Brain Cancer Detection From MRI: A Machine Learning Approach (Tensorflow)

IRJET Journal

International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 04 | Apr-2018 www.irjet.net p-ISSN: 2395-0072

BRAIN CANCER DETECTION FROM MRI: A MACHINE LEARNING APPROACH (TENSORFLOW)

Aaswad Sawant[#1], Mayur Bhandari [#2], Ravikumar Yadav[#3], Rohan Yele[#4], Mrs. Sneha Bendale[#5]

[1,2,3,4,5]*Department Of Computer Engineering, Terna Engineering College, Mumbai University, India*

Abstract - *Cancer is one of the most harmful disease. MRI is one of the procedures of detecting cancer. Machine learning with image classifier can be used to efficiently detect cancer cells in brain through MRI resulting in saving of valuable time of radiologists and surgeons. This research paper focuses on the use of tensorflow for the detection of brain cancer using MRI. In tensorflow we implemented convolutional neural network with 5 layers. Here total 1800 MRI were used in dataset out of which 900 were cancerous and 900 were non-cancerous. The training accuracy was found to be 99% and validation accuracy was 98.6% in 35 epochs. This system is still in development. The system can be used as a second decision by surgeons and radiologists to detect brain tumor easily and efficiently.*

Key Words: **Tensorflow, MRI, Epoch, Softmax, Rectified Linear Unit (RELU), Convolution Neural Network (CNN).**

1. INTRODUCTION

A brain tumor is a collection, or mass, of abnormal cells in your brain. Symptoms of brain tumors depend on the location and size of the tumor. It is very crucial to detect the brain cancer as early as possible.

MRI can be used to detect the brain cancer by analyzing the MRI but this procedure is vary time consuming for vast number of cases. The literature survey we did lead us to the use of convolutional neural network classifier and we implemented this by using tensorflow[1].

1.1 Tensorflow

TensorFlow [1] is an open source software library released in 2015 by Google to make it easier for developers to design, build, and train deep learning models. TensorFlow originated as an internal library that Google developers used to build models in-house, and we expect additional functionality to be added to the open source version as they are tested and vetted in the internal flavor. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. These arrays are referred to as "tensors". In June 2016, Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google

Although TensorFlow is only one of several options available to developers, we choose to use it here because of its thoughtful design and ease of use.

At a high level, TensorFlow is a Python library that allows users to express arbitrary computation as a graph of data flows. Nodes in this graph represent mathematical operations, whereas edges represent data that is communicated from one node to another. Data in TensorFlow are represented as tensors, which are multidimensional arrays. Although this framework for thinking about computation is valuable in many different fields, TensorFlow is primarily used for deep learning in practice and research.

2. PROCEDURE

2.1 Dataset Acquisition

For any machine learning system data is the single most important thing. For our case the data we acquired was from various online resources [3], [4], [5], [6]. The dataset was in dicom format. We used a tool called mango to obtain their equivalent JPG/PNG image.

2.1.1 Data Augmentation

Plentiful high-quality data is the key to great machine learning models. But good data doesn't grow on trees, and that scarcity can impede the development of a model. One way to get around a lack of data is to augment dataset. Smart approaches to programmatic data augmentation can increase the size of your training set. Even better, the model will often be more robust (and prevent overfitting) and can even be simpler due to a better training set.

Here we flipped each image horizontally and rotation by 30^0 left as well as right by 30^0 in training set allowing us to work on wide variety of data

```
images[n]=cv2.imread(join(canc_path,onlyfiles[n]))
path = "./aug_cancerous/"

file="canc%d.jpg"%d
#saving the Original image as OG
cv2.imwrite(os.path.join( path , file), images[n])
d+=1

#OG rotated 30deg left
file="canc%d.jpg"%d
rotated = imutils.rotate_bound(images[n], 330)
cv2.imwrite(os.path.join( path , file), rotated)
d+=1

#OG rotated 30deg right
file="canc%d.jpg"%d
rotated = imutils.rotate_bound(images[n], 30)
cv2.imwrite(os.path.join( path , file), rotated)
d+=1

#OG's Vertical flip as VF
file="canc%d.jpg"%d
vertical_img = images[n].copy()
vertical_img = cv2.flip( images[n], 1 )
cv2.imwrite(os.path.join( path , file), vertical_img)
d+=1

#VF rotated 30deg left
file="canc%d.jpg"%d
rotated = imutils.rotate_bound(vertical_img, 330)
cv2.imwrite(os.path.join( path , file), rotated)
d+=1

#VF rotated 30deg right
file="canc%d.jpg"%d
rotated = imutils.rotate_bound(vertical_img, 30)
cv2.imwrite(os.path.join( path , file), rotated)
d+=1
```

Fig -1: Data Augmentation script.

The result of each operation can be seen below.

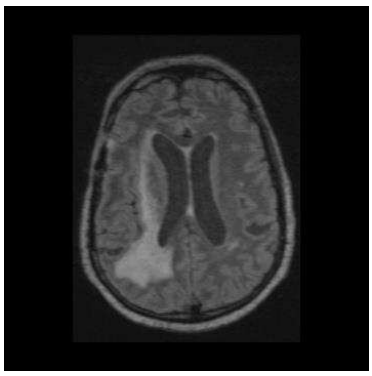

Fig -2: The original MRI Brain image.

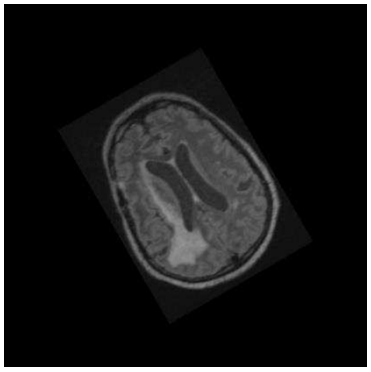

Fig -3: 30^0 Left rotated MRI Brain image.

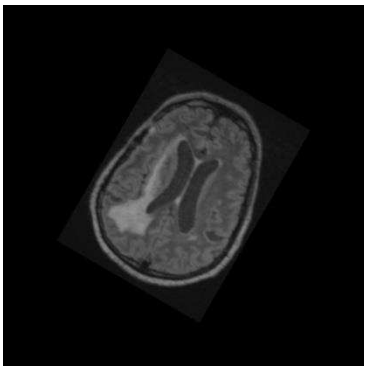

Fig -4: 30^0 Right rotated MRI Brain image.

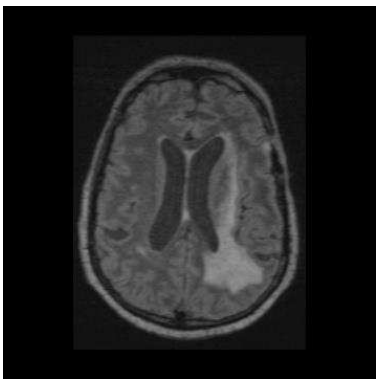

Fig -5: Vertical Flip of original MRI Brain image.

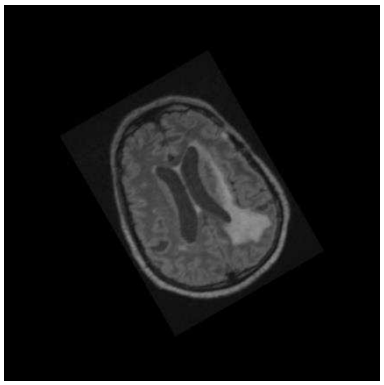

Fig -6: 30^0 left rotated Vertical flipped MRI Brain image.

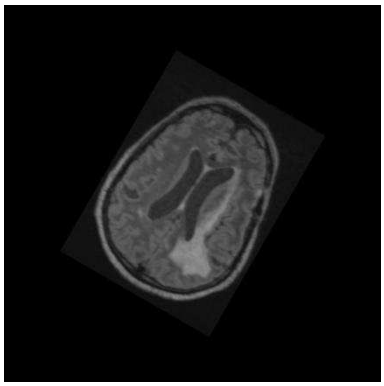

Fig -7: 30^0 right rotated Vertical flipped MRI Brain image.

2.1.2 Anisotropic Diffusion Filter

2.2 Model Creation

The convolutional neural network architecture used here is the most basic lenet architecture. This architecture contains 5 layers, out of which 2 layer are convolutional layer, the other 2 layers are pooling layer and last layer is fully connecting layer.

The TensorFlow layers module provides a high-level API that makes it easy to construct a neural network. It provides methods that facilitate the creation of dense (fully connected) layers and convolutional layers, adding activation functions, and applying dropout regularization.

2.2.1 Convolutional Neural Network:

Convolutional neural networks (CNNs) are the current state-of-the-art model architecture for image classification tasks. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification. CNNs contains three components:

2.2.1.1 Convolutional layers:

Convolutional layers, which apply a specified number of convolution filters to the image. For each sub-region, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a ReLU activation function to the output to introduce nonlinearities into the model.

2.2.1.2 Pooling layers:

Pooling layers, which down sample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts sub-regions of the feature map (e.g., 2x2-pixel tiles), keeps their maximum value, and discards all other values.

2.2.1.3 Dense (fully connected) layers:

Dense (fully connected) layers, which perform classification on the features extracted by the convolutional layers and down sampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

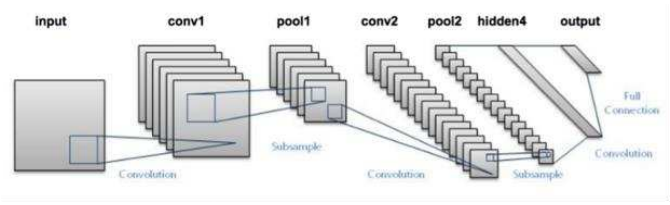

Fig -8: LeNet architecture

Typically, a CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification. The final dense layer in a CNN contains a single node for each target class in the model (all the possible classes the model may predict), with a softmax activation function to generate a value between 0–1 for each node (the sum of all these softmax values is equal to 1). We can interpret the softmax values for a given image as relative measurements of how likely it is that the image falls into each target class.

2.2.2 CNN Architecture:

The tf.layers module contains methods to create each of the three layer types above:

- **conv2d()** : Constructs a two-dimensional convolutional layer. Takes number of filters, filter kernel size, padding, and activation function as arguments.
- **max_pooling2d()** : Constructs a two-dimensional pooling layer using the max-pooling algorithm. Takes pooling filter size and stride as arguments.
- **dense()** : Constructs a dense layer. Takes number of neurons and activation function as arguments.

The CNN architecture graph made using above mentioned methods is shown in below figures. The network graph was very big so we cropped it into four parts as seen below.

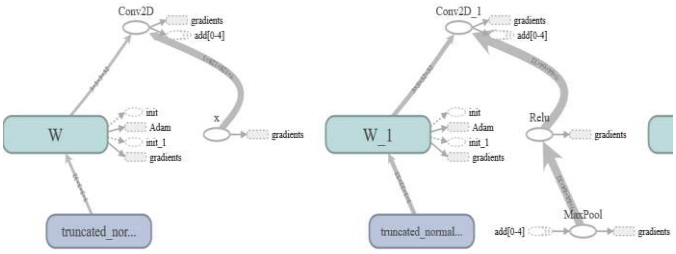

Fig -9: Network Graph Part1

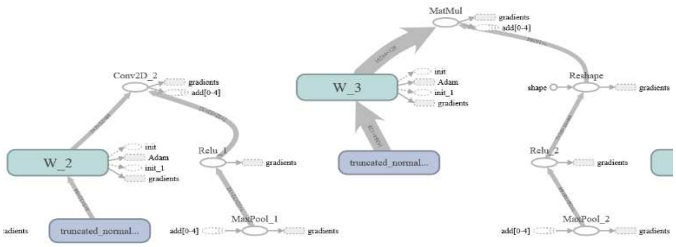

Fig -10: Network Graph Part2

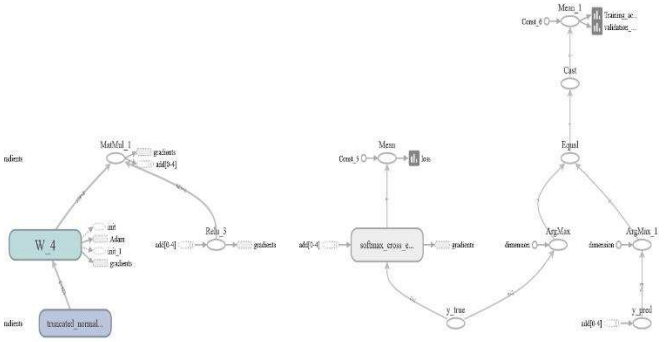

Fig -11: Network Graph Part3

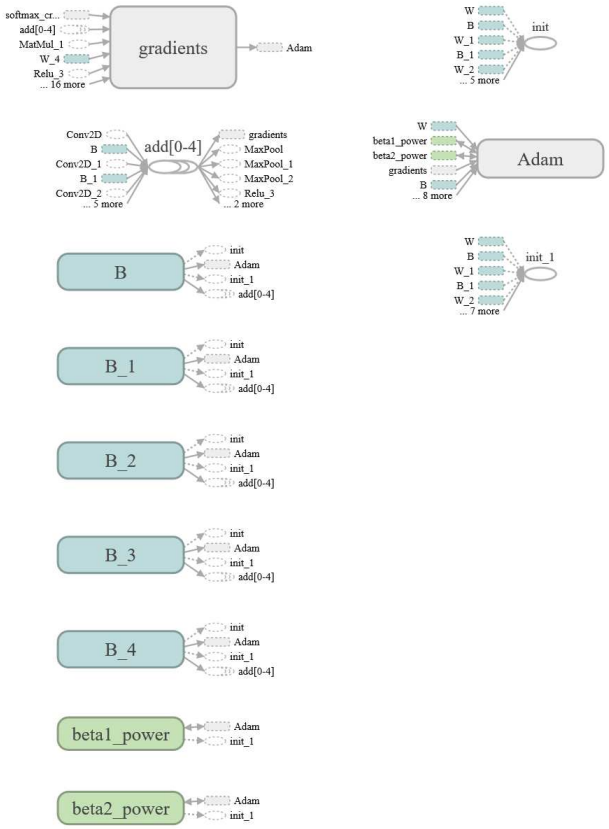

Fig -12: Network Graph Part4

The various layers are using weights and biases. The weights and biases are created using following code snippet.

```
def create_weights(shape):
    w = tf.Variable(tf.truncated_normal(shape, stddev=0.05),name="W")
    #Making a histogram of weights with name W
    tf.summary.histogram("weights",w)
    return w
def create_biases(size):
    b = tf.Variable(tf.constant(0.05, shape=[size]),name="B")
    #Making a histogram of biases with name B
    tf.summary.histogram("biases",b)
    return b
```

Fig -13: Weights and Biases creation

The filter size and number of filters in first two convolutional layers are 3 and 32 respectively. While the third convolutional layer had same number of filter size but the number of filters were doubled to 64. The fully connected layer size used is 128.

```
## Creating the convolutional layer
layer = tf.nn.conv2d(input=input,filter=weights,strides=[1, 1, 1, 1],padding='SAME')

layer += biases

## We shall be using max-pooling.
layer = tf.nn.max_pool(value=layer,ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1],padding='SAME')
## Output of pooling is fed to Relu which is the activation function for us.
layer = tf.nn.relu(layer)
```

Fig -14: Creation of Convolutional layer.

2.3 Epochs, Batch Size and Iterations:

One epoch is one forward pass and one backward pass of all the training examples through the network. Batch size is the number of training examples in one forward/backward pass. The higher the batch size, the more memory space is required. Number of iterations is the number of passes, each pass using (batch size) number of examples. To be clear, one pass is one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

In this case we had 1800 MRI in jpg/png format, so the memory requirement was above average around 8GB. The batch size is 60 and the number of iterations were 840 which resulted in 840 epochs.

2.4 Learning rate and Optimizer:

The learning rate decided here is α=0.0001 and this learning rate is utilized by Adam Optimizer [2].

2.4.1 Adam Optimizer:

Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

Parameters required (The numbers in bracket are the optimal values):

β1—This is used for decaying the running average of the gradient (0.9)

β2—This is used for decaying the running average of the square of gradient (0.999)

α—Step size parameter (0.001)

ε- It is to prevent Division from zero error. (10^-8)

Algorithm:

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Fig -14: ADAM optimizer algorithm

The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods.

3. RESULTS:

The 35 epochs resulted in very high Training accuracy and validation accuracy. All of the accuracies are recorded in tf.summary and plotted in tensorboard [1]. The graph curve show below were calculated on 840 iterations/steps and their corresponding accuracies.

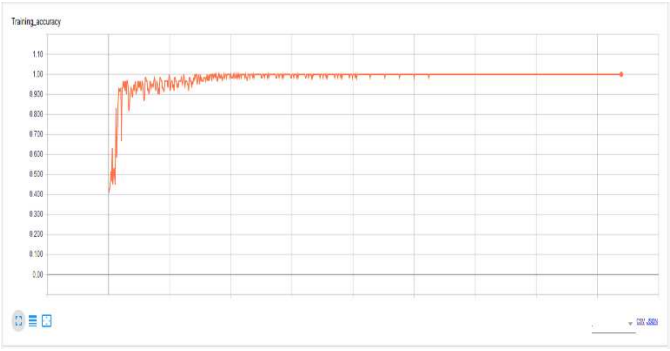

Fig -15: Training Accuracy

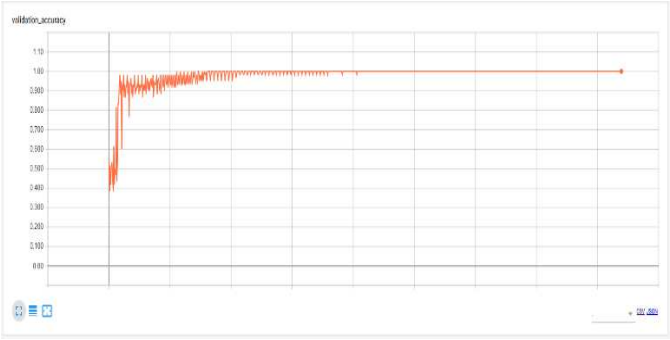

Fig -16: Validation Accuracy

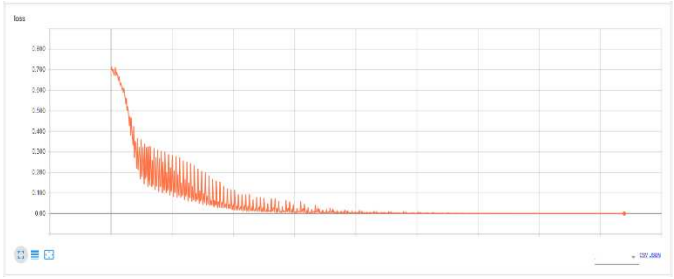

Fig -17: Validation Loss

The figure shows the validation loss over the iterations and it is quite uneven in the beginning but eventually as iterations go on the loss reaches 0.

4. CONCLUSION AND FUTURE WORK

Brain tumor detection is done using MRI and analyzing it. The machine learning is very powerful strategy for the detection of the cancer tumor from MRI. Here we achieved the training accuracy of 99% and validation accuracy of 98.6%, with validation loss from 0.704 to 0.000 over 35 epochs.

The method show in this paper is very basic image classification method of lenet architecture. The more powerful approaches are available. Our model was created on the CPU based tensorflow and GPU version of tensorflow is much faster to train, which will result in much faster model creation. The more sophisticated system should take MRI images in dicom format directly and operate on them.

REFERENCES

[1] Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man´e, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vi´egas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. “TensorFlow: Large-scale machine learning on heterogeneous systems”, 2015. Software available from tensorflow.org.

[2] Adam: A Method for Stochastic Optimization Diederik P. Kingma, Jimmy Ba arXiv:1412.6980v9 [cs.LG].

[3] Menze BH, Jakab A, Bauer S, Kalpathy-Cramer J, Farahani K, Kirby J, Burren Y, Porz N, Slotboom J, Wiest R, Lanczi L, Gerstner E, Weber MA, Arbel T, Avants BB, Ayache N, Buendia P, Collins DL, Cordier N, Corso JJ, Criminisi A, Das T, Delingette H, Demiralp Γ‡,

Durst CR, Dojat M, Doyle S, Festa J, Forbes F, Geremia E, Glocker B, Golland P, Guo X, Hamamci A, Iftekharuddin KM, Jena R, John NM, Konukoglu E, Lashkari D, Mariz JA, Meier R, Pereira S, Precup D, Price SJ, Raviv TR, Reza SM, Ryan M, Sarikaya D, Schwartz L, Shin HC, Shotton J, Silva CA, Sousa N, Subbanna NK, Szekely G, Taylor TJ, Thomas OM, Tustison NJ, Unal G, Vasseur F, Wintermark M, Ye DH, Zhao L, Zhao B, Zikic D, Prastawa M, Reyes M, Van Leemput K. "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)", IEEE Transactions on Medical Imaging 34(10), 1993-2024 (2015) DOI: 10.1109/TMI.2014.2377694

[4] Bakas S, Akbari H, Sotiras A, Bilello M, Rozycki M, Kirby JS, Freymann JB, Farahani K, Davatzikos C. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features", Nature Scientific Data, 4:170117 (2017) DOI: 10.1038/sdata.2017.117

[5] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Martin Rozycki, Justin Kirby, John Freymann, Keyvan Farahani, and Christos Davatzikos. (2017) Segmentation Labels and Radiomic Features for the Pre-operative Scans of the TCGA-GBM collection. The Cancer Imaging Archive. https://doi.org/10.7937/K9/TCIA.2017.KLXWJJ1Q

[6] Spyridon Bakas, Hamed Akbari, Aristeidis Sotiras, Michel Bilello, Michel Rozycki, Justin S Kirby, John B Freymann, Keyvan Farahani, Christos Davatzikos. "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features", Nature Scientific Data, 4:170117 doi: 10.1038/sdata.2017.117 (2017).

[7] Isselmou, A. , Zhang, S. and Xu, G. (2016) A Novel Approach for Brain Tumor Detection Using MRI Images. Journal of Biomedical Science and Engineering, 9, 44-52. doi: 10.4236/jbise.2016. 910B006.

[8] E. F. Badran, E. G. Mahmoud and N. Hamdy, "An algorithm for detecting brain tumors in MRI images," The 2010 International Conference on Computer Engineering & Systems, Cairo, 2010, pp. 368-373. doi: 10.1109/ICCES.2010.5674887

[9] Komal Sharma, Akwinder Kaur and Shruti Gujral. Article: Brain Tumor Detection based on Machine Learning Algorithms. International Journal of Computer Applications 103(1):7-11, October 2014.

[10] Athiwaratkun, Ben & Kang, Keegan. (2015). Feature Representation in Convolutional Neural Networks. .

[11] J. T. Kwak and S. M. Hewitt, "Nuclear Architecture Analysis of Prostate Cancer via Convolutional Neural Networks," in IEEE Access, vol. 5, pp. 18526-18533, 2017. doi: 10.1109/ACCESS.2017.2747838

[12] Mahmoud Al-Ayyoub, Ghaith Husari, Ahmad Alabed-alaziz and Omar Darwish, "Machine Learning Approach for Brain Tumor Detection", ICICS '12 Proceedings of the 3rd International Conference on Information and Communication Systems, Article-23, 2012-04-03. ISBN:978-1-4503-1327-8 doi>10.1145/ 2222444.2222467

[13] N. Subash and J. Rajeesh, "Brain Tumor Classification Using Machine Learning" In International Science Press, I J C T A, 8(5), 2015, pp. 2335-2341