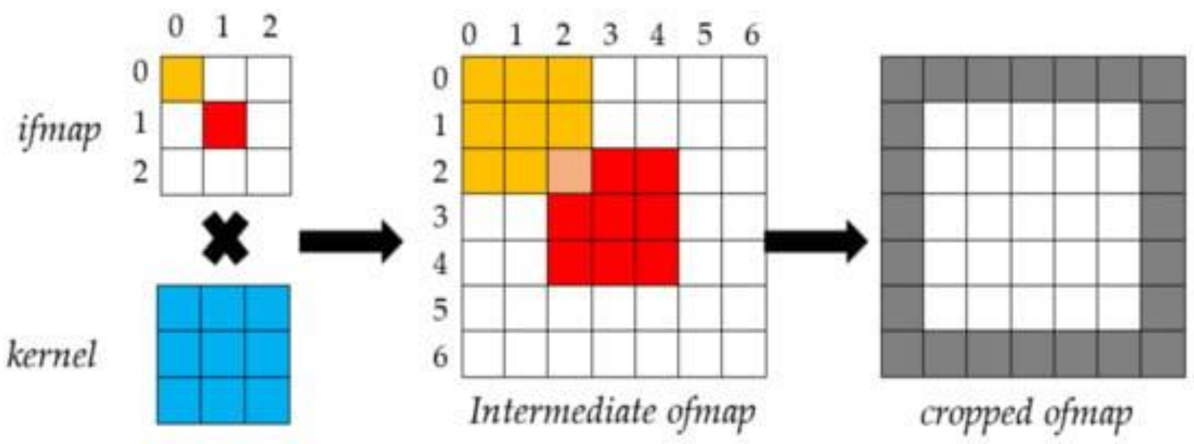
· **Feature Map**: The result of applying the filter across the entire input is called the feature map or activation map. It highlights the presence of specific features in the input.



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.

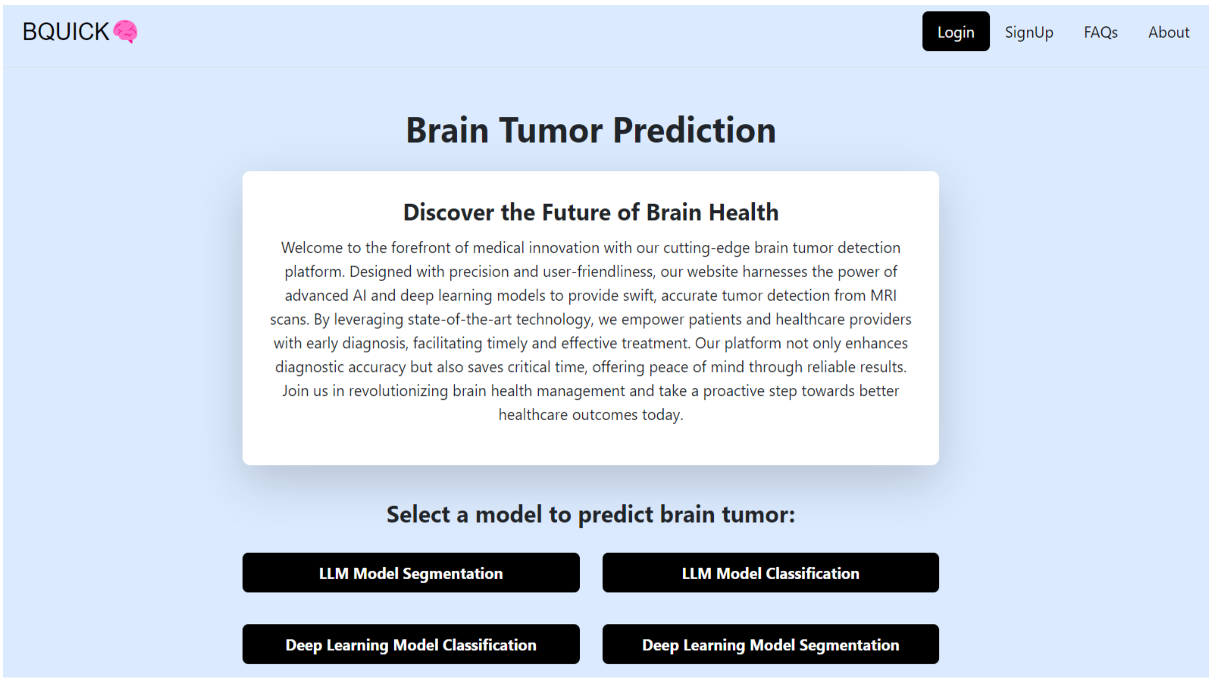
1. **Filter (Kernel)**: Similar to convolution, deconvolution uses a filter (or kernel), but instead of sliding over the input to reduce its size, the filter is applied to upsample the input.
2. **Striding**: In convolution, striding refers to the steps the filter takes as it moves across the input. 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.



BQUICK

Login SignUp FAQs About

Brain Tumor Prediction

Discover the Future of Brain Health

Welcome to the forefront of medical innovation with our cutting-edge brain tumor detection platform. Designed with precision and user-friendliness, our website harnesses the power of advanced AI and deep learning models to provide swift, accurate tumor detection from MRI scans. By leveraging state-of-the-art technology, we empower patients and healthcare providers with early diagnosis, facilitating timely and effective treatment. Our platform not only enhances diagnostic accuracy but also saves critical time, offering peace of mind through reliable results. Join us in revolutionizing brain health management and take a proactive step towards better healthcare outcomes today.

Select a model to predict brain tumor:

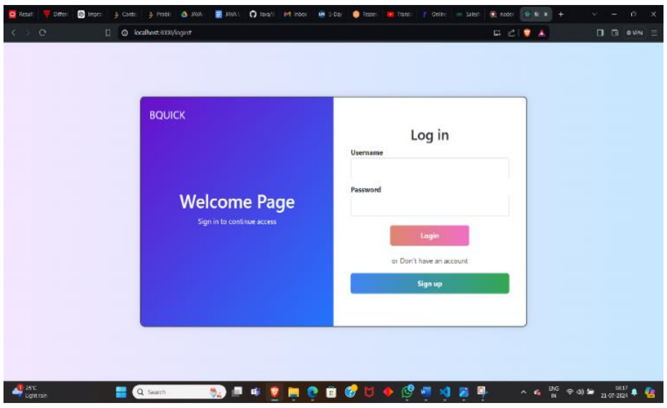
LLM Model Segmentation | LLM Model Classification

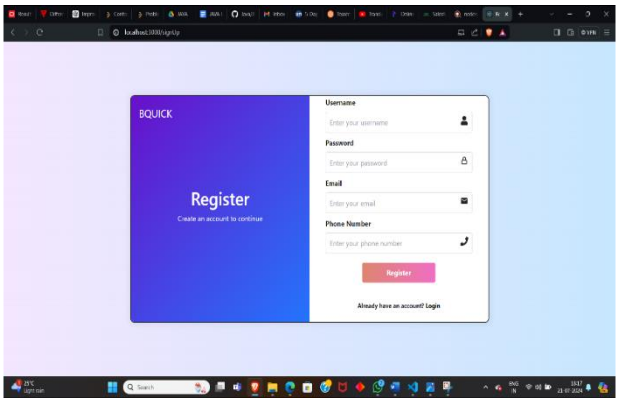
Deep Learning Model Classification | Deep Learning Model Segmentation

Fig : HOME PAGE

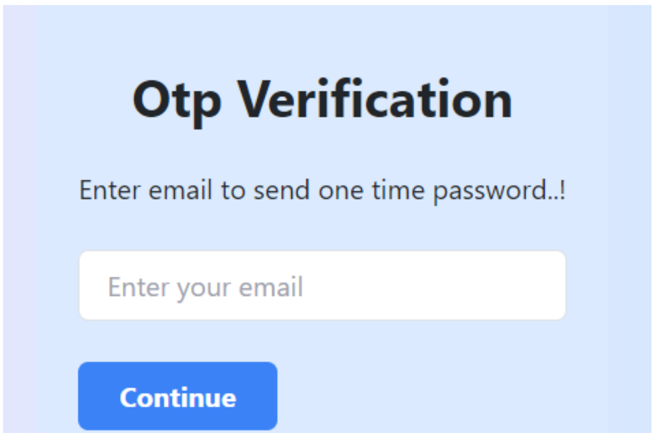
Fig: LOGIN PAGE

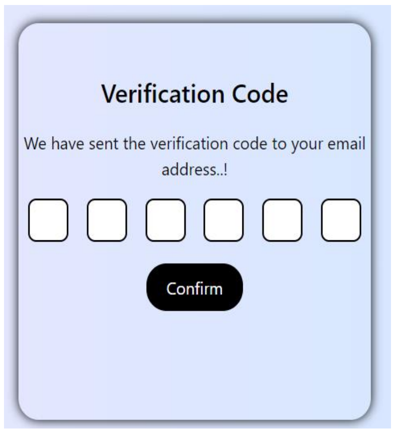
Fig: SIGN UP PAGE

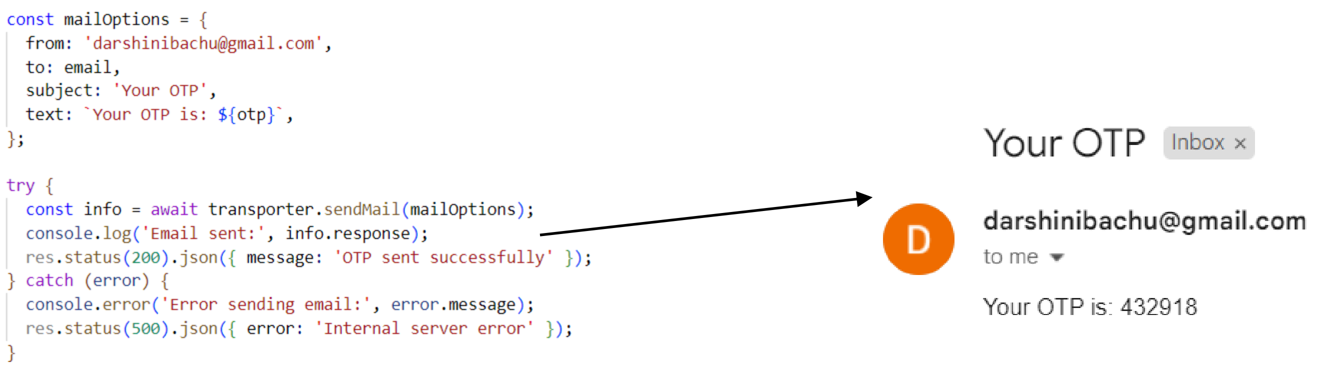




AUTHENTICATION







```
const mailOptions = {
  from: 'darshinibachu@gmail.com',
  to: email,
  subject: 'Your OTP',
  text: `Your OTP is: ${otp}`,
};

try {
  const info = await transporter.sendMail(mailOptions);
  console.log('Email sent:', info.response);
  res.status(200).json({ message: 'OTP sent successfully' });
} catch (error) {
  console.error('Error sending email:', error.message);
  res.status(500).json({ error: 'Internal server error' });
}
```

Your OTP Inbox ×

D darshinibachu@gmail.com
to me

Your OTP is: 432918

PREDICTION AND RESULT DISPLAY

1.UNETR MODEL

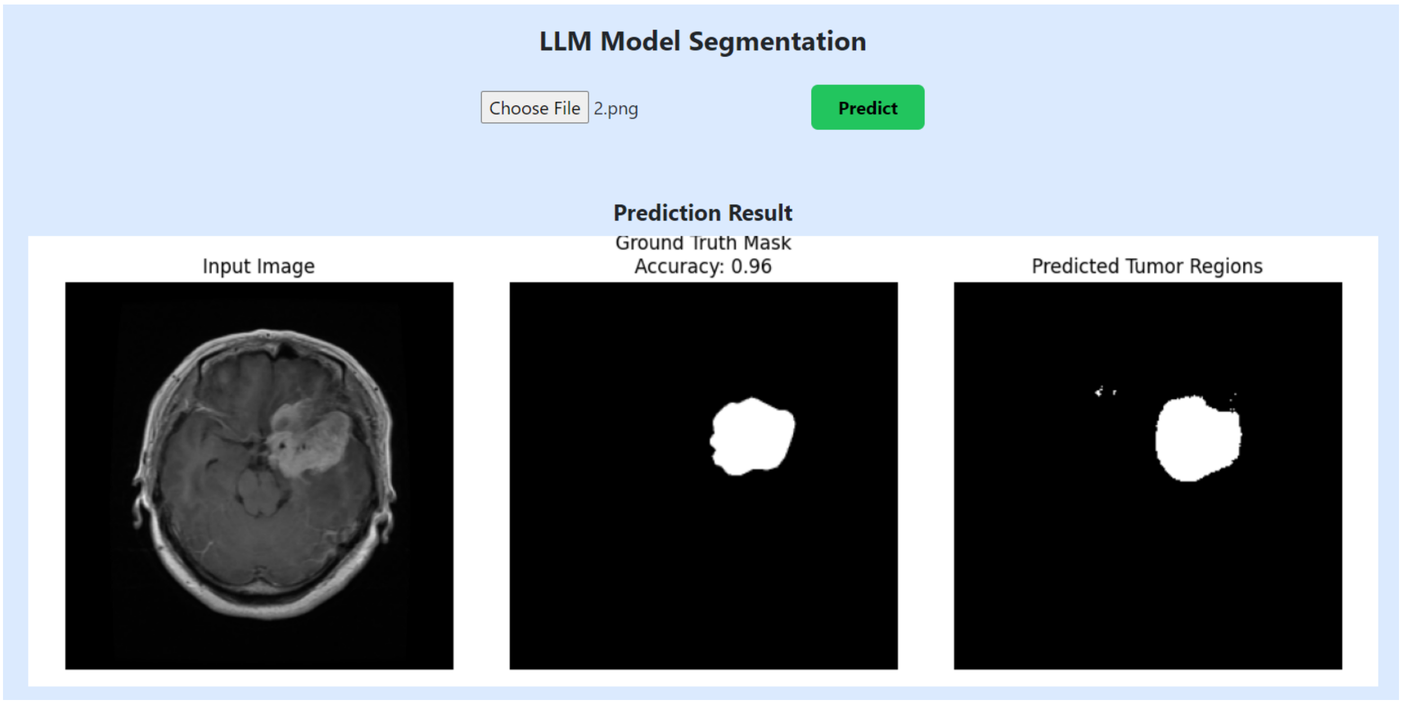


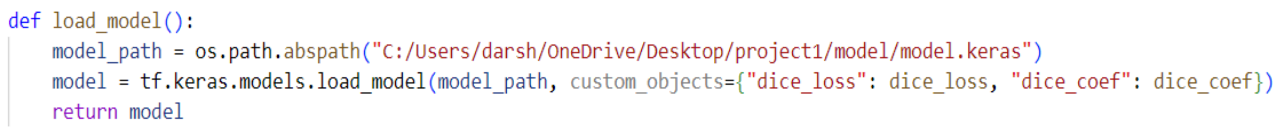
Fig : Prediction result on frontend

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.

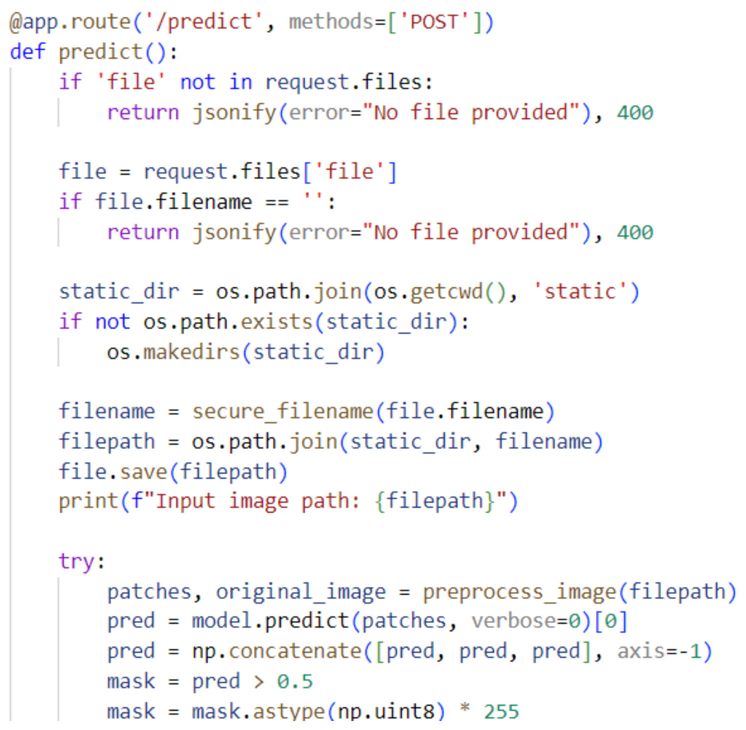
1. **Initialize Flask App : app = Flask(__name__)**
   This line initializes a new Flask application.

2. **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



```
def load_model():
    model_path = os.path.abspath("C:/Users/darsh/OneDrive/Desktop/project1/model/model.keras")
    model = tf.keras.models.load_model(model_path, custom_objects={"dice_loss": dice_loss, "dice_coef": dice_coef})
    return model
```

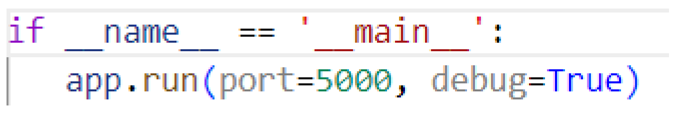
3. **Define the Prediction Endpoint**
   Flask routes are used to define the endpoints of your web service. In this case, we will create an endpoint to handle predictions



```
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify(error="No file provided"), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify(error="No file provided"), 400

    static_dir = os.path.join(os.getcwd(), 'static')
    if not os.path.exists(static_dir):
        os.makedirs(static_dir)

    filename = secure_filename(file.filename)
    filepath = os.path.join(static_dir, filename)
    file.save(filepath)
    print(f"Input image path: {filepath}")

    try:
        patches, original_image = preprocess_image(filepath)
        pred = model.predict(patches, verbose=0)[0]
        pred = np.concatenate([pred, pred, pred], axis=-1)
        mask = pred > 0.5
        mask = mask.astype(np.uint8) * 255
```

Fig : Example of route

4. **Run the Flask Application**



```
if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

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:

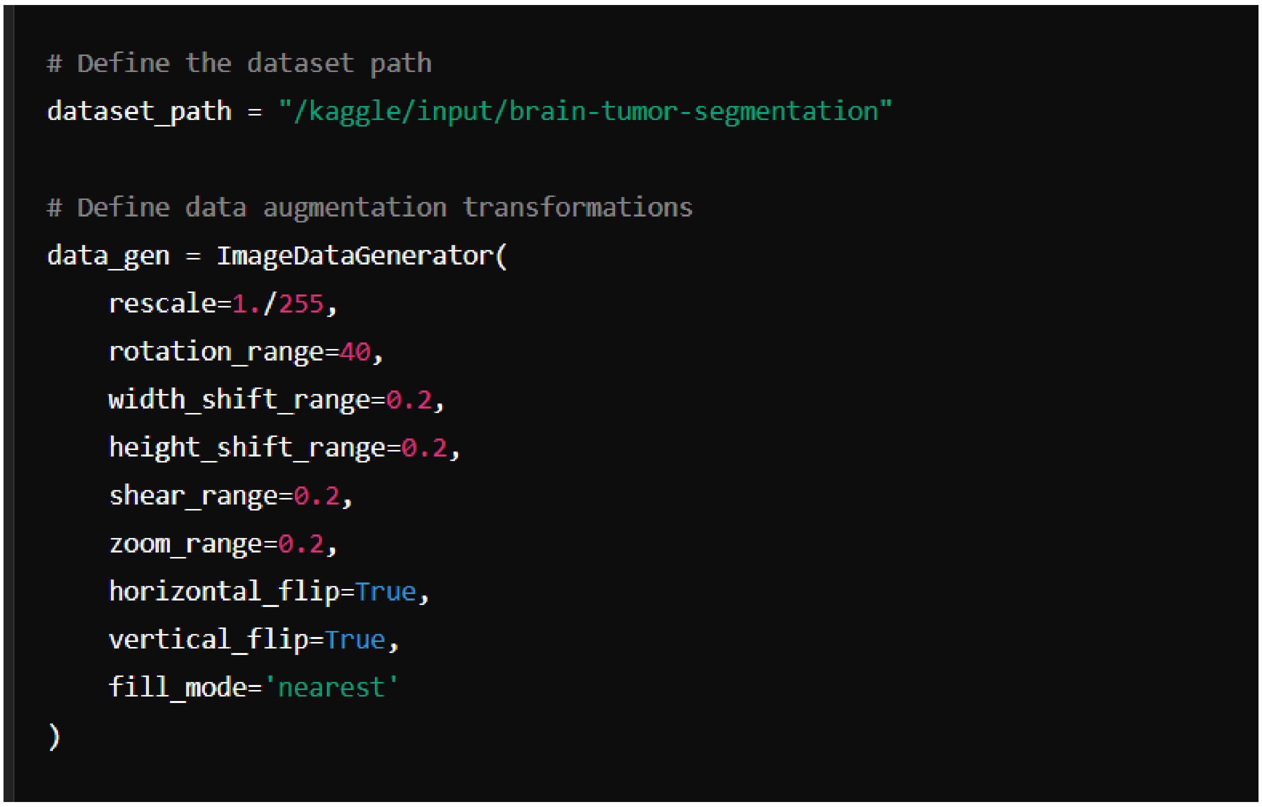
Name: Brain tumour Segmentation

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**

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.



```
# Define the dataset path
dataset_path = "/kaggle/input/brain-tumor-segmentation"

# Define data augmentation transformations
data_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

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.

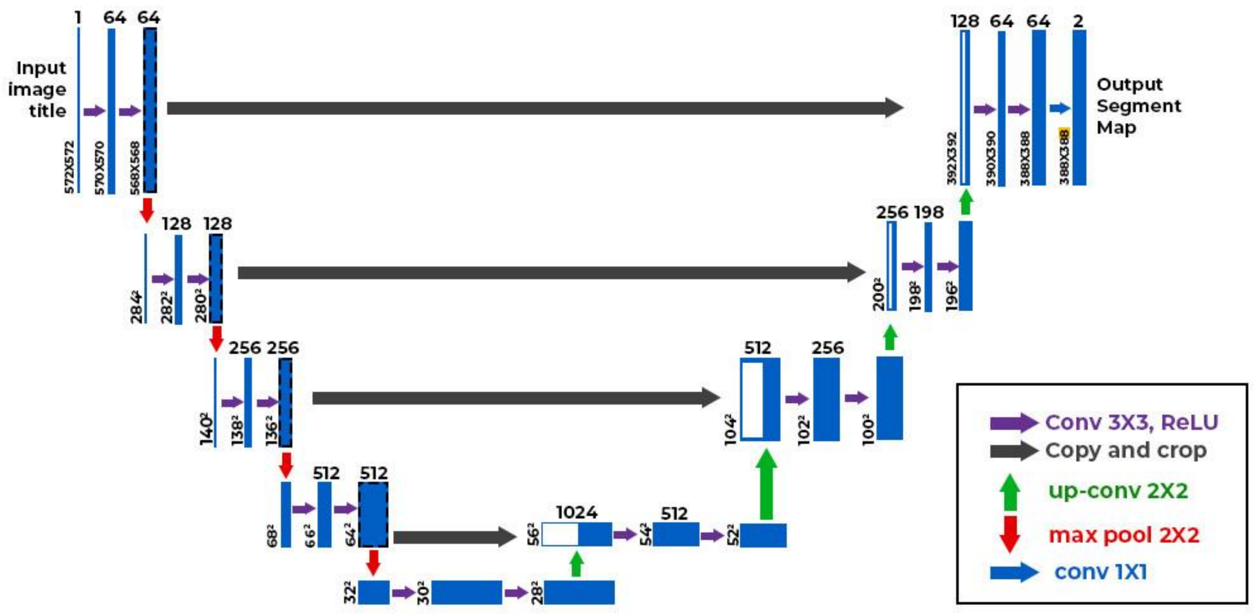


Fig: UNET ARCHITECTURE

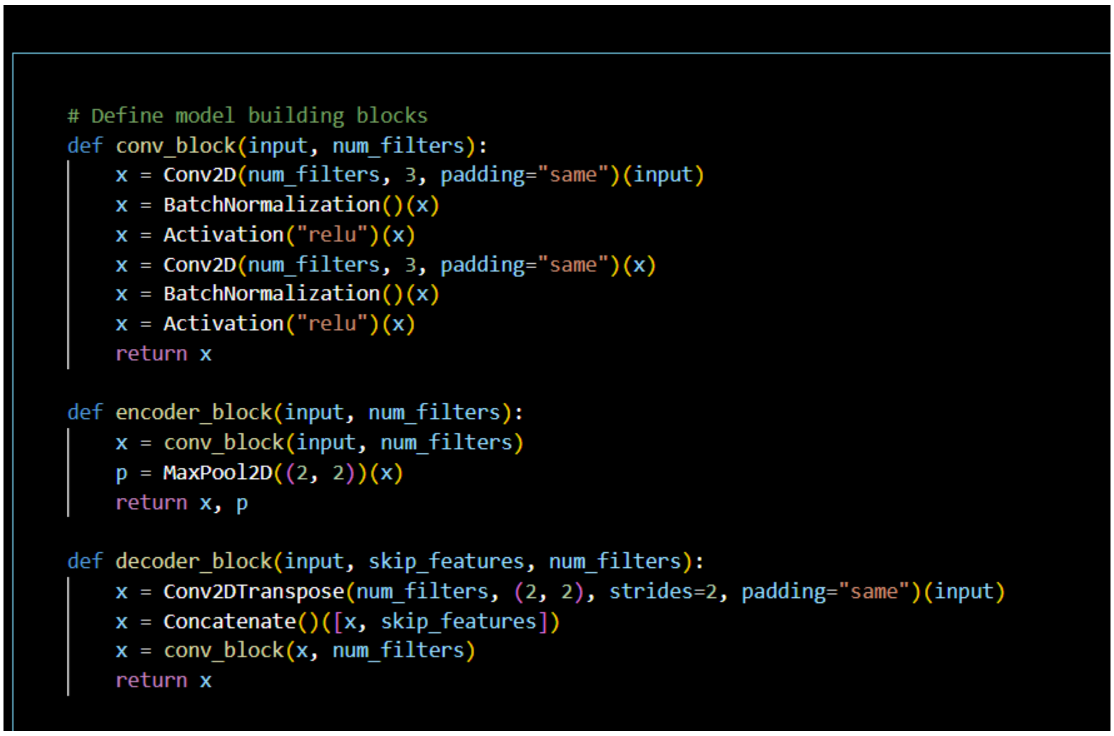

```
# Define model building blocks
def conv_block(input, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(input)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = Conv2D(num_filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    return x

def encoder_block(input, num_filters):
    x = conv_block(input, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x
```

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.

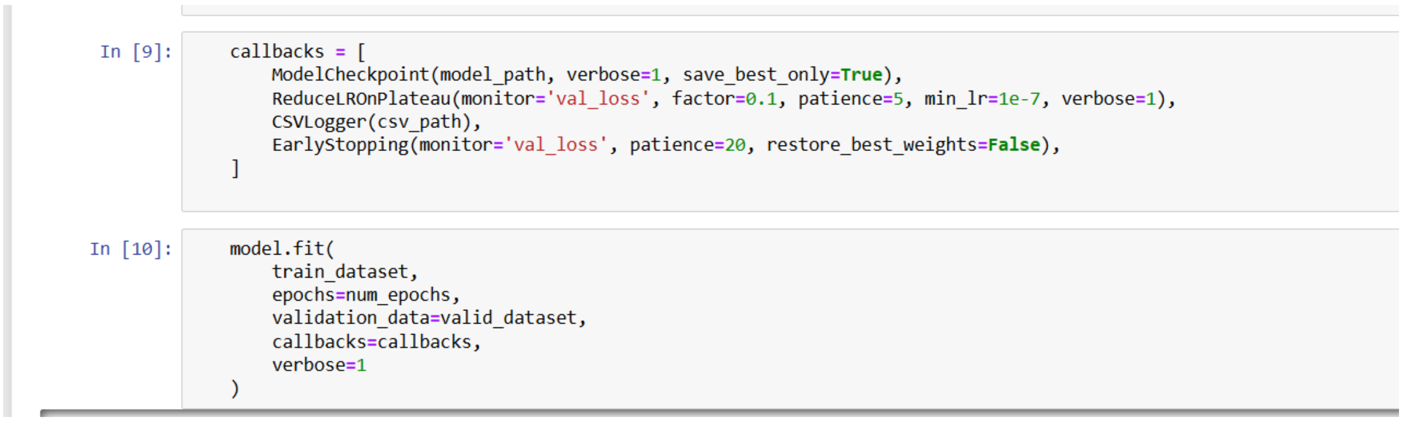
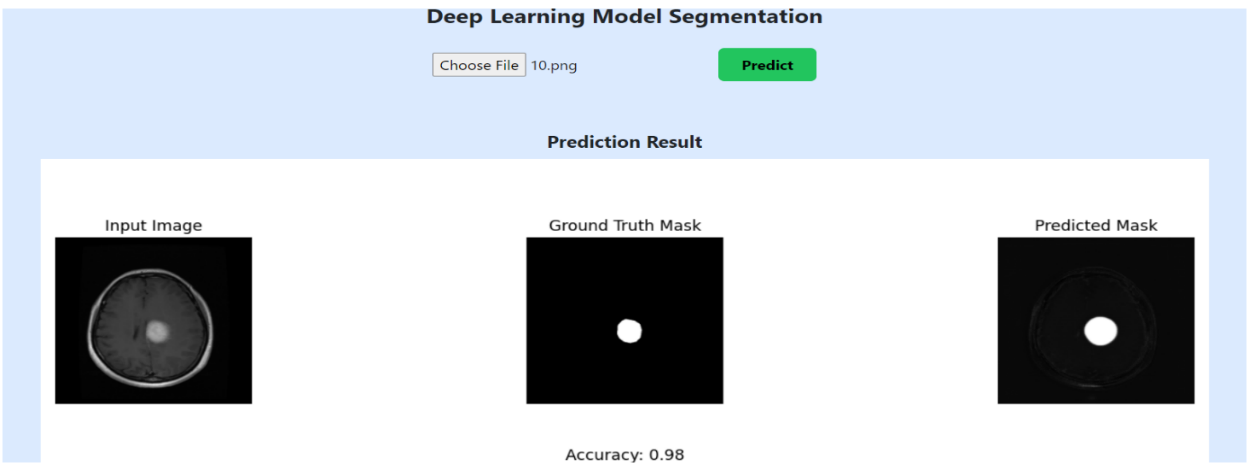

```
In [9]:   callbacks = [
              ModelCheckpoint(model_path, verbose=1, save_best_only=True),
              ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, min_lr=1e-7, verbose=1),
              CSVLogger(csv_path),
              EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=False),
          ]

In [10]:  model.fit(
              train_dataset,
              epochs=num_epochs,
              validation_data=valid_dataset,
              callbacks=callbacks,
              verbose=1
          )
```

Fig: ModelCheckpoint

Few Model Predictions:



Flask framework:

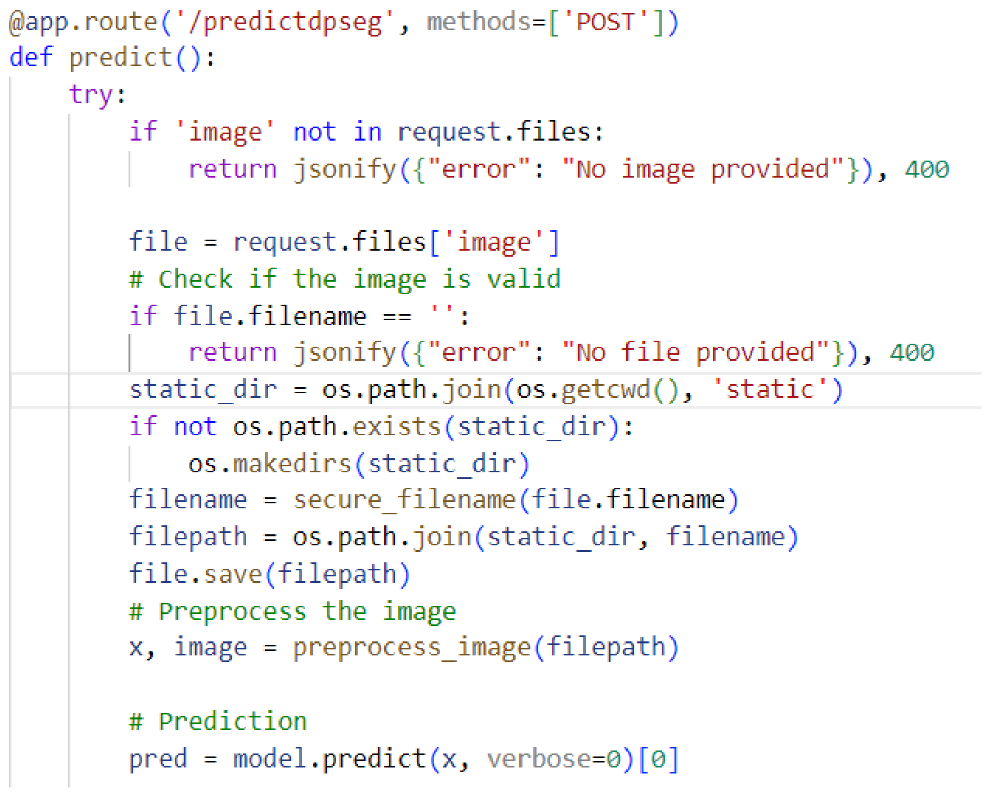
Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(__name__)**
   This line initializes a new Flask application.

2. **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

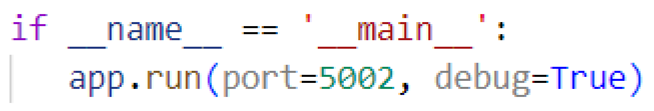
```
# Load the model
model_path = os.path.abspath("C:/Users/darsh/OneDrive/Desktop/project1/model/model1.keras")
model = tf.keras.models.load_model(model_path)
```

3. **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



```
@app.route('/predictdpseg', methods=['POST'])
def predict():
    try:
        if 'image' not in request.files:
            return jsonify({"error": "No image provided"}), 400

        file = request.files['image']
        # Check if the image is valid
        if file.filename == '':
            return jsonify({"error": "No file provided"}), 400
        static_dir = os.path.join(os.getcwd(), 'static')
        if not os.path.exists(static_dir):
            os.makedirs(static_dir)
        filename = secure_filename(file.filename)
        filepath = os.path.join(static_dir, filename)
        file.save(filepath)
        # Preprocess the image
        x, image = preprocess_image(filepath)

        # Prediction
        pred = model.predict(x, verbose=0)[0]
```

4. **Run the Flask Application**



```
if __name__ == '__main__':
    app.run(port=5002, debug=True)
```

PART-3: <u>Brain Tumour Classification Using VGG-16 Architecture</u>

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:

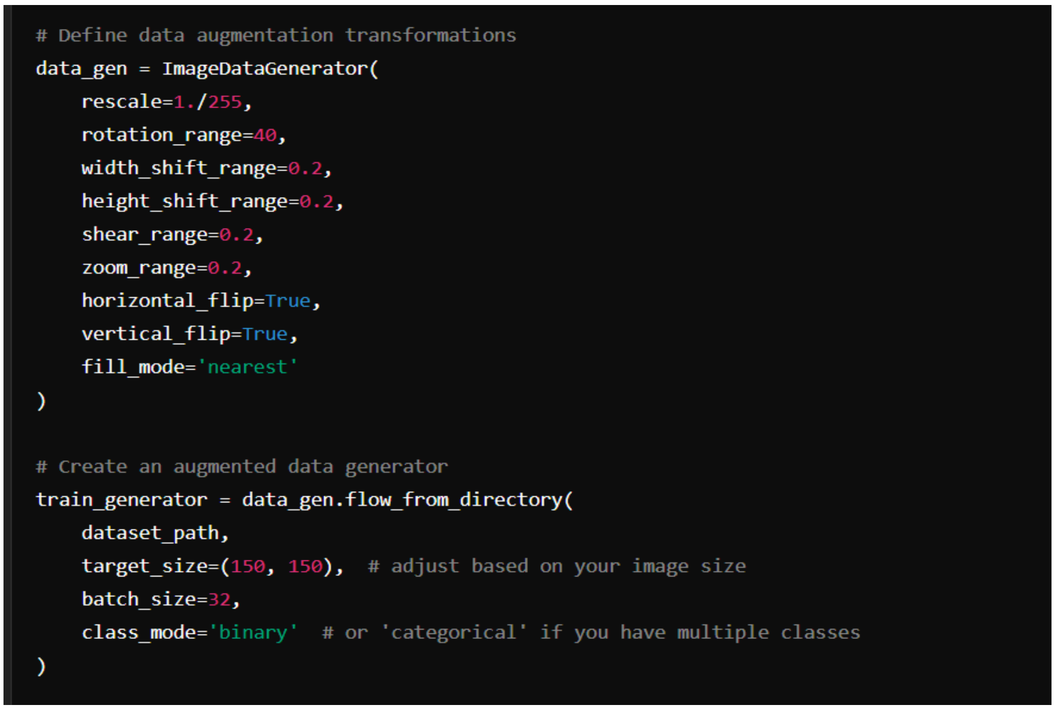
Name: Brain tumour Classification

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.



```
# Define data augmentation transformations
data_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

# Create an augmented data generator
train_generator = data_gen.flow_from_directory(
    dataset_path,
    target_size=(150, 150),  # adjust based on your image size
    batch_size=32,
    class_mode='binary'  # or 'categorical' if you have multiple classes
)
```

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.

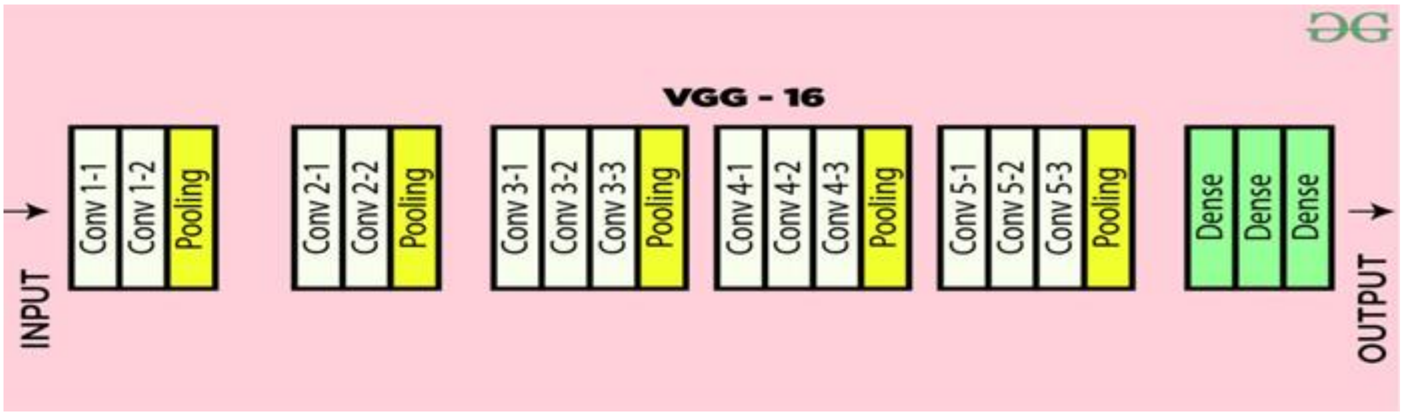
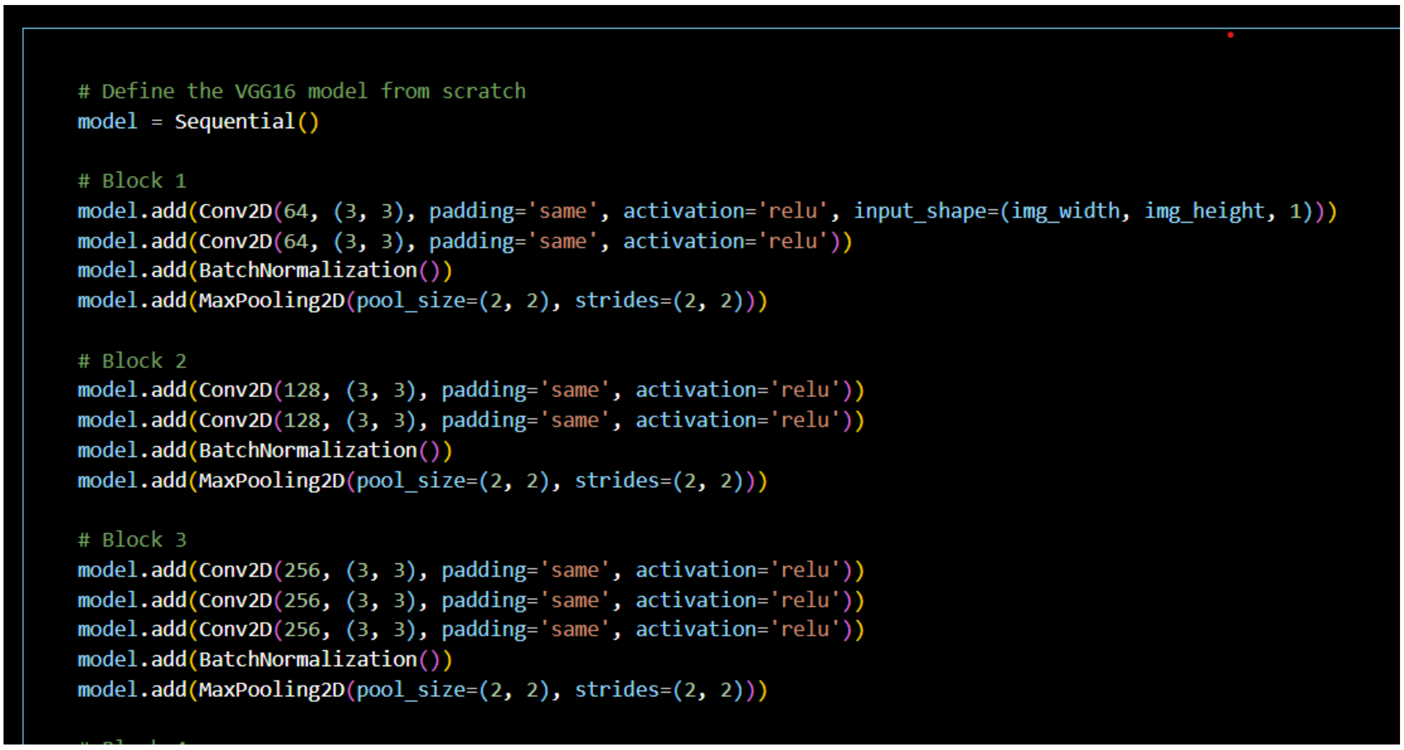


Fig: VGG-16 Architect

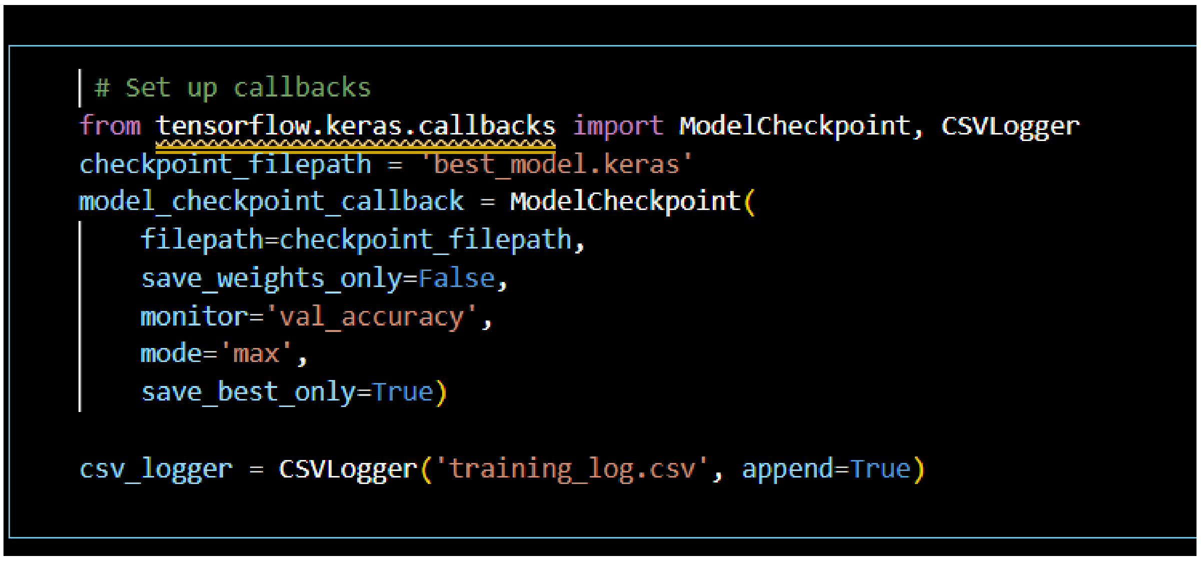


```python
# Define the VGG16 model from scratch
model = Sequential()

# Block 1
model.add(Conv2D(64, (3, 3), padding='same', activation='relu', input_shape=(img_width, img_height, 1)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Block 2
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Block 3
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

Fig: VGG-16 Architecture Code Snippet

Model Checkpoint:

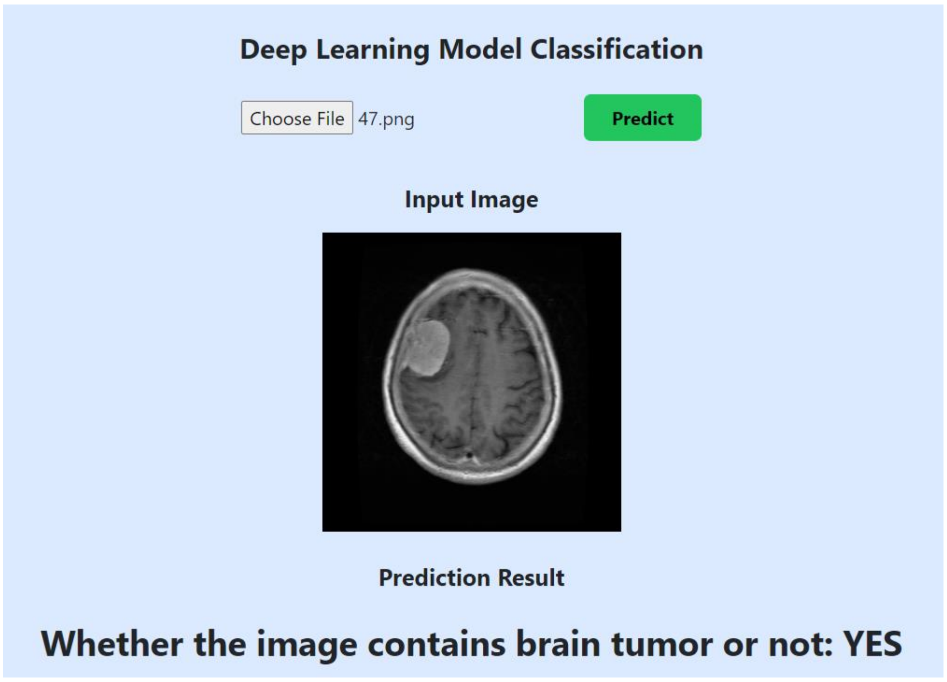
A model checkpoint is a saved state of a machine learning model at a specific point during the training process. It includes the model's weights, optimizer state, and sometimes other relevant information like the current epoch number and the training state. Checkpoints allow the training process to be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



```python
# Set up callbacks
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
checkpoint_filepath = 'best_model.keras'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

csv_logger = CSVLogger('training_log.csv', append=True)
```

Fig: ModelCheckpoint

FEW MODEL PREDICTIONS:


Deep Learning Model Classification

Choose File 47.png Predict

Input Image

Prediction Result

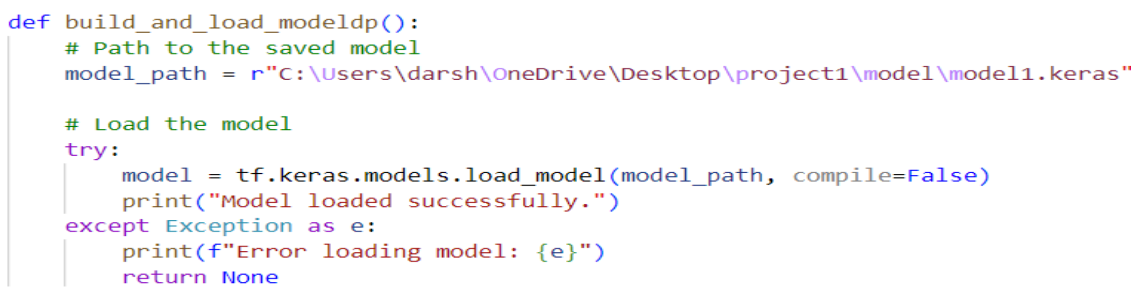
Whether the image contains brain tumor or not: YES

Flask framework:

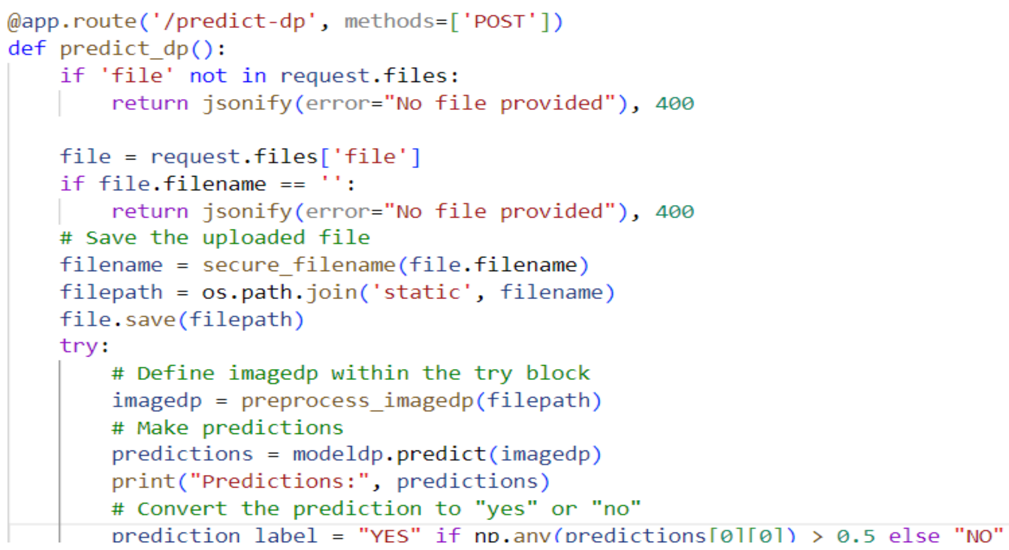
Flask is a lightweight WSGI web application framework in Python. It is designed with simplicity and flexibility in mind, making it a popular choice for small to medium-sized applications. Flask provides the essentials to build web applications, including routing, request handling, and response formatting.

1. **Initialize Flask App : app = Flask(__name__)**
   This line initializes a new Flask application.

2. **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

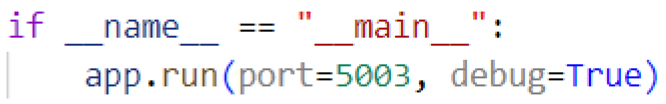

```
def build_and_load_modeldp():
    # Path to the saved model
    model_path = r"C:\Users\darsh\OneDrive\Desktop\project1\model\model1.keras"

    # Load the model
    try:
        model = tf.keras.models.load_model(model_path, compile=False)
        print("Model loaded successfully.")
    except Exception as e:
        print(f"Error loading model: {e}")
        return None
```

3. **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



```python
@app.route('/predict-dp', methods=['POST'])
def predict_dp():
    if 'file' not in request.files:
        return jsonify(error="No file provided"), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify(error="No file provided"), 400
    # Save the uploaded file
    filename = secure_filename(file.filename)
    filepath = os.path.join('static', filename)
    file.save(filepath)
    try:
        # Define imagedp within the try block
        imagedp = preprocess_imagedp(filepath)
        # Make predictions
        predictions = modeldp.predict(imagedp)
        print("Predictions:", predictions)
        # Convert the prediction to "yes" or "no"
        prediction_label = "YES" if np.any(predictions[0][0]) > 0.5 else "NO"
```

4. **Run the Flask Application**



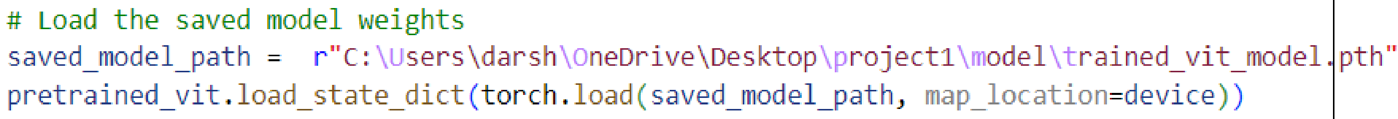
```python
if __name__ == "__main__":
    app.run(port=5003, debug=True)
```

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.

2. **Load the VIT TRANSFORMER Model**
   Save your pre-trained model using a library like torch. Here, we use torch.load to load the model.



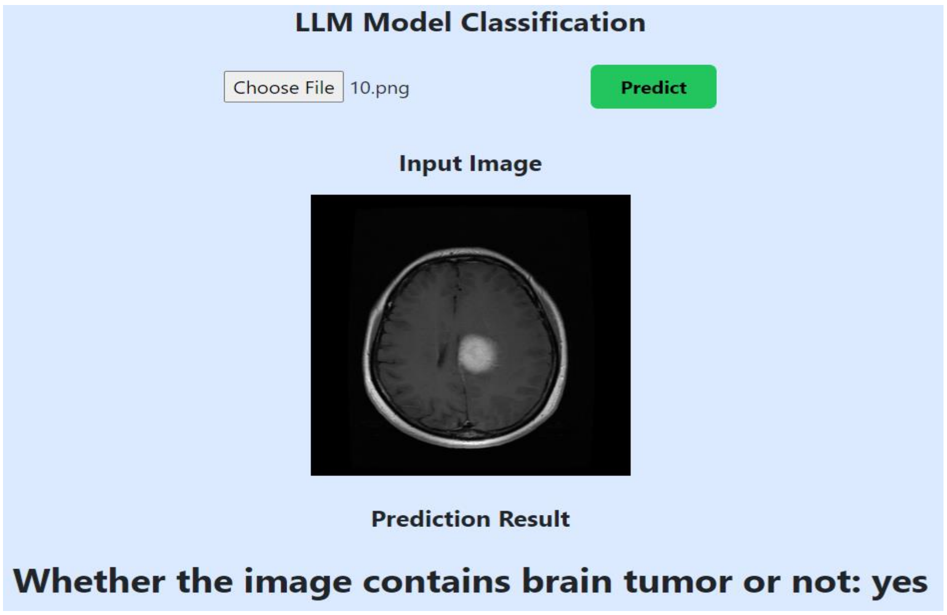
```python
# Load the saved model weights
saved_model_path =  r"C:\Users\darsh\OneDrive\Desktop\project1\model\trained_vit_model.pth"
pretrained_vit.load_state_dict(torch.load(saved_model_path, map_location=device))
```

3. **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

```python
@app.route('/predictllm', methods=['POST'])
def predict():
    if 'image' not in request.files:
        return jsonify({'error': 'No file part'})

    file = request.files['image']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})

    try:
        image_bytes = file.read()
        predicted_class, predicted_prob = classify_image(image_bytes)

        return jsonify({'prediction': predicted_class, 'Accuracy': predicted_prob})
    except Exception as e:
        print(f'Error processing image: {str(e)}')
        return jsonify({'error': 'Error processing image'})
```

4. **Run the Flask Application**

```python
if __name__ == '__main__':
    app.run(port=5004, debug=True)
```



LLM Model Classification

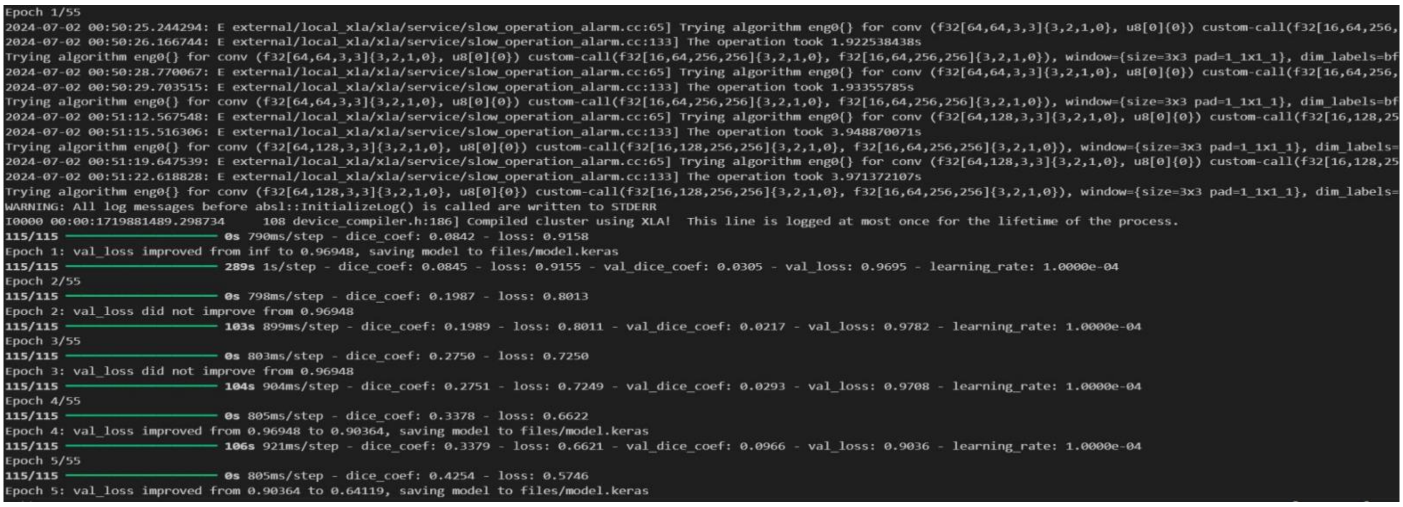
Input Image

Prediction Result

Whether the image contains brain tumor or not: yes

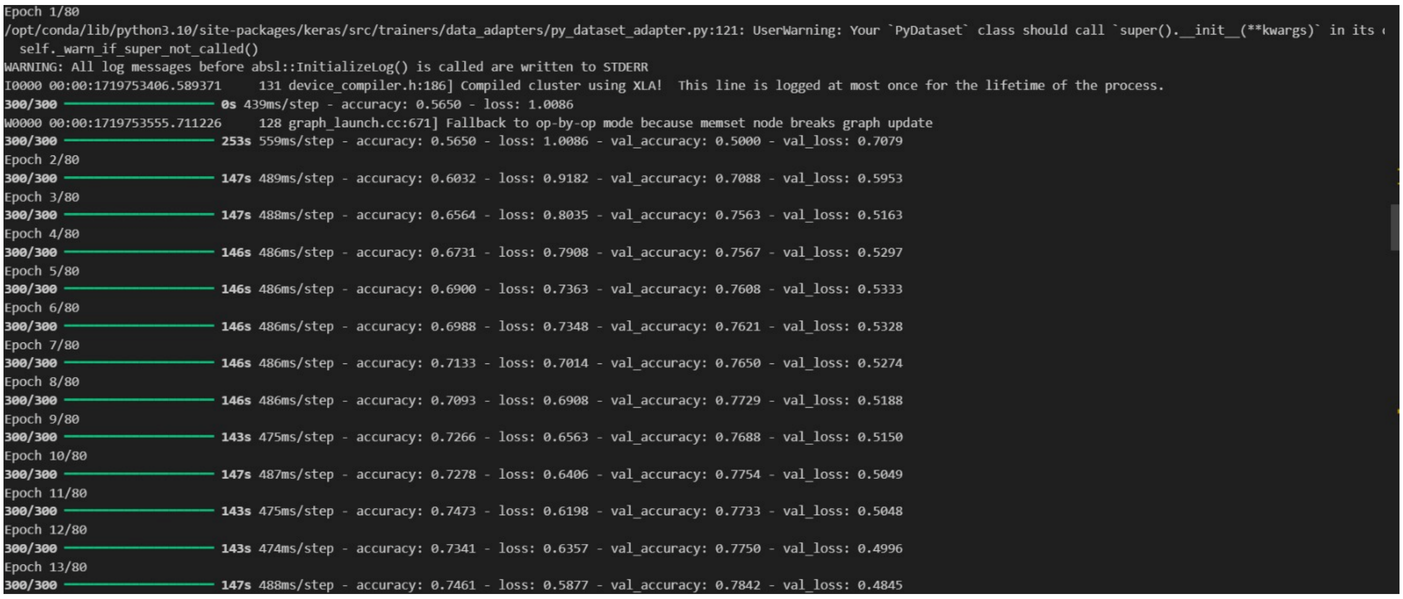
CPU VS GPU

CPU:



```
Epoch 1/55
2024-07-02 00:50:25.244294: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[64,64,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,64,256,
2024-07-02 00:50:26.166744: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 1.922538438s
Trying algorithm eng0{} for conv (f32[64,64,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,64,256,256]{3,2,1,0}, f32[16,64,256,256]{3,2,1,0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf
2024-07-02 00:50:28.770067: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[64,64,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,64,256,
2024-07-02 00:50:29.703515: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 1.93355785s
Trying algorithm eng0{} for conv (f32[64,64,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,64,256,256]{3,2,1,0}, f32[16,64,256,256]{3,2,1,0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf
2024-07-02 00:51:12.567548: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[64,128,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,128,25
2024-07-02 00:51:15.516306: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 3.948870071s
Trying algorithm eng0{} for conv (f32[64,128,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,128,256,256]{3,2,1,0}, f32[16,64,256,256]{3,2,1,0}), window={size=3x3 pad=1_1x1_1}, dim_labels=
2024-07-02 00:51:19.647539: E external/local_xla/xla/service/slow_operation_alarm.cc:65] Trying algorithm eng0{} for conv (f32[64,128,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,128,25
2024-07-02 00:51:22.618828: E external/local_xla/xla/service/slow_operation_alarm.cc:133] The operation took 3.971372107s
Trying algorithm eng0{} for conv (f32[64,128,3,3]{3,2,1,0}, u8[0]{0}) custom-call(f32[16,128,256,256]{3,2,1,0}, f32[16,64,256,256]{3,2,1,0}), window={size=3x3 pad=1_1x1_1}, dim_labels=
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1719881489.298734     108 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 790ms/step - dice_coef: 0.0842 - loss: 0.9158
Epoch 1: val_loss improved from inf to 0.96948, saving model to files/model.keras
115/115 ━━━━━━━━━━━━━━━━━━━━ 289s 1s/step - dice_coef: 0.0845 - loss: 0.9155 - val_dice_coef: 0.0305 - val_loss: 0.9695 - learning_rate: 1.0000e-04
Epoch 2/55
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 798ms/step - dice_coef: 0.1987 - loss: 0.8013
Epoch 2: val_loss did not improve from 0.96948
115/115 ━━━━━━━━━━━━━━━━━━━━ 103s 899ms/step - dice_coef: 0.1989 - loss: 0.8011 - val_dice_coef: 0.0217 - val_loss: 0.9782 - learning_rate: 1.0000e-04
Epoch 3/55
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 803ms/step - dice_coef: 0.2750 - loss: 0.7250
Epoch 3: val_loss did not improve from 0.96948
115/115 ━━━━━━━━━━━━━━━━━━━━ 104s 904ms/step - dice_coef: 0.2751 - loss: 0.7249 - val_dice_coef: 0.0293 - val_loss: 0.9708 - learning_rate: 1.0000e-04
Epoch 4/55
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 805ms/step - dice_coef: 0.3378 - loss: 0.6622
Epoch 4: val_loss improved from 0.96948 to 0.90364, saving model to files/model.keras
115/115 ━━━━━━━━━━━━━━━━━━━━ 106s 921ms/step - dice_coef: 0.3379 - loss: 0.6621 - val_dice_coef: 0.0966 - val_loss: 0.9036 - learning_rate: 1.0000e-04
Epoch 5/55
115/115 ━━━━━━━━━━━━━━━━━━━━ 0s 805ms/step - dice_coef: 0.4254 - loss: 0.5746
Epoch 5: val_loss improved from 0.90364 to 0.64119, saving model to files/model.keras
```

GPU:



```
Epoch 1/80
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its
  self._warn_if_super_not_called()
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1719753406.589371     131 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
300/300 ━━━━━━━━━━━━━━━━━━━━ 0s 439ms/step - accuracy: 0.5650 - loss: 1.0086
W0000 00:00:1719753555.711226     128 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
300/300 ━━━━━━━━━━━━━━━━━━━━ 253s 559ms/step - accuracy: 0.5650 - loss: 1.0086 - val_accuracy: 0.5000 - val_loss: 0.7079
Epoch 2/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 147s 489ms/step - accuracy: 0.6032 - loss: 0.9182 - val_accuracy: 0.7088 - val_loss: 0.5953
Epoch 3/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 147s 488ms/step - accuracy: 0.6564 - loss: 0.8035 - val_accuracy: 0.7563 - val_loss: 0.5163
Epoch 4/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 146s 486ms/step - accuracy: 0.6731 - loss: 0.7908 - val_accuracy: 0.7567 - val_loss: 0.5297
Epoch 5/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 146s 486ms/step - accuracy: 0.6900 - loss: 0.7363 - val_accuracy: 0.7608 - val_loss: 0.5333
Epoch 6/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 146s 486ms/step - accuracy: 0.6988 - loss: 0.7348 - val_accuracy: 0.7621 - val_loss: 0.5328
Epoch 7/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 146s 486ms/step - accuracy: 0.7133 - loss: 0.7014 - val_accuracy: 0.7650 - val_loss: 0.5274
Epoch 8/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 146s 486ms/step - accuracy: 0.7093 - loss: 0.6908 - val_accuracy: 0.7729 - val_loss: 0.5188
Epoch 9/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 143s 475ms/step - accuracy: 0.7266 - loss: 0.6563 - val_accuracy: 0.7688 - val_loss: 0.5150
Epoch 10/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 147s 487ms/step - accuracy: 0.7278 - loss: 0.6406 - val_accuracy: 0.7754 - val_loss: 0.5049
Epoch 11/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 143s 475ms/step - accuracy: 0.7473 - loss: 0.6198 - val_accuracy: 0.7733 - val_loss: 0.5048
Epoch 12/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 143s 474ms/step - accuracy: 0.7341 - loss: 0.6357 - val_accuracy: 0.7750 - val_loss: 0.4996
Epoch 13/80
300/300 ━━━━━━━━━━━━━━━━━━━━ 147s 488ms/step - accuracy: 0.7461 - loss: 0.5877 - val_accuracy: 0.7842 - val_loss: 0.4845
```

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.