Brain Tumor Detection Using Deep Learning

CNN Project

Author
- Omkar Umesh Gadade

1. Introduction

Brain tumor detection using MRI imaging is a vital computer-aided diagnosis task.
This report covers a full pipeline including exploratory data analysis, preprocessing, modeling, training, evaluation, and comparison of **CNN** and **ResNet50** architectures. The task provided to us is to:

- **The Task**

"**Build a lightweight deep learning system that detects and classifies brain tumors from MRI scans using compact CNN architectures. The model should efficiently identify tumor presence and type on limited hardware**."

- **Expected Outcome**:

"A trained classifier that labels MRI images as no tumor or specific tumor types, with suitable evaluation metrics."

So, to make full justification to the Task, I planned to create a **custom CNN** model from scratch; by adding each layer one by one and compiling it.

I took the liberties to load pretrained Model like the **ResNet50** which is a skip connection model and to compare its performance after training it on the MRI images with our own custom-made CNN model; to make the study more detailed and versatile.

The **Dataset** I used to train both the Models is taken from Kaggle from below link:

https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data

2. Literature Survey

Deep learning models—especially CNN-based architectures—have shown strong performance in medical imaging.

State-of-the-art approaches include transfer learning with **ResNet50, Efficient Net, Dense Net, and Vision Transformers.**

But the goal is to justify the Task and create a light CNN model architecture from scratch on a limited hardware.

Early brain tumor detection is life-saving, but the tools to automate it face a fundamental data barrier.

[A] The Clinical Imperative:

Research has shown that the most effective means of loIring death from Brain Cancer is early diagnosis and treatment**. Magnetic Resonance Imaging (MRI)** is one of the most efficient methods currently used for tumor detection.

[B] The Technical Barrier

State-of-the-art Deep Learning models, such as **VGG16, ResNet50**, often rely on large annotated datasets. These are challenging to collect in the medical field due to privacy concerns and high labelling costs, and training deep learning models from scratch can lead to **overfitting**.

A light weight, intelligently designed CNN can outperform data-hungry models on specialized Tasks like Brain Tumor Detection.

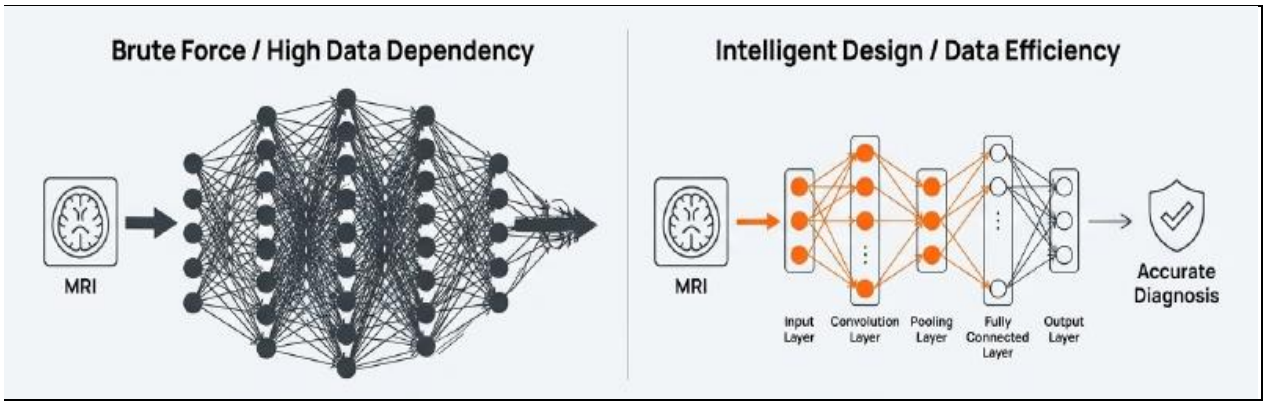


Fig: Design Choice

Instead of relying on massive pre trained networks, I build a lean CNN network from the ground up. The architecture was selected to balance classification accuracy and computational efficiency, making it suitable for prospective clinical use where resources can be **constrained.**

From simple edges to complex tumor shapes, the model learns a hierarchy of features

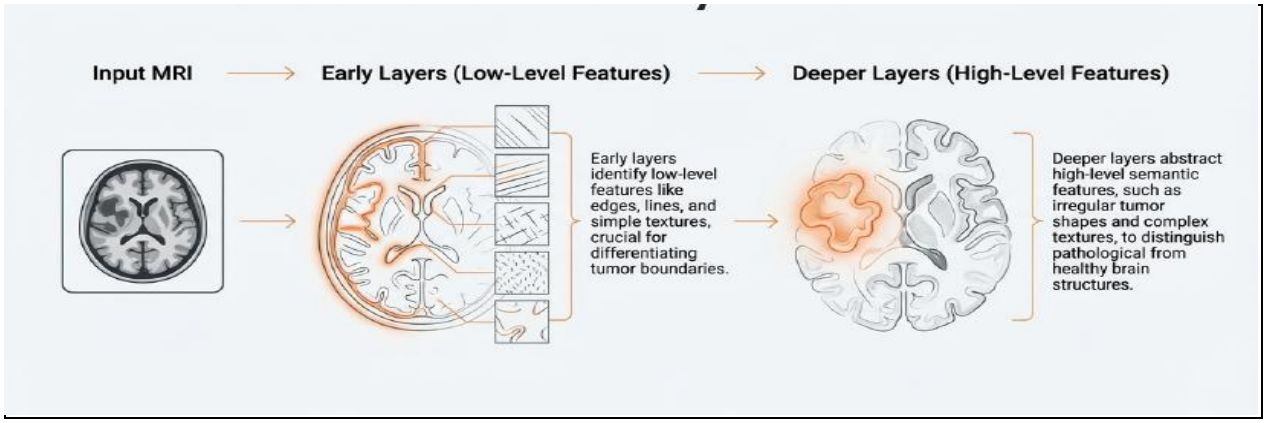


Fig: Feature learning layer by layer

The layers start's learning rich features as I go deeper and deeper in a traditional CNN model.

The Model's efficiency and accuracy create a clear path towards clinical integration.

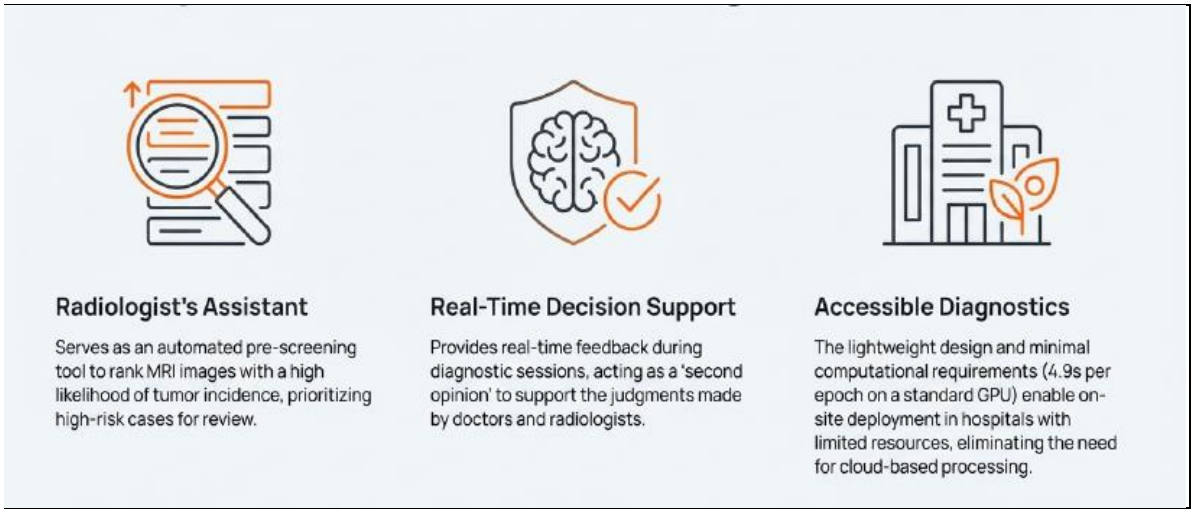


Fig: Benefits of a light model

Hence; the benefits of a light weight CNN model over pretrained heavy model tell us that the:

- **Training Speed** — Much Faster
- **Low Computational Cost** (RAM + VRAM)
- **Low Risk of Overfitting** on Small Datasets

Feature	**Light weight CNN**	**ResNet50/VGG**
Parameters	0.1M – 4M	25M-138M
Training Speed	Very Fast	Slow
GPU Memory	Low	High
Overfitting Risk	Low	High on small data
Interpretability	Easy	Hard
Deployment	Excellent	Poor
Real-Time-Use	Yes	Mostly No

3. Dataset Description & EDA

The dataset contains MRI images categorized into Training and Testing Folders.
EDA analyzes class balance, duplicates, corrupted images, and Plots.

The **EDA** and **Model** work is done on **Google Colab/Google Drive**

Please adhere to the following list:

The following list of files and Work distribution is present in the zip folder:

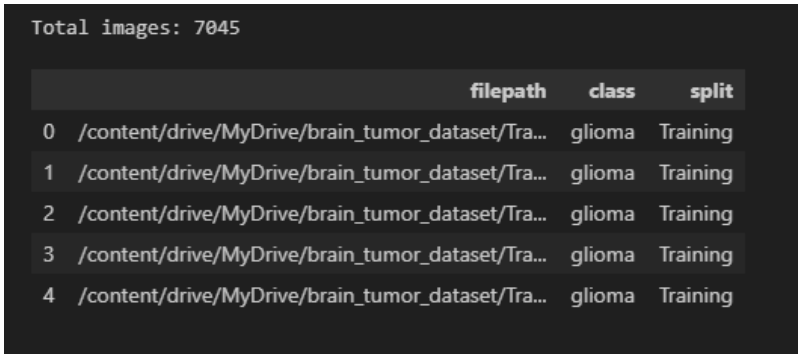
1. EDA_Brain_Tumor_Detection_Omkar.ipynb
2. Model_Brain_Tumor_Detection_Omkar.ipynb
3. Custom_CNN Model File (brain_tumor_model.h5)
4. ResNet50 Model File (resnet50_brain_tumor_model.h5)
5. **train_balanced_dataset** folder (obtained from EDA, for training the model)

EDA – Work:

In the EDA performed inside **EDA_Brain_Tumor_Detection_Omkar_Gadade.ipynb** I followed a certain list of steps:

The main aim is to save a **Balanced Training Data Folder** in the Google Drive:

1. I first Import the necessary Modules and libraries and Mount the Drive in Colab to read Files from Google Drive
2. I mention the File Paths for Train and Test Folders uploaded in our Google Drive
3. Then I setup and scan the Folder (Training+Test) into a Pandas DataFrame



Total images: 7045

	filepath	class	split
0	/content/drive/MyDrive/brain_tumor_dataset/Tra...	glioma	Training
1	/content/drive/MyDrive/brain_tumor_dataset/Tra...	glioma	Training
2	/content/drive/MyDrive/brain_tumor_dataset/Tra...	glioma	Training
3	/content/drive/MyDrive/brain_tumor_dataset/Tra...	glioma	Training
4	/content/drive/MyDrive/brain_tumor_dataset/Tra...	glioma	Training

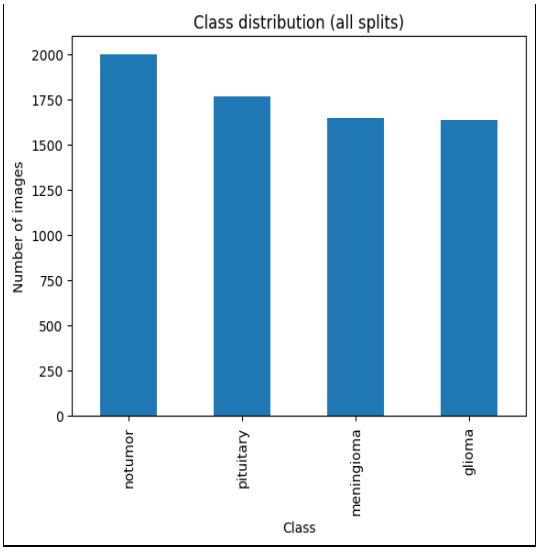
4. Basic EDA is performed like Split Count, Class Distribution and Class Distribution by split:

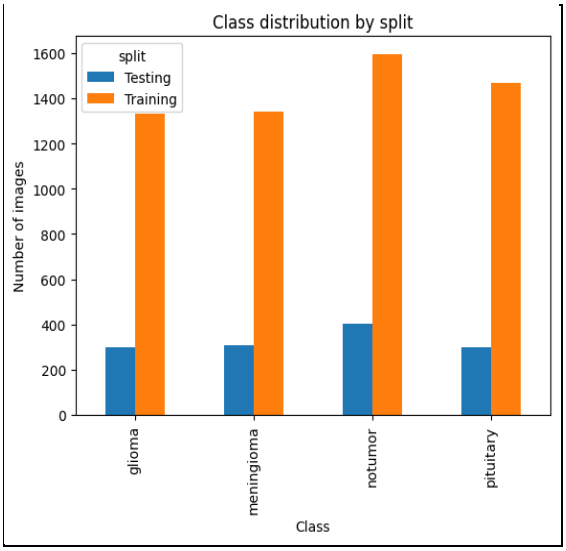
```
Split counts:
split
Training    5734
Testing     1311
Name: count, dtype: int64
```

```
Class distribution (overall):
class
notumor       2000
pituitary     1766
meningioma    1645
glioma        1634
Name: count, dtype: int64
```

```
Class distribution by split:
split     class
Testing   glioma          300
          meningioma      306
          notumor         405
          pituitary       300
Training  glioma         1334
          meningioma     1339
          notumor        1595
          pituitary      1466
dtype: int64
```

5. Plots are been plotted for visualization purpose:





6. I further Verified whether the Images are corrupted and removed them

```
Checking images: 100%|██████████| 7045/7045 [15:36<00:00,  7.52img/s]
Corrupted / unreadable images: 0


   filepath   class   split   is_ok
```

7. I Hashed the Images to remove any Duplicate Images lurking in the Dataset.

```
Detecting duplicates: 100%|██████████| 7045/7045 [00:00<00:00, 1610468.26img/s]
Number of exact duplicate images: 319
```

8. After Removing the Duplicates, I did Under Sampling for equal representation of MRI images per class category only for Training Data: -

```
Train counts per class:
class
pituitary     1445
notumor       1422
meningioma    1333
glioma        1321
Name: count, dtype: int64

Using min_count = 1321 for balanced sampling.
```

```
Balanced train counts per class:
class
glioma        1321
meningioma    1321
notumor       1321
pituitary     1321
Name: count, dtype: int64
```

9. I saved this Balanced Dataset Folder by creating a **new Balanced Training Dataset Folder** in the Drive apart from the Training and Testing Folders already present Now I use this Balanced Train Dataset to Train our Model in the **Model_Omkar_Gadade.ipynb** file

4. Methodology

Three models are built**: a Custom CNN** and a **transfer-learning ResNet50**

Model.ipynb – Work:

I created two models;

1. **Custom Lightweight CNN Model** from scratch
2. Loaded a Pretrained **ResNet50** Model

[A] Custom CNN Model code and architecture is as Follows:

```
model = Sequential()
```

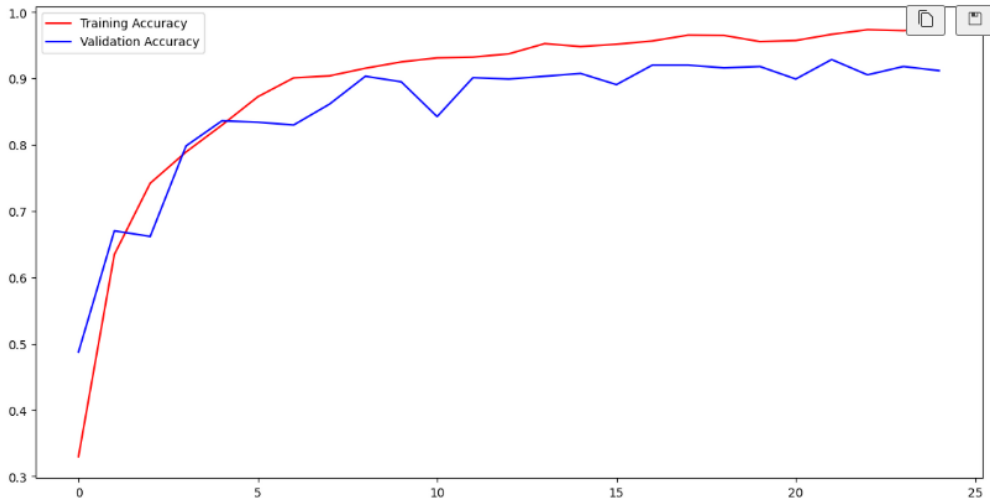
```
model.add(Conv2D(32,(3,3),activation = 'relu',input_shape=(150,150,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512,activation = 'relu'))
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(4,activation='softmax'))
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
conv2d_1 (Conv2D)	(None, 146, 146, 64)	18,496
max_pooling2d (MaxPooling2D)	(None, 73, 73, 64)	0
dropout (Dropout)	(None, 73, 73, 64)	0
conv2d_2 (Conv2D)	(None, 71, 71, 64)	36,928
conv2d_3 (Conv2D)	(None, 69, 69, 64)	36,928
dropout_1 (Dropout)	(None, 69, 69, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 34, 34, 64)	0
dropout_2 (Dropout)	(None, 34, 34, 64)	0
conv2d_4 (Conv2D)	(None, 32, 32, 128)	73,856
conv2d_5 (Conv2D)	(None, 30, 30, 128)	147,584
conv2d_6 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_3 (Dropout)	(None, 14, 14, 128)	0
conv2d_7 (Conv2D)	(None, 12, 12, 128)	147,584
conv2d_8 (Conv2D)	(None, 10, 10, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 256)	0
dropout_4 (Dropout)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 512)	3,277,312
dense_1 (Dense)	(None, 512)	262,656
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 4)	2,052

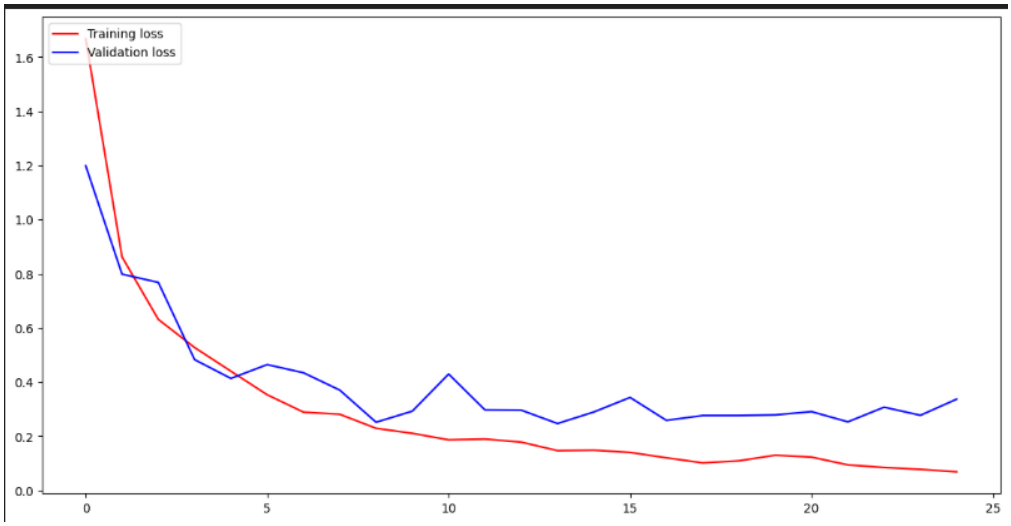
```
Total params: 4,447,044 (16.96 MB)

Trainable params: 4,447,044 (16.96 MB)

Non-trainable params: 0 (0.00 B)
```

The Metrics Such **as Training/Validation Accuracy** as Ill **as Training/Validation Loss** are plotted as below:

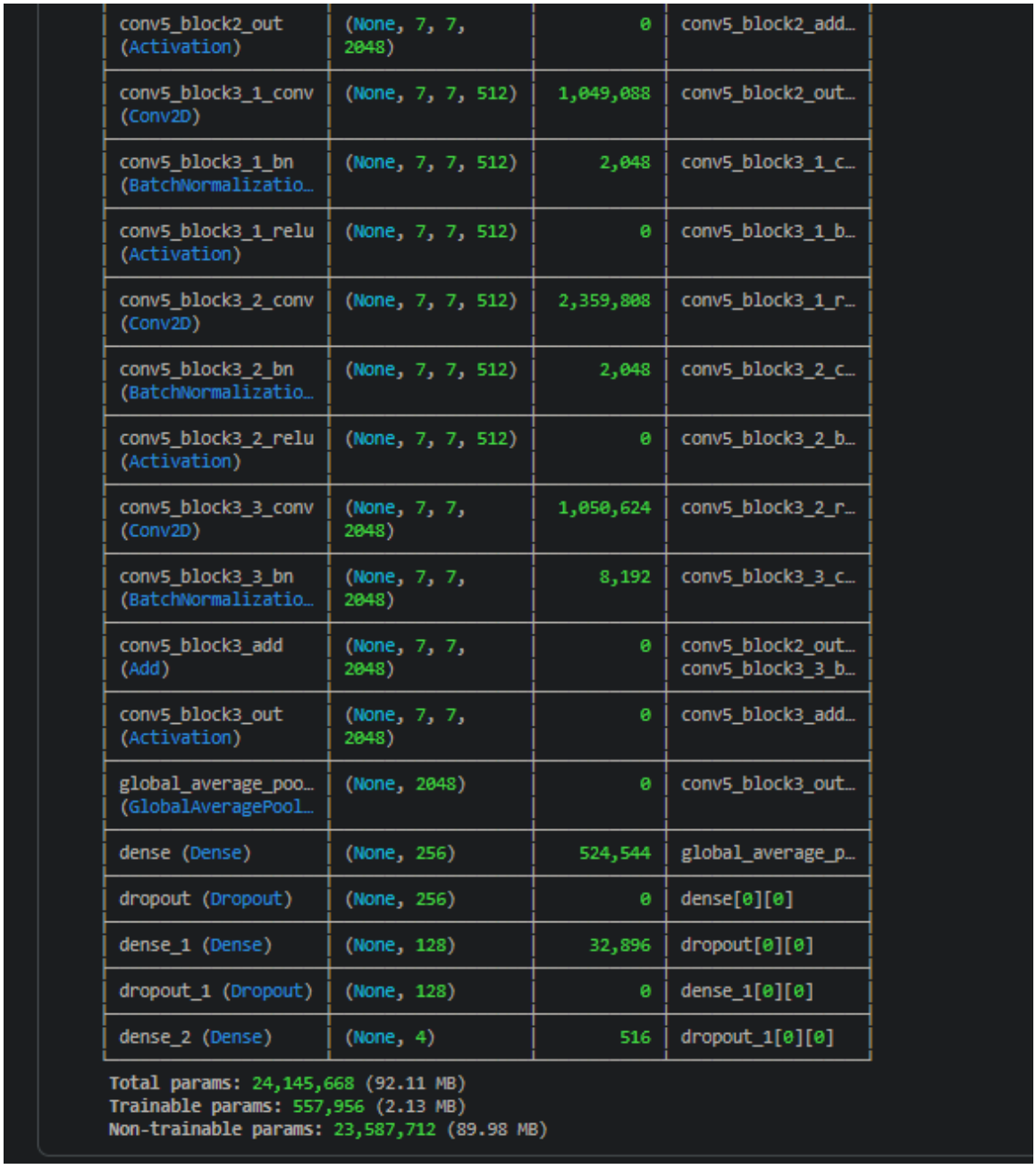
- **Training/Validation Accuracy of Custom CNN Model**



- **Training/Validation Loss of Custom CNN Model**



[B] <u>Pretrained ResNet50 Model Architecture</u>:



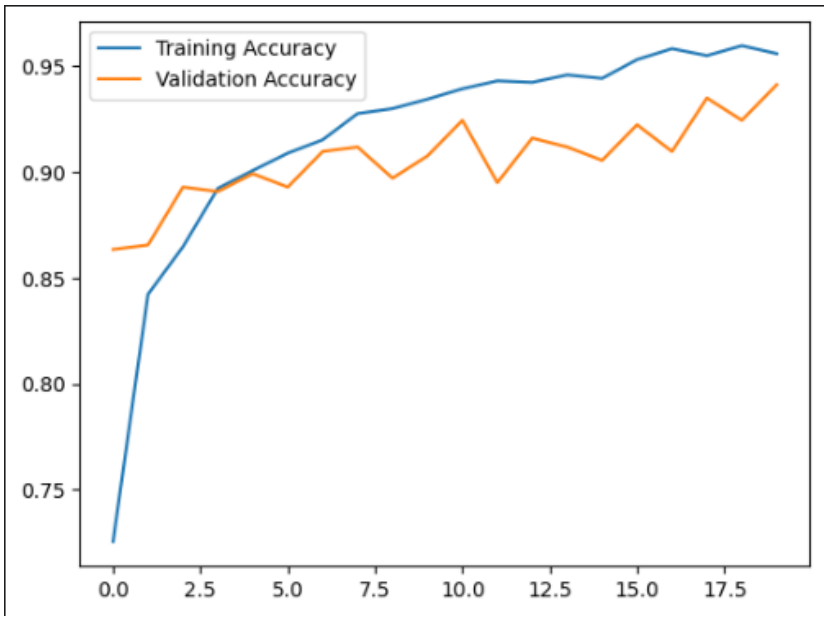
conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_block2_add…
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1,049,088	conv5_block2_out…
conv5_block3_1_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block3_1_c…
conv5_block3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_1_b…
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block3_1_r…
conv5_block3_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block3_2_c…
conv5_block3_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_2_b…
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block3_2_r…
conv5_block3_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block3_3_c…
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out… conv5_block3_3_b…
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add…
global_average_poo… (GlobalAveragePool…	(None, 2048)	0	conv5_block3_out…
dense (Dense)	(None, 256)	524,544	global_average_p…
dropout (Dropout)	(None, 256)	0	dense[0][0]
dense_1 (Dense)	(None, 128)	32,896	dropout[0][0]
dropout_1 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 4)	516	dropout_1[0][0]

Total params: 24,145,668 (92.11 MB)
Trainable params: 557,956 (2.13 MB)
Non-trainable params: 23,587,712 (89.98 MB)

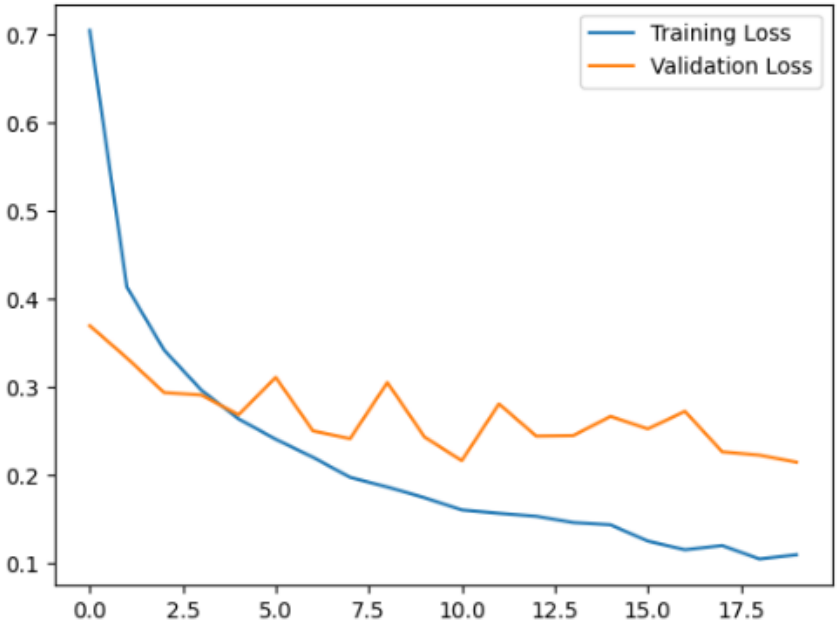
Since the model is huge, only a part of it has been added to the document file. The Entire code is available in Model.ipynb code file.

The Metrics Such **as Training/Validation Accuracy** as well as **Training/Validation Loss** for **ResNet50 Model** are plotted as below:

- **<u>Training/Validation Accuracy for ResNet50 Model</u>:**



- **<u>Training/Validation Loss for ResNet50 Model</u>:**



5. Experiments

Hyperparameters include **batch size 32**, **learning rates 0.1** for CNN model and **0.0005** for ResNet50 Model, and 20–30 epochs.
All experiments are run on GPU. Model summaries are included in tables below.

[**A**] **Custom CNN** Model params:

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

Fitting the Model with Data

```
history = model.fit(X_train,y_train,epochs=25,validation_split=0.1)
```

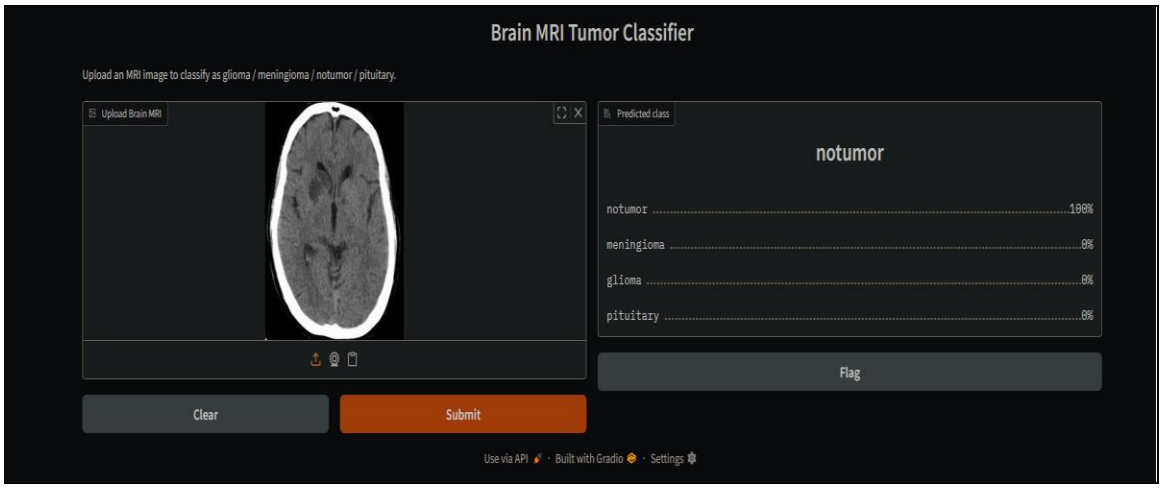
[**B**] **ResNet50** Model Params:

```
model.compile(
    loss="categorical_crossentropy",
    optimizer=keras.optimizers.Adam(learning_rate=0.0005),
    metrics=["accuracy"]
)

history = model.fit(
    X_train, y_train,
    epochs=20,
    validation_split=0.1,
    batch_size=32
)
```

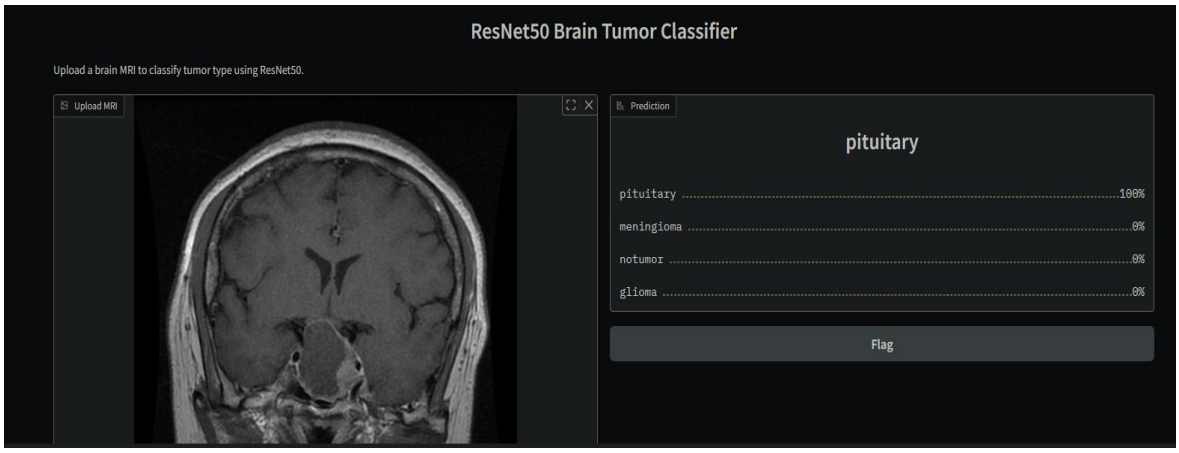
PREDICTION DEMO

The Prediction Demo was carried out on **Gradio** Framework; some of the snaps are as follows, the input is an Image after submitting the image I get the detected class with other classes probabilities

[A] <u>Custom CNN Model</u>:



[B] <u>ResNet50 Model</u>

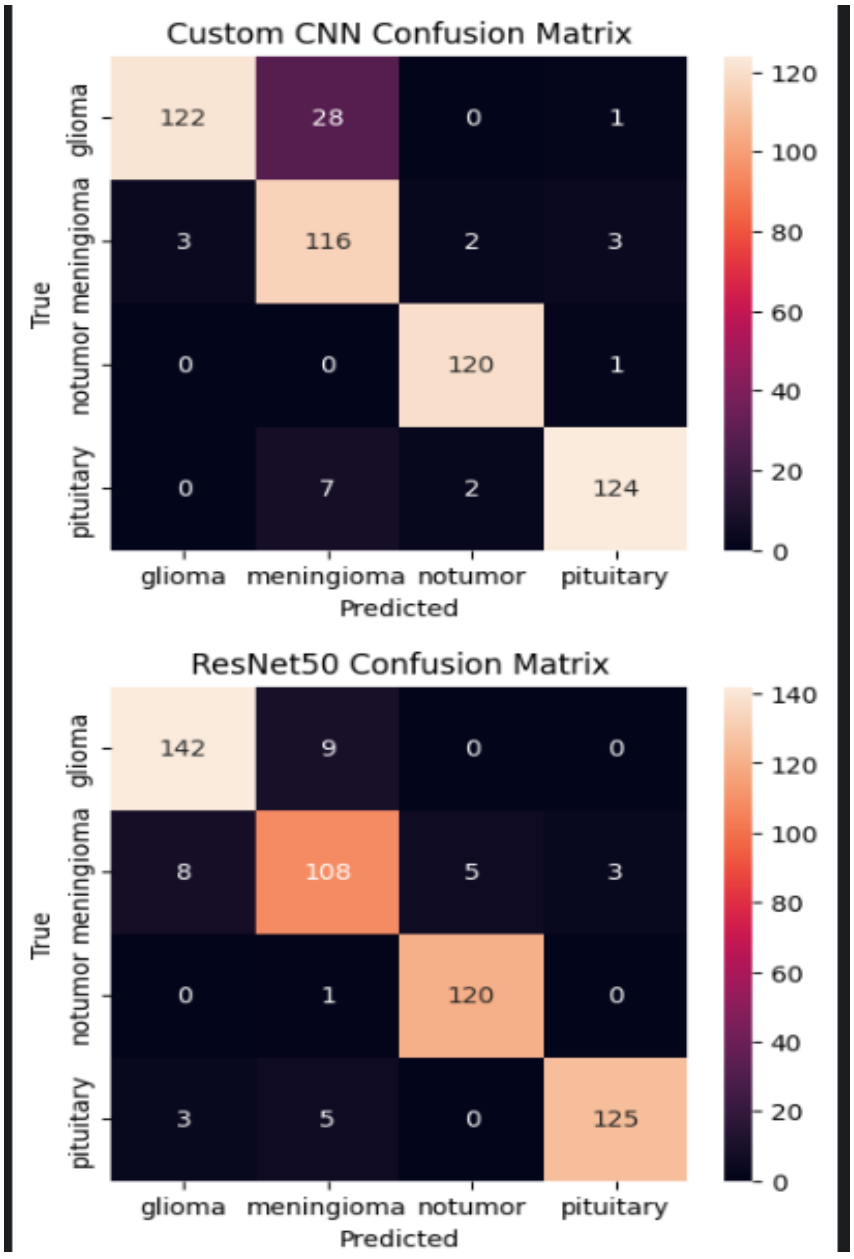


6. Results

Performance metrics for Custom CNN and ResNet50 models include accuracy, precision, recall, F1-score.
ResNet50 shows highest accuracy. Confusion matrices and training curves illustrate improvements.

```
=== Custom CNN ===
Accuracy: 0.9111531190926276

Classification Report:

              precision    recall  f1-score   support

      glioma       0.98      0.81      0.88       151
  meningioma       0.77      0.94      0.84       124
     notumor       0.97      0.99      0.98       121
   pituitary       0.96      0.93      0.95       133

    accuracy                           0.91       529
   macro avg       0.92      0.92      0.91       529
weighted avg       0.92      0.91      0.91       529



=== ResNet50 ===
Accuracy: 0.9357277882797732

Classification Report:

              precision    recall  f1-score   support

      glioma       0.93      0.94      0.93       151
  meningioma       0.88      0.87      0.87       124
     notumor       0.96      0.99      0.98       121
   pituitary       0.98      0.94      0.96       133

    accuracy                           0.94       529
   macro avg       0.94      0.94      0.94       529
weighted avg       0.94      0.94      0.94       529
```

Confusion Matrix and Model Accuracy:



	Model	Accuracy
0	Custom CNN	0.911153
1	ResNet50	0.935728

7. Conclusion

An efficient brain tumor diagnosis system is necessary for the early treatment of the patient. To help this problem, we designed 3 models of which -

Resnet50 generalizes best, **Custom CNN** created from scratch performs adequately.

The Following Final Comparison between all 3 Models is given Below:

Table: Model Comparison

Model	Accuracy	Precision	Recall
Custom CNN	0.91	0.92	0.91
ResNet50	0.93	0.94	0.94