VIT EXTERNSHIP DELIVERABLE (PROJECT) REPORT

Programme / Course : Artificial Intelligence

Project Title : Brain Tumor Classification using IBM Watson

Mentor : Mr. Prince

Batch : 2023

Title: Brain Tumor Classification using IBM Watson

Team Members:

VINAY VITTAL MOOLYA	20BCE1318
ARYAN PUROHIT	20BCE1333
JAGRIT TANEJA	20BRS1002
JAGDISH SINDHI	20BCE1611

Date Of Submission: 1/07/2023

1. INTRODUCTION

1.1. OVERVIEW

Brain tumors are one of the most often diagnosed malignant tumors in persons of all ages. Recognizing its grade is challenging for radiologists in health monitoring and automated determination; however, IoT can help. It is critical to detect and classify contaminated tumor locations using Magnetic Resonance Imaging (MRI) images. Numerous tumors exist, including glioma tumor, meningioma tumor, pituitary tumor, and no tumor (benign). Detecting the type of tumor and preventing it is one of the most challenging aspects of brain tumor categorization. With the rapid development of deep learning technology, automatic classification of brain tumors by magnetic resonance imaging (MRI) has become a promising research area.

Numerous deep learning-based approaches for categorizing brain tumors have been published in the literature. In this study, we present a deep learning-based approach for brain tumor classification by MRI. The proposed method is based on convolutional neural networks (CNN). It is a type of deep learning algorithm that has achieved remarkable results in various image classification tasks. Our approach uses a pre-trained CNN model and refines it using a large annotated brain MRI dataset to extract relevant features from scans. These features are populated into fully connected classes for classification into different tumor types.

In this project, multiclass brain tumor classification will be performed by using a diversified dataset of around 3264 MR images of different types of brain tumors (2870 images of training sets and 394 images of testing sets), which are classified into glioma, meningioma, pituitary and benign tumor. At the end of this project, our project will be able to classify these different types of tumors with satisfiable accuracy given a MRI image set.

1.2. PURPOSE

Brain tumors are one of the most often diagnosed malignant tumors in persons of all ages. Recognizing its grade is challenging for radiologists in health monitoring and automated determination; however, IoT can help. It is critical to detect and classify contaminated tumor locations using Magnetic Resonance Imaging (MRI) images. Numerous tumors exist, including glioma tumor, meningioma tumor, pituitary tumor, and no tumor (benign). Detecting the type of tumor and preventing it is one of the most challenging aspects of brain tumor categorization. With the rapid development of deep learning technology, automatic classification of brain tumors by magnetic resonance imaging (MRI) has become a promising research area.

Brain tumors are abnormal growths of cells that can develop in different areas of the brain. Tumor classification is an essential task in the diagnosis and treatment of brain tumors. Accurately identifying the type of tumor is crucial for determining the appropriate treatment plan and predicting the patient's prognosis. However, manual classification of brain tumors based on imaging studies, such as Magnetic Resonance Imaging (MRI), can be challenging and time-consuming for clinicians. Therefore, computer-aided diagnosis systems using machine learning algorithms have been developed to assist clinicians in classifying brain tumors accurately and efficiently. In this project, we aim to build a brain tumor classification model using deep learning techniques that can classify brain tumors into different types based on MRI images. Multiclass brain tumor classification will be performed by using a diversified dataset of around 3264 MR images of different types of brain tumors (2870 images of training sets and 394 images of testing sets), which are classified into glioma, meningioma, pituitary and benign tumor. At the end of this project, our project will be able to classify these different types of tumors with satisfiable accuracy given a MRI image set.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

Multimodal Brain Tumor Classification Using Deep Learning and Robust Feature Selection: A Machine Learning Application for Radiologists

Manual identification of brain tumors is an error-prone and tedious process for radiologists; therefore, it is crucial to adopt an automated system. The multimodal brain tumor classification (T1, T2, T1CE, and Flair) is a challenging task for radiologists. Here, the authors present an automated multimodal classification method using deep learning for brain tumor type classification. The proposed method consists of five core steps. In the first step, linear contrast stretching is employed. In the second step, deep learning feature extraction is performed. In the third step, a correntropy-based joint learning approach is used for the selection of best features. In the fourth step, the partial least square (PLS)-based robust covariant features were fused in one matrix. The combined matrix was fed to an Extreme Learning Machine for final classification. The proposed method was validated on the BraTS datasets and an accuracy of 97.8%, 96.9%, 92.5% for BraTs2015, BraTs2017, and BraTs2018, respectively, was achieved.

Brain tumor detection and multi‑classification using advanced deep learning techniques

This article presents segmentation through Unet architecture with ResNet50 as a backbone on the Figshare data set and achieved a level of 0.9504 of the intersection over union (IoU). The preprocessing and data augmentation concept were introduced to enhance the classification rate. The multi-classification of brain tumors is performed using evolutionary algorithms and reinforcement learning through transfer learning. Other deep learning methods such as ResNet50, DenseNet201, MobileNet V2, and InceptionV3 are also applied. Results thus obtained exhibited that the proposed research framework performed better than reported in state of the art. Different CNN models applied for tumor classification such as MobileNet V2, Inception V3, ResNet50, DenseNet201, NASNet and attained accuracy 91.8, 92.8, 92.9, 93.1, 99.6%, respectively. NASNet exhibited the highest accuracy.

A Transfer Learning approach for AI-based classification of brain tumors

Magnetic Resonance Imaging (MRI) is generally utilized for such a task because of its unrivaled quality of the image and the reality that it does not depend on ionizing radiation. In this work, the AI-based classification of BT using Deep Learning Algorithms are proposed for the classifying types of brain tumors utilizing openly accessible datasets. These datasets classify BTs into (malignant and benign). The datasets comprise 696 images on T1-weighted images for testing purposes. The projected arrangement accomplishes a noteworthy performance with the finest accuracy of 99.04%. The achieved outcome signifies the capacity of the proposed algorithm for the classification of brain tumors.

Segmentation, Feature Extraction, and Multiclass Brain Tumor Classification

In this paper, multiclass brain tumor classification was performed by using a diversified dataset of 428 post-contrast T1-weighted MR images from 55 patients. In this study, principal component analysis (PCA) is used for reduction of dimensionality of the feature space. The six classes obtained are then classified by an artificial neural network (ANN). Three sets of experiments have been performed. In the first experiment, classification accuracy by ANN approach is performed. In the second experiment, PCA-ANN approach with random sub-sampling has been used in which the SROIs from the same patient may get repeated during testing. It is observed that the classification accuracy has increased from 77 to 91 %. PCA-ANN has delivered high accuracy for each class: AS—90.74 %, MET—96.67 %, etc. The third experiment was to remove bias and to test the robustness of the proposed system. In this case also, the proposed system has performed well by delivering an overall accuracy of 85.23 %.

A decision support system for multimodal brain tumor classification using deep learning

In this article, a new automated deep learning method is proposed for the classification of multiclass brain tumors. To realize the proposed method, the Densenet201 Pre-Trained Deep Learning Model is fine-tuned and later trained using a deep transfer of imbalanced data learning. The features of the trained model are extracted from the average pool layer, which represents the very deep information of each type of tumor. Two techniques for the selection of features are proposed. The first technique is Entropy–Kurtosis-based High Feature Values (EKbHFV) and the second technique is a modified genetic algorithm (MGA) based on metaheuristics. Finally, both EKbHFV and MGA-based features are fused using a non-redundant serial-based approach and classified using a multiclass SVM cubic classifier. For the experimental process, two datasets, including BRATS2018 and BRATS2019, are used without increase and have achieved an accuracy of more than 95%. The precise comparison of the proposed method with other neural nets shows the significance of this work.

A Brain Tumor Identification and Classification Using Deep Learning based on CNN-LSTM Method

A CNN (Convolutional Neural Network), one of the most advanced methods in deep learning, was used to detect a tumor using brain MRI images in this paper. However, there were many issues with the training procedure detected. They developed an IoT computational system based on deep learning for detecting brain tumors in MRI images. The paper suggests combining A CNN(Convolutional Neural Network) with an STM(Long Short Term Memory), LSTMs can supplement the ability of CNN to extract features. When used for image classification, the layered LSTM-CNN design outperforms standard CNN classification. Experiments were undertaken to forecast the proposed model's performance using the Kaggle data set, which contains 3264 MRI scans. The dataset was separated into two sections: 2870 photos of training sets and 394 images of testing sets. The experimental findings demonstrate that the proposed model outperforms earlier CNN and RNN models in terms of accuracy.

2.2. PROPOSED SOLUTION

2.2.1. DATA SOURCE

The dataset that we will be using for this project is publicly available on kaggle. More details about the Dataset are as follows.

Brain tumors are one of the most feared tumors ,and are classified as Benign , Malignant , glioma , meningioma etc. The best technique to detect brain tumors is Magnetic Resonance Imaging (MRI). Hence in this particular dataset we have 2 directories Training and Testing already split in the dataset to make work easier. In the training directory the MRI images of the brain are already organized into various classes which include glioma(826) , meningioma(822) , pituitary(827) and no tumor(395). Also in the testing directory the images are classified into different directories based on the classes which are Glioma(100) , Meningioma(115) , Pituitary(74) , no tumor(105).

In a nutshell our model will be working on 2870 training images and 394 images for testing.

Dataset: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

2.2.2. DATA PREPROCESSING

As we have a large amount of data in the form of images that are of various dimensions and categories, there was a need to preprocess our data so as to have a uniformity in the data when we pass it as input to our various models.

The various preprocessing techniques used are given in the below code:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_gen = ImageDataGenerator(rescale=(1./255),horizontal_flip=True,shear_range=0.2)
test_gen = ImageDataGenerator(rescale=(1./255))
TL_train = train_gen.flow_from_directory("/kaggle/input/brain-tumor-classification-mri/Training",
                                target_size=(224,224),
                                batch_size=15,
                                class_mode='categorical')
TL_test = test_gen.flow_from_directory('/kaggle/input/brain-tumor-classification-mri/Testing',
                                target_size=(224,224),
                                batch_size=15,
                                class_mode='categorical')
```

The above code snippet is used to prepare image data for a deep learning model by using the Keras ImageDataGenerator and flow_from_directory functions. The following is what each parameter does:

1. **rescale:** This parameter scales the pixel values of the images by a factor of 1/255, which normalizes the pixel values to be in the range [0,1].
2. **zoom_range, height_shift_range, width_shift_range:** These parameters are used for data augmentation. They randomly zoom and shift the images horizontally and vertically to create new training examples.
3. **fill_mode:** This parameter determines how the pixels that are shifted outside the image boundaries are filled. In this case, the 'nearest' mode is used, which fills the shifted pixels with the nearest pixel value.
4. **directory:** This parameter specifies the path to the directory containing the training or test images. Here, we have made a separate directory to store data in an orderly fashion, with train and test data separate, with certain number of images in each subdirectory, symbolizing different types of cancers, so that we can achieve a certain accuracy while also keeping in mind our resources limits

5. **generator:** This parameter is set to either train_datagen or test_datagen, which are instances of the ImageDataGenerator class with specific parameters defined.
6. **class_mode:** This parameter determines the type of labels to be generated by the generator. In this case, it is set to 'categorical', which means the labels will be one-hot encoded vectors.
7. **target_size:** This parameter specifies the size of the input images that will be fed into the model. The images will be resized to this size.
8. **batch_size:** This parameter specifies the number of images that will be fed into the model at once during training or testing.
9. **shuffle:** This parameter determines whether to shuffle the images in the test directory before generating the labels. In this case, it is set to FALSE, which means the images will not be shuffled.

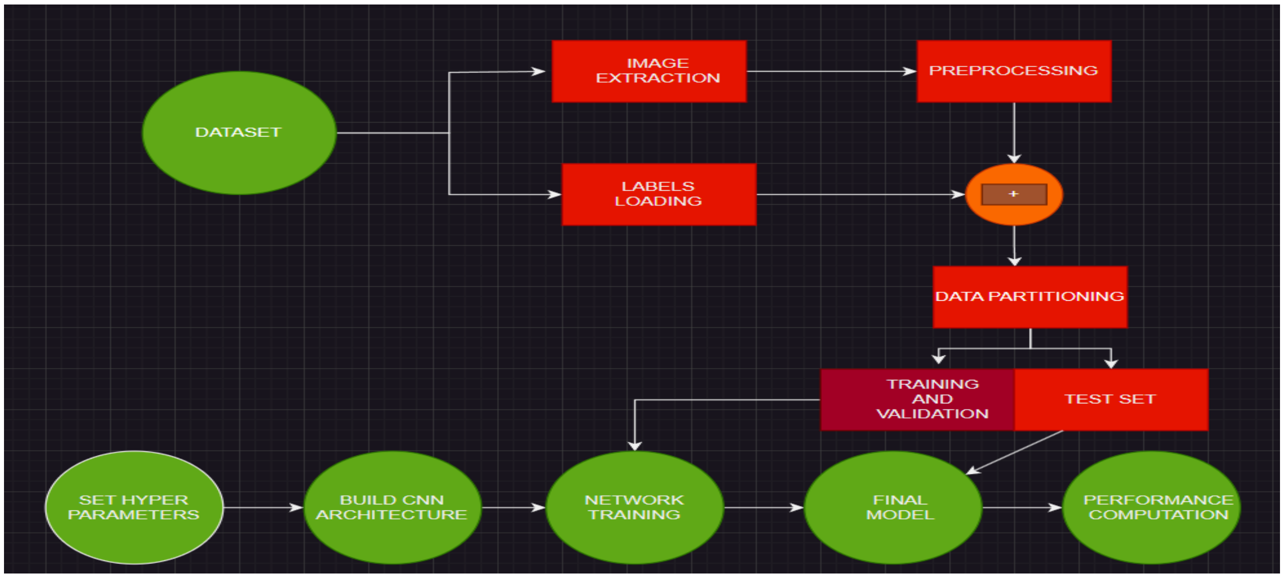
2.2.3. DATA ANALYTICS MODEL

The problem statement at hand is that of Brain Tumor Classification using Deep Learning Techniques such as Convolutional Neural Networks or CNNs. In such a setup, we are required to collect images from the data source, feed it into the model to train it, improve the feasibility of the model by attaching to it a feedback cycle based on classification accuracy and finally have it work on new data.

For the CNN model, we will be applying the 2 variants of the VGGNET architecture - VGG16 and VGG19 and our own custom Brute Model too.. These CNN models have been found to work well for the task of image feature extraction and classification. The images are thus preprocessed and fed to the model for the training process. Then hyperparameter tuning and layer details are tweaked as per the feedback cycle setup based on classification accuracy. After achieving feasible results, the model can be exported and applied to real world applications such as in medical scans and early detection of brain tumors.

3. THEORETICAL ANALYSIS

3.1. BLOCK DIAGRAM



3.2. HARDWARE/SOFTWARE DESIGNING

3.2.1. VGG-Net Architecture

VGGNet, or the Very Deep Convolutional Networks, is a deep neural network architecture that was introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford in 2014. It is a groundbreaking architecture that achieved state-of-the-art results in various computer vision tasks, including image classification, object detection, and semantic segmentation.

VGGNet has been trained on ImageNet, which is a large database of quality controlled, human-annotated images that help test algorithms that are built to store, retrieve, or annotate multimedia data. ImageNet has approximately 1.2 million images to train the model on with around 1000 classes of images. VGGNet based models have been able to achieve an accuracy of 92.7% on ImageNet images classification.

The key characteristic of the VGGNet architecture is its depth. It consists of 16 or 19 layers of convolutional neural network (CNN) that are stacked on top of each other, making it one of the deepest CNN architectures at the time of its release. The network architecture is simple and consists of only 3x3 convolutional filters and 2x2 max-pooling layers. This simple architecture was designed to achieve better accuracy than previous models while reducing the number of trainable parameters.

The VGGNet architecture is divided into two parts: the convolutional layers and the fully connected layers. The convolutional layers perform feature extraction, while the fully connected layers perform classification based on the features extracted from the convolutional layers.

In summary, the VGGNet architecture is a deep neural network architecture that achieved state-of-the-art results in various computer vision tasks, including image classification, object detection, and semantic segmentation. Its simple architecture with 3x3 convolutional filters and 2x2 max-pooling layers, and its depth of 16 or 19 layers make it one of the most popular and widely used CNN architectures in computer vision.

Here is a quick outline of important terminologies of VGG architecture:

Input—VGGNet receives a 224×224 image input. In the ImageNet competition, the model's creators kept the image input size constant by cropping a 224×224 section from the center of each image.

Convolutional layers—the convolutional filters of VGG use the smallest possible receptive field of 3×3. VGG also uses a 1×1 convolution filter as the input's linear transformation.

ReLu activation—next is the Rectified Linear Unit Activation Function (ReLU) component, AlexNet's major innovation for reducing training time. ReLU is a linear function that provides a matching output for positive inputs and outputs zero for negative inputs. VGG has a set convolution stride of 1 pixel to preserve the spatial resolution after convolution (the stride value reflects how many pixels the filter "moves" to cover the entire space of the image).

Hidden layers—all the VGG network's hidden layers use ReLU instead of Local Response Normalization like AlexNet. The latter increases training time and memory consumption with little improvement to overall accuracy.

Pooling layers–A pooling layer follows several convolutional layers—this helps reduce the dimensionality and the number of parameters of the feature maps created by each convolution step. Pooling is crucial given the rapid growth of the number of available filters from 64 to 128, 256, and eventually 512 in the final layers.

Fully connected layers—VGGNet includes three fully connected layers. The first two layers each have 4096 channels, and the third layer has 1000 channels, one for every class.

3.2.1.1. VGG16 Model:

VGG16 refers to the VGG model, also called VGGNet. It is a convolution neural network (CNN) model supporting 16 layers. The VGG16 model can achieve a test accuracy of 92.7% in ImageNet, a dataset containing more than 14 million training images across 1000 object classes. It is one of the top models from the ILSVRC-2014 competition. This model differs from previously high-performing models in several ways. Firstly, it used a tiny 3×3 receptive field with a 1-pixel stride for comparison whereas AlexNet used an 11×11 receptive field with a 4-pixel stride. The 3×3 filters combine to provide the function of a larger receptive field. The benefit of using multiple smaller layers rather than a single large layer is that more non-linear activation layers accompany the convolution layers, improving the decision functions and allowing the network to converge quickly.

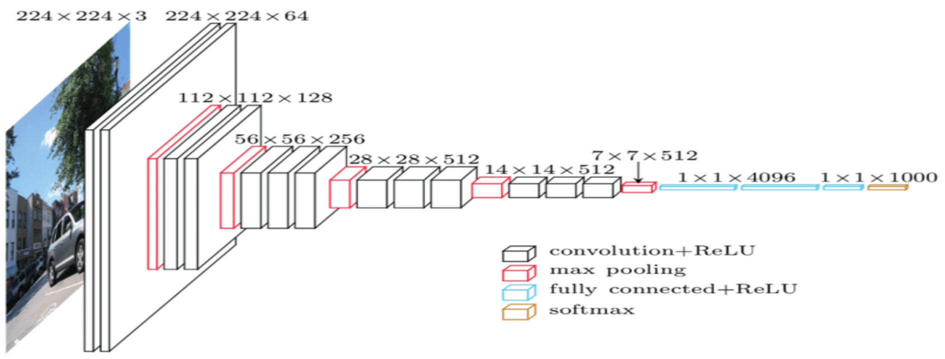
Secondly, VGG uses a smaller convolutional filter, which reduces the network's tendency to over-fit during training exercises. A 3×3 filter is the optimal size because a smaller size cannot capture left-right and up-down information. Thus, VGG is the smallest possible model to understand an image's spatial features. Consistent 3×3 convolutions make the network easier to manage.

VGG16 Architecture:

VGG16, as its name suggests, is a 16-layer deep neural network. VGG16 is thus a relatively extensive network with a total of 138 million parameters—it's huge even by today's standards. However, the simplicity of the VGGNet16 architecture is its main attraction. The VGGNet architecture incorporates the most important convolutional neural network features.

VGG16 Model Code:

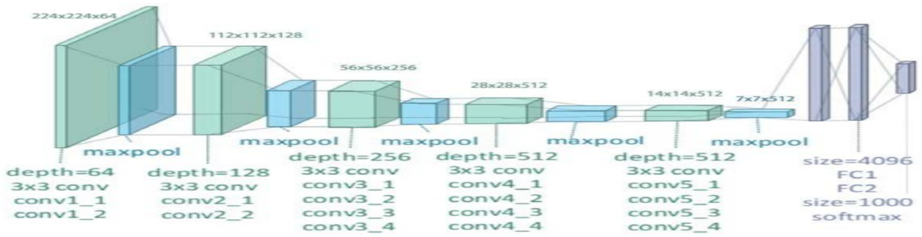
```
vgg = VGG16(include_top=False,weights='imagenet',input_shape=(224,224,3))

for layer in vgg.layers:
  layer.trainable=False

x = Flatten()(vgg.output)

prediction = Dense(4,activation='softmax')(x)

vggmodel = Model(inputs=vgg.input,outputs=prediction)

vggmodel.summary()

vggmodel.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

vggmodel.fit_generator(TL_train,validation_data=TL_test,epochs=15,steps_per_epoch=len(TL_train),
              validation_steps=len(TL_test))
```



Source: ResearchGate

3.2.1.2. VGG19 Model:

VGG19 is a variant of the VGG model which in short consists of 19 layers(16 convolutional layers, 3 Fully Connected layers, 5 MaxPool layers and 1 SoftMax layer). VGG19 has 19.6 billion FLOPs. We can understand VGG as a successor of the AlexNet but was created by a different group named as Visual Geometry Group at Oxford's and hence the name VGG. It carries and uses some ideas from its predecessors and improves on them and uses deep Convolutional neural layers to improve accuracy.

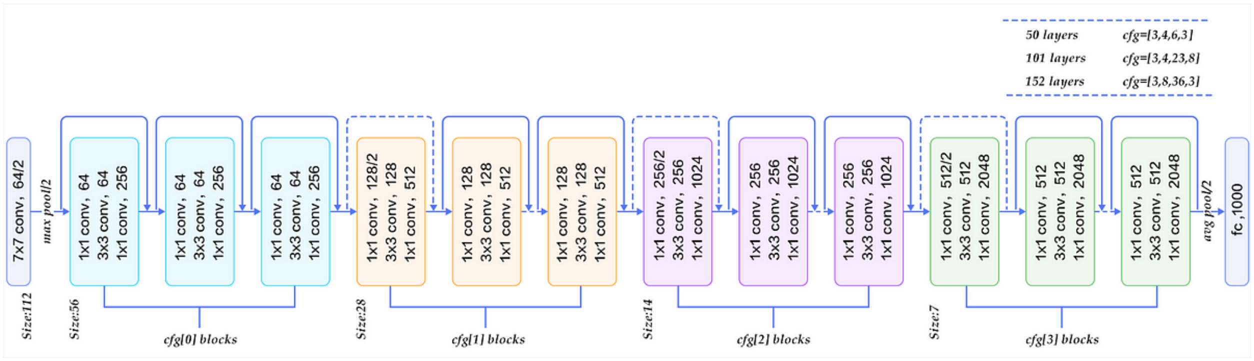


VGG19 Model Code:

```
vgg19 = VGG19(include_top=False,weights='imagenet',input_shape=(224,224,3))

for layer in vgg19.layers:
  layer.trainable=False

x = Flatten()(vgg19.output)

prediction = Dense(4,activation='softmax')(x)

vgg19model = Model(inputs=vgg19.input,outputs=prediction)

vgg19model.summary()

vgg19model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

vgg19model.fit_generator(TL_train,validation_data=TL_test,epochs=15,steps_per_epoch=len(TL_train),
              validation_steps=len(TL_test))
```

3.2.1.3. Resnet50 Model

ResNet-50 is a 50-layer convolutional neural network architecture developed by Microsoft Research in 2015. It is widely used for image recognition tasks such as object detection and classification because it outperforms traditional models. ResNet-50 uses residual blocks to solve the vanishing gradient problem that occurs in deep neural networks. This allows for effective learning from multi-layer images. These residual blocks contain links that allow information to bypass multiple layers, thus allowing the network to learn more efficiently. Models are trained on large datasets such as ImageNet and can be fine-tuned for specific image recognition tasks.


50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 128/2 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 256/2 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 512/2 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
avg pool/2
fc, 1000
Size:112
Size:56
cfg[0] blocks
Size:28
cfg[1] blocks
Size:14
cfg[2] blocks
Size:7
cfg[3] blocks

The ResNet-50 architecture consists of 50 layers, and its main innovation is the use of residual connections.

The residual connection, also known as the hop connection, allows the network to learn the rest of the functions instead of trying to learn the original mapping directly. This alleviates the problem of trailing gradients and allows for much deeper training of neural networks.

In ResNet-50, residual connections are added between each of the two convolutional layers, allowing the network to learn features at multiple levels of abstraction. The network also uses batch normalization after each convolution layer to normalize the input and reduce internal covariate shifts, which improves network convergence.

The detailed architecture of ResNet 50:

- Input: 224x224x3 (RGB image)
- Convolutional layer with 64 filters, kernel size of 7x7, and stride of 2x2, followed by batch normalization and ReLU activation.
- Max pooling layer with kernel size of 3x3 and stride of 2x2.
- Residual blocks:
    - Block 1: 3 convolutional layers with 64 filters, kernel size of 1x1, 3x3, and 1x1 respectively, followed by batch normalization and ReLU activation. The output is added to the input through a residual connection. This block is repeated 3 times.

  - Block 2: 3 convolutional layers with 128 filters, kernel size of 1x1, 3x3, and 1x1 respectively, followed by batch normalization and ReLU activation. The output is added to the input through a residual connection. This block is repeated 4 times.
  - Block 3: 3 convolutional layers with 256 filters, kernel size of 1x1, 3x3, and 1x1 respectively, followed by batch normalization and ReLU activation. The output is added to the input through a residual connection. This block is repeated 6 times.
  - Block 4: 3 convolutional layers with 512 filters, kernel size of 1x1, 3x3, and 1x1 respectively, followed by batch normalization and ReLU activation. The output is added to the input through a residual connection. This block is repeated 3 times.
- Global average pooling layer to compute the average of each feature map.
- Fully connected layer with 1000 nodes (one for each class in the ImageNet dataset).
- Softmax activation function to produce the final output probabilities.

Resnet50 Code:

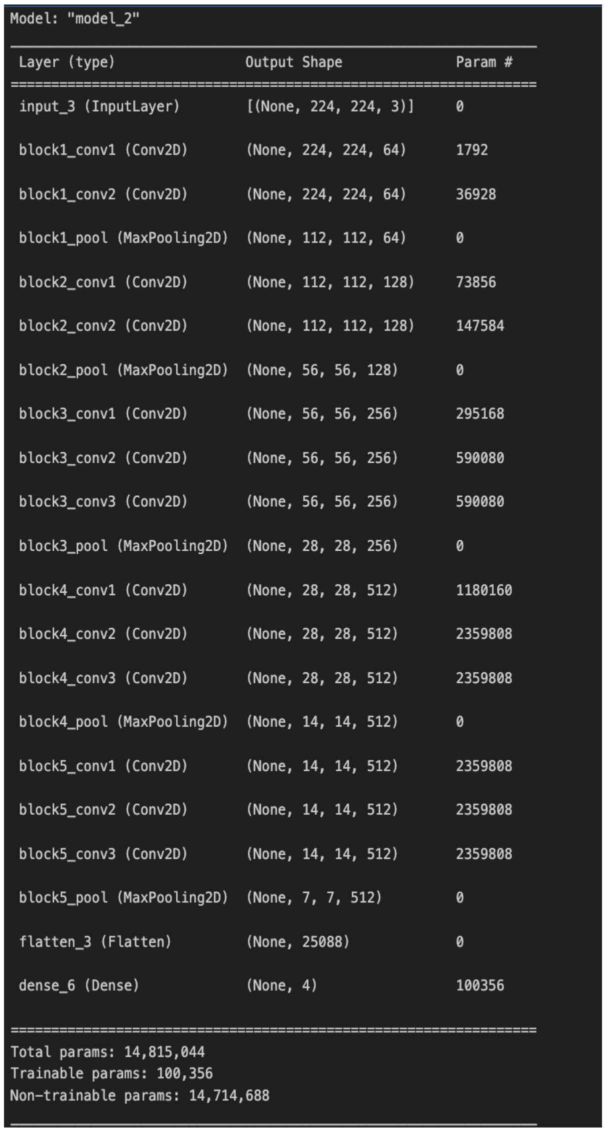
```
resnet = ResNet50(include_top=False,input_shape=(224,224,3))

for layer in resnet.layers:
  layer.trainable=False

x = Flatten()(resnet.output)

out = Dense(4, activation='softmax')(x)

res_model = Model(inputs=resnet.input,outputs=out)

res_model.summary()

res_model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

res_model.fit(TL_train,epochs=10,validation_data=TL_test,steps_per_epoch=len(TL_train),
         validation_steps=len(TL_test))
```

3.2.1.4 Custom-Final-Model Code:

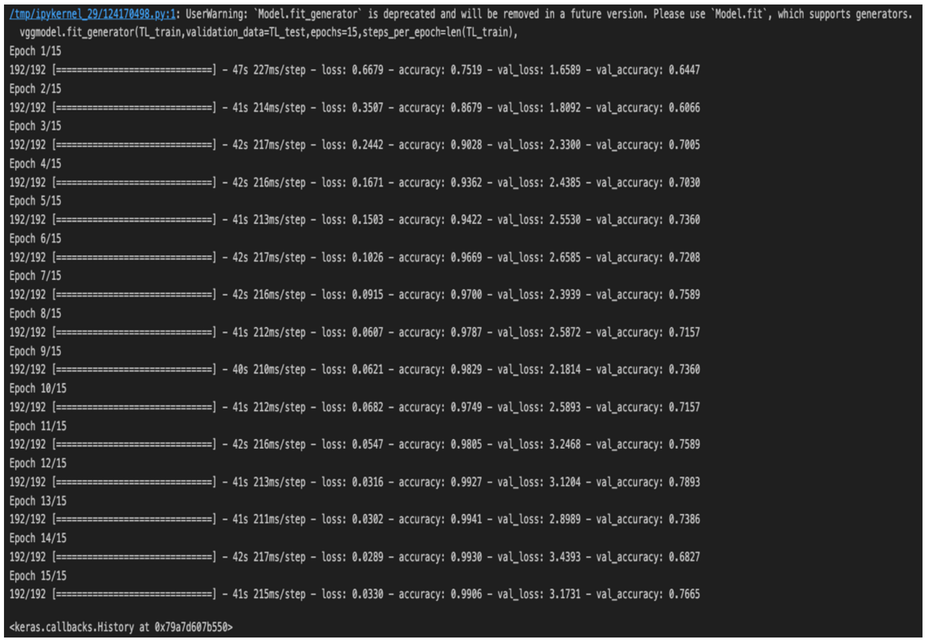
```
model = Sequential()
model.add(Conv2D(32,(3,3),activation = 'relu',input_shape=(150,150,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Dropout(0.3))
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'))
```

4. EXPERIMENTAL INVESTIGATIONS

4.1. VGG16 Model Architecture Summary



```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_3 (Flatten)         (None, 25088)             0

 dense_6 (Dense)             (None, 4)                 100356

=================================================================
Total params: 14,815,044
Trainable params: 100,356
Non-trainable params: 14,714,688
_________________________________________________________________
```

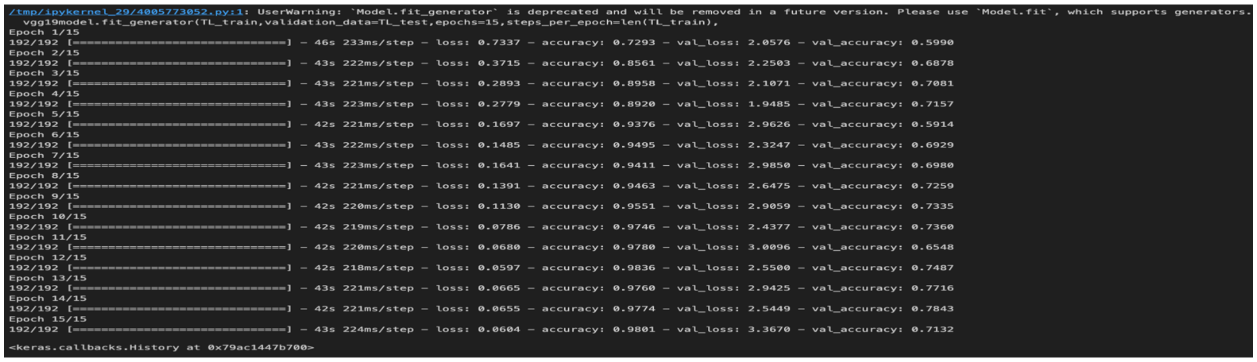


```
/tmp/ipykernel_29/124170498.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  vggmodel.fit_generator(TL_train,validation_data=TL_test,epochs=15,steps_per_epoch=len(TL_train)),
Epoch 1/15
192/192 [==============================] - 47s 227ms/step - loss: 0.6679 - accuracy: 0.7519 - val_loss: 1.6589 - val_accuracy: 0.6447
Epoch 2/15
192/192 [==============================] - 41s 214ms/step - loss: 0.3507 - accuracy: 0.8679 - val_loss: 1.8092 - val_accuracy: 0.6066
Epoch 3/15
192/192 [==============================] - 42s 217ms/step - loss: 0.2442 - accuracy: 0.9028 - val_loss: 2.3300 - val_accuracy: 0.7005
Epoch 4/15
192/192 [==============================] - 42s 216ms/step - loss: 0.1671 - accuracy: 0.9362 - val_loss: 2.4385 - val_accuracy: 0.7030
Epoch 5/15
192/192 [==============================] - 41s 213ms/step - loss: 0.1503 - accuracy: 0.9422 - val_loss: 2.5530 - val_accuracy: 0.7360
Epoch 6/15
192/192 [==============================] - 42s 217ms/step - loss: 0.1026 - accuracy: 0.9669 - val_loss: 2.6585 - val_accuracy: 0.7208
Epoch 7/15
192/192 [==============================] - 42s 216ms/step - loss: 0.0915 - accuracy: 0.9700 - val_loss: 2.3939 - val_accuracy: 0.7589
Epoch 8/15
192/192 [==============================] - 41s 212ms/step - loss: 0.0607 - accuracy: 0.9787 - val_loss: 2.5872 - val_accuracy: 0.7157
Epoch 9/15
192/192 [==============================] - 40s 210ms/step - loss: 0.0621 - accuracy: 0.9829 - val_loss: 2.1814 - val_accuracy: 0.7360
Epoch 10/15
192/192 [==============================] - 41s 212ms/step - loss: 0.0682 - accuracy: 0.9749 - val_loss: 2.5893 - val_accuracy: 0.7157
Epoch 11/15
192/192 [==============================] - 42s 216ms/step - loss: 0.0547 - accuracy: 0.9805 - val_loss: 3.2468 - val_accuracy: 0.7589
Epoch 12/15
192/192 [==============================] - 41s 213ms/step - loss: 0.0316 - accuracy: 0.9927 - val_loss: 3.1204 - val_accuracy: 0.7893
Epoch 13/15
192/192 [==============================] - 41s 211ms/step - loss: 0.0302 - accuracy: 0.9941 - val_loss: 2.8989 - val_accuracy: 0.7386
Epoch 14/15
192/192 [==============================] - 42s 217ms/step - loss: 0.0289 - accuracy: 0.9930 - val_loss: 3.4393 - val_accuracy: 0.6827
Epoch 15/15
192/192 [==============================] - 41s 215ms/step - loss: 0.0330 - accuracy: 0.9906 - val_loss: 3.1731 - val_accuracy: 0.7665

<keras.callbacks.History at 0x79a7d607b550>
```

Accuracy Based on the Test Data

This model gave us an accuracy of 76% on the validation dataset so in order to achieve better accuracy we tried other Transfer Learning models such as VGG19 , Resnet and also we tried writing a model on our own whose performance will be discussed later.

4.2. VGG19 Model Summary (Performance)

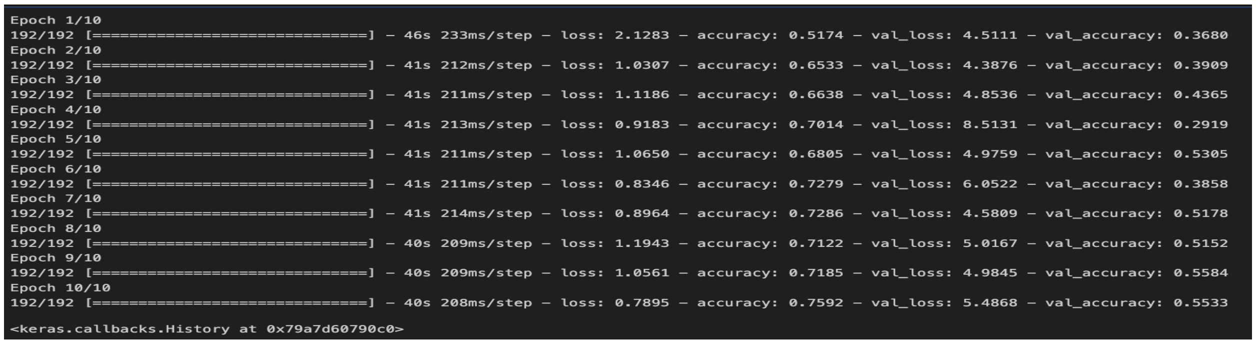


```
/tmp/ipykernel_29/4005773052.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  vgg19model.fit_generator(TL_train,validation_data=TL_test,epochs=15,steps_per_epoch=len(TL_train),
Epoch 1/15
192/192 [==============================] - 46s 233ms/step - loss: 0.7337 - accuracy: 0.7293 - val_loss: 2.0576 - val_accuracy: 0.5990
Epoch 2/15
192/192 [==============================] - 43s 222ms/step - loss: 0.3715 - accuracy: 0.8561 - val_loss: 2.2503 - val_accuracy: 0.6878
Epoch 3/15
192/192 [==============================] - 43s 221ms/step - loss: 0.2893 - accuracy: 0.8958 - val_loss: 2.1071 - val_accuracy: 0.7081
Epoch 4/15
192/192 [==============================] - 43s 223ms/step - loss: 0.2779 - accuracy: 0.8920 - val_loss: 1.9485 - val_accuracy: 0.7157
Epoch 5/15
192/192 [==============================] - 42s 221ms/step - loss: 0.1697 - accuracy: 0.9376 - val_loss: 2.9626 - val_accuracy: 0.5914
Epoch 6/15
192/192 [==============================] - 43s 222ms/step - loss: 0.1485 - accuracy: 0.9495 - val_loss: 2.3247 - val_accuracy: 0.6929
Epoch 7/15
192/192 [==============================] - 43s 223ms/step - loss: 0.1641 - accuracy: 0.9411 - val_loss: 2.9850 - val_accuracy: 0.6980
Epoch 8/15
192/192 [==============================] - 42s 221ms/step - loss: 0.1391 - accuracy: 0.9463 - val_loss: 2.6475 - val_accuracy: 0.7259
Epoch 9/15
192/192 [==============================] - 42s 220ms/step - loss: 0.1130 - accuracy: 0.9551 - val_loss: 2.9059 - val_accuracy: 0.7335
Epoch 10/15
192/192 [==============================] - 42s 219ms/step - loss: 0.0786 - accuracy: 0.9746 - val_loss: 2.4377 - val_accuracy: 0.7360
Epoch 11/15
192/192 [==============================] - 42s 220ms/step - loss: 0.0680 - accuracy: 0.9780 - val_loss: 3.0096 - val_accuracy: 0.6548
Epoch 12/15
192/192 [==============================] - 42s 218ms/step - loss: 0.0597 - accuracy: 0.9836 - val_loss: 2.5500 - val_accuracy: 0.7487
Epoch 13/15
192/192 [==============================] - 43s 221ms/step - loss: 0.0665 - accuracy: 0.9760 - val_loss: 2.9425 - val_accuracy: 0.7716
Epoch 14/15
192/192 [==============================] - 42s 221ms/step - loss: 0.0655 - accuracy: 0.9774 - val_loss: 2.5449 - val_accuracy: 0.7843
Epoch 15/15
192/192 [==============================] - 43s 224ms/step - loss: 0.0604 - accuracy: 0.9801 - val_loss: 3.3670 - val_accuracy: 0.7132

<keras.callbacks.History at 0x79ac1447b700>
```

Accuracy Based On The Test Data

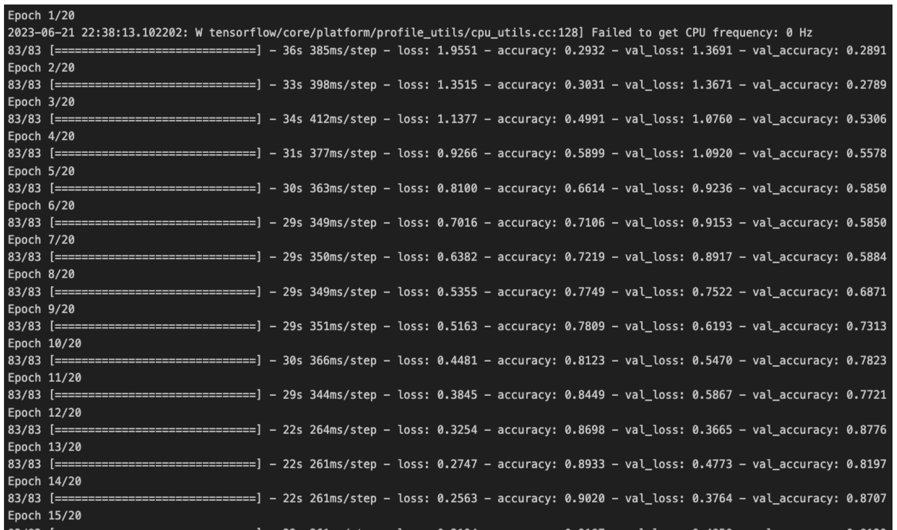
We have achieved an accuracy of 71.32% on the validation data on applying this model for the classification purpose. So we moved ahead with the Resnet Model.

4.3. Restnet50:

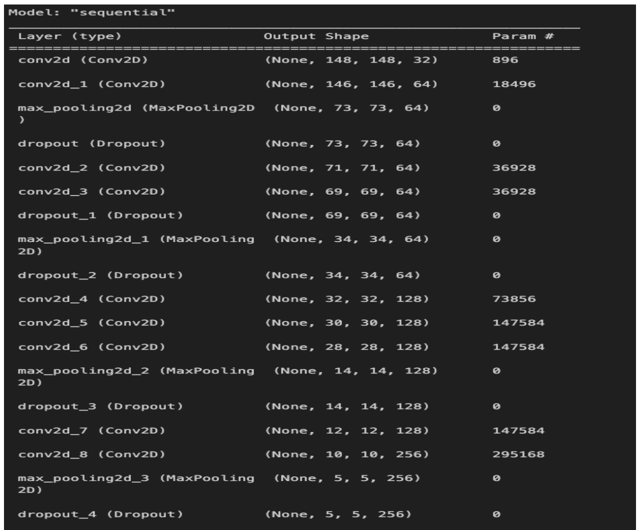


```
Epoch 1/10
192/192 [==============================] - 46s 233ms/step - loss: 2.1283 - accuracy: 0.5174 - val_loss: 4.5111 - val_accuracy: 0.3680
Epoch 2/10
192/192 [==============================] - 41s 212ms/step - loss: 1.0307 - accuracy: 0.6533 - val_loss: 4.3876 - val_accuracy: 0.3909
Epoch 3/10
192/192 [==============================] - 41s 211ms/step - loss: 1.1186 - accuracy: 0.6638 - val_loss: 4.8536 - val_accuracy: 0.4365
Epoch 4/10
192/192 [==============================] - 41s 213ms/step - loss: 0.9183 - accuracy: 0.7014 - val_loss: 8.5131 - val_accuracy: 0.2919
Epoch 5/10
192/192 [==============================] - 41s 211ms/step - loss: 1.0650 - accuracy: 0.6805 - val_loss: 4.9759 - val_accuracy: 0.5305
Epoch 6/10
192/192 [==============================] - 41s 211ms/step - loss: 0.8346 - accuracy: 0.7279 - val_loss: 6.0522 - val_accuracy: 0.3858
Epoch 7/10
192/192 [==============================] - 41s 214ms/step - loss: 0.8964 - accuracy: 0.7286 - val_loss: 4.5809 - val_accuracy: 0.5178
Epoch 8/10
192/192 [==============================] - 40s 209ms/step - loss: 1.1943 - accuracy: 0.7122 - val_loss: 5.0167 - val_accuracy: 0.5152
Epoch 9/10
192/192 [==============================] - 40s 209ms/step - loss: 1.0561 - accuracy: 0.7185 - val_loss: 4.9845 - val_accuracy: 0.5584
Epoch 10/10
192/192 [==============================] - 40s 208ms/step - loss: 0.7895 - accuracy: 0.7592 - val_loss: 5.4868 - val_accuracy: 0.5533

<keras.callbacks.History at 0x79a7d60790c0>
```

4.4 Custom-Final-Model Code



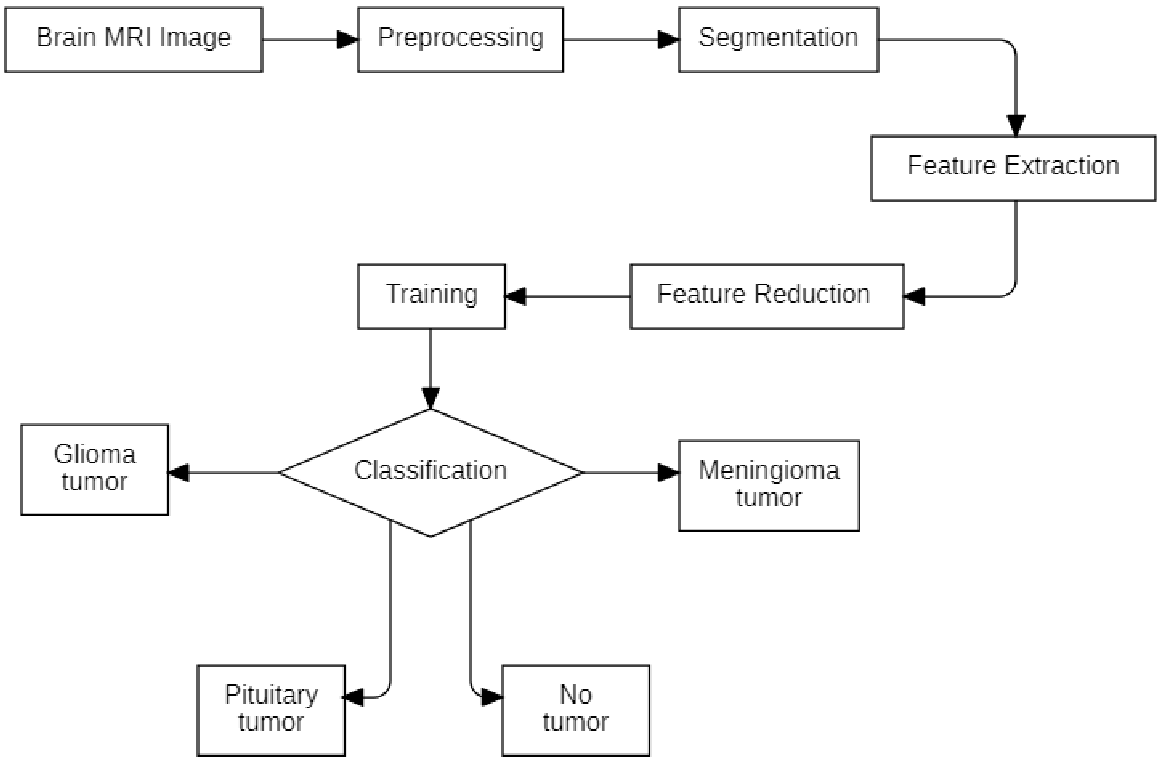
```
Epoch 1/20
2023-06-21 22:38:13.102202: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
83/83 [==============================] - 36s 385ms/step - loss: 1.9551 - accuracy: 0.2932 - val_loss: 1.3691 - val_accuracy: 0.2891
Epoch 2/20
83/83 [==============================] - 33s 398ms/step - loss: 1.3515 - accuracy: 0.3031 - val_loss: 1.3671 - val_accuracy: 0.2789
Epoch 3/20
83/83 [==============================] - 34s 412ms/step - loss: 1.1377 - accuracy: 0.4991 - val_loss: 1.0760 - val_accuracy: 0.5306
Epoch 4/20
83/83 [==============================] - 31s 377ms/step - loss: 0.9266 - accuracy: 0.5899 - val_loss: 1.0920 - val_accuracy: 0.5578
Epoch 5/20
83/83 [==============================] - 30s 363ms/step - loss: 0.8100 - accuracy: 0.6614 - val_loss: 0.9236 - val_accuracy: 0.5850
Epoch 6/20
83/83 [==============================] - 29s 349ms/step - loss: 0.7016 - accuracy: 0.7106 - val_loss: 0.9153 - val_accuracy: 0.5850
Epoch 7/20
83/83 [==============================] - 29s 350ms/step - loss: 0.6382 - accuracy: 0.7219 - val_loss: 0.8917 - val_accuracy: 0.5884
Epoch 8/20
83/83 [==============================] - 29s 349ms/step - loss: 0.5355 - accuracy: 0.7749 - val_loss: 0.7522 - val_accuracy: 0.6871
Epoch 9/20
83/83 [==============================] - 29s 351ms/step - loss: 0.5163 - accuracy: 0.7809 - val_loss: 0.6193 - val_accuracy: 0.7313
Epoch 10/20
83/83 [==============================] - 30s 366ms/step - loss: 0.4481 - accuracy: 0.8123 - val_loss: 0.5470 - val_accuracy: 0.7823
Epoch 11/20
83/83 [==============================] - 29s 344ms/step - loss: 0.3845 - accuracy: 0.8449 - val_loss: 0.5867 - val_accuracy: 0.7721
Epoch 12/20
83/83 [==============================] - 22s 264ms/step - loss: 0.3254 - accuracy: 0.8698 - val_loss: 0.3665 - val_accuracy: 0.8776
Epoch 13/20
83/83 [==============================] - 22s 261ms/step - loss: 0.2747 - accuracy: 0.8933 - val_loss: 0.4773 - val_accuracy: 0.8197
Epoch 14/20
83/83 [==============================] - 22s 261ms/step - loss: 0.2563 - accuracy: 0.9020 - val_loss: 0.3764 - val_accuracy: 0.8707
Epoch 15/20
```



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

 conv2d_1 (Conv2D)           (None, 146, 146, 64)      18496

 max_pooling2d (MaxPooling2D  (None, 73, 73, 64)       0
 )

 dropout (Dropout)           (None, 73, 73, 64)        0

 conv2d_2 (Conv2D)           (None, 71, 71, 64)        36928

 conv2d_3 (Conv2D)           (None, 69, 69, 64)        36928

 dropout_1 (Dropout)         (None, 69, 69, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 34, 34, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 34, 34, 64)        0

 conv2d_4 (Conv2D)           (None, 32, 32, 128)       73856

 conv2d_5 (Conv2D)           (None, 30, 30, 128)       147584

 conv2d_6 (Conv2D)           (None, 28, 28, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 dropout_3 (Dropout)         (None, 14, 14, 128)       0

 conv2d_7 (Conv2D)           (None, 12, 12, 128)       147584

 conv2d_8 (Conv2D)           (None, 10, 10, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 dropout_4 (Dropout)         (None, 5, 5, 256)         0
```

We have achieved the best performance by the model by giving us the accuracy of 88.78% on the validation data. We have used this model for our classification project and proceeded further with the Flask Integration & deployment of the project.

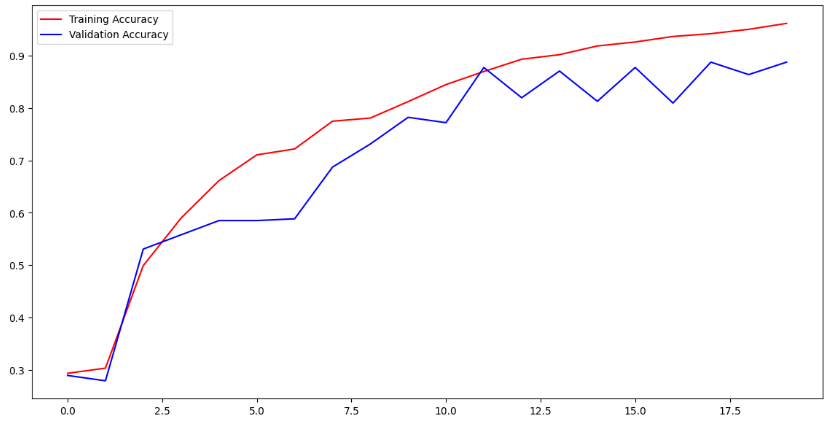
5. FLOWCHART



The flowchart shows all the system phases. It is used for brain tumor detection, and is dedicated for brain tumor classification. Brain MRIs are provided to the input, then preprocessing and segmentation are applied. After that feature extraction and feature reduction are used to select the relevant features. Classification is applied to the features to classify images into the four classes i.e. glioma_tumor, meningioma_tumor, no_tumor and pituitary_tumor.

6. RESULTS

Fig 1: Plots of accuracy and loss vs. epochs of the models used for classification



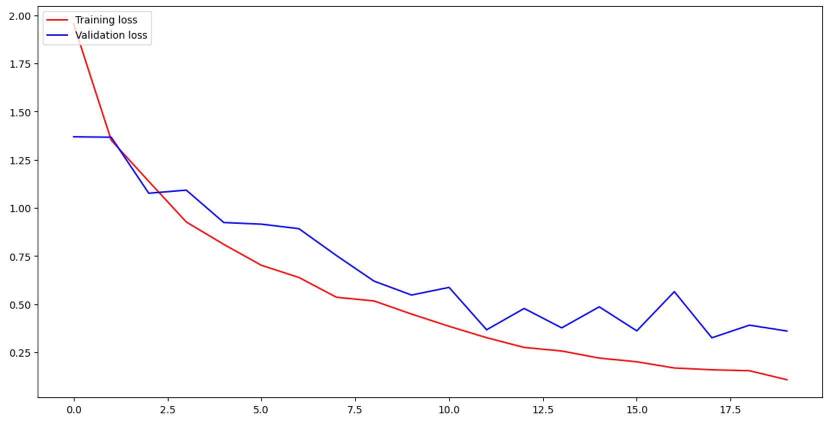
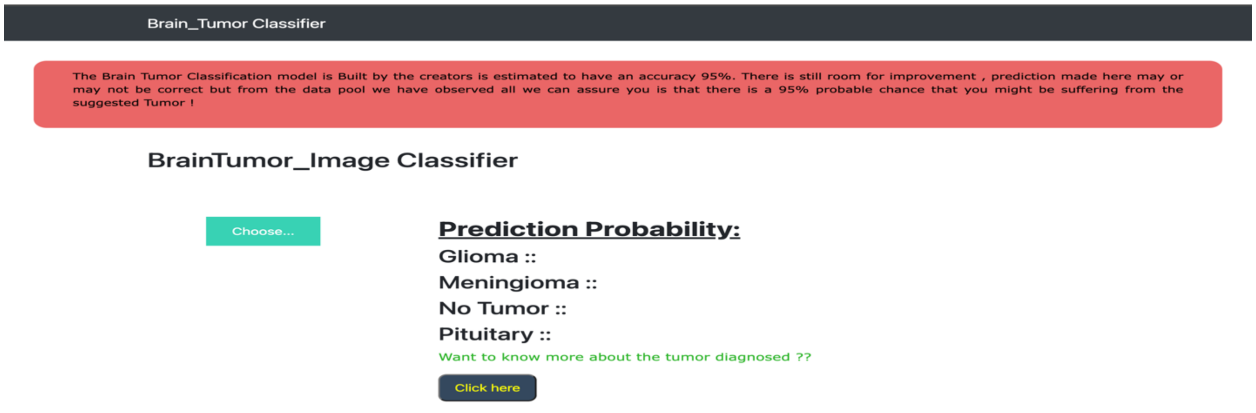


Table 1: Comparative study of performance of models

No.	**Model**	**Training Acc (in %)**	**Training Loss**	**Validation Acc (in %)**	**Validation Loss**
1	VGG16	99.06	0.0330	76.65	3.1731
2	VGG19	98.01	0.0604	71.32	3.3670
3	Resnet 50	75.92	0.7895	55.33	5.4868
4	Custom_final_Model	96.18	0.1070	88.78	0.3600

From the above tabulation it is clearly observed that the Transfer Learning model such as VGG16 , VGG19 and Resnet50 that we have applied performed a bit poorer than the custom brute force model that we created from scratch. The model that we created from scratch gave us the overall accuracy of 88.78 % which has been then saved and used for the Brain Tumor Classification purpose.

6.1 FRONT END:


Brain_Tumor Classifier

The Brain Tumor Classification model is Built by the creators is estimated to have an accuracy 95%. There is still room for improvement , prediction made here may or may not be correct but from the data pool we have observed all we can assure you is that there is a 95% probable chance that you might be suffering from the suggested Tumor !

BrainTumor_Image Classifier

Prediction Probability:

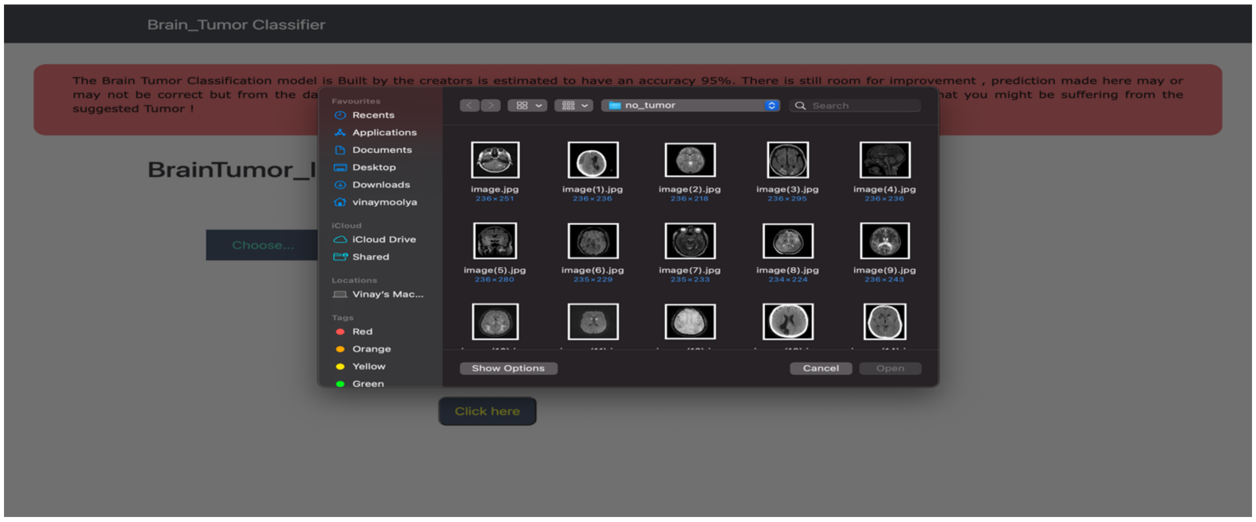
Glioma ::

Meningioma ::

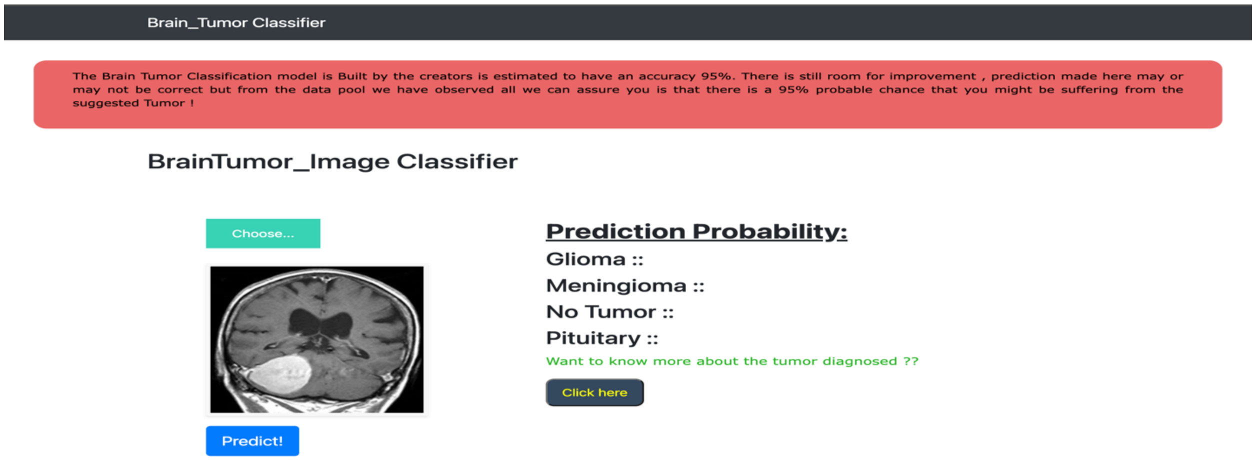
No Tumor ::

Pituitary ::

Want to know more about the tumor diagnosed ??


Brain_Tumor Classifier

The Brain Tumor Classification model is Built by the creators is estimated to have an accuracy 95%. There is still room for improvement , prediction made here may or may not be correct but from the data ... hat you might be suffering from the suggested Tumor !


Brain_Tumor Classifier

The Brain Tumor Classification model is Built by the creators is estimated to have an accuracy 95%. There is still room for improvement , prediction made here may or may not be correct but from the data pool we have observed all we can assure you is that there is a 95% probable chance that you might be suffering from the suggested Tumor !

BrainTumor_Image Classifier

Prediction Probability:

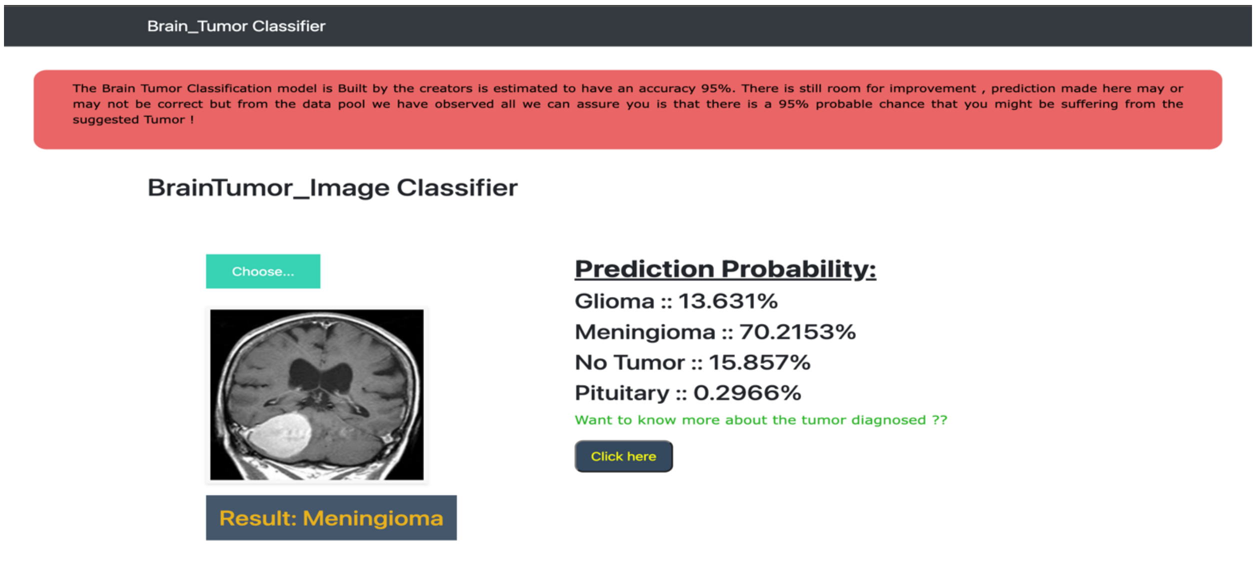
Glioma ::

Meningioma ::

No Tumor ::

Pituitary ::

Want to know more about the tumor diagnosed ??



Brain_Tumor Classifier

The Brain Tumor Classification model is Built by the creators is estimated to have an accuracy 95%. There is still room for improvement , prediction made here may or may not be correct but from the data pool we have observed all we can assure you is that there is a 95% probable chance that you might be suffering from the suggested Tumor !

BrainTumor_Image Classifier

Prediction Probability:

Glioma :: 13.631%

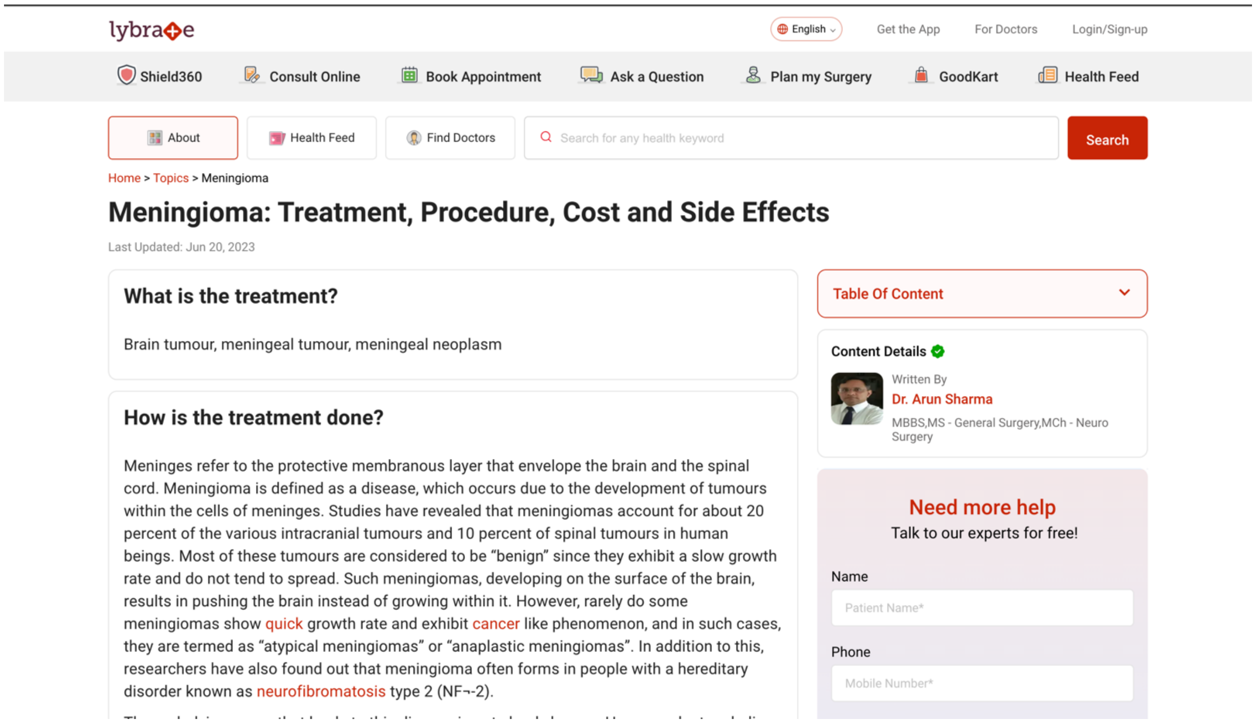
Meningioma :: 70.2153%

No Tumor :: 15.857%

Pituitary :: 0.2966%

Want to know more about the tumor diagnosed ??

Result: Meningioma



Home > Topics > Meningioma

Meningioma: Treatment, Procedure, Cost and Side Effects

Last Updated: Jun 20, 2023

What is the treatment?

Brain tumour, meningeal tumour, meningeal neoplasm

How is the treatment done?

Meninges refer to the protective membranous layer that envelope the brain and the spinal cord. Meningioma is defined as a disease, which occurs due to the development of tumours within the cells of meninges. Studies have revealed that meningiomas account for about 20 percent of the various intracranial tumours and 10 percent of spinal tumours in human beings. Most of these tumours are considered to be "benign" since they exhibit a slow growth rate and do not tend to spread. Such meningiomas, developing on the surface of the brain, results in pushing the brain instead of growing within it. However, rarely do some meningiomas show quick growth rate and exhibit cancer like phenomenon, and in such cases, they are termed as "atypical meningiomas" or "anaplastic meningiomas". In addition to this, researchers have also found out that meningioma often forms in people with a hereditary disorder known as neurofibromatosis type 2 (NF¬-2).

Content Details

Written By

Dr. Arun Sharma

MBBS,MS - General Surgery,MCh - Neuro Surgery

Need more help

Talk to our experts for free!

Name

Patient Name*

Phone

Mobile Number*

The project we have created classifies the MRI image taken as an input from the user and classifies it into one of the types and then makes the prediction about which class the particular tumor belongs to and presents it to the user of the probability as well. Based on the prediction the patients will also be directed to a webpage where they can know more about the diagnosed tumor and be aware of it and take the necessary steps.

Link to the Github Dataset : https://github.com/VinayMoolya/BrainTumor_SI

Link to the Demo Video: https://youtu.be/Oe4KauYR7R8

7. ADVANTAGES AND DISADVANTAGES

7.1 Advantages:

1. **Accurac**y: MRI images provide detailed and accurate information about the brain, making them an effective tool for detecting brain tumors. Integrating MRI image analysis with Flask allows for efficient processing and analysis, leading to improved accuracy in tumor detection.
2. **Non-invasive**: MRI scans are non-invasive, meaning they do not require any surgical procedures or injections. This makes them a safer option for patients compared to invasive methods like biopsies. Integrating MRI with Flask enables easy access and analysis of these non-invasive scans.
3. **Visualization**: Flask provides a flexible framework for building web-based interfaces. By integrating MRI image analysis with Flask, medical professionals can visualize and interpret the results conveniently through a web interface. This enables easier collaboration and communication among healthcare providers.
4. **Accessibility**: Flask allows for the deployment of applications on the web, making the brain tumor detection system accessible from various devices and locations. Medical professionals can access the system remotely, facilitating faster diagnosis and treatment planning.
5. **Scalability**: Flask is a lightweight framework that can handle high traffic and requests efficiently. It allows for easy scaling of the application, ensuring that multiple users can access the brain tumor detection system simultaneously without performance issues.

7.2 Disadvantages:

1. **Expertise and Training**: Developing a brain tumor detection system using MRI images and Flask requires expertise in medical imaging, machine learning, and web development. Acquiring and maintaining the necessary skills can be challenging and time-consuming.
2. **False Positives and Negatives**: Despite the accuracy of MRI images, false positives and false negatives can still occur in tumor detection. Integration with Flask does not eliminate these potential errors and requires careful validation and continuous improvement of the detection algorithms.
3. **Computational Resources**: Processing and analyzing MRI images can be computationally intensive, especially when dealing with large datasets. The integration

with Flask needs to consider the computational resources required to handle the image analysis and ensure the system can handle the load effectively.

4. **Regulatory Compliance**: Medical imaging systems are subject to regulatory requirements and privacy considerations. Integrating MRI image analysis with Flask requires compliance with relevant regulations, such as data security and patient privacy, which may involve additional complexity and cost.
5. **Dependency on Internet Connection**: Deploying the brain tumor detection system with Flask on the web relies on a stable internet connection. In areas with limited connectivity, accessing and utilizing the system may pose challenges.

8. APPLICATIONS:

1. **Medical Diagnosis and Treatment**: The system can assist radiologists and oncologists in accurately identifying and diagnosing brain tumors based on MRI images. It can provide valuable information about the tumor's size, location, and characteristics, helping in treatment planning and monitoring the tumor's progression.
2. **Screening and Early Detection:** Integrating the solution with Flask allows for the development of web-based screening tools that can be used by healthcare providers to identify potential brain tumors at an early stage. Early detection increases the chances of successful treatment and improves patient outcomes.
3. **Telemedicine and Remote Consultations:** With the web-based interface provided by Flask, the brain tumor detection system can be accessed remotely, enabling telemedicine consultations. This is particularly valuable in areas with limited access to specialized medical facilities, allowing patients to receive expert opinions without the need for physical travel.
4. **Research and Data Analysis**: The system can be utilized in medical research to analyze large datasets of MRI images and extract insights related to brain tumors. Researchers can use the tool to study patterns, develop new algorithms, and contribute to advancements in brain tumor detection and treatment.
5. **Educational and Training Purposes**: The integrated solution can be used as an educational tool for medical students, residents, and healthcare professionals. It can provide a platform for learning and practicing brain tumor detection skills, aiding in the training of future radiologists and oncologists.

9. CONCLUSION

In our study, we utilized a dataset of Brain MRI images and applied four Convolutional Neural Network (CNN) models for the task of classifying the scans into four different classes: Glioma, Meningioma, No tumor, and Pituitary. The purpose was to evaluate the performance of these models in accurately identifying brain tumors.

Table 1 provides the results of our experiments, showcasing the performance metrics achieved by each model. These metrics typically include accuracy, precision, recall, and F1 score, among others. By examining these metrics, we can assess the effectiveness of each model in correctly classifying the Brain MRI scans.

Additionally, we have included Fig 1, which likely represents a visual representation of the performance comparison among the different models. It might display a chart or graph illustrating the performance metrics or accuracy scores of each model, making it easier to visualize the differences in their performance.

Based on the provided information, our Brute Force Custom Model emerged as the top-performing model among the listed options. This indicates that the custom model demonstrated the highest accuracy and most reliable results in identifying brain tumors from the MRI images. The reasons behind its superior performance might include its unique architecture, optimized hyperparameters, or specialized features designed specifically for this task.

10. FUTURE SCOPE

The future scope of the project can involve several aspects to enhance the accuracy of brain tumor classification using custom Convolutional Neural Networks (CNNs). Here are some potential areas to explore:

1. **Data Augmentation Technique**s: Experiment with various data augmentation techniques to increase the diversity of the training data. This can include random rotations, translations, flips, and zooms. Additionally, consider applying more advanced techniques such as elastic deformations, intensity transformations, or generative adversarial networks (GANs) to further augment the dataset.
2. **Transfer Learning:** Although you mentioned creating models from scratch, it may still be worthwhile to explore transfer learning. Fine-tuning pre-trained CNN models on large-scale image datasets (e.g., ImageNet) and then adapting them to brain tumor classification can potentially leverage the learned features and improve performance. Compare the performance of custom-built models with those utilizing transfer learning to determine the best approach.
3. **Ensemble Methods:** Investigate the use of ensemble methods to combine predictions from multiple CNN models. Ensemble techniques, such as majority voting or averaging, can help reduce errors and enhance overall classification accuracy. Consider training multiple CNN models with different architectures or hyperparameters and combining their predictions for improved performance.

11. BIBLIOGRAPHY:

11.1 REFERENCES

[1] Khan, M.A., Ashraf, I., Alhaisoni, M., Damaševičius, R., Scherer, R., Rehman, A. and Bukhari, S.A.C., 2020. Multimodal brain tumor classification using deep learning and robust feature selection: A machine learning application for radiologists. *Diagnostics*, *10*(8), p.565.

[2] Sadad, T., Rehman, A., Munir, A., Saba, T., Tariq, U., Ayesha, N. and Abbasi, R., 2021. Brain tumor detection and multi‐classification using advanced deep learning techniques. *Microscopy Research and Technique*, *84*(6), pp.1296-1308.

[3] Mehrotra, R., Ansari, M.A., Agrawal, R. and Anand, R.S., 2020. A transfer learning approach for AI-based classification of brain tumors. *Machine Learning with Applications*, *2*, p.100003.

[4] Sachdeva, J., Kumar, V., Gupta, I., Khandelwal, N. and Ahuja, C.K., 2013. Segmentation, feature extraction, and multiclass brain tumor classification. Journal of digital imaging, 26, pp.1141-1150.

[5] Sharif, M.I., Khan, M.A., Alhussein, M., Aurangzeb, K. and Raza, M., 2021. A decision support system for multimodal brain tumor classification using deep learning. Complex & Intelligent Systems, pp.1-14.

[6] Vankdothu, R., Hameed, M.A. and Fatima, H., 2022. A brain tumor identification and classification using deep learning based on CNN-LSTM method. Computers and Electrical Engineering, 101, p.107960.

[7] Toğaçar, M., Ergen, B., Cömert, Z. and Özyurt, F., 2020. A deep feature learning model for pneumonia detection applying a combination of mRMR feature selection and machine learning models. Irbm, 41(4), pp.212-222.

[8] Hashmi, M.F., Katiyar, S., Keskar, A.G., Bokde, N.D. and Geem, Z.W., 2020. Efficient pneumonia detection in chest xray images using deep transfer learning. Diagnostics, 10(6), p.417.

[9] Rinesh, S., Maheswari, K., Arthi, B., Sherubha, P., Vijay, A., Sridhar, S., Rajendran, T. and Waji, Y.A., 2022. Investigations on brain tumor classification using hybrid machine learning algorithms. Journal of Healthcare Engineering, 2022.

[10] Díaz-Pernas, F.J., Martínez-Zarzuela, M., Antón-Rodríguez, M. and González-Ortega, D., 2021, February. A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network. In Healthcare (Vol. 9, No. 2, p. 153). MDPI.

11.2 APPENDIX

11.2.1 SOURCE CODE

App.py:

```
 import os
import numpy as np
import json
# Keras
import keras
from keras.models import load_model
from keras.preprocessing import image

# Flask utils
from flask import Flask, request, render_template
from werkzeug.utils import secure_filename
import tensorflow_hub as hub


# Define a flask app
app = Flask(__name__)

# Model saved with Keras model.save()
model = load_model(('BrainTumor_SI/braintumor.h5'), custom_objects={'KerasLayer': hub.KerasLayer})


def model_predict(img_path, model):
    test_image = image.load_img(img_path, target_size = (150,150))
    test_image = image.img_to_array(test_image)
    test_image = np.array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = model.predict(test_image)

    labels = ['Glioma','Meningioma','No Tumor','Pituitary']

    return labels[result.argmax()] , result


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


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']
        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)

        # Make prediction
        preds , res = model_predict(file_path, model)

        data = {
            "pred" : preds,
            "glioma" : str(round(res[0][0]*100,4)),
            "meningioma" :str(round(res[0][1]*100,4)),
            "notumor": str(round(res[0][2]*100,4)),
            "pituitary":str(round(res[0][3]*100,4)),
            "glink": "https://www.lybrate.com/topic/glioma",
            "mlink" : "https://www.lybrate.com/topic/meningioma",
            "nlink" : "https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Brain-Tumors",
                                                                                                    "plink"            :
"https://medsurgeindia.com/cost/pituitary-tumor-treatment-cost-in-india/#:~:text=A%20pituitary%20tumor%2
0is%20a,that%20regulate%20vital%20bodily%20processes."
        }

        y = json.dumps(data)
        return y
    return None


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

main.js

```
$(document).ready(function () {
  // Init
  $(".image-section").hide();
  $(".loader").hide();
  $("#result").hide();

  // Upload Preview
  function readURL(input) {
    if (input.files && input.files[0]) {
      var reader = new FileReader();
      reader.onload = function (e) {
        $("#imagePreview").css(
          "background-image",
          "url(" + e.target.result + ")"
        );
        $("#imagePreview").hide();
        $("#imagePreview").fadeIn(650);
      };
    reader.readAsDataURL(input.files[0]);
  }
}
$("#imageUpload").change(function () {
  $(".image-section").show();
  $("#btn-predict").show();
  $("#result").text("");
  $("#result").hide();
  readURL(this);
});

// Predict
$("#btn-predict").click(function () {
  var form_data = new FormData($("#upload-file")[0]);

  // Show loading animation
  $(this).hide();
  $(".loader").show();

  // Make prediction by calling api /predict
  $.ajax({
    type: "POST",
    url: "/predict",
    data: form_data,
    contentType: false,
    cache: false,
    processData: false,
    async: true,
    success: function (y) {
      // Get and display the result
      obj = JSON.parse(y);
      labels = ["Glioma", "Meningioma", "No Tumor", "Pituitary"];
      ind = 0;
      for (let i = 0; i < labels.length; i++) {
        if (labels[i] === obj["pred"]) {
          ind = i;
          break;
        }
      }
      links = [obj["glink"], obj["mlink"], obj["nlink"], obj["plink"]];
      $(".loader").hide();
      $("#result").fadeIn(600);
      $("#result").text(" Result:  " + obj["pred"]);
      $("#glioma").text(obj["glioma"] + "%");
      $("#meningioma").text(obj["meningioma"] + "%");
      $("#notumor").text(obj["notumor"] + "%");
      $("#pituitary").text(obj["pituitary"] + "%");
      var ele = document.getElementById("linktoknow");
      ele.href = links[ind];
      var dia = document.getElementById("diag");
      if (ind === 2) {
          dia.innerHTML = `<span>
              "Hurray you are not diagnosed with Any tumor... But still click on
              the link below to be aware of Brain Tumors";
            </span>1`;
        }
        console.log("Success!");
      },
    });
  });
});
```