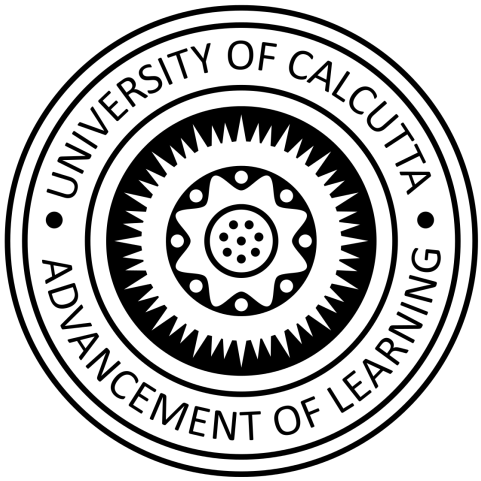
Brain Tumor Detection



Department of Computer Science
Gurudas College
Calcutta University
25/07/2021

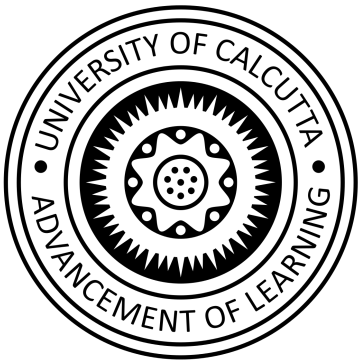
Detection of tumorous cells using machine learning models

Submitted in partial fulfillment for the requirements for the degree Bachelor of Science (Honors) in Computer Science.
Academic Year: **2018-2021**

Authors:
• Shoptorshi • Brahmajit • Rajarshi • Bhargav

Supervisor: Kajari Bhattacharjeee

Certificate



Department of Computer Science
Gurudas College
Calcutta University

This is to certify that the project entitled "Brain tumor detection using Machine Learning models" is a bona fide work of **Shoptorshi**, **Brahmajit**, **Rajarshi** and **Bhargav** submitted to Gurudas College, University of Calcutta; in partial fulfillment of the requirement for the award of the degree "Bachelor of Science (Honors)" in Computer Science.

Supervisor:

_____

Kajari Bhattacharjeee

Principal:

_____

Dr. Mausumi Chatterjee

Department
Head:

_____

Srijeeta Charkraborty

Acknowledgement

I would like to express my special thanks of gratitude to our supervisor **Kajari Bhattacharjeee**, teachers of our department **Sonali Gupta** and **Srijeeta Charkraborty** (current department head); as well as our principal ma'am **Dr. Mausumi Chatterjee** who gave us the golden opportunity to do this wonderful project on *Brain tumor detection*, which has also helped us in doing a lot of research and we came to know about many new things.

We are really thankful to them.

Secondly we would like to thank the incredible authors of the research papers in our references/citations for providing us with the information.

Table of Content

1 Abstract

Brain Tumor Detection

Using various machine learning models to detect brain tumor.

Tumors are cancerous or non-cancerous mass or growth of abnormal cells in brain. Tumors can start in brain, or cancer elsewhere in the body can spread to brain. There are many way to control the occurrence of these abnormal cells. A tumor can be denoted as a malformed mass of tissues wherein the cells multiply abruptly and ceaselessly, that is there is no control over the growth of the cells.

The process of Image segmentation is adopted for extracting abnormal tumor region within the brain. In the MRI (magnetic resonance image), segmentation of brain tissue holds very significant in order to identify the presence of outlines concerning the brain tumor. There is abundance of hidden information in stored in the Health care sector. With appropriate use of accurate data mining classification techniques, early prediction of any disease can be effectively performed.

The project examines list of risk factors that are being traced out in brain tumor surveillance systems. Also the method proposed assures to be highly efficient and precise for brain tumor detection, classification and segmentation. To achieve this precise automatic or semi-automatic methods are needed. The project proposes an automatic segmentation method that relies upon *CNN (Convolution Neural Networks)* , *VGG 16* and *Resent 50* , determining small 7 x 7 kernels. By incorporating this single technique, segmentation and classification is accomplished. CNN (a ML technique) from NN (Neural Networks)wherein it has layer based for results classification.

Various levels involved in the proposed mechanisms are:

1. **Data collection**
2. **Pre-processing**
3. **Average filtering**
4. **segmentation**
5. **feature extraction**
6. **CNN (or any other model) via classification and identification. By utilizing the DM (data mining) techniques, significant relations and patterns from the data can be extracted. The techniques of ML (machine learning) and Data mining are being effectively employed for brain tumor detection and prevention at an early stage.**

2 Introduction

2.1 Domain Description

1. **Neurological Examination**: It is a series of test to measures the function of the patients nervous system and also his/her physical and mental alertness.

2. **Machine Learning**: Machine learning approaches address these problems by mainly using hand-crafted features (or pre-defined features). As an initial step in this kind of segmentation, the key information is extracted from the input image using some feature extraction algorithm, and then a discriminative model is trained to recognize the tumor from normal tissues. The designed machine learning techniques generally employ hand-crafted features with various classifiers, such as random forest, support vector machine (SVM), fuzzy clustering. The designed methods and features extraction algorithms have to extract features, edge-related details, and other necessary information—which is time-consuming. Moreover, when boundaries between healthy tissues and tumors are fuzzy/vague, these methods demonstrate poorer performances.

3. **Brain Scan**: Brain scan is a picture of the internal structure of the brain. A specialized machine takes a scan in the same way as a digital camera takes a photograph. Using computer technology, a scan compiles an image of the brain by photographing it from various angles. Some types of scan uses contrast agent (or contrast dye), which helps the doctor to see the difference between normal and abnormal brain tissues.

   MRI (Magnetic Resonance Imaging): It is a scanning device that uses magnetic field and computer to capture images of the brain on films. It does not use x-rays. It provides pictures from various planes, which permits doctor to create a three-dimensional image of the tumor. The MRI detects signals emitted from normal and abnormal tissues, providing clear images of almost all tumors.

2.2 Motivation

The motivation is to develop a software with better segmentation capability for use in medical imaging to detect diseases like brain tumor. Image segmentation has been identified as the key problem of medical image analysis and remains a popular and challenging area of research. Image segmentation is increasingly used in many clinical and research applications to analyze medical imaging datasets; which motivated us to present a snapshot of dynamically changing field of medical image segmentation.

CT (Computed Tomography), MRI (Magnetic Resonance Imaging), PET (Positron Emission Tomography) etc. generates a large amount of image information. With the improved technology, not only does the size and resolution of the images grow but also the number of dimensions increases. In the future, we would like to have algorithms which can automatically detect diseases, lesions and tumors, and highlight their locations in the large pile of images.

The motivation of this work is to increase patient safety by providing better and more precise data for medical decision.

2.3 Scope of Work

2.3.1 Deliverables

- Working program to take an MRI scan as input and predict presence of tumorous cells with $\geq 90\%$ accuracy.

2.3.2 Scope

- The working program has external dependencies (libraries) and it's expected to have a them installed for the program to work.

2.3.3 Timeline

- **April 27, 2021** Project Assigned
- **May 2, 2021** Project finalized by supervisor, and group is divided into groups of two.
- **May 3, 2021** Data collection started.
- **May 12, 2021** Project Repository created and coding is started.
- **July 7, 2021** Coding is finished, documentation is started.
- **Just 21, 2021** Documentation complete.

2.3.4 Reports

- Constantly updating and pushing code to repository.
- Both teams staying in touch with each other to keep up with each others progress.
- Report back to supervisor every once a week.

3 Background

Natarajan [6] proposed brain tumor detection method for MRI brain images. The MRI brain images are first preprocessed using median filter, then segmentation of image is done using threshold segmentation and morphological operations are applied and then finally, the tumor region is obtained using image subtraction technique. This approach gives the exact shape of tumor in MRI brain image. Joshi [4] proposed brain tumor detection and classification system in MR images by first extracting the tumor portion from brain image, then extracting the texture features of the detected tumor using Gray Level Co- occurrence Matrix (GLCM) and then classified using neuro-fuzzy classifier. Amin and Mageed [8] proposed neural network and segmentation base system to automatically detect the tumor in brain MRI images. The Principal Component Analysis (PCA) is used for feature extraction and then Multi-Layer Perceptron (MLP) is used classify the extracted features of MRI brain image. The average recognition rate is 88.2% and peak recognition rate is 96.7%. Sapra [9] proposed image segmentation technique to detect brain tumor from MRI images and then Probabilistic Neural Network (PNN) is used for automated brain tumor classification in MRI scans. PNN system proposed handle the process of brain tumor classification more accurately. Suchita and Lalit [2] proposed unsupervised neural network learning technique for classification of brain MRI images. The MRI brain images are first preprocessed which include noise filtering, edge detection, then the tumor is extracted using segmentation. The texture features are extracted using Gray-Level Co-occurrence Matrix(GLCM) and then Self-Organizing Maps (SOM) are used to classify the brain as normal or abnormal brain, that is, whether it contain tumor or not. Rajeshwari and Sharmila [7] proposed preprocessing techniques which are used to improve the quality of MRI image before using it into an application. The average, median and wiener filters are used for noise removal and interpolation based Discrete Wavelet Transform (DWT) technique is used for resolution enhancement. The Peak Signal to Noise Ratio (PSNR) is used for evaluation of these techniques.

4 Methodology

As per literature survey, it was found that automated brain tumor detection is very necessary as high accuracy is needed when human life is involved. Automated detection of tumor in MR images involves feature extraction and classification using machine learning algorithm. In this paper, a system to automatically detect tumor in MR images is proposed as shown in Figure 1

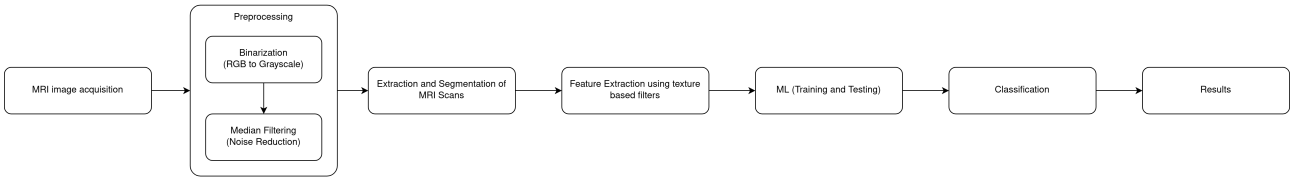
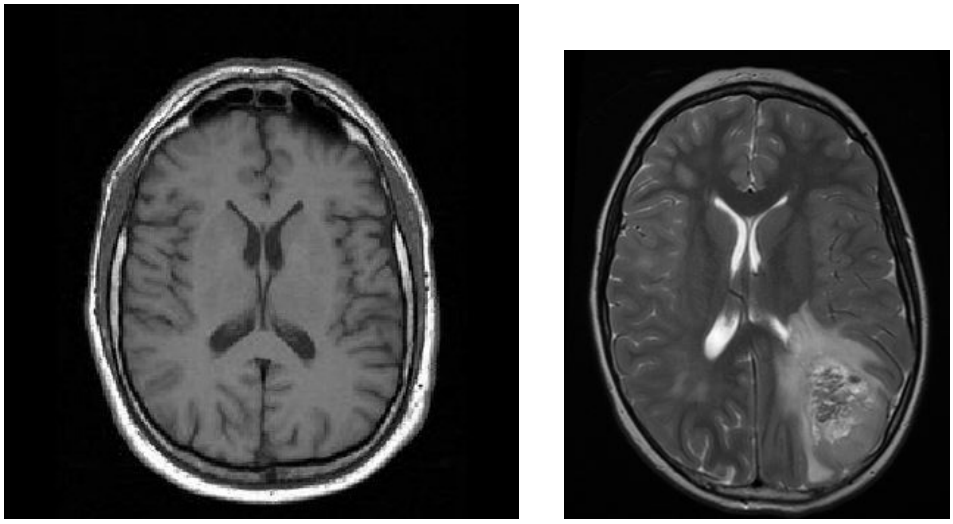


Figure 1: Proposed Methodology

4.1 Image Acquisition

The MRI brain images are acquired and are given as input to pre-processing stage. The sample brain MR images are shown in Figure 2.



(a) MRI scan shown no presence of tumor (b) MRI scan of a tumorous cell

Figure 2: MRI Scans

4.2 Preprocessing

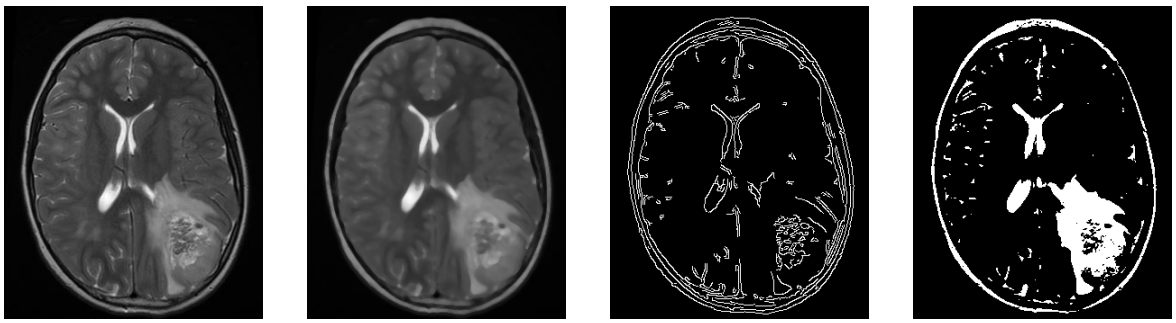
Preprocessing is needed as it provides improvement in image data which enhances some of the image features which are important for further processing. The pre-processing steps that are applied to MR image are as follows:

1. The RGB MR image is converted to gray scale image and then median filter is applied for noise removal from brain MR images as shown in Figure 3b. The noise is to removed for further processing as high accuracy is needed.

2. Then edges are detected from filtered image using canny edge detection as shown in Figure 3c. The edge detected image is needed for segmentation of the image

3. Then watershed segmentation is done for finding the location of the tumor in the brain image as shown in Figure 3d. Segmentation is the process of dividing an image into multiple segments. The aim of segmentation is to change representation of image into something

which is more easy to analyze. The result of watershed segmentation is label image. In label image, all the different objects identified will have different pixel values , all the pixels of first object will have value 1, all the pixels of second object will have value 2 and so on.



(a) Original Image (b) Filtered Imaage (c) Edge Detection (d) Segmentation

Figure 3: preprocessing operations

4.3 Feature Extraction

When input to an algorithm is very large and redundant to be processed, it is transformed into reduced representative set of features called feature vector. Transformation of input data into set of features is called feature extraction. In this step, the important features needed for image classification are extracted. The segmented brain MR image is used and texture features are extracted from the segmented image which shows the texture property of the image. These features are extracted using Gray Level Co-occurrence Matrix (GLCM) as it is robust method with high performance.

The GLCM features are extracted as follows:

1. Energy: It gives a measure of textural uniformity, that is, measure of pixel pair repetitions.

$$E = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} p(i,j)^2 \text{ here, range = [0, 1]} \tag{1}$$

2. Contrast: It gives a measure of intensity contrast between a pixel and its neighbor over the whole image.

$$Con = \sum_{n=0}^{N_g-1} n^2 \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} p(i,j)^2 \text{ here, range = [0, 1]} \tag{2}$$

3. Correlation: It gives a measure of how correlated a pixel to its neighbor over the whole image.

$$C = \frac{1}{\sigma^x \sigma^y} \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} (i,j) p(i,j)^2 - \mu_x \mu_y \text{ here, range = [-1, 1]} \tag{3}$$

4. Homogeneity: It gives a measure of closeness of distribution of elements in GLCM to GLCM diagonal.

$$H = \sum_{i=0}^{N_g-1} \sum_{j=0}^{N_g-1} \frac{p(i,j)}{1 + \pmod{i,j}} \text{ here, range = [0, 1]} \tag{4}$$

4.4 Classification

The Machine learning algorithms are used for classification of MR brain image either as normal or abnormal. The major aim of ML algorithms is to automatically learn and make intelligent decisions. The feature set formed by above specified method was applied to Multi-Layer Perceptron (MLP) and Naive Bayes for classification. MLP is a feed forward artificial neural network model that maps sets of input data into a set of appropriate output. It is known as feed forward because it does not contain any cycles and network output depends only on the current input instance. In MLP, each node is a neuron with a nonlinear activation function. It is based on supervised learning technique. Learning take place by changing connection weights after each piece of data is processed, based on the amount of error in the target output as compared to the expected result. The goal of the learning procedure is to minimize error by improving the current values of the weight associated with each edge. Because of this backward changing process of the weights, model is named as back-propagation.

Naive bayes is a supervised learning as well as statistical method for classification. It is simple probabilistic classifier based on Bayes theorem. It assumes that the value of a particular feature is unrelated to the presence or absence of any other feature. The prior probability and likelihood are calculated in order to calculate the posterior probability. The method of maximum posterior probability is used for parameter estimation. This method requires only a small amount of training data to estimate the parameters which are needed for classification. The time taken for training and classification is less.

For classification three models are used; **CNN**, **VGG 16** and **Resnet 50**.

4.5 Convolutional Neural Network

CNN or Convolutional Neural Network a class of artificial neural network, most commonly applied to analyze visual imagery.They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are only equivariant, as opposed to invariant, to translation. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

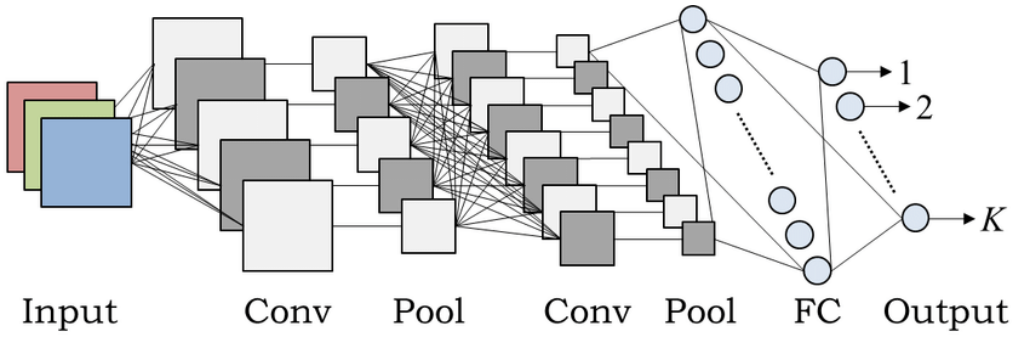


Figure 4: CNN architecture

4.5.1 Architecture

A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU.

As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers [1].

Convolutional Layer: In a CNN, the input is a tensor with a shape: (number of inputs) x (input height) x (input width) x (input channels). After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape: (number of inputs) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a CNN generally has the following attributes:

- Convolutional filters/kernels defined by a width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameters). One layer's input channels must equal the number of output channels (also called depth) of its input.
- Additional hyperparameters of the convolution operation, such as: padding, stride, and dilation.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus.

Pooling Layer: Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2 x 2 are commonly used. Global pooling acts on all the neurons of the feature map. There are two common types of pooling in popular use: max and average. Max pooling uses the maximum value of each local cluster of neurons in the feature map, while average pooling takes the average value.

ReLU Layer:

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function

$$f(x) = \max(0, x) \tag{5}$$

.It effectively removes negative values from an activation map by setting them to zero. It introduces nonlinearities to the decision function and in the overall network without affecting the receptive fields of the convolution layers.

Other functions can also be used to increase nonlinearity, for example the saturating hyperbolic tangent

$$f(x) = \tanh(x) f(x) = |\tanh(x)| \tag{6}$$

and the sigmoid function

$$\sigma(x) = (1 + e^{-x})^{-1} \tag{7}$$

ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy [5].

Fully Connected Layer

After several convolutional and max pooling layers, the final classification is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

Loss Layer

The "loss layer", or "loss function", specifies how training penalizes the deviation between the predicted output of the network, and the true data labels (during supervised learning). Various loss functions can be used, depending on the specific task.

The Softmax loss function is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values is $[0, 1]$. Euclidean loss is used for regressing to real-valued labels $(-\infty, \infty)$.

Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a convolutional layer, each neuron receives input from only a restricted area of the previous layer called the neuron's receptive field. Typically the area is a square (e.g. 5 by 5 neurons). Whereas, in a fully connected layer, the receptive field is the entire previous layer. Thus, in each convolutional layer, each neuron takes input from a larger area in the input than previous layers. This is due to applying the convolution over and over, which takes into account the value of a pixel, as well as its surrounding pixels. When using dilated layers, the number of pixels in the receptive field remains constant, but the field is more sparsely populated as its dimensions grow when combining the effect of several layers.

Weights

Each neuron in a neural network computes an output value by applying a specific function to the input values received from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning consists of iteratively adjusting these biases and weights.

The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces the memory footprint because a single bias and a single vector of weights are used across all receptive fields that share that filter, as opposed to each receptive field having its own bias and vector weighting.

4.6 VGG 16

VGG 16 is is a significantly more accurate ConvNet architecture, which not only achieve state-of-the-art accuracy on ILSVRC classification and localisation tasks, but are also applicable to other image recognition datasets, where they achieve excellent performance even when used as a part of a relatively simple pipelines (e.g. deep features classified by a linear SVM without fine-tuning) [10, p. 1].

4.6.1 Architecture

During training, the input to our ConvNets is a fixed-size 224 × 224 RGB image. The only pre- processing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2.

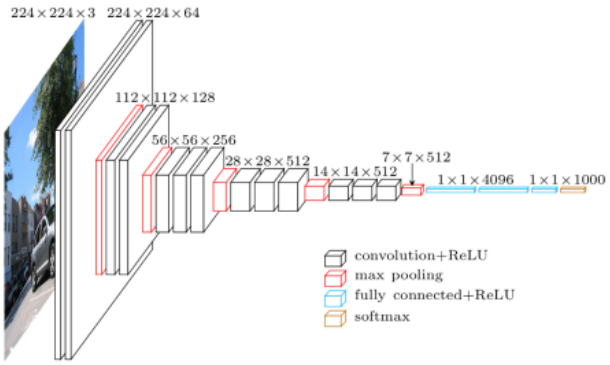


Figure 5: VGG Architecture

A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each class).

The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation: as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory con- sumption and computation time. [10, p. 1]

4.6.2 Configuration

The ConvNet configurations, evaluated in this paper, are outlined in Figure 6, one per column. In the following we will refer to the nets by their names (A–E). All configurations follow the generic design presented in Section 4.6.1 and differ only in the depth: from 11 weight layers in the network A (8 conv. and 3 FC layers) to 19 weight layers in the network E (16 conv. and 3 FC layers). The width of conv. layers (the number of channels) is rather small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until it reaches 512. [10, p. 3]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 6: VGG16 configuration

4.7 ResNet 50

4.7.1 Residual Learning

Let us consider $H(x)$ as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers. If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., $H(x) - x$ (assuming that the input and output are of the same dimensions). So rather than expect stacked layers to approximate $H(x)$, we explicitly let these layers approximate a residual function

$$F(x) = H(x) - x \tag{8}$$

The original function thus becomes $F(x) + x$. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different [3, p. 3].

4.7.2 Network Architecture

The plain/residual network was tested as follows:

Plain Network

Our plain baselines are mainly inspired by the philosophy of VGG nets. The convolutional layers mostly have 3×3 filters and follow two simple design rules:

1. for the same output feature map size, the layers have the same number of filters
2. if the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer.

We perform downsampling directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layers are 34. It is worth noticing that our model has fewer filters and lower complexity than VGG nets. Our 34-layer baseline has 3.6 billion FLOPs (multiply-adds), which is only 18% of VGG-19 (19.6 billion FLOPs) [3, p. 3] .

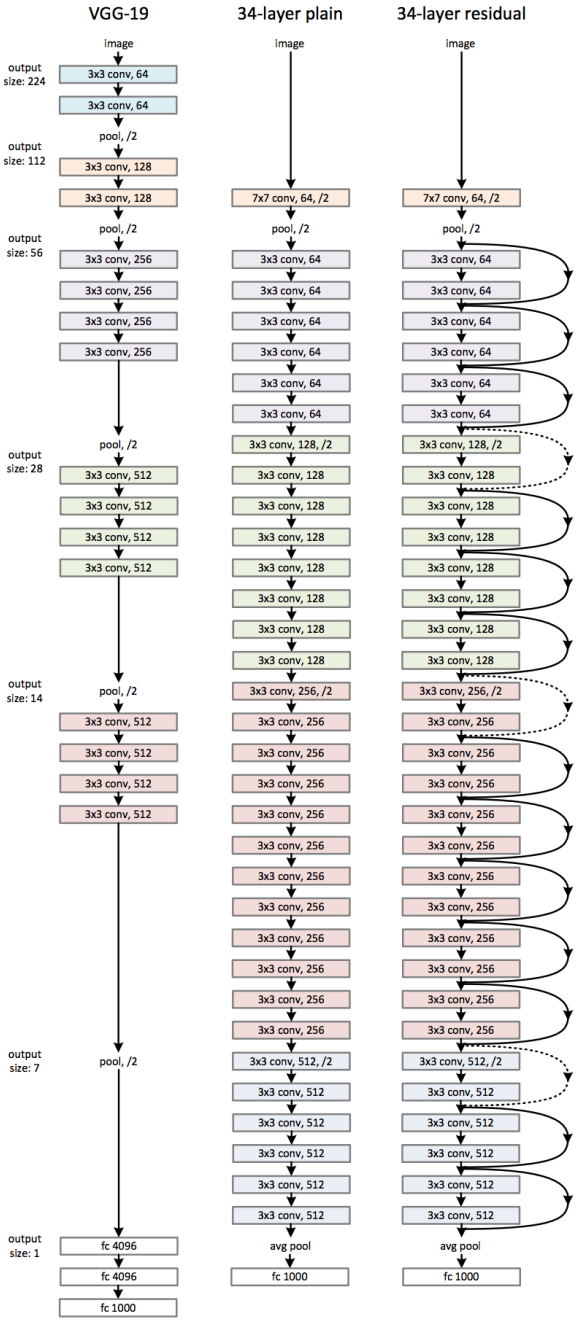


Figure 7: Resnet Architecture

Residual Network Based on the above plain network, we insert shortcut connections which turn the network into its counterpart residual version. The identity shortcuts can be directly used when the input and output are of the same dimensions. When the dimensions increase, we consider two options:

1. The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter

2. The projection shortcut is used to match dimensions (done by 1×1 convolutions).

For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2 [3, p. 4] .

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 8: resnet50 configuration

5 Implementation

5.1 Assumptions and Dependencies

It is assumed that the MRI scans are collected and processed before feeding into the module.

The program is dependent on external modules and are expected to be pre-installed on the system. The modules namely include:

- fast.ai
- tensorflow
- OpenCV
- matplotlib

5.2 Implementation Methods

After acquiring the data, the data is classified into two classes (binary class) of yes, consisting of tumorous cell and no, not consisting tumorous cells. A second batch of images is kept separate for prediction, the prediction batch.

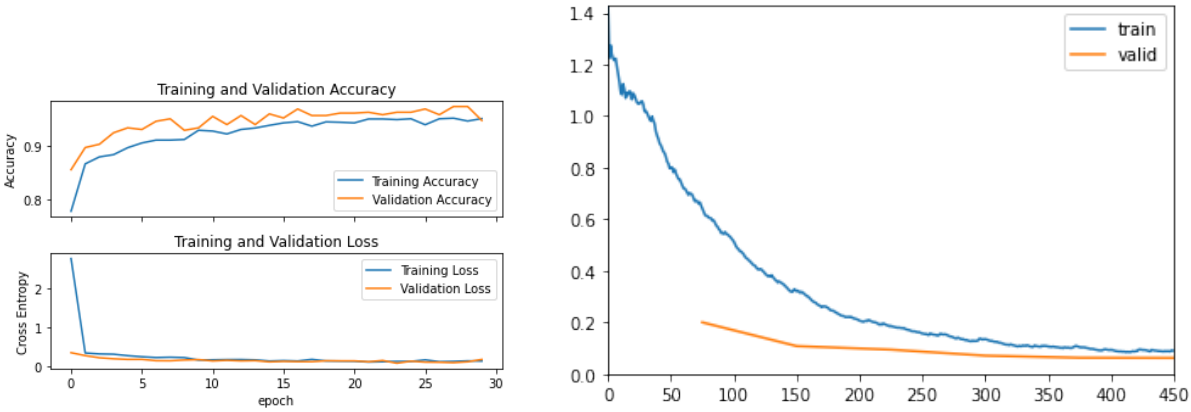
Next three machine learning models are created. A **CNN** or a convolutional neural network, **VGG 16** and **Resnet 50**. The models are implemented in both fast.ai and tensorflow, to be compared later.

While training the images are resized to (150×150) and augmented at runtime, thus removing bias as much as possible. Since there are only two possible classes (yes and no), a binary class is sufficient; dividing the images into training and testing (validation set) in $80:20$ ratio. Giving us 2603 and 650 images for the *yes* and *no* classes, respectively.

After training is complete the models are then evaluated (further discussed in Section 6)

6 Result

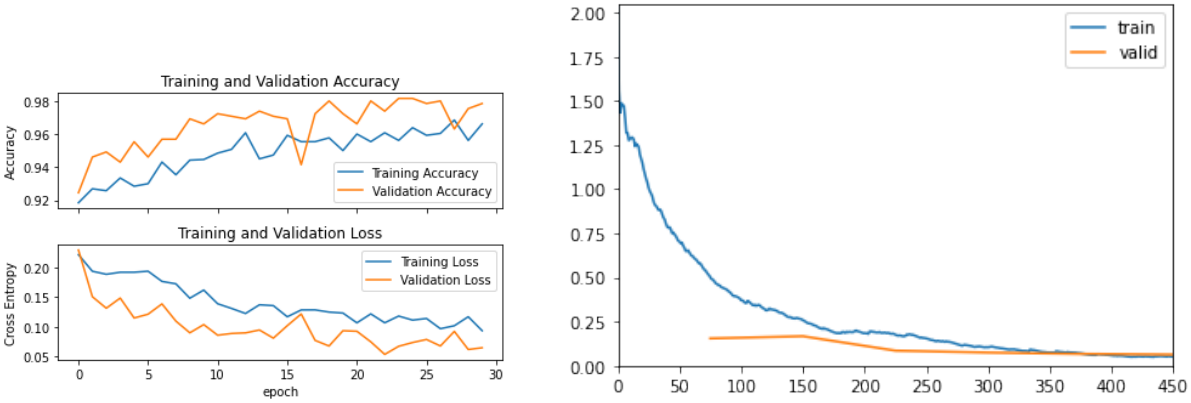
The evaluated models are plotted on graph for better understanding of data.



(a) vgg16 using tensorflow

(b) vgg16 using fast.ai

Figure 9: VGG16 implemented in fast.ai and tensorflow



(a) resnet50 using tensorflow

(b) resnet50 using fast.ai

Figure 10: Resnet 50 implemented in fast.ai and tensorflow

We have achieved around 96% and 98% accuracy for for the VGG16 and ResNet, respectively; using *tensorflow*, and 97% (98% after unfreezing) and 97% (99% after unfreezing) using *fast.ai*. While using CNN accuracy was almost same when implemented by the both the libraries.

There are lot of room for improvement but due to time and hardware constrains we were able to only implement till such.

As it can be seen from Figure 11 there are some false positives in our predictions.

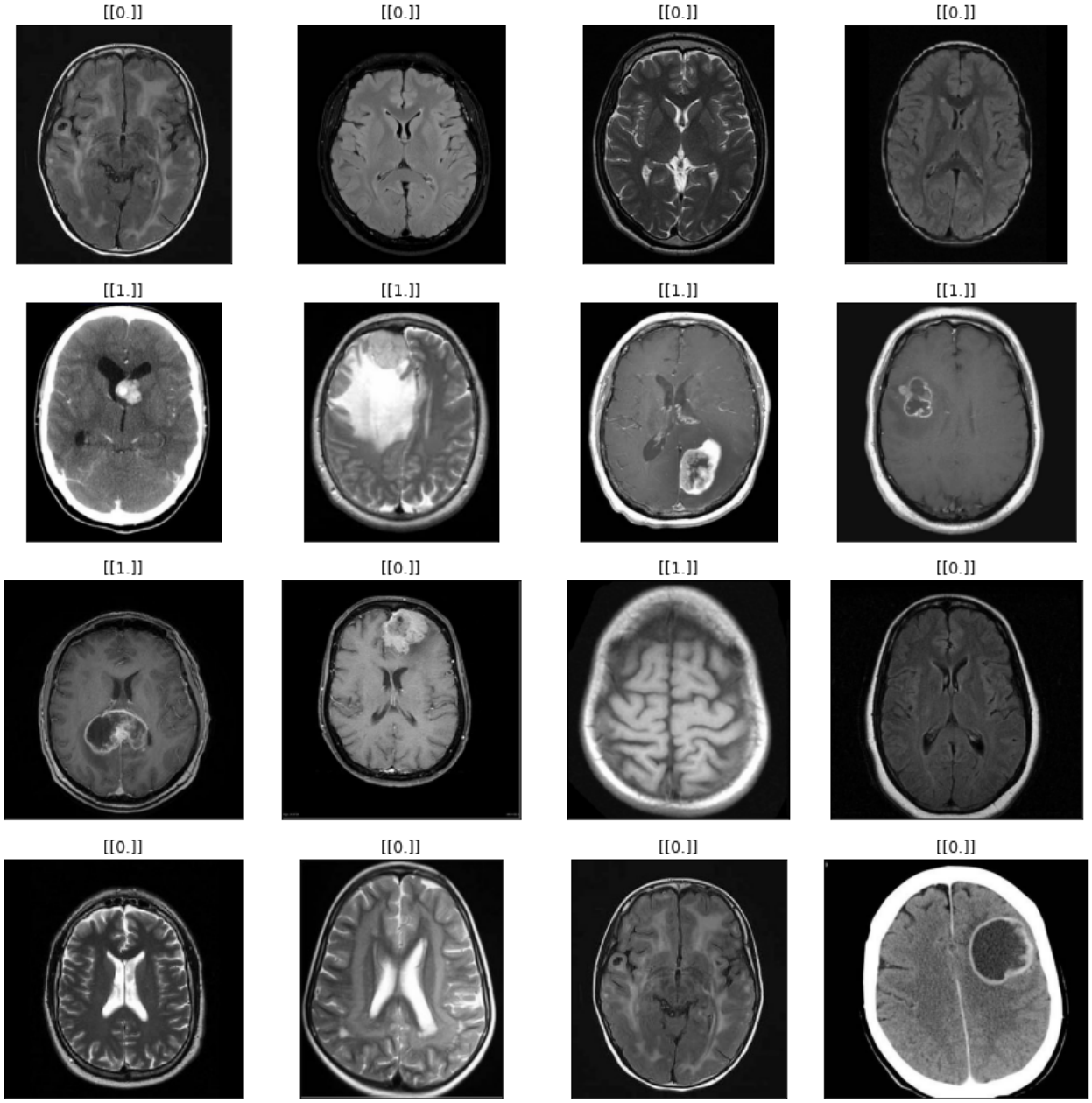


Figure 11: Predictions made by the VGG16 model in tensorflow on random data

7 Appendix

7.1 Tensorflow Implementation

7.1.1 CNN

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dropout, Dense
from matplotlib import pyplot as plt

training_dataset_directory = '/content/dataset/training'
testing_dataset_directory = '/content/dataset/testing'

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1. / 255,
    fill_mode='nearest',
    validation_split=0.2
)

img_size = (150, 150)
img_shape = img_size + (3,)

training_set = train_datagen.flow_from_directory(
    training_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='training'
)

testing_set = train_datagen.flow_from_directory(
    testing_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='validation'
)

model = tf.keras.models.Sequential()
model.add(
    tf.keras.layers.Conv2D(
        16,
        (3, 3),
        activation='relu',
        input_shape=(150, 150, 3)
    )
)
model.add(
    tf.keras.layers.MaxPool2D(2, 2)
```

```
)

model.add(
    tf.keras.layers.Conv2D(
        32,
        (3, 3),
        activation='relu'
    )
)
model.add(
    tf.keras.layers.MaxPool2D(2, 2)
)

model.add(
    tf.keras.layers.Conv2D(
        64,
        (3, 3),
        activation='relu'
    )
)
model.add(
    tf.keras.layers.MaxPool2D(2, 2)
)

model.add(
    tf.keras.layers.Flatten()
)
model.add(
    tf.keras.layers.Dense(
        512,
        activation='relu')
)
model.add(
    tf.keras.layers.Dense(
        1, activation='sigmoid'
    )
)

model.summary()

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

history = model.fit(
    training_set,
    epochs=30,
    verbose=1,
    validation_data=testing_set
)
```

7.1.2 VGG 16

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dropout, Dense
from matplotlib import pyplot as plt

training_dataset_directory = '/content/dataset/training'
testing_dataset_directory = '/content/dataset/testing'

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    preprocessing_function=preprocess_input,
    validation_split=0.2
)

img_size = (150, 150)
img_shape = img_size + (3,)
random_seed = 123

training_set = train_datagen.flow_from_directory(
    training_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='training'
)

testing_set = train_datagen.flow_from_directory(
    testing_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='validation'
)

img_size = (150, 150)
img_shape = img_size + (3,)
random_seed = 123

vgg16 = VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=img_shape
)

num_classes = 1
model = Sequential()
model.add(vgg16)
model.add(Flatten())
```

```
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='sigmoid'))
model.layers[0].trainable = False
model.summary()

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

epochs = 30
history = model.fit(
    training_set,
    epochs=epochs,
    validation_data=testing_set,
)
```

7.1.3 ResNet 50

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dropout, Dense
from tensorflow.keras.optimizers import RMSprop, Adam, SGD
from matplotlib import pyplot as plt

training_dataset_directory = '/content/dataset/training'
testing_dataset_directory = '/content/dataset/testing'

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    preprocessing_function=preprocess_input,
    validation_split=0.2
)

img_size = (150, 150)
img_shape = img_size + (3,)
random_seed = 123

training_set = train_datagen.flow_from_directory(
    training_dataset_directory,
    target_size=img_size,
    batch_size=32,
    class_mode='binary',
    subset='training'
)

testing_set = train_datagen.flow_from_directory(
```

```
    testing_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='validation'
)

img_size = (150, 150)
img_shape = img_size + (3,)
random_seed = 123

resnet50 = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=img_shape
)

num_classes = 1
model = Sequential()
model.add(resnet50)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='sigmoid'))
model.layers[0].trainable = False
model.summary()

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

epochs = 30
history = model.fit(
    training_set,
    epochs=epochs,
    validation_data=testing_set,
)
```

7.1.4 Prediction

```
import tensorflow as tf
from keras_preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing import image_dataset_from_directory
import matplotlib.image as mpimg
from keras.preprocessing import image
import cv2
import os
import glob

import numpy as np
```

```
import matplotlib.pyplot as plt

model = tf.keras.models.load_model('/content/gdrive/MyDrive/model/vgg_imp.h5')

training_dataset_directory = '/content/dataset/training'
testing_dataset_directory = '/content/dataset/testing'

train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    preprocessing_function=preprocess_input,
    validation_split=0.2
)

img_size = (150, 150)
img_shape = img_size + (3,)
random_seed = 123

training_set = train_datagen.flow_from_directory(
    training_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='training'
)

testing_set = train_datagen.flow_from_directory(
    testing_dataset_directory,
    target_size=img_size,
    class_mode='binary',
    subset='validation'
)

loss, acc = model.evaluate(training_set)

IMG_SIZE = (150,150)
BATCH_SIZE = 32
new_dataset = image_dataset_from_directory(
    '/content/gdrive/MyDrive/dataset',
    image_size=IMG_SIZE
)

predictions = model.predict(new_dataset,batch_size=BATCH_SIZE)

img_dir = "/content/gdrive/MyDrive/dataset/pred"
data_path = os.path.join(img_dir,'*g')
files = glob.glob(data_path)
data = []
result = []
for f1 in files:
    test_image = image.load_img(f1, target_size = (150, 150))
```

```
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis = 0)
    images = np.vstack([test_image])
    classes = model.predict(images, batch_size=10)
    classes = np.round(classes)
    data.append(f1)
    result.append(classes)

for x,y in zip(data, result):
  print(x, y)

img = cv2.imread(data[0])
plt.imshow(img)
plt.title(result[0])

L = 4
W = 4
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel()
for i in np.arange(0, L * W):
    img = cv2.imread(data[i])
    axes[i].imshow(img)
    axes[i].set_title(result[i])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
```

References

[1] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. Sebastopol, CA: O'Reilly Media, 2019. ISBN: 978-1-492-03264-9.

[2] Suchita Goswami and L.K.P. Bhaiya. "Brain Tumour Detection Using Unsupervised Learning Based Neural Network". In: Apr. 2013, pp. 573–577. ISBN: 978-1-4673-5603-9. DOI: 10.1109/CSNT.2013.123.

[3] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[4] Dipali Joshi, N.K. Rana, and V.M. Misra. "Classification of Brain Cancer using Artificial Neural Network". In: June 2010, pp. 112–116. DOI: 10.1109/ICECTECH.2010.5479975.

[5] A. Sutskever Krizhevsky and G. E. I. Hinton. "Imagenet classification with deep convolutional neural networks". In: 2012, pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[6] P. Natarajan et al. "Tumor detection using threshold operation in MRI brain images". In: *2012 IEEE International Conference on Computational Intelligence and Computing Research*. 2012, pp. 1–4. DOI: 10.1109/ICCIC.2012.6510299.

[7] S. Rajeshwari and T. Sharmila. "Efficient quality analysis of MRI image using preprocessing techniques". In: Apr. 2013, pp. 391–396. ISBN: 978-1-4673-5759-3. DOI: 10.1109/CICT.2013.6558127.

[8] Mohammed Abdel-Megeed Mohammed Salem. "Brain Tumor Diagnosis Systems Based on Artificial Neural Networks and Segmentation using MRI". In: Jan. 2012.

[9] Pankaj Sapra, Rupinderpal Singh, and Shivani Khurana. "Brain Tumor Detection Using Neural Network". In.

[10] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].