Chapter 8

Execution of CNN Model

Representation of CNN as a part of the brain tumor detection & classification system

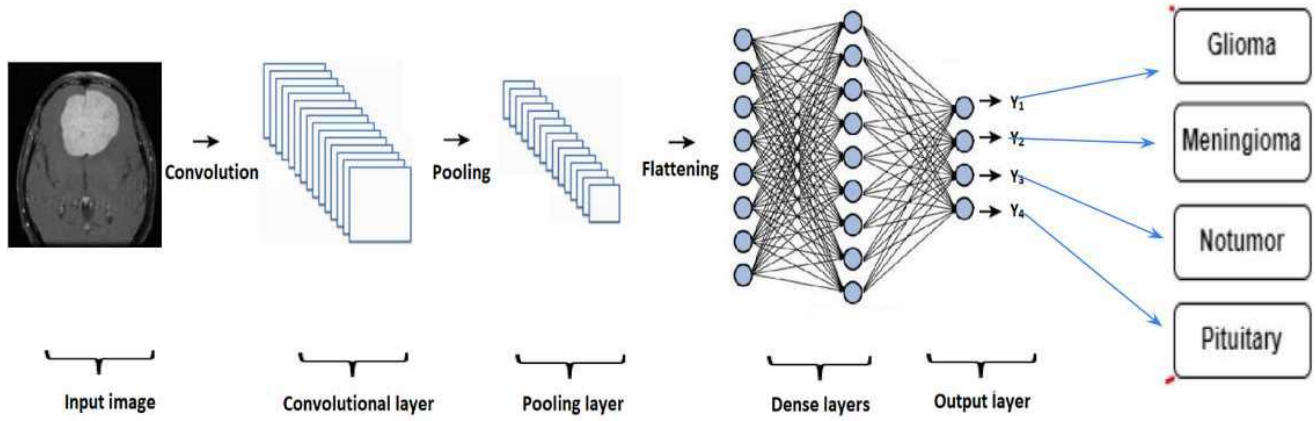


Fig. 8.1 CNN model representation of our system

The above image shows the path of work of CNN in our system "Brain Tumor Detection and Classification system using DeepNets". A brain MRI image is given as input and finally the output is classified into 4 types of tumors.

8.1 Implementation Steps

Implementation of an algorithm is the first step towards mastering the algorithm. This will give us a deep and practical appreciation for how the algorithm works. This knowledge can also help us to internalize the mathematical description of the algorithm by thinking of the vectors and matrices as arrays and the computational intuitions for the transformations on those structures.

This section shows a complete implementation of a brain tumor detection and classification model using a Convolutional Neural Network (CNN) in TensorFlow.

- ***Step 1 : IMPORTING ALL THE NECESSARY LIBRARIES***

At first, all the required libraries for data manipulation, visualization and building the Convolutional Neural Network (CNN) model are imported with highest possible accuracy using Tensorflow

The modules imported are :-

1. *os*
2. *numpy*
3. *pandas*
4. *matplotlib*
5. *seaborn*
6. *tensorflow library*

ImageDataGenerator *class is used as it helps to generate batches of tensor image data with real-time data augmentation, which can be useful for training models with more generalized data.*

Sequential*, a linear stack of layers is used which llows you to build a model layer by layer*

by adding one layer at a time.

- ***Step 2* : SETTING UP THE DATASET PATHS AND DIRECTORIES**

The root path of the brain tumor MRI dataset is stored in a variable. Similarly, the root path of the training and the testing directories are also stored in individual variables. The training and testing directories are defined by joining the dataset path with the specific subdirectories.

- ***Step 3* : LOADING AND PREPROCESSING THE DATASET**

The images are read from each category in the training directory, counts the number of images in each category, and creates a Pandas DataFrame to store the image filenames, corresponding categories, and counts. These dataframes are stored in empty lists.

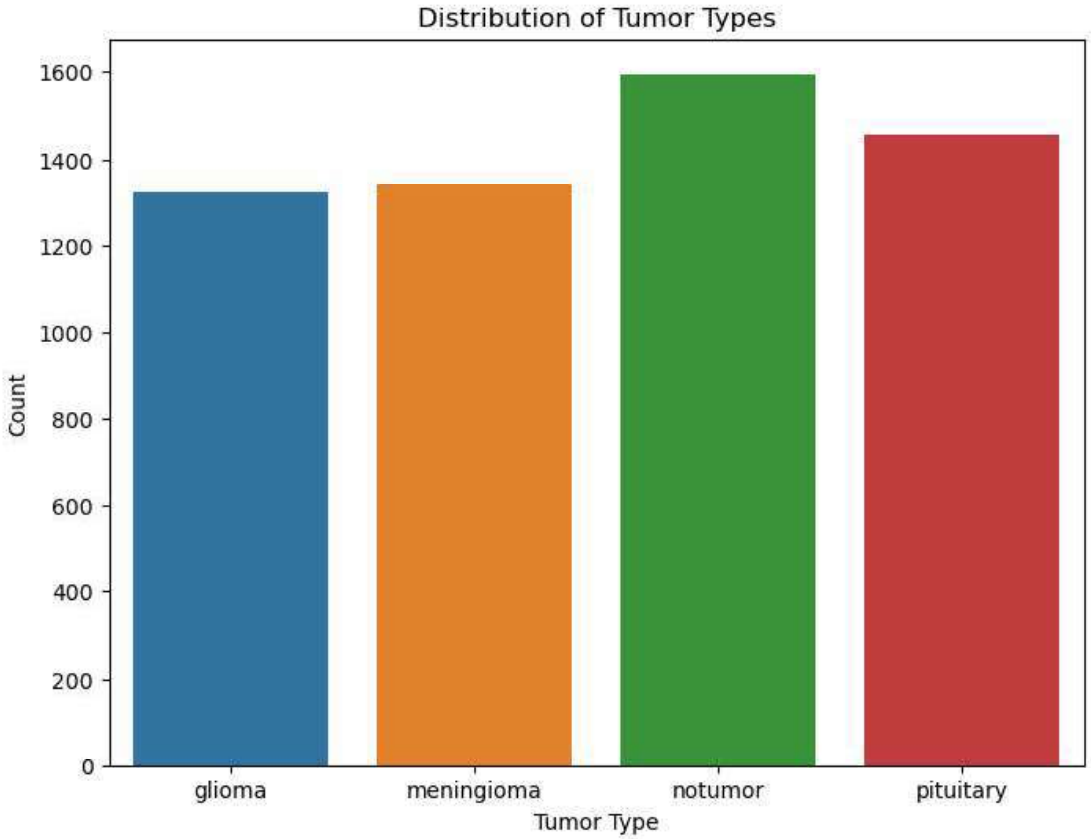


Fig. 8.2 Graphical Representation of the Tumor Types

X-axis represents the different tumor categories.

Y-axis represents the count of images in each category.

- ***Step 4 : VISUALIZING IMAGES FOR EACH TUMOR TYPES***

A new 12 inch by 8 inch figure is formed.

The sample images for each tumor type are displayed using a grid of 2X2 subplots.

The layout of the subplots is adjusted to fit neatly within the figure.

These visualizations helps in quickly inspecting sample images from each tumor category, allowing for an initial qualitative assessment of the dataset.

The image below shows some sample MRI images from our dataset.

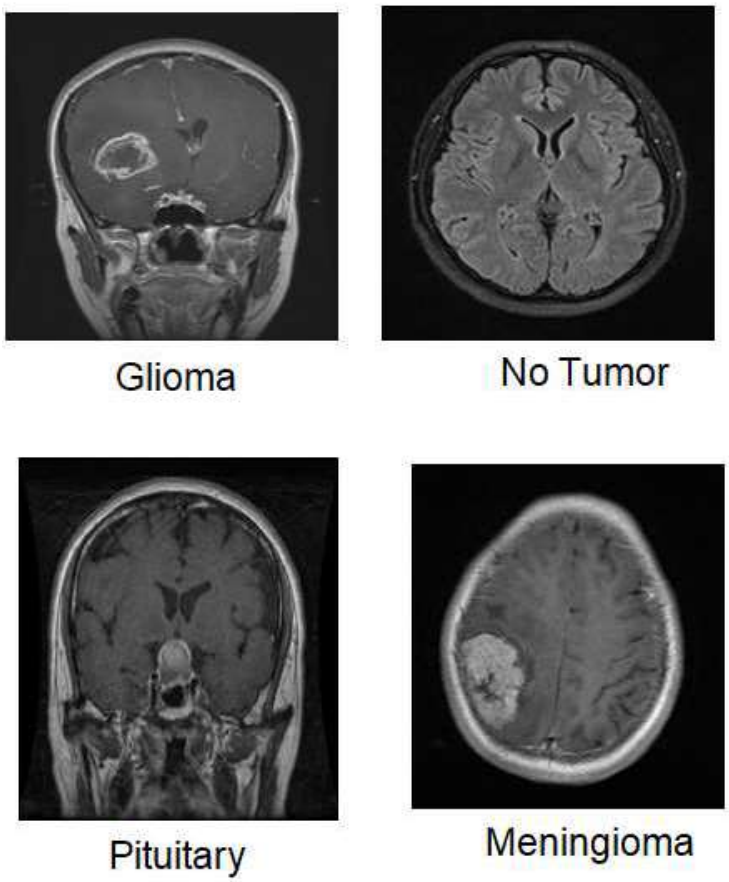


Fig. 8.3 Visualization of the dataset images

- ***Step 5 : SETTING UP THE IMAGE SIZE, BATCH SIZE AND EPOCHS FOR THE MODEL***

A variable defines the desired size for the input images in the CNN.
Another variable specifies the number of images to be processed in each training batch (batch size)
Epochs determines the number of times the entire training dataset is iterated during training.

Image size used - (150, 150)
Batch size used - 32
No. of epochs used - 50

Standardizing the input size ensures that all images have the same dimensions, which is a requirement for CNNs. It also helps in reducing the computational load.

Using a batch of images instead of a single image at a time helps in stabilizing and speeding up the training process. A batch size of 32 is a common choice that balances memory usage and training speed.

More epochs generally improve the model's performance up to a point. After a certain number of epochs, the model might start overfitting, learning the training data too well and not generalizing to new data. 50 epochs is a reasonable starting point to see how well the model performs.

- ***Step 6 : DATA AUGMENTATION AND PREPROCESSING***

Data augmentation is performed using ImageDataGenerator class from Keras.
Generates batches of tensor image data with real-time data augmentation.

It applies various transformations to the training imags to artificially increase the size of the dataset and improve the generalization.

Rescaling the pixel values, rotation, shifting, etc. are some of the most commonly used augmentation parameters.

The train class is created using the augmented data.

The test class is created with only pixel rescaling for the test dataset.

Augmentation parameters used in our model are :-

Rescaling	***Rotation***	***Shearing***
Width shifting	***Height shifting***	***Zooming***
Horizontal flipping	***Vertical flipping***	***Fill mode***

- ***Step 7 : BUILDING THE MODEL ARTITECHURE***

  In this step, the model architecture is designed.

Model - Sequential

It consists of a series of ***convolutional (Conv2D)*** *and* ***max pooling (MaxPooling2D)*** *layers, followed by a* ***flattening*** *layer, two* ***fully connected (Dense)*** *layers, and a* ***dropout*** *layer for regularization.*

Activation function - ReLU *for convolutional layers*

$$Relu(z) = max(0, z)$$

Softmax *for dense layer*

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

K = total number of classes

zi = input to the softmax function for the i-th class

sigma(zi) = utput of the softmax function for the i-th class.

Optimizer - Adam (adaptive learning rate optimization algorithm)

Loss function - Categorical Cross-Entropy

If M is greater tha 2 (i.e, multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-(y \log(p) + (1 - y) \log(1 - p)) - \sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

M - number of classes

log - the natural log

y - binary indicator (0 or 1) if class label c is the correct classification for observation o

p - predicted probability observation o is of class c

Output image size after convolution = ***(n + 2p - f + 1)/s***

n = size of the input image (nxn)

p = size of padding

f = size of filter/kernel

s = *stride value*

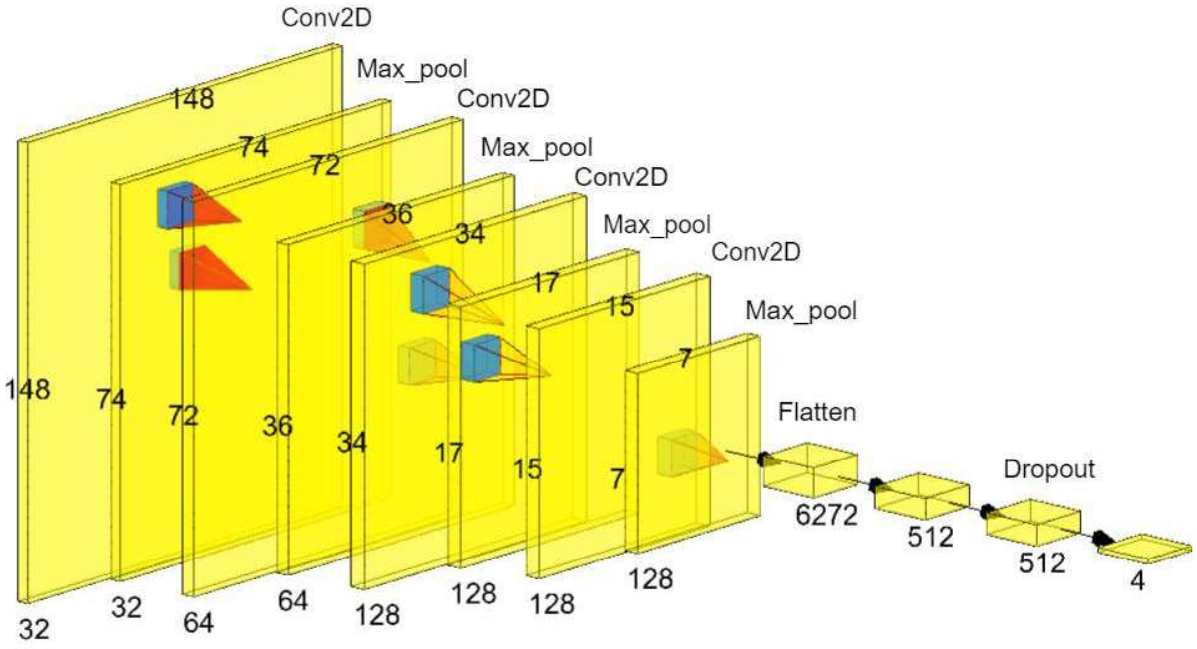


Fig. 8.4 Implemented CNN model

This CNN architecture processes the input image through a series of convolutional and max-pooling layers to extract features. The features are then flattened and passed through fully connected layers for classification. The dropout layer is used to improve generalization by reducing overfitting. The final output is a set of probabilities corresponding to the different classes.

The flowchart of the implemented CNN architecture is shown below :-

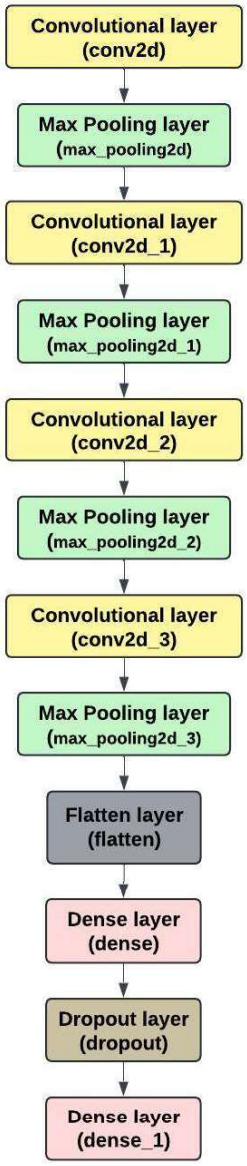


Fig. 8.5 Visualization of the dataset images

- ***Step 8 : VISUALIZATION THROUGH GRAPH***

The training and validation accuracy over epochs using the data is calculated. It helps visualize the model's learning progress and check for overfitting or underfitting.
The training and validation loss over epochs using the data is calculated. It helps visualize how the model's loss decreases over time, indicating improved performance.

- ***Step 9 : EVALUATION***

Here, the trained model is evaluated on the test dataset.
It calculates the loss and accuracy of the model's predictions on the test data..
These evaluation metrics provide insights into the model's performance on unseen data and help assess its generalization capabilities.

- ***Step 10 : CONFUSION MATRIX AND EXPLANATION***

The model is used to make predictions on the test dataset.
A confusion matrix is created using TensorFlow's tf.math.confusion matrix function. It compares the true and predicted categories and provides a count of correct and incorrect predictions for each class.
Random sample images, their corresponding predictions, and true labels are selected for visualization.
A variable stores the filenames of test images.
Random indices are chosen and the corresponding images, predictions, and true labels are extracted.
A grid of subplots is created to display the sample images along with their predicted and true labels.

8.2 Experimental Results

This section provides a comprehensive overview of the experimental results for the brain tumor detection and classification system with the implementation of the CNN architecture.

The following subsections are some of the implementation steps which contains the results after experimentation.

- ***DATA AUGMENTATION & PREPROCESSING***

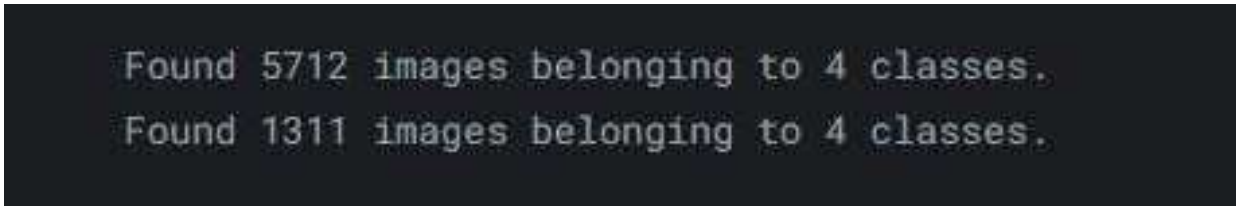


Fig. 8.6 Results of augmentation and preprocessing

The above pictures indicates the results of the images after augmentation and preprocessing of the dataset.

5712 training images are distributed across 4 classes.

1311 testing images are distributed across 4 classes.

- ***BUILDING THE MODEL ARCHITECTURE***

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 3,454,660
Trainable params: 3,454,660
Non-trainable params: 0
```

Fig. 8.7 Results of augmentation and preprocessing

1. *1st convolutional layer has 32 filters of size 3x3. The output shape is 148v148x32 with 3 color channels RGB. The activation function used is ReLU.*

2. *1st max pooling layer has 2x2 filter with output shape 74x74x32 reducing the spatial dimensions.*

3. *2nd convolutional layer has 64 filters of size 3x3. The activation function used is ReLU.*

4. *2nd max pooling layer performs max pooling with a 2x2 filter, reducing the spatial dimensions by half.*

5. *3rd convolutional layer has 128 filters of size 3x3. The activation function used is ReLU.*

6. *Next max pooling layer performs max pooling with a 2x2 filter*

7. *Next convolutional layer has 128 filters of size 3x3. The activation function used is ReLU.*

8. *Max pooling with a 2x2 filter is used.*

9. *Flatten layer lattens the 3D tensor into a 1D tensor.*

10. *Fully connected (dense) layer has 512 units with Softmax activation.*

11. *Dropout layer prevents overfitting, randomly setting 50% of the input units to 0.*

12. *This output layer has 4 units with softmax activation, corresponding to the 4 classes for classification.*

- ***MODEL TRAINING***

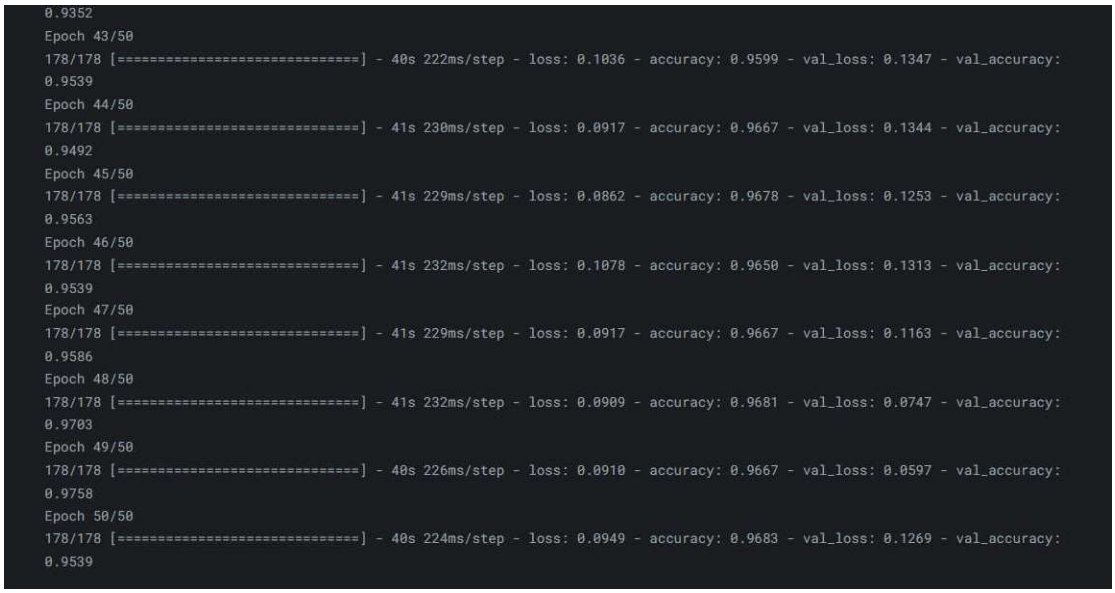


Fig. 8.8 Few epochs for model training

The image in the previous page is a snapshot of the epochs while the model is being trained.

- ***VISUALIZATION***

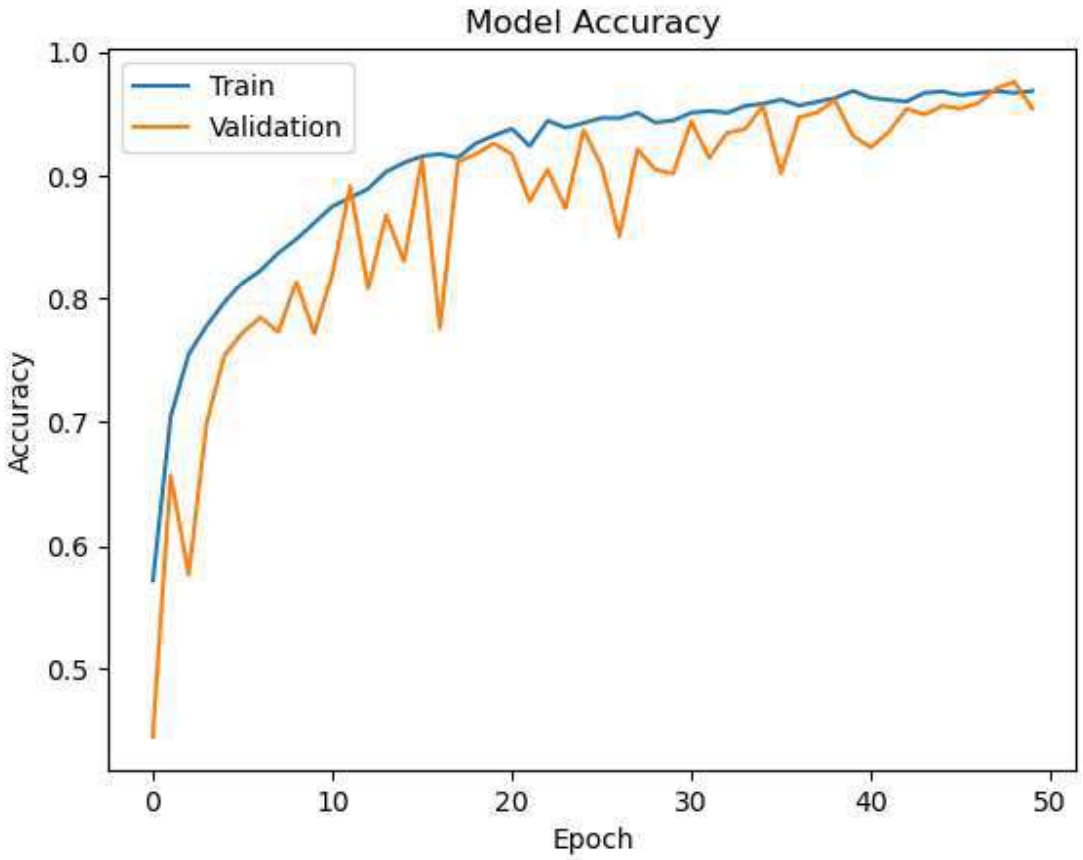


Fig. 8.9 Train vs Validation accuracy

The above shows the training and validation accuracy of a convolutional neural network (CNN) over 50 epochs. It indicates that the CNN model is performing well, with high training and validation accuracy. The slight fluctuations in validation accuracy are typical and do not suggest significant overfitting. By following the recommendations, the model's performance can be further optimized.

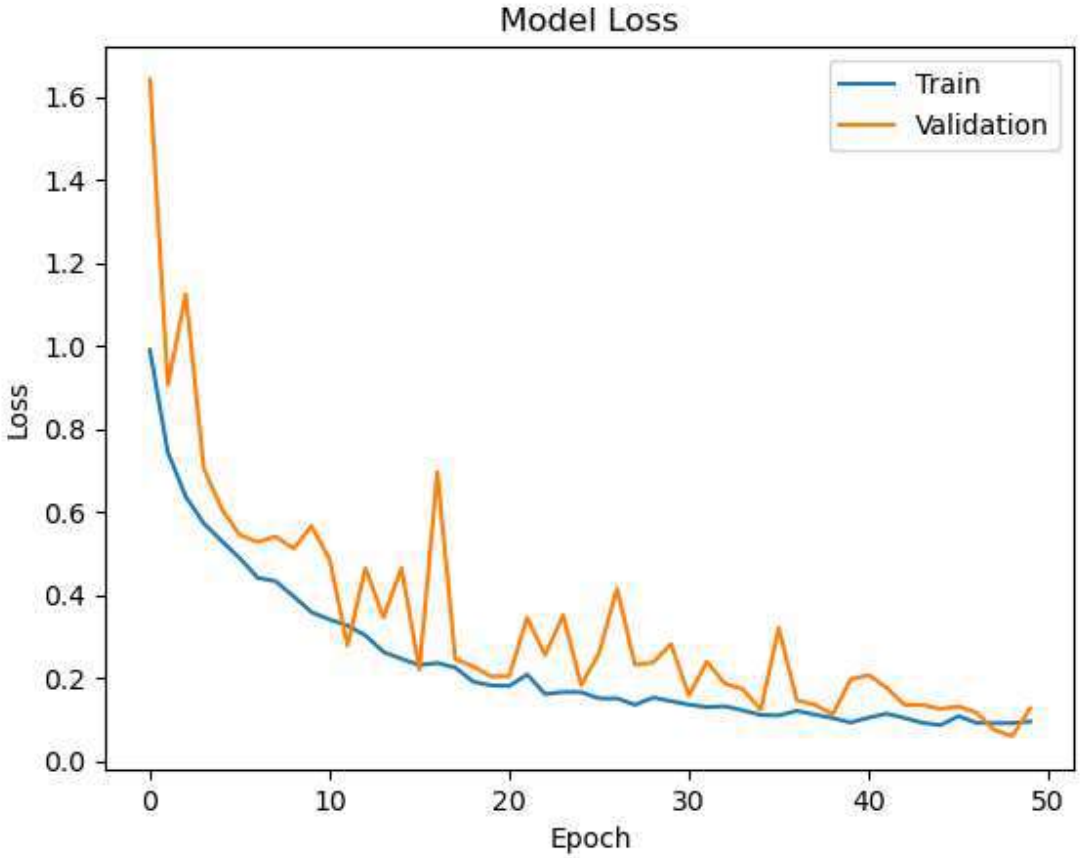


Fig. 8.10 Train vs Validation loss

This plot shows the training and validation loss of a convolutional neural network (CNN) over 50 epochs. It indicates that the CNN model is performing well, with both training and validation loss decreasing over epochs. The slight fluctuations in validation loss are typical and do not suggest significant overfitting. By following the recommendations, the model's performance can be further optimized.

- ***EVALUATION***

The test loss is 0.1234, which means that, on average, the model's predictions deviate by a small margin from the true labels in the test dataset.

The test accuracy is 0.9602, indicating that the model has achieved an accuracy of ap-

```
40/40 [==============================] - 3s 68ms/step - loss: 0.1269 - accuracy: 0.9539
Test Loss: 0.12692564725875854
Test Accuracy: 0.953906238079071
```

Fig. 8.11 Evaluation of the model

proximately 96.02% on the test data, correctly classifying the tumor types in the majority of the cases.

- ***CONFUSION MATRIX***

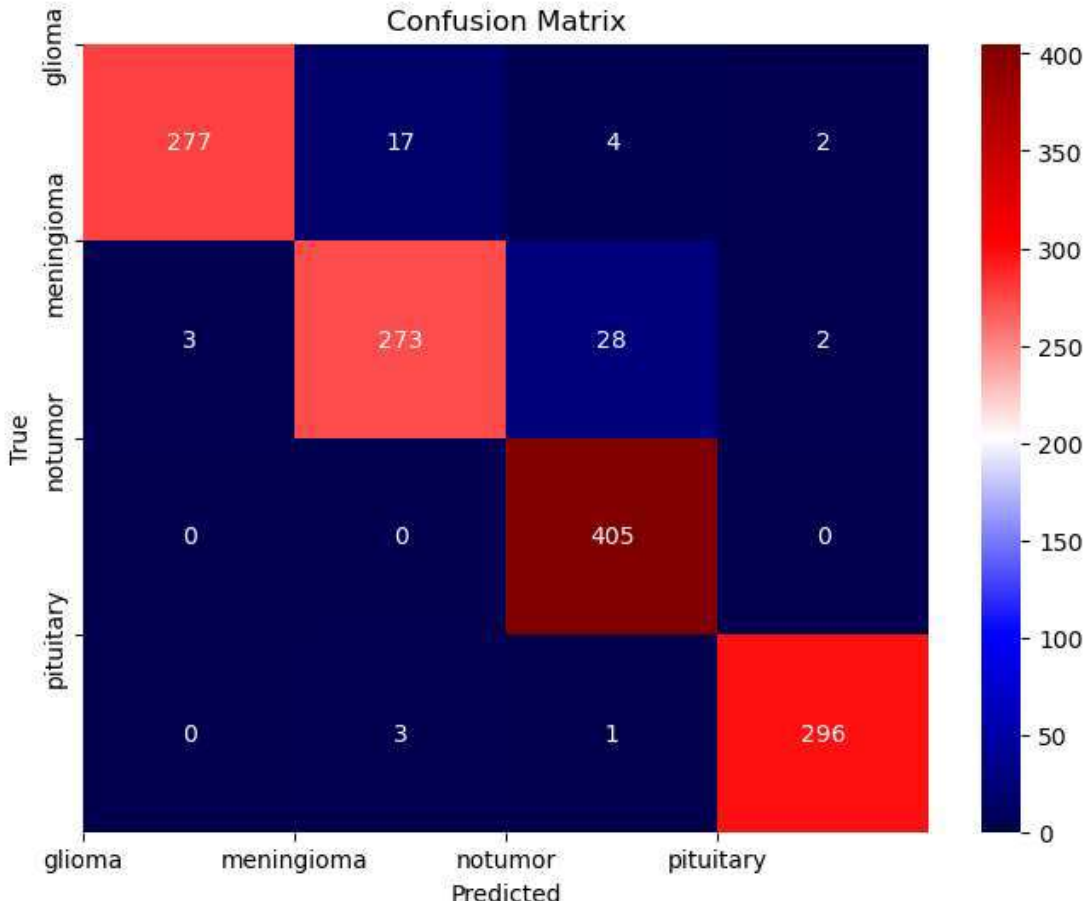


Fig. 8.12 Confusion matrix of the implemented CNN model

1. *Glioma*

   Correctly classified 277 times.

   Misclassified as Meningioma 17 times, Notumor 4 times, and Pituitary 2 times.

2. *Meningioma*

   Correctly classified 273 times.

   Misclassified as Glioma 3 times, Notumor 28 times, and Pituitary 2 times.

3. *No tumor*

   Correctly classified 405 times.

   No misclassifications.

4. *Pituitary*

   Correctly classified 296 times.

   Misclassified as Meningioma 3 times and Notumor 1 time.

The image below shows the sample images along with their predicted and true labels

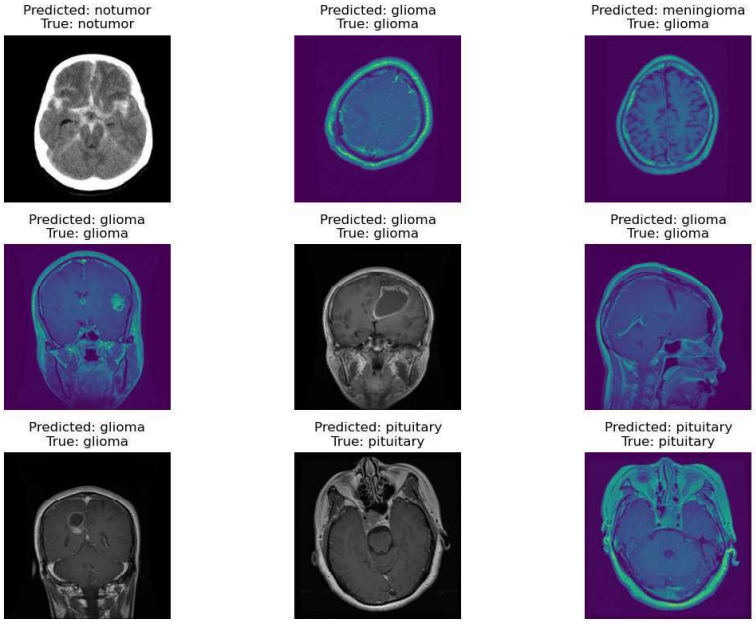


Fig. 8.13 True vs predicted images

- ***EVALUATION METRICS***

The evaluation metrics used are :- Precision, recall and F1-score.

Class	Precision	Recall	F1-score
Glioma	0.9892	0.9233	0.9551
Meningioma	0.9317	0.8921	0.9115
No tumor	0.9246	1.0	0.9608
Pituitary	0.9866	0.9866	0.9866

Table 8.1 Tabular representation of evaluation metrices

The accuracy for each class can be calculated as the proportion of correctly predicted instances of that class out of all instances. Here are the accuracies for each class:

Glioma: 86.33%, Meningioma: 98.04%, No Tumor: 100%, Pituitary: 98.00%

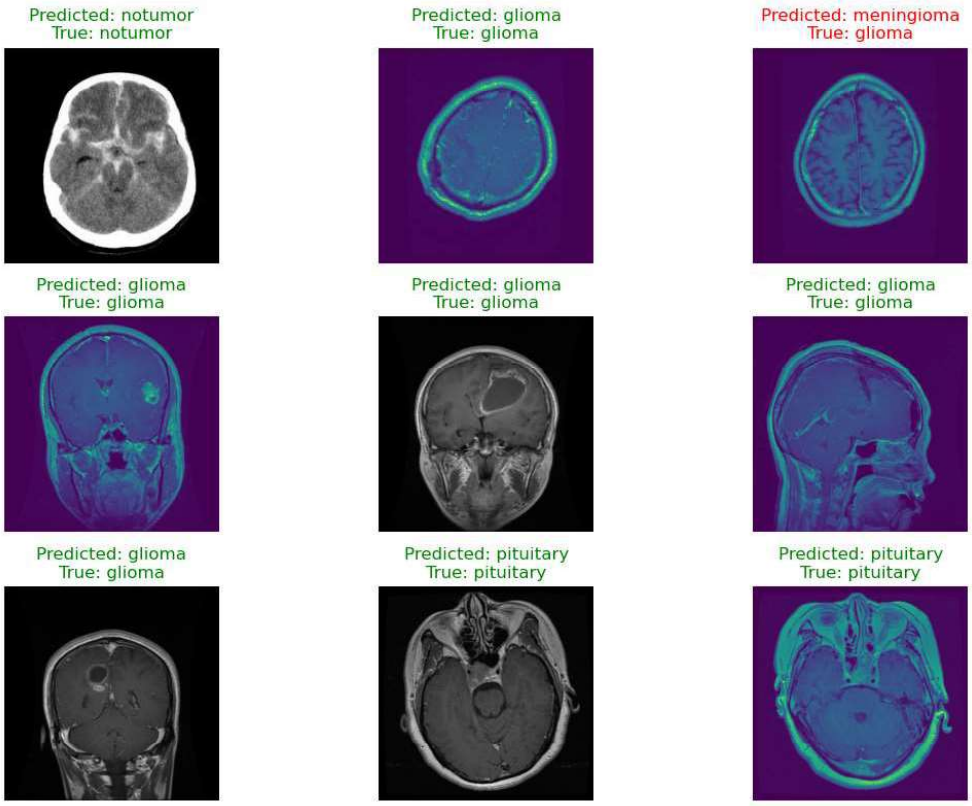


Fig. 8.14 True vs predicted labels

Chapter 9

Execution of VGG-19 Architecture

Representation of VGG-19 as a part of the brain tumor detection and classification system

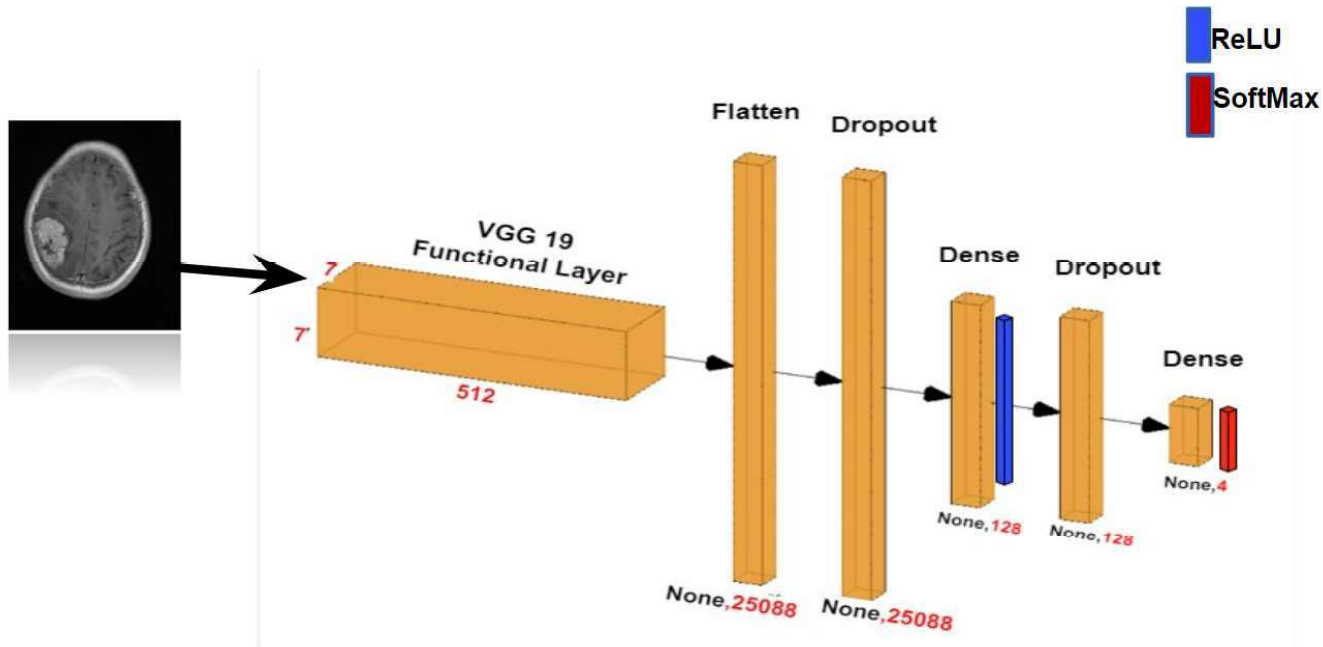


Fig. 9.1 VGG-19 model representation of our system

The above image shows the path of work of VGG-19 in our system "Brain Tumor Detection and Classification system using DeepNets".

9.1 Implementation Steps

Implementation is a crucial part of a project because it's the phase when the project team puts the plan into action to achieve its goals. It's the process of turning ideas and evidence into policies and practices that work in the real world. It is one of the most crucial steps towards successfully developing an algorithm.

This section shows a complete implementation of a brain tumor detection and classification model using a VGG-19 in TensorFlow.

- ***Step 1 : IMPORTING ALL THE NECESSARY LIBRARIES***

Several libraries and modules necessary for building and evaluating the VGG-19 for image classification, specifically for detecting and classifying brain tumors.
The modules imported are :-

1. *os*
2. *numpy*
3. *pandas*
4. *matplotlib*
5. *seaborn*
6. *random*
7. *itertools*
8. *tensorflow*

ImageDataGenerator *class is used for augmenting image data to increase the variety and quantity of training data.*

Sequential *defines the architecture.*

- ***Step 2 : LOADING THE TRAINING DATA***

1. *This part loads the training data from a directory containing MRI brain tumor images, and it organizes the data into a DataFrame for further processing.*
2. *Training data path is defined and file paths of images and their corresponding labels (categories) are stored.*
3. *Iterations are made through the files and folders. Then file path and labels series are concatenated.*

- ***Step 3 : LOADING THE TESTING DATA***

1. *The path to the directory containing the testing data is defined.*
2. *The file paths of images and their corresponding labels (categories) are define in two lists.*
3. *Two series i.e, file path of images and labels are concatenated along the columns.*
4. *The resulting DataFrame can be used with data generators or custom data loaders for feeding the data into machine learning models, particularly for evaluating the model performance on the test set.*

- ***Step 4 : SPLITTING THE DATASET***

1. *The dataset is splitted into 4 classes :-*

   Glioma, Meningioma, No tumor and Pituitary

   The image shown below depicts the division of the entire dataset into all the 4 classes of tumors.

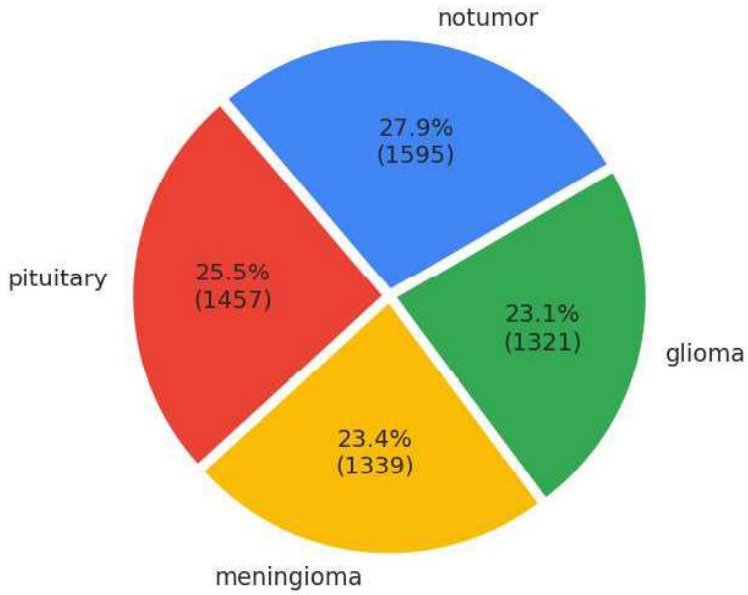


Fig. 9.2 Division of the dataset

- ***Step 5 : PREPROCESSING AND AUGMENTATION***

1. *ImageDataGenerator class is used to handle data augmentation and preprocessing.*
2. *Data generators are directly created from Pandas DataFrames containing image file paths and corresponding labels.*
3. *Separate generators are created for training, validation, and test datasets, each configured to resize images, handle batches, and (for training and validation) shuffle data.*
4. *Batch processing of the images are done.*
5. *After this, the processed and augmented images are plotted.*

Image size used - (240, 240)

Batch size used - 32

- ***Step 6 : DATASET DIVISION***

A pie chart is created to visualize the distribution of a dataset into training, validation, and testing sets.

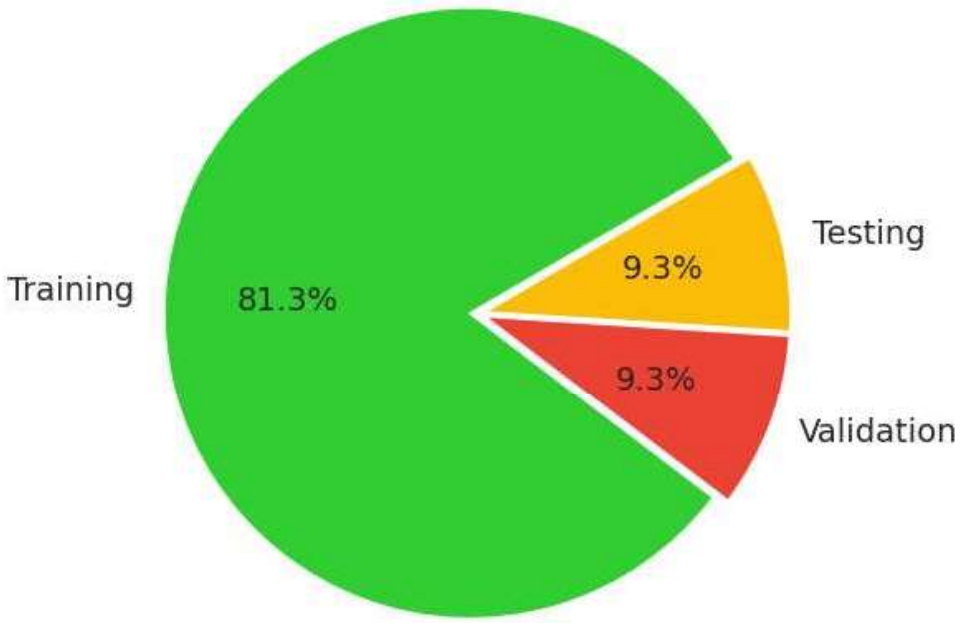


Fig. 9.3 Dataset splitting

The resulting pie chart will show the proportion of samples in each set, with the training set, validation set, and testing set represented by different colors and labels.

- ***Step 7 : CREATING THE MODEL ARCHITECTURE***

1. *Image sizes are set and batches of augmented images are are generated for training.*
2. *The base model i.e, a pre-trained VGG-19 network is created which is being trained on the ImageNet dataset.*
3. *After this all the layers like flatten, droput etc. are designed according to the requirements.*

Model : ***Sequential***

Layers : ***Flatten, Dropout and Fully-connected (Dense) layers***

Activation function : ***ReLU (dense layer), Softmax (Output layer)***

Equation of ReLU :-

$$Relu(z) = max(0, z)$$

Equation of Softmas :-

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

- ***Step 8 : COMPILING THE MODEL***

The model is compiled and trained on the dataset.

Batch size : ***32***

No. of epochs : ***50***

Optimizer : ***Adam***

Loss function : ***Categorical cross entropy***

Equation of categorical cross-entropy loss :-

$$-(y \log(p) + (1-y) \log(1-p)) - \sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

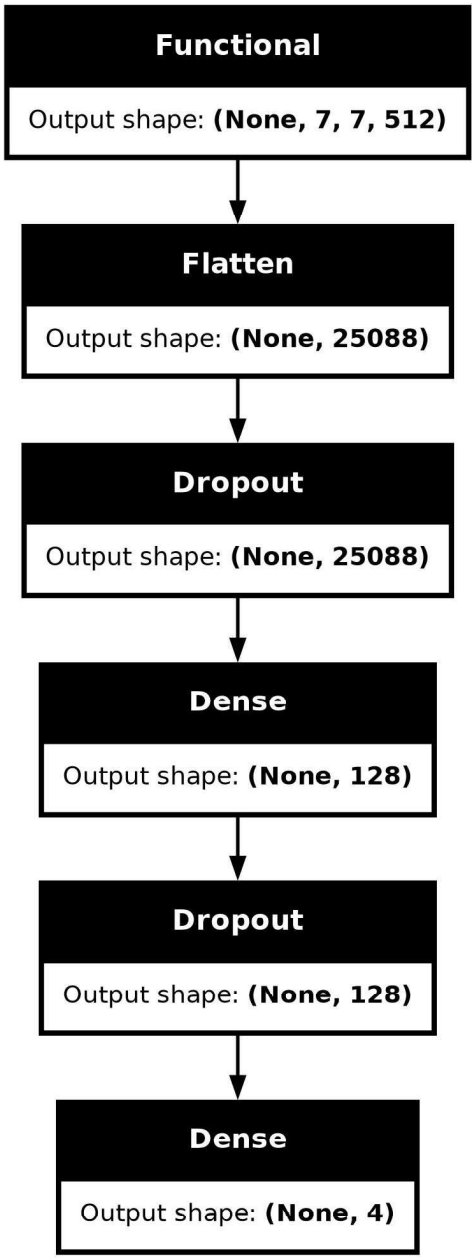


Fig. 9.4 Used VGG-19 architecture

This architecture shown above is typical for a classification task where the final output corresponds to 4 classes. The dropout layers are used to prevent overfitting. The functional layer at the beginning likely represents some pre-trained model that outputs a feature map of shape (7, 7, 512).

9.2 Experimental Results

This section provides a comprehensive overview of the experimental results for the brain tumor detection and classification system with the implementation of the VGG-19 architecture.

- ***DATA AUGMENTATION & PREPROCESSING***

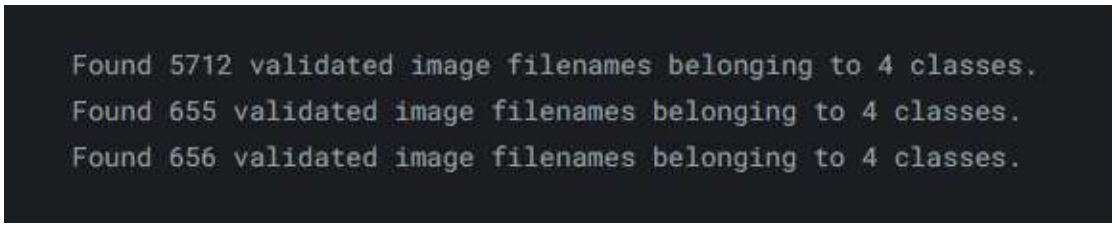


Fig. 9.5 Results of augmentation and preprocessing of VGG-19

It is showing the number of images found and the number of categories they belong to. It seems the data was split into three parts:

Part 1 : *5712 training images belonging to 4 classes*

Part 2 : *655 training images belonging to 4 classes*

Part 3 : *656 training images belonging to 4 classes*

- ***PLOTTING THE PROCESSED IMAGES***

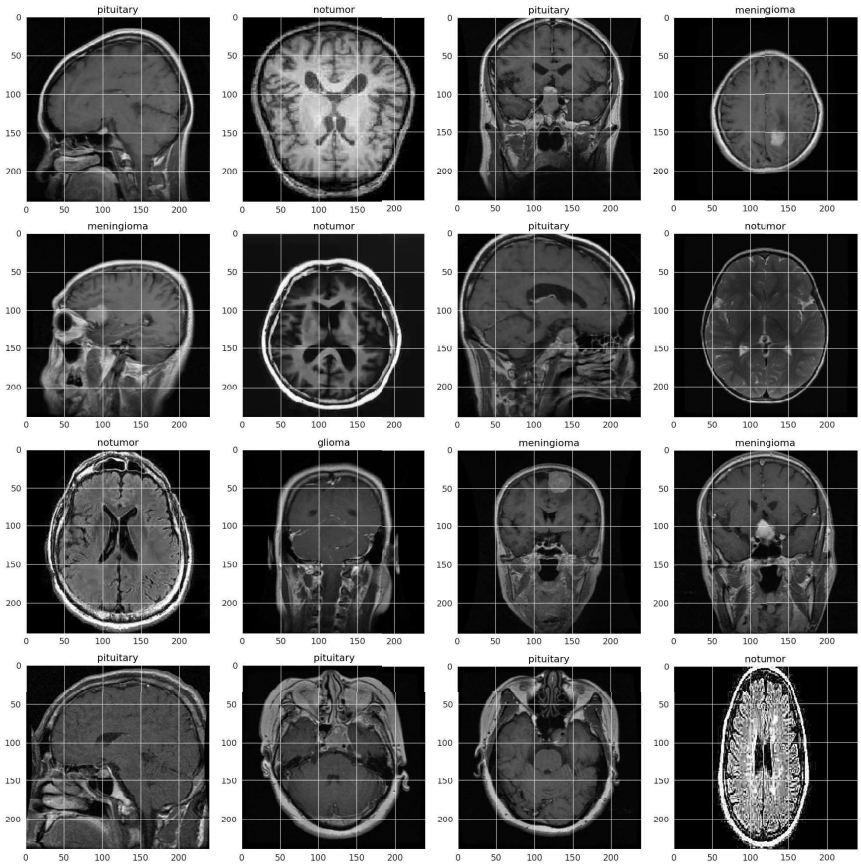


Fig. 9.6 Results of pre-processed images

The above image appears is a collection of MRI scans labeled with different conditions. Here is a breakdown of the labels and their corresponding MRI scans:

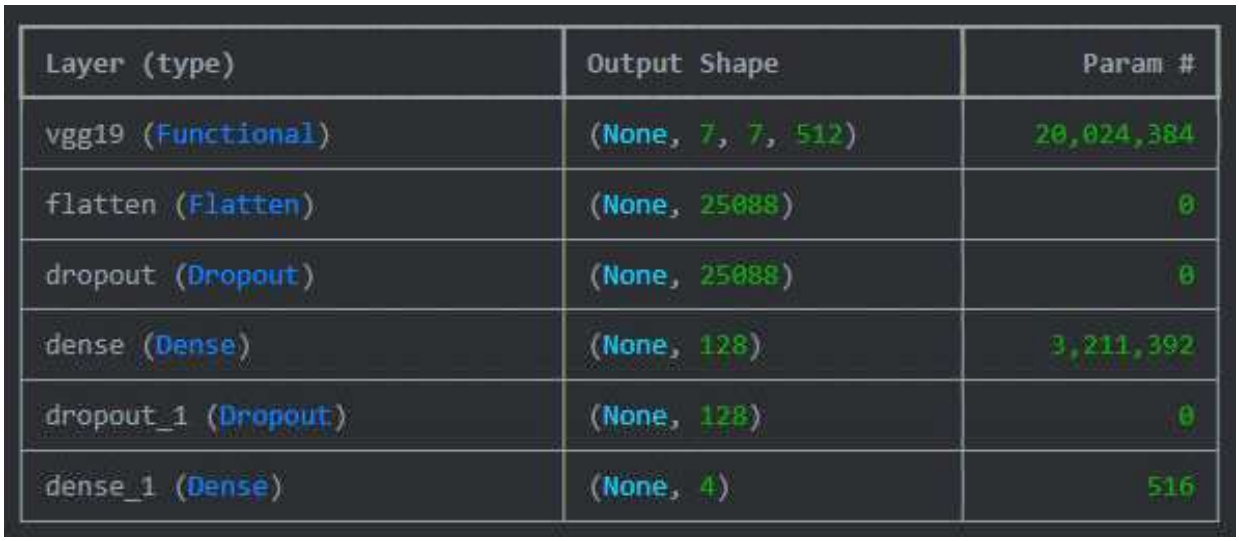
1. ***Top row*** *: 1st image - pituitary, 2nd image - no tumor, 3rd image - pituitary, 4th image : meningioma*

2. ***Second row*** *: 1st image - meningioma, 2nd image - no tumor, 3rd image - pituitary, 4th image: no tumor*

3. ***Third row*** *: 1st image - notumor, 2nd image - glioma, 3rd image - meningioma, 4th image: meningioma*

4. ***Bottom row*** *: 1st image - pituitary, 2nd image - pituitary, 3rd image - pituitary, 4th image: no tumor*

- ***CREATING MODEL ARCHITECTURE***



Layer (type)	Output Shape	Param #
vgg19 (Functional)	(None, 7, 7, 512)	20,024,384
flatten (Flatten)	(None, 25088)	0
dropout (Dropout)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

Fig. 9.7 Model summary of VGG-19 architecture

- ***VGG19 Functional Layer :*** *re-trained VGG19 network truncated to output a tensor of shape (7, 7, 512).*
- ***Flatten Layer :*** *Flattens the output to a 1D tensor of shape (25088)*
- ***Dropout Layer :*** *Applied to prevent overfitting, keeping the tensor shape as (25088)*
- ***Dense Layer :*** *Fully connected layer reducing the dimension to (128)*
- ***Dropout Layer :*** *Another dropout layer to prevent overfitting, maintaining the shape (128)*
- ***Dense Layer :*** *Final fully connected layer reducing the output to 4 classes, resulting in a shape of (4).*

- ***MODEL TRAINING***

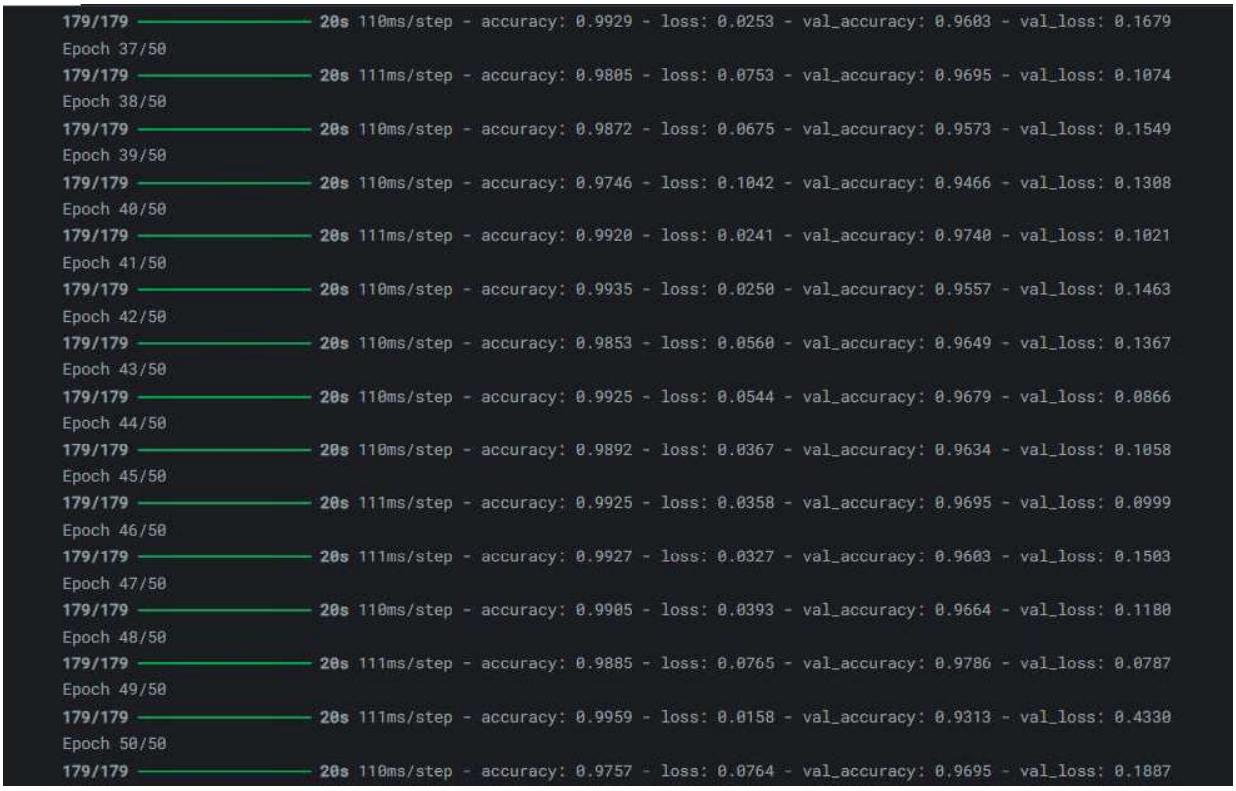


Fig. 9.8 Few epochs for VGG-19 model training

- ***VISUALIZATION***

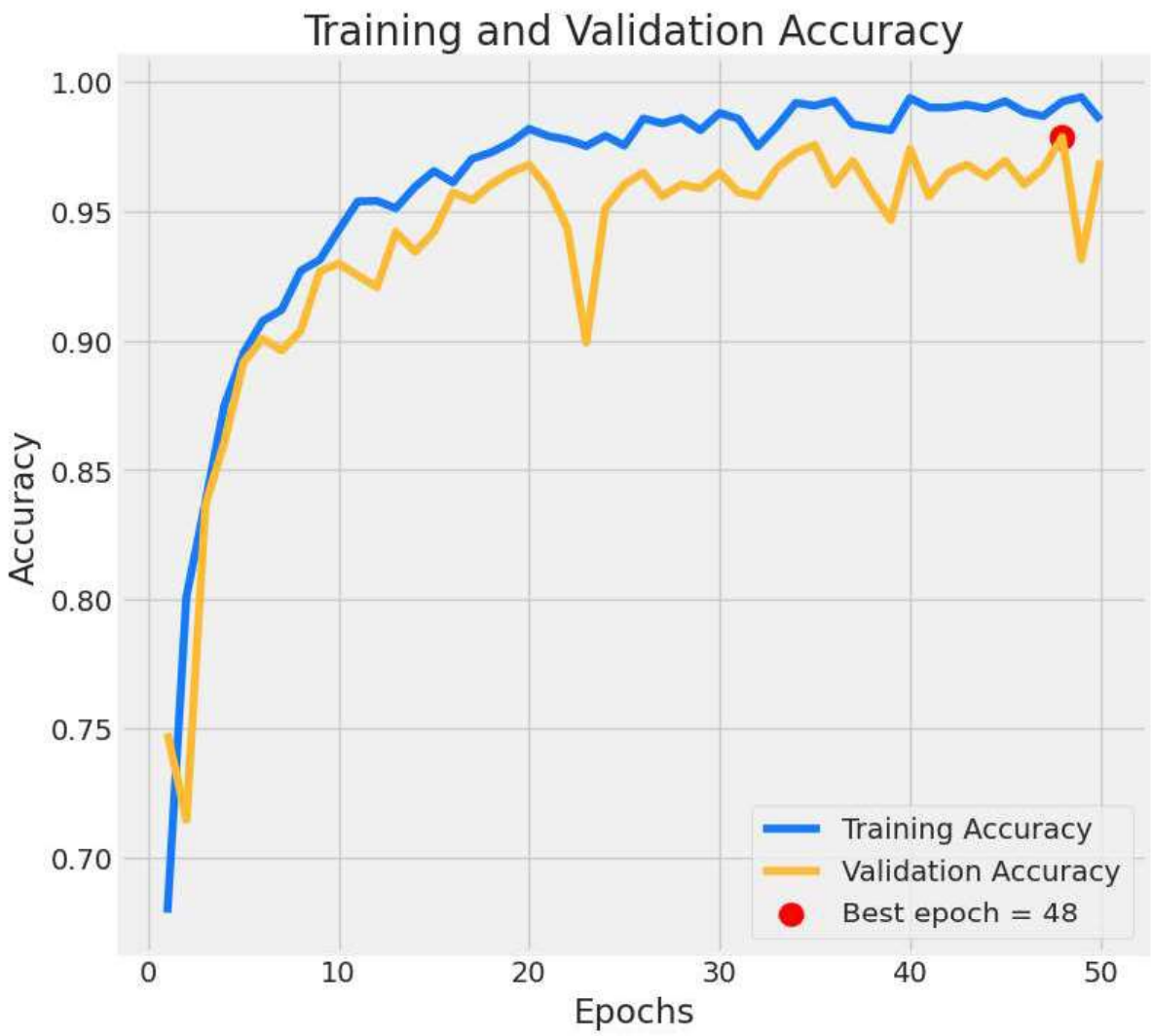


Fig. 9.9 Training vs validation accuracy

From the plot, we can see that the training accuracy steadily increases over the course of training. The validation accuracy initially increases, but it eventually plateaus. This is a common pattern that can be observed when training a machine learning model. The model is becoming increasingly accurate on the training data, but it is not necessarily improving on the validation data. This suggests that the model may be overfitting to the training data. Overfitting occurs when a model learns the training data too well and is not able to generalize to new data.

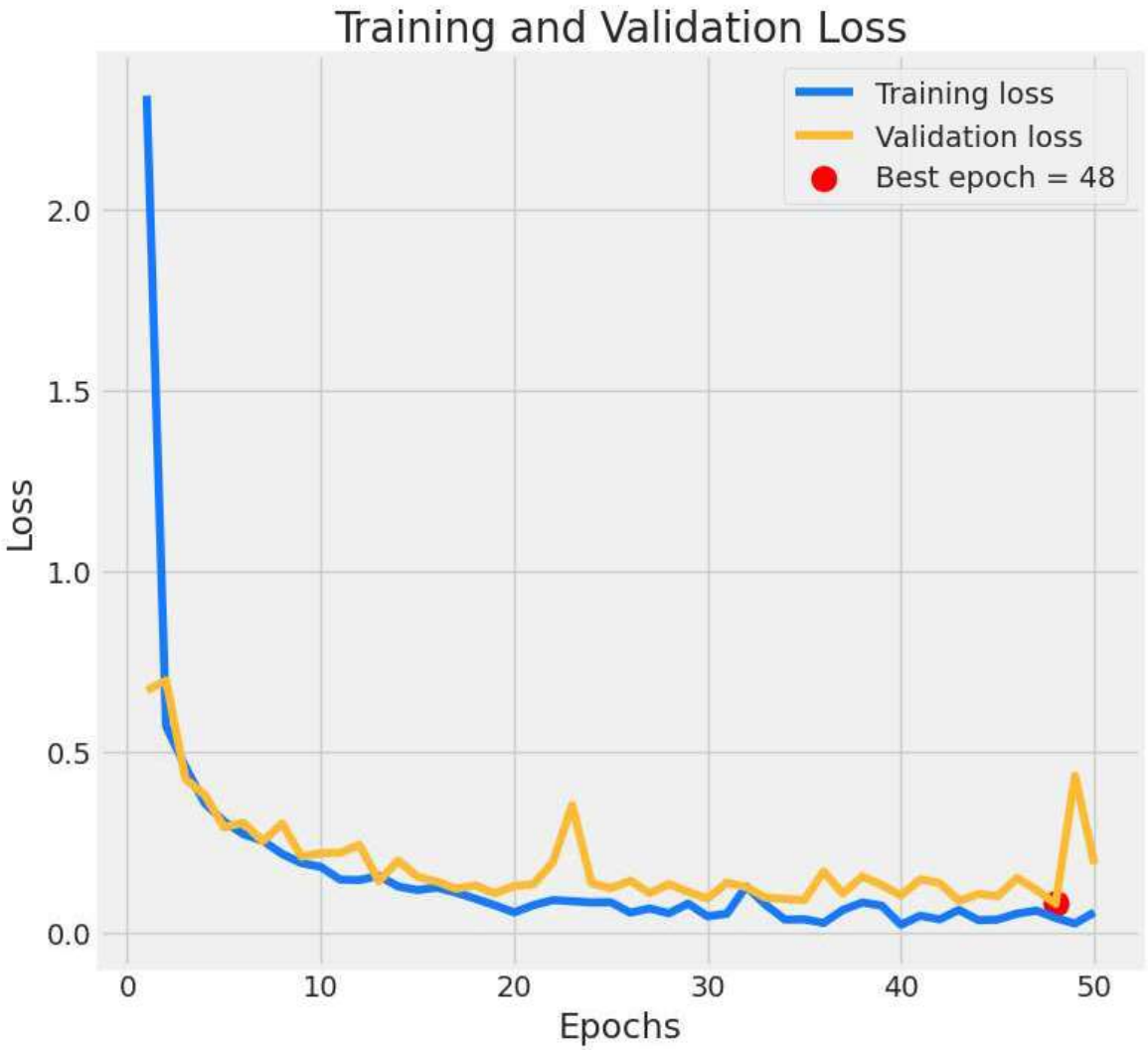


Fig. 9.10 Training vs validation loss

The plot shows that the training loss decreases steadily over the epochs, while the validation loss fluctuates. It suggests that the model starts to overfit the training data after around 20 epochs, as the validation loss starts to increase while the training loss continues to decrease. The best epoch for the model is epoch 48, which suggests that the model performs well on new data after being trained for 48 epochs.

- ***EVALUATION***

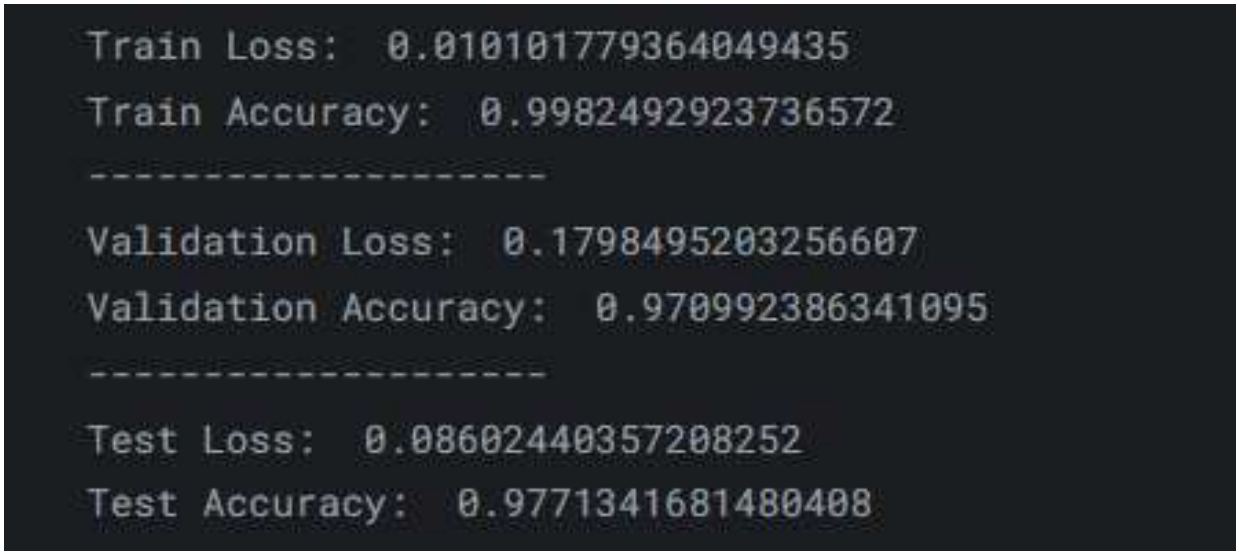


Fig. 9.11 Evaluation of the model

DATA	ACCURACY	LOSS
TRAIN	99.8%	0.1%
VALIDATION	97%	1.79%
TEST	97.7%	0.8%

Table 9.1 Table of evaluated results of VGG-19

1. *Train Accuracy represents how accurately the model predicts the labels of the training data.*

2. *Train Loss measures how well the model is learning from the training data. A lower loss is better.*

3. *Validation accuracy is similar to train accuracy, but it's calculated on the validation set.*

4. *Validation loss is similar to train loss, but it is calculated on a separate dataset called the validation set.*

5. *Test accuracy is the same as validation accuracy but calculated on the test set.*

6. *Test loss is similar to validation loss but calculated on the test set.*

- ***CONFUSION MATRIX GENERATION***

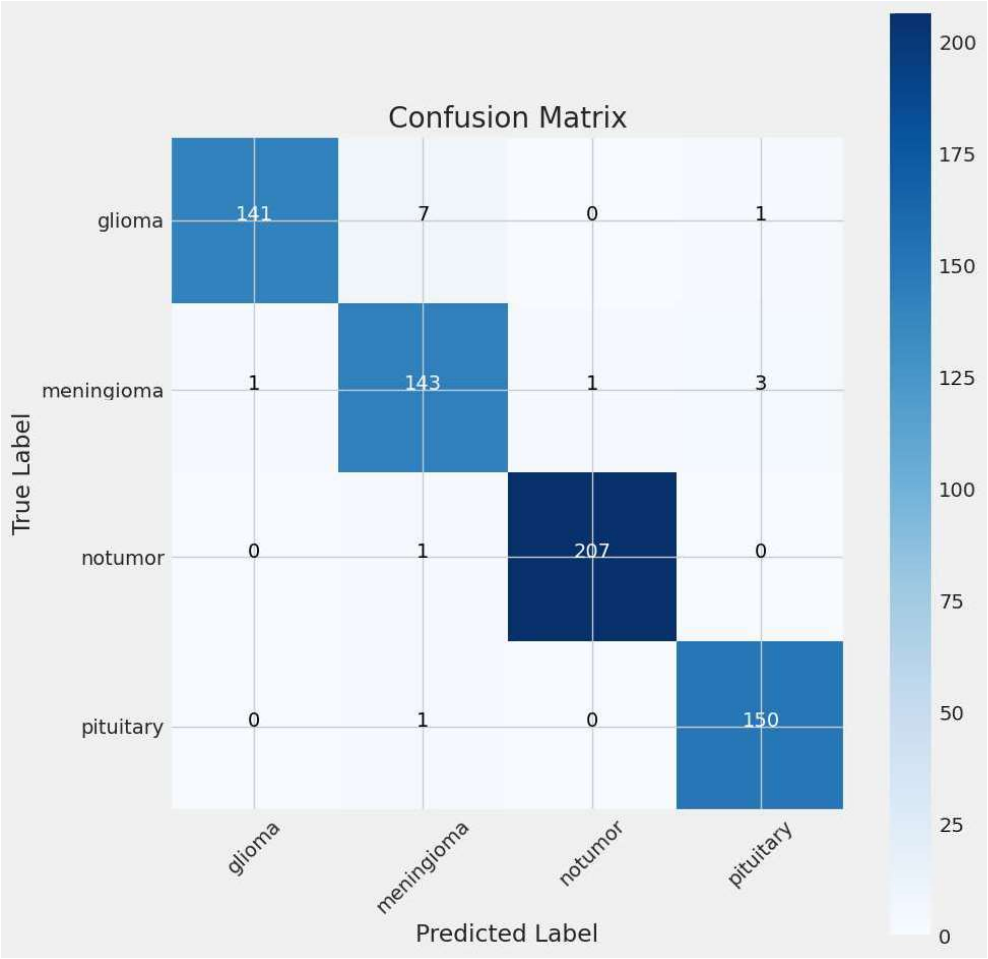


Fig. 9.12 Confusion matrix of the implemented VGG-19 model

1. *Glioma*

   Correctly classified 141 times.

   Misclassified as Meningioma 7 times, No tumor 0 times, and Pituitary 1 time.

2. *Meningioma*

   Correctly classified 143 times.

   Misclassified as Glioma 1 times, No tumor 1 time, and Pituitary 3 times.

3. *No tumor*

   Correctly classified 205 times.

   Misclassified as meningioma 1 time

4. *Pituitary*

   Correctly classified 150 times.

   Misclassified as Meningioma 1 time

- ***EVALUATION METRICES***

	precision	recall	f1-score	support
glioma	0.99	0.95	0.97	149
meningioma	0.94	0.97	0.95	148
notumor	1.00	1.00	1.00	208
pituitary	0.97	0.99	0.98	151
accuracy			0.98	656
macro avg	0.98	0.98	0.98	656
weighted avg	0.98	0.98	0.98	656

Fig. 9.13 Resultant evaluation metrices

Chapter 10

Execution by EfficientNet

Representation of EfficientNet B3 as a part of the brain tumor detection and classification system

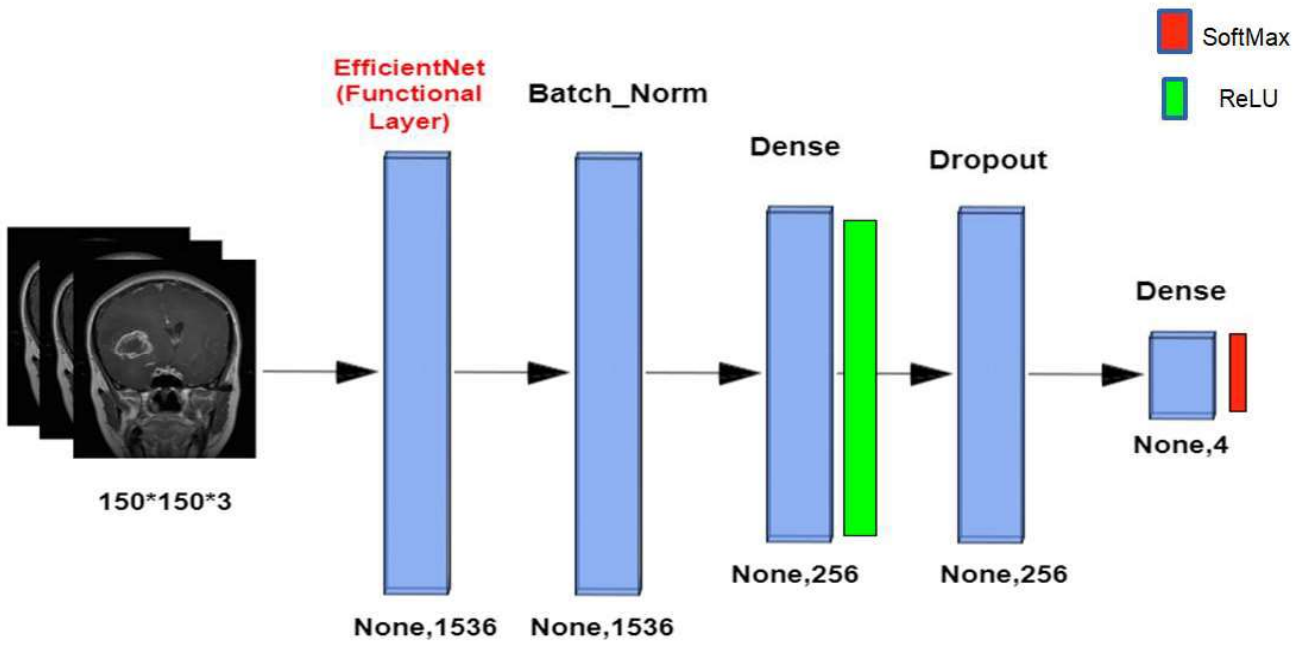


Fig. 10.1 EfficientNet B3 model representation of our system

The above image shows the path of work of EfficientNet B3 in our system "Brain Tumor Detection and Classification system using DeepNets"

10.1 Implementation steps

- ***Step 1* : IMPORTING NECESSARY LIBRARIES & MODULES**

The first stage towards implementing the EfficientNet algorithm in the brain tumor detetcion and classification system includes importing a variety of libraries and modules for system operations, data handling, visualization, and deep learning.

Here's a breakdown of what each section does:

System Libraries -

1. *os*
2. *time*
3. *shutil*
4. *pathlib*
5. *itertools*

Data Handling Tools -

1. *cv2*
2. *numpy*
3. *pandas*
4. *seaborn*
5. *matplotlib*

Deep Learning Libraries -

1. *tensorflow*
2. *keras*

- ***Step 2* : LOADING THE TRAINING DATA**

1. *Here, the image file paths and their corresponding labels are read from a directory structure and stored them in a pandas DataFrame.*
2. *The training data along with its subfolders are defined.*

3. *A pandas series created with the file paths and labels are created and concatenated.*

4. *The DataFrame created is useful for further processing and model training, as it consolidates all necessary information about the images and their labels in a structured format.*

- ***Step 3 : LOADING THE TESTING DATA***

1. *Here the image file paths and their corresponding labels from a test data directory and stores them in a pandas DataFrame.*

2. *The path to the test data directory is defined.*

3. *On iterating through the directory, a pandas Dataframe is created as it is useful for further processing and evaluation of the test data and consolidates all necessary information about the images and their labels in a structured format*

- ***Step 4 : TRAINING, VALIDATION AND TESTING DATA GENERATION***

1. *The batch size, image size, number of channels, etc. are generated.*

   ***Batch size** : 16*

   ***Image size** : (224, 224)*

   ***No. of channels** : 3*

2. *ImageDataGenerator() is used for training and testing data.*

3. *Batches of augmented image data are generated from the training, validation and testing DataFrames.*

4. *The images are resized to 224x224 pixels and loaded in RGB color mode.*

- ***Step 5 : MODEL DEVELOPMENT***

1. *A deep learning model is created for image classification using transfer learning with the EfficientNetB3 architecture.*

   Image shape used : *224x224 pixels with 3 color channels.*

2. *The pre-trained EfficientNet pre-trained model with linear stack of layers and batch normalization is used.*

3. *Fully connected layer (Dense) is used with 256 units.*

4. *Model is compiled.*

   Model : *Sequential*

   Optimizer : *Adam*

   Loss function *: Categorical cross-entropy loss*

   Metrics used : *Accuracy*

- ***Step 6 : RESULT GENERATION***

The results like training accuracy and loss curve plotting, confusion matrix, etc. are generated in order to identify the overall performance of the algorithm.

10.2 Experimental Results

This section provides a comprehensive overview of the experimental results for the brain tumor detection and classification system with the implementation of the EfficientNet architecture.

- ***RESULTS OF GENERATING TRAINING, VALIDATION & TESTING DATA***

```
Found 5712 validated image filenames belonging to 4 classes.
Found 656 validated image filenames belonging to 4 classes.
Found 655 validated image filenames belonging to 4 classes.
```

Fig. 10.2 Results of data split

This is a log output from a program which is performing image classification.
The output suggests that the program has found 5712, 656, and 655 image filenames, respectively.
These image files belong to 4 different classes i.e, Glioma, Meningioma, No tumor and Pituitary

- ***DISPLAYING SAMPLE IMAGES***

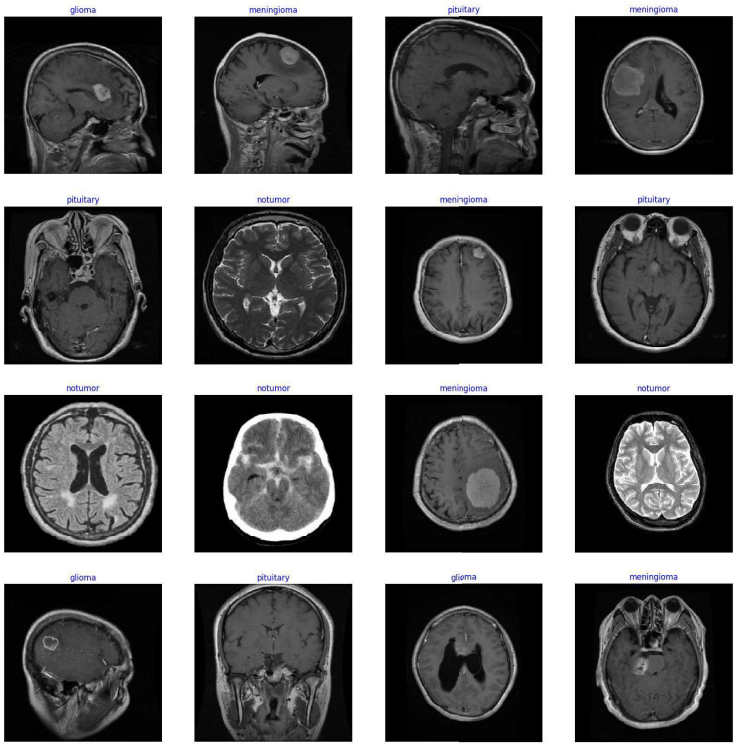


Fig. 10.3 Sample images from dataset

The above image shows the collection of MRI scans of brain images, each labeled with a specific category related to brain tumors or the absence of tumors.

These images are the ones before training the EfficientNet architecture.

The image is organized into a grid with multiple rows and columns.

The labels above each MRI scan indicate the category of the image i.e Glioma, Meningioma, Pituitary or no tumor.

These images are a collection of all the 4 types of tumors present in the dataset.

- ***MODEL SUMMARY***

The model given in the next page shows the summary of a Keras Sequential model.

This model is designed for image classification. It uses a pre-trained efficientnet B3

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb3 (Functional)  (None, 1536)             10783535

 batch_normalization (BatchN  (None, 1536)             6144
 ormalization)

 dense (Dense)               (None, 256)               393472

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 4)                 1028

=================================================================
Total params: 11,184,179
Trainable params: 11,093,804
Non-trainable params: 90,375
_________________________________________________________________
```

Fig. 10.4 Model summary after applying EfficientNet

network to extract features from an image, then uses fully connected layers to classify those features into one of 4 categories (based on the output shape of dense 1).

- ***MODEL TRAINING***

```
Epoch 17/25
357/357 [==============================] - 1599s 4s/step - loss: 0.0998 - accuracy: 0.9982 - val_loss: 0.0946 - val_accura
cy: 0.9954
Epoch 18/25
357/357 [==============================] - 1587s 4s/step - loss: 0.0913 - accuracy: 0.9991 - val_loss: 0.0919 - val_accura
cy: 0.9954
Epoch 19/25
357/357 [==============================] - 1565s 4s/step - loss: 0.0941 - accuracy: 0.9977 - val_loss: 0.0885 - val_accura
cy: 0.9939
Epoch 20/25
357/357 [==============================] - 1582s 4s/step - loss: 0.0850 - accuracy: 0.9991 - val_loss: 0.0988 - val_accura
cy: 0.9908
Epoch 21/25
357/357 [==============================] - 1597s 4s/step - loss: 0.0832 - accuracy: 0.9989 - val_loss: 0.0854 - val_accura
cy: 0.9969
Epoch 22/25
357/357 [==============================] - 1566s 4s/step - loss: 0.0847 - accuracy: 0.9984 - val_loss: 0.0784 - val_accura
cy: 0.9969
Epoch 23/25
357/357 [==============================] - 1546s 4s/step - loss: 0.0827 - accuracy: 0.9982 - val_loss: 0.0878 - val_accura
cy: 0.9969
Epoch 24/25
357/357 [==============================] - 1559s 4s/step - loss: 0.0801 - accuracy: 0.9989 - val_loss: 0.0870 - val_accura
cy: 0.9954
Epoch 25/25
357/357 [==============================] - 1559s 4s/step - loss: 0.0781 - accuracy: 0.9988 - val_loss: 0.0651 - val_accura
cy: 0.9985
```

Fig. 10.5 Few epochs for model training

- ***VISUALIZATION OF THE RESULTS***

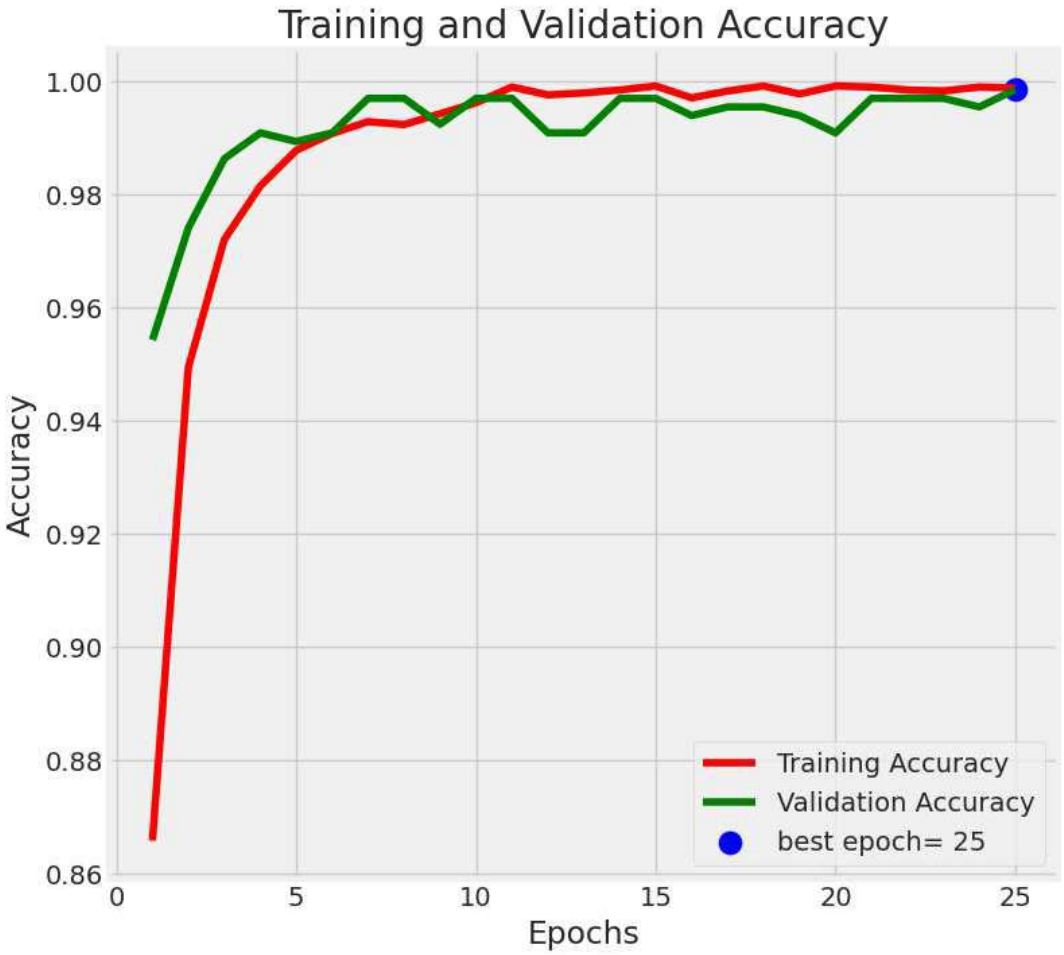


Fig. 10.6 Training & validation accuracy graph

This graph shows the training and validation accuracy of a machine learning model over 25 epochs. Both the training and validation accuracies increase steadily in the early epochs, indicating that the model is learning effectively. The graph suggests that the model has achieved a good level of accuracy, but it may be overfitting to the training data. This could be addressed by using techniques such as early stopping or regularization.

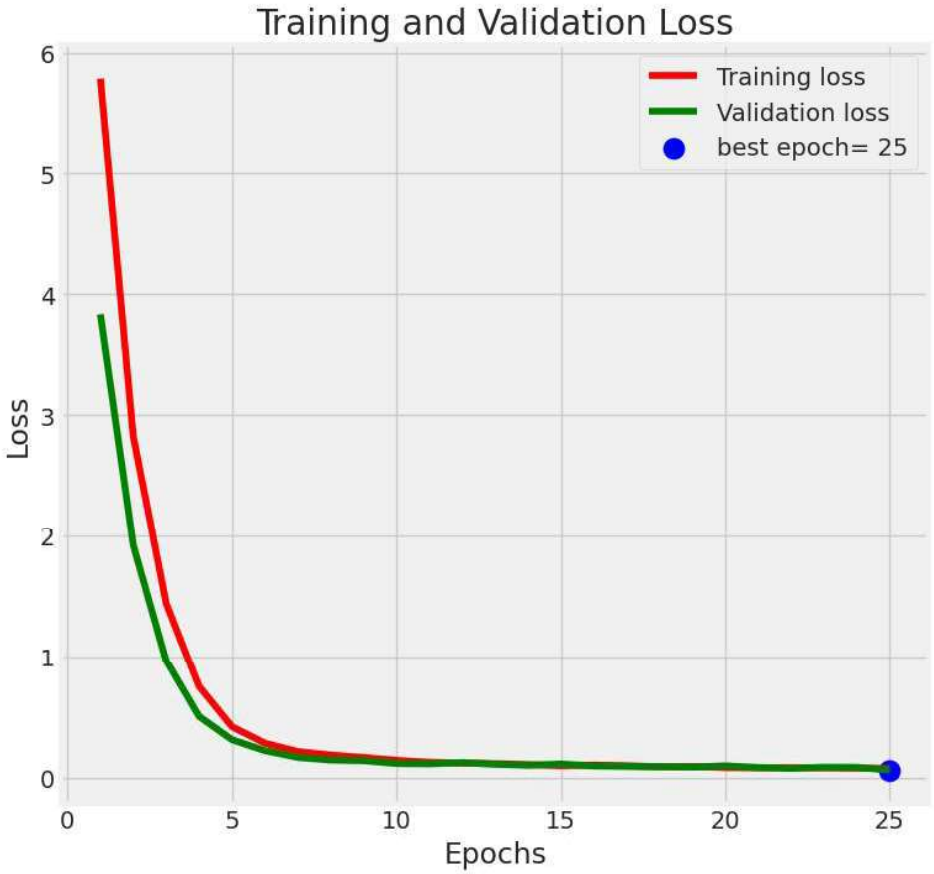


Fig. 10.7 Training & validation loss graph

We can see that the training loss starts high and decreases rapidly in the beginning, while the validation loss remains relatively flat. This is a common pattern in machine learning training, as the model initially learns to fit the training data very well but does not yet generalize well to unseen data. As training progresses, both the training loss and validation loss continue to decrease but at a slower pace.

- ***CONFUSION MATRIX***

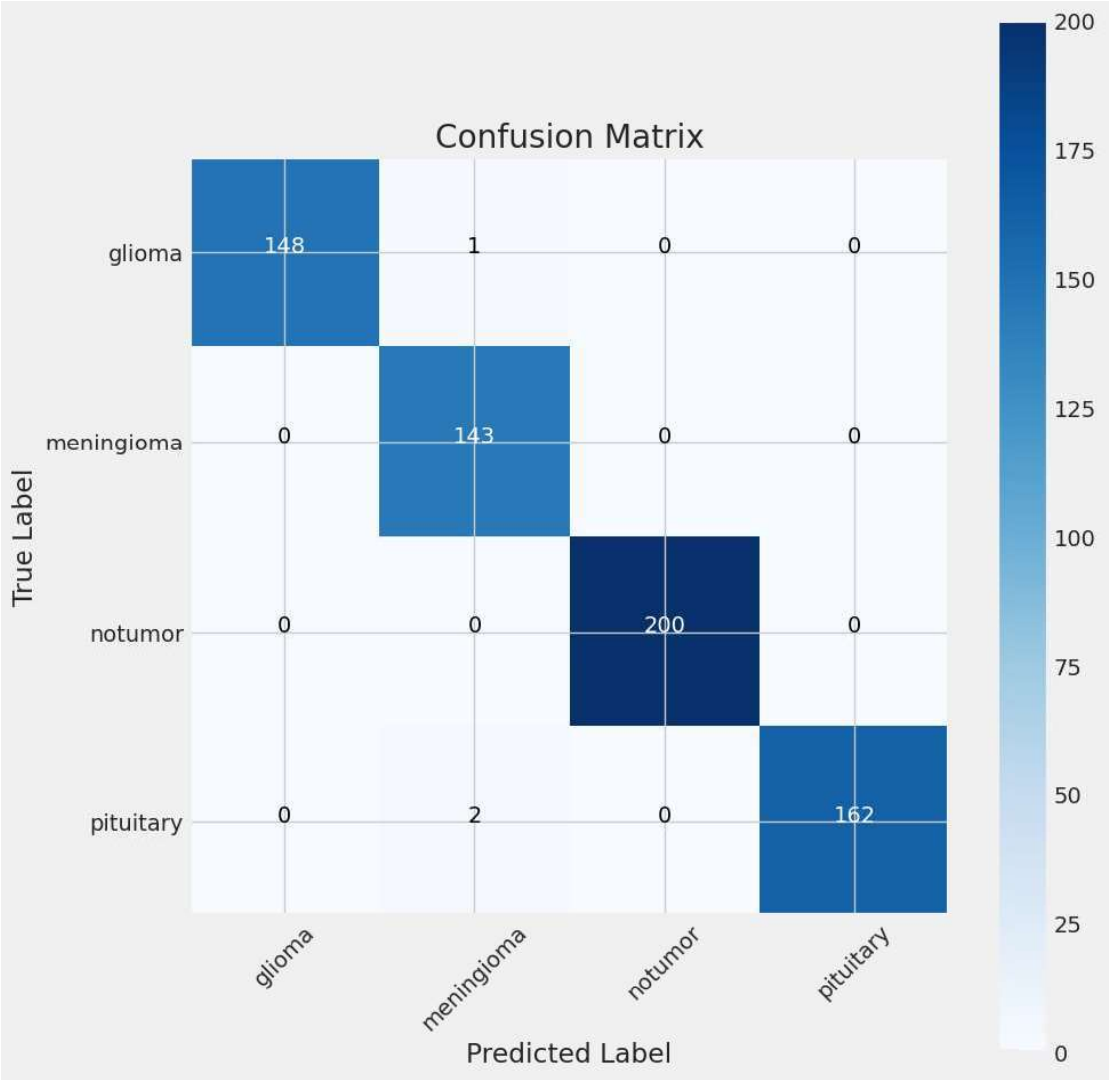


Fig. 10.8 Generated confusion matrix

1. *Glioma : Correctly classified 141 times*

2. *Meningioma : Correctly classified 143 times*

3. *No tumor : Correctly classified 205 times.*

4. *Pituitary : Correctly classified 150 times.*

Chapter 11

Execution by Yolo v8

Representation of Yolo v8 as a part of the brain tumor detection and classification system

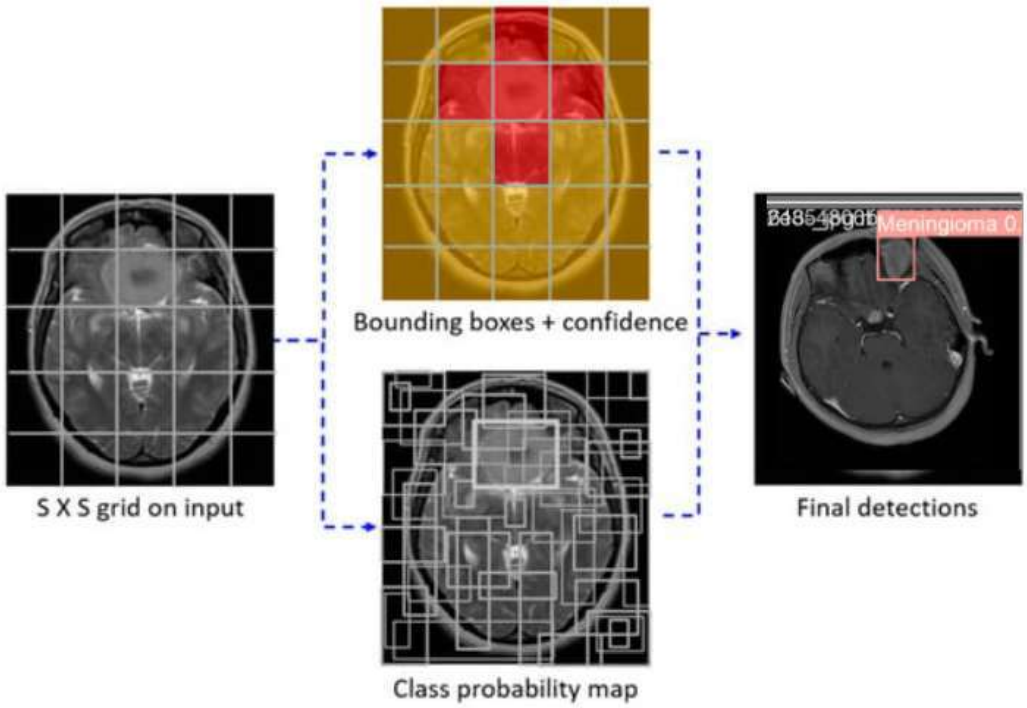


Fig. 11.1 Yolo v8 model representation of our system

he above image shows the path of work of Yolo v8 in our system "Brain Tumor Detection and Classification system using DeepNets"

YOLO v8 is an important tool for brain tumor detection and classification systems due to its efficiency, accuracy, real-time processing, deep learning capabilities, transfer learning, multi-scale prediction, and end-to-end system.
In the "Brain Tumor Detection and Classification System using DeepNets" project, we have used the yolo v8 algorithm in order to detect the brain tumors specifically from the custom made dataset.

11.1 Implementation steps

1. *Installing YOLOv8*
2. *CLI Basics*
3. *Inference with Pre-trained COCO Model*
4. *Roboflow Universe*
5. *Preparing a custom dataset*
   Step 1 : *Creating project*
   Step 2 : *Uploading images*
   Step 3 : *Labelling*
   Step 4 : *Generating new dataset versions*
   Step 5 : *Exporting dataset*
6. *Custom training*
7. *Validating inference model*
8. *Inferencing with custom model*
9. *Deploying model on Roboflow*

11.2 Experimental results

This section gives an overview of the experimental results that are obtained after executing the YOLO v8 algorithm on the custom brain tumor training dataset generated from Roboflow.

- ***ANNOTATIONS MADE IN BRAIN MRI SCANS***

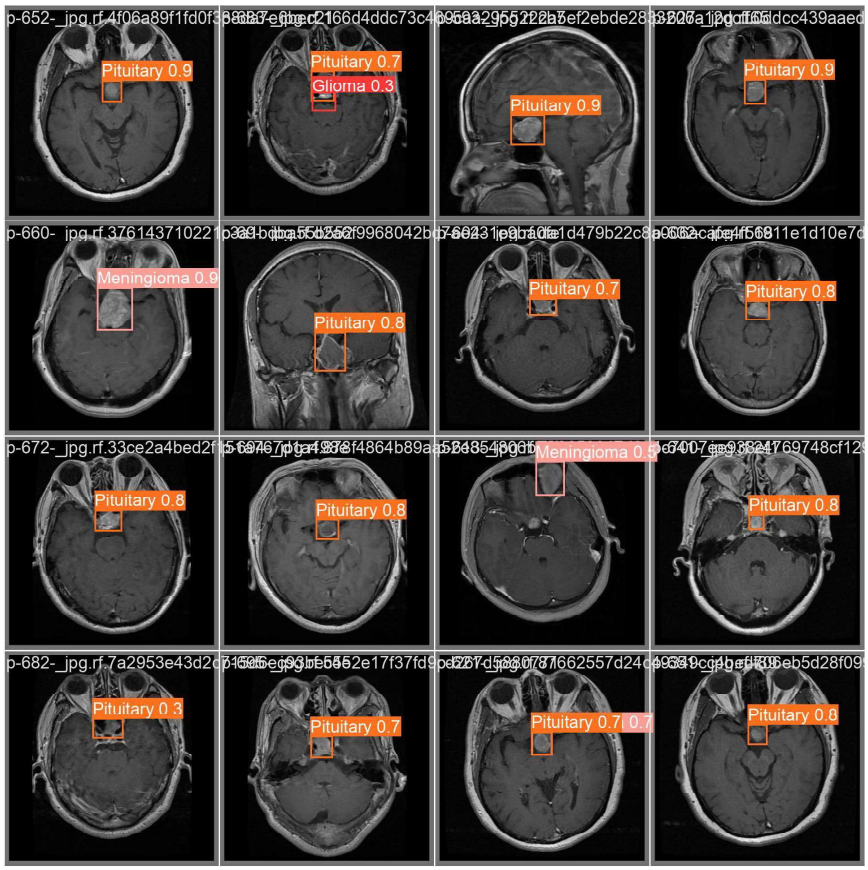


Fig. 11.2 Annotated MRI images

Here is an explanation of the annotations:

1. Types of Tumors identified :-

- ***Pituitary Tumor :*** *Annotated with "Pituitary" followed by a confidence score.*

- ***Glioma :*** *Annotated with "Glioma" followed by a confidence score.*
- ***Meningioma :*** *Annotated with "Meningioma" followed by a confidence score.*

2. Confidence Scores :-

- *Each annotation is followed by a numerical value between 0 and 1.*
- *This value represents the confidence level of the model's prediction. For example, "Pituitary 0.9" means the model is 90% confident that the identified region is a pituitary tumor.*

3. Bounding Boxes :-

- *Each detected tumor is marked with a colored bounding box.*
- *The color and positioning highlight the area where the model detected the tumor.*

The architecture is typically divided into three main parts:

- ***Backbone :*** *The part of the network responsible for feature extraction from the input image.*
- ***Neck :*** *The part that aggregates features from different scales and passes them to the head.*
- ***Head :*** *The part that performs the final detection task, producing bounding box coordinates, objectness scores, and class probabilities.*

- *VISUALIZATIONS THROUGH GRAPH*

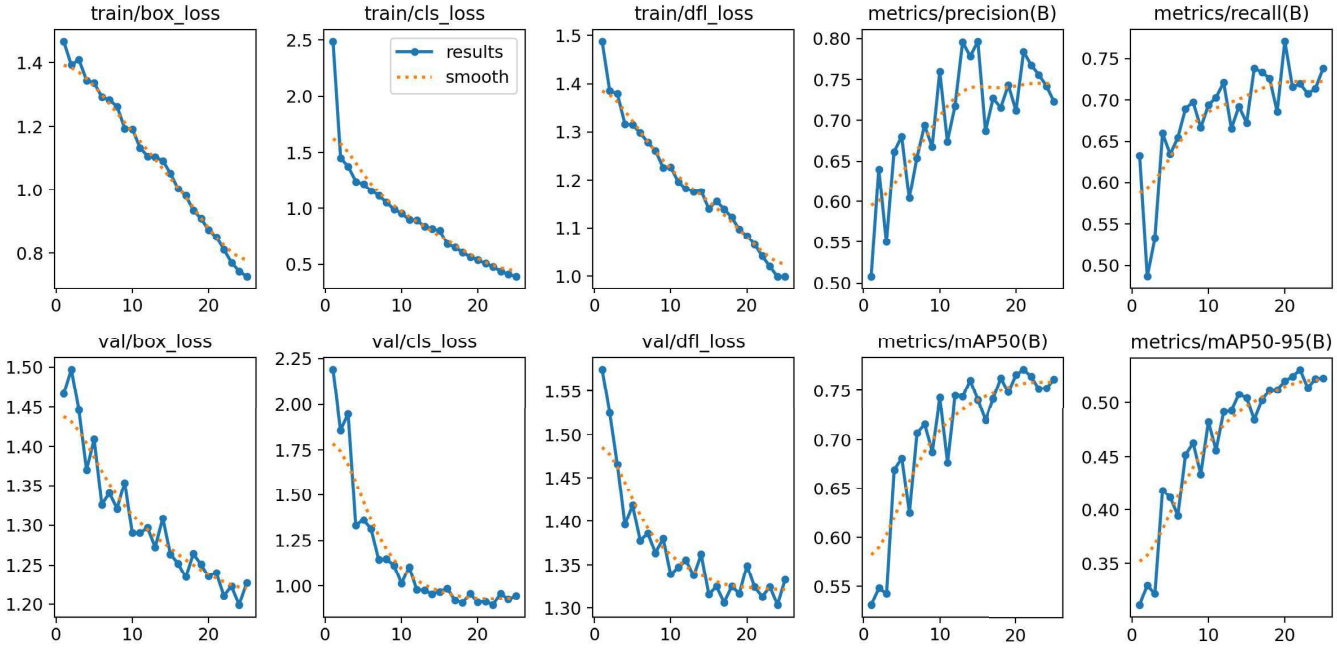


Fig. 11.3 Graphical view of results

The above image contains a set of graphs that appear to represent the training and validation performance metrics of a YOLO (You Only Look Once) object detection model over a number of epochs.

The general observations from these graphs are :-

1. ***Training Losses :*** *All are showing a consistent decreasing trend, which is a positive sign that the model is learning effectively during training.*

2. ***Validation Losses :*** *Similar decreasing trends in validation losses indicate that the model is not only learning but also generalizing well to unseen data.*

3. ***Precision and Recall :*** *Increasing trends in both metrics suggest that the model is improving in both identifying true positives and reducing false positives.*

4. ***mAP Metrics :*** *Increasing trends in mAP50 and mAP50-95 confirm that the overall detection performance of the model is improving across different IoU thresholds.*

- *CONFUSION MATRIX*

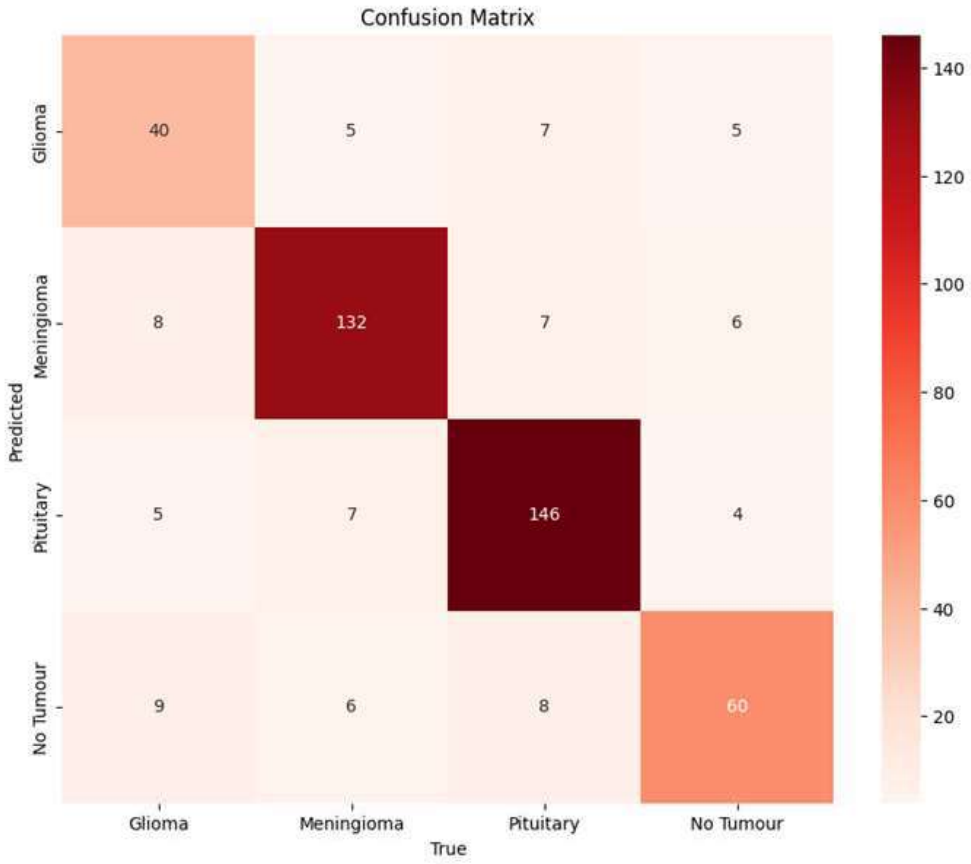


Fig. 11.4 Generated confusion matrix

In this confusion matrix, we can see that the model is doing a good job at classifying meningiomas, with 132 out of 147 being correctly classified. It is also doing a good job at classifying pituitary tumors, with 146 out of 167 being correctly classified. The model is not doing as well at classifying gliomas and backgrounds, with 40 out of 62 gliomas and 14 out of 43 backgrounds being correctly classified.

Here are some more detailed observations from the confusion matrix:

1. ***True Positives (TP) :*** *The number of instances that were correctly classified as belonging to the positive class. For example, there are 132 true positives for meningiomas.*

2. ***True Negatives (TN) :*** *The number of instances that were correctly classified as belonging to the negative class. For example, there are 146 true negatives for pituitary tumors.*

3. ***False Positives (FP) :*** *The number of instances that were incorrectly classified as belonging to the positive class. For example, there are 7 false positives for meningiomas.*

4. ***False Negatives (FN) :*** *The number of instances that were incorrectly classified as belonging to the negative class. For example, there are 8 false negatives for meningiomas.*

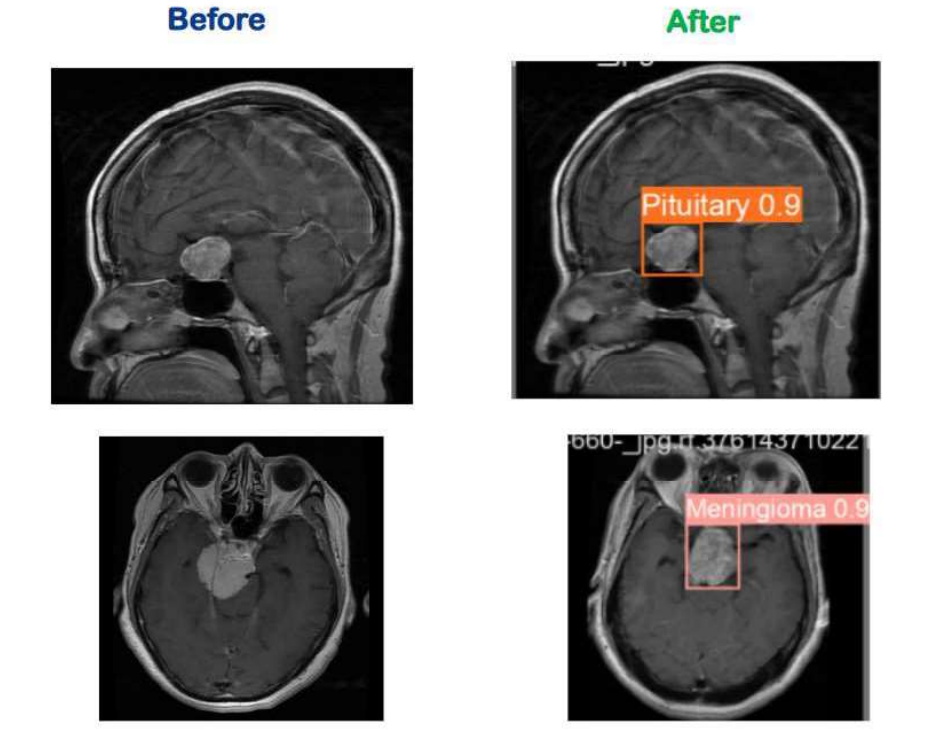


Fig. 11.5 Before vs after images representation of Yolo v8

Chapter 12

Conclusion and Future Work

CONCLUSION :

In conclusion, a brain tumor detection project using deep learning involves the development and training of a neural network, typically a convolutional neural network (CNN), to analyze medical images and identify the presence of tumors.

In brain tumor detection we have studied about feature based existing work. In feature based we have study about image processing techniques likes image pre-processing, image segmentation, features extraction, classification and also study about deep learning techniques CNN and VGG16. In this system we have detect the tumor is present or not if the tumour is present then model return's yes otherwise it return no.

In summary, a successful brain tumor detection project involves a combination of data quality, model development, ethical considerations, and collaboration with medical professionals. It is an iterative process that requires continuous improvement and validation to ensure the model's effectiveness and safety in real-world healthcare settings.

COMPARITIVE ANALYSIS OF THE RESULTS OF ALL THE ALGORITHMS USED IN THIS SYSTEM

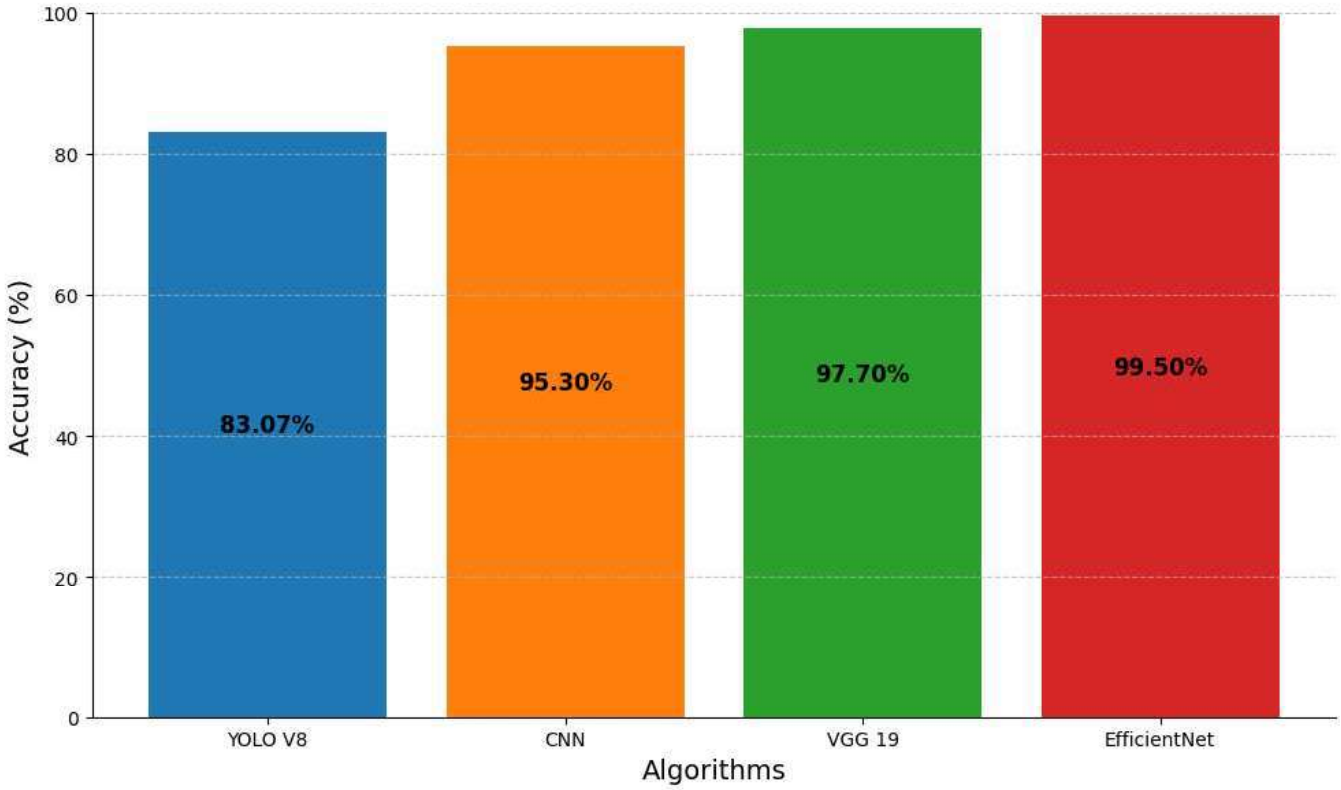


Fig. 12.1 Comparative analysis

FUTURE WORK :

1. ***Explainable AI :*** *To use XAI methods like, LIME (Local Interpretable Model-agnostic Explanations) and SHAP (Shapley Additive explanations)*

2. ***Website creation :*** *To create a website for deploying the model by using Flask.*

3. ***Secondary tumors :*** *To identify on secondary brain tumors for indirectly saving people's lives in the long run.*

4. ***Provide diagonosis :*** *To provide clinical advices to patients on identifying brain tumors.*

5. ***Visual explanations*** **:** *Using techniques like Grad-CAM and saliency maps to highlight areas of importance in brain scans.*

6. ***Patient-Specific Models*** **:** *Developing models tailored to individual patients based on their unique genetic and clinical profiles.*

References

[AH18] *Ali Ari and Davut Hanbay. Deep learning based brain tumor classification and detection system.* Turkish Journal of Electrical Engineering and Computer Sciences, *26(5):2275–2286, 2018.*

[BNK15] *Vipin Y Borole, Sunil S Nimbhore, and Dr Seema S Kawthekar. Image processing techniques for brain tumor detection: A review.* International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), *4(5):2, 2015.*

[CLR14] *Said Charfi, Redouan Lahmyed, and Lalitha Rangarajan. A novel approach for brain tumor detection using neural network.* International Journal of Research in Engineering and Technology, *2(7):93–104, 2014.*

[DSS21] *Nadim Mahmud Dipu, Sifatul Alam Shohan, and KMA Salam. Deep learning based brain tumor detection and classification. In* 2021 International conference on intelligent technologies (CONIT), *pages 1–6. IEEE, 2021.*

[EVB+21] *Morteza Esmaeili, Riyas Vettukattil, Hasan Banitalebi, Nina R Krogh, and Jonn Terje Geitung. Explainable artificial intelligence for human-machine interaction in brain tumor localization.* Journal of Personalized Medicine, *11(11):1213, 2021.*

[KPSP20] *Thiruvenkadam Kalaiselvi, Thiyagarajan Padmapriya, Padmanaban Sriramakrishnan, and Venugopal Priyadharshini. Development of automatic glioma brain*

tumor detection system using deep convolutional neural networks. International Journal of Imaging Systems and Technology*, 30(4):926–938, 2020.*

[ÖSA20] Fatih Özyurt, Eser Sert, and Derya Avcı. An expert system for brain tumor detection: Fuzzy c-means with super resolution and convolutional neural network with extreme learning machine. Medical hypotheses*, 134:109433, 2020.*

[RPSM21] G Ramkumar, R Thandaiah Prabu, Ngangbam Phalguni Singh, and U Maheswaran. Withdrawn: Experimental analysis of brain tumor detection system using machine learning approach, 2021.

[RS19] PG Rajan and C Sundar. Brain tumor detection and segmentation by intensity adjustment. Journal of medical systems*, 43(8):282, 2019.*

[RV18] G Sethuram Rao and D Vydeki. Brain tumor detection approaches: a review. In 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 479–488. IEEE, 2018.*

[SANI16] Md Sujan, Nashid Alam, Syed Abdullah Noman, and Mohammed Jahirul Islam. A segmentation based automated system for brain tumor detection. International Journal of Computer Applications*, 153(10):41–49, 2016.*

[SGK15] P Shanthakumar and P Ganesh Kumar. Computer aided brain tumor detection system using watershed segmentation techniques. International Journal of Imaging Systems and Technology*, 25(4):297–301, 2015.*

[SSK+20] Md Abu Bakr Siddique, Shadman Sakib, Mohammad Mahmudur Rahman Khan, Abyaz Kader Tanzeem, Madiha Chowdhury, and Nowrin Yasmin. Deep cnvolutional neural networks model-based brain tumor detection in brain mri images. In 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 909–914. IEEE, 2020.*