BRAIN CANCER DETECTION USING MRI SCANS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Shanthanreddy Thotapally

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

December 2019

Fargo, North Dakota

North Dakota State University
Graduate School

Title

BRAIN CANCER DETECTION USING MRI SCANS

By

Shanthanreddy Thotapally

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Kendall E. Nygard
Chair

Dr. Vasant Ubhaya

Dr. Joel K. Ransom

Approved:

December 6, 2019
Date

Dr. Kendall Nygard
Department Chair

ABSTRACT

An estimate of about 700,000 Americans today live with a brain tumor. Of these, 70% are benign and 30% are malicious. The average survival rate of all the malicious brain tumor patients is 35%. Diagnosing these tumors early on gives the best chance for survival. The Doctors use MRI scans to identify the presence of a tumor and it's characteristics like the type and size. In this paper, I implemented a Deep learning convolutional neural network model that classifies the brain tumors using MRI scans. We shall use VGG-16 deep-learning approach to implement the machine learning algorithm. The proposed system can be divided into 3 parts: data input and preprocessing, building the VGG-16 model, image classification using the built model. Using this approach, I have achieved 80% accuracy. The accuracy of the model developed will depend on how correctly the affected brain tumor images can be classified from the unaffected.

Keywords: Brain Tumor, Magnetic Resonance Imaging (MRI), Data Preprocessing, Data Augmentation, VGG-16, Jupyter notebook, Machine Learning.

ACKNOWLEDGEMENTS

I would like to express my feeling of gratitude to Dr. Kendall Nygard, Department Chair, Computer Science, North Dakota State University, Fargo, for giving me the opportunity to pursue my research and develop this machine learning project. During my time here at NDSU, Dr. Nygard has generously offered his advice in almost everything related to my academics. He has set aside valuable time to be a member of my supervisory committee, and for this I am grateful. Students would benefit from more professors like him.

I am equally grateful to my supervisory committee members, Dr. Vasant Ubhaya, Senior Professor, Computer Science, NDSU, Fargo and Dr. Joel K. Ransom, Professor and Extension Agronomist, Plant Sciences, NDSU, Fargo for the support and their precious time.

Last but not least, I must extend gratitude to the remainder of the Computer Science staff for their advice and guidance throughout my Master's program.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

In this era of unprecedented changes, with the increase in ailments, the field of heath demands the use of new technology to provide solutions to these problems. The brain tumor is one such malady that is the most life-threatening disease known to mankind. Even if this disease is detected in the primal stage, it's not completely possible to find a cure within the time frame and save the patient's life. The Tumor can be classified into two types: Malignant (meaning the cells are cancerous and lethal to patient's life) and Benign (meaning the presence of tumor won't affect the health of the patient in any manner). It is commonly accepted that the most important part of the human body is the human brain, therefore, if anything shall happen to it, it will directly impact the life expectancy of the patient. The development of a prediction model for the diagnose of brain tumors would greatly benefit the medical community. In this project, I will build a convolutional neural network model that classifies an image set of MRI scans based on the presence of tumors by comparing it with the previous scans of a brain tumor. With this project, I seek to help and devise a solution that could be useful to detect this lethal disease. Furthermore, this model may be trained iteratively on larger and larger datasets, using a process called Ensemble Learning.

CHAPTER 2. PROBLEM DEFINITION AND ALGORITHM

A Brain tumor is a collection or growth of a mass of abnormal cells within the brain. The skull that covers the brain is very strong. Within such a restricted space, any growth may cause problems. Two main types of tumors occur namely, Cancerous (malignant) or non-cancerous (benign) tumors in the brain. The pressure inside your skull may increase when benign or malignant tumors grow. The growth time of the tumors can differ greatly. The positioning, growth rate and the size of the tumor determine how the nervous system will be affected in the body [1].

2.1. Brain Tumor

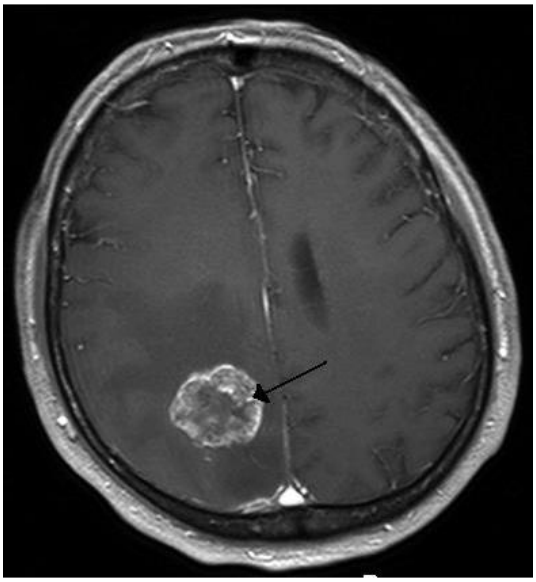


Figure 1. Brain metastasis from lung cancer [3]

Usually, brain tumors are classified as primary or secondary. A primary tumor is the type of tumor, that originates in the brain itself. Many of the primary tumors aren't that harmful and are mild that can be treated. A secondary brain tumor that is also known as a metastatic brain tumor occurs first in other organs like lungs or breast and then it spreads(metastasizes) to your brain cells. The symptoms vary depending on the type of brain tumor and the part involved. Few

include vomiting, blurred vision, change in mental functioning, memory loss, muscle weakness, etc. [2]

2.2. VGG-16 Model

A convolution neural network is a deep learning algorithm that takes in an input image, assigns characteristics to different aspects of the image and be able to classify that image. The pre-processing required in the Convolution network is much lower as compared with other classification problems. VGG-16 is one such type of convolution neural network. It is 16 layers deep making the model heavier that could increase the training time i.e. the input images are processed via 16 layers. In this model, the output is classified into different categories. For example, suppose a VGG-16 network takes input images of 100 different objects, the network then classifies the images based on the requirements such as presence/absence of a tumor, classify objects in photographs, etc. This model was proposed by K Simonyan and A Zisserman and achieved 92.7% accuracy in the top 5 test accuracy in ImageNet Database [10]. For the brain cancer detection project, the image set of MRI scans is taken into consideration and it is classifying different images based on whether the tumor is present or not.

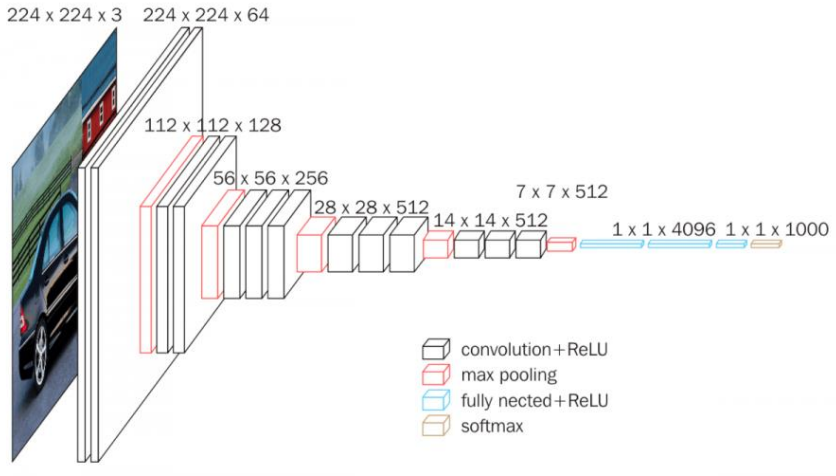


Figure 2. VGG-16 model [10]

2.2.1. Layers and Steps Involved in VGG-16

VGG_16 model has 16 layers. In each and every layer, the input image is processed through various steps. Let's see what these steps do in detail.

1) **Input the image**: First, the input image of size 224 by 224 pixels is provided to the model.
2) **Convolution:** Every layer has a 3x3 filter/kernel. In this step, the filter is convolved with a stride of pixels 2 across the input image and the feature map is obtained. The feature map is usually the dot product of the input pixel values and the filter. The feature map obtained is passed as an input to the next layer.
3) **Max pooling:** The pixel density is increased in this step, reducing the dimension of the feature map. Max pooling is nothing but taking the maximum value from a 3 x 3 filter with a stride of 2 pixels. The core features are extracted through max pooling.
4) **Normalization:** The negative pixel values are converted to 0's, this is done by taking the $\max(x_{ij},0)$ function. This activation function is called Rectified Linear Unit (ReLU). Through this step, all the irrelevant pixels will be disabled.
5) **Dropout:** In the dropout step, randomly selected layers are labeled as zeroes, so that the network learns new pathways to represent the final image. This helps to improve the learning. The softmax function is usually applied in the very last step in the neural network. Softmax converts the output produced in the last layer into a probability distribution function. This is done based on features extracted in the middle layers of the VGG_16 network. The predictions made are based on the results produced by this function.

2.3. Objective

The objective is to develop a machine learning algorithm that identifies the parameters that help in detecting the brain tumor and be able to classify the image as having a tumor or not. This algorithm is developed by building the VGG_16 model based on MRI scans that are provided as the data sets. After looking at the results, the Doctor can easily identify the tumors as being malignant or benign. This is achieved using a Convolutional Neural Network model (CNN), an approach that specifically helps with the image classification problems. There are multiple CNN models out of those I chose VGG_16 as this is the most effective and has a standard training process especially in case of image classification problems. I have used the Jupyter notebook application to implement the model. Jupyter notebook is an open-source web application that can be used for deep learning, data visualization, numerical and statistical simulation and much more.

CHAPTER 3. EXPERIMENTAL EVALUATION

As far as health care is concerned, automated Brain cancer detection techniques are found to be highly important as higher accuracy is needed in safety-critical processes. Automated Brain cancer detection involves the collection of the MRI scans and the preprocessing of the data.

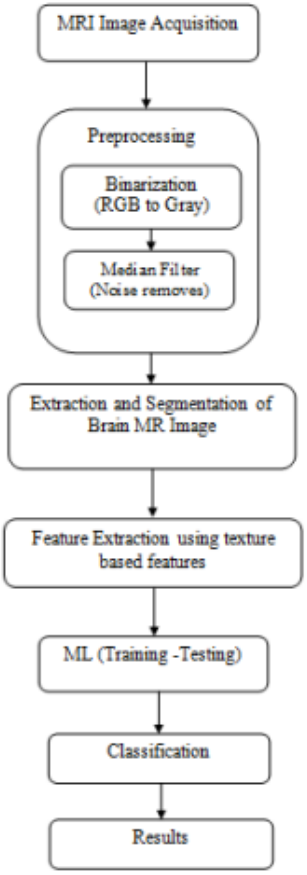


Figure 3. The proposed method for brain cancer detection from MRI scans [4]

First, any RGB images are first converted into grayscale. Since the dataset is small, the process of Data Augmentation is used, which helps to increase the size of the training set. Data preprocessing is explained more clearly in further sections. Next, a CNN model is to be built. In our case, we used the VGG_16 model. The dataset is then used to train the built model. After the model training i.e. running the model through many iterations called epochs, the test dataset is

provided as an input to the trained algorithm and the accuracy of the results produced are noted. In my case, I have produced results that are close to 80% accuracy.

3.1. Setting Up the Environment

The environment required for the implementation has to be setup before building the model. This includes importing and installing all the necessary packages and libraries.

```
from IPython.display import clear_output
!pip install imutils
clear_output()
```

Figure 4. Sample screenshot of package installs

I have used TensorFlow backend to build, train and run the model. TensorFlow is developed by the Google Brain team. Tensorflow is an open-source machine learning library that provides API, used in applications of neural networks and production of deep learning models. As per the Tensorflow website, companies like Airbnb, Uber, Snapchat, Dropbox use Tensorflow as their software library [11].

Keras is another high-level neural network library, that I used to run on the Tensorflow backend. Keras is considered as easy to learn and use. Keras helps in fast and easy prototyping. It runs well on the Central Processing Unit (CPU) as well as the Graphics Processing Unit (GPU) [7]. There are 2 types of models in Keras i.e. Sequential and Model. The core data structure used in Keras is “Model”. The Sequential model is a linear stack of layers, that can be described very simply. I also used Keras optimization algorithms like Adaptive Moment Optimization (Adam) and Root Mean square Propagation (RMRprop) [8].

The reason for using the Keras libraries instead of TensorFlow libraries is that Keras libraries are written in python language. This makes the implementation easier because the whole project was developed using python code as this is the most popular language for the deep

learning projects. Even though one can find the libraries written in python in Tensorflow, most of the Tensorflow libraries are written in C++ and CUDA.

```
import numpy as np
from tqdm import tqdm
import cv2
import os
import shutil
import itertools
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import layers
from keras.models import Model, Sequential
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping

init_notebook_mode(connected=True)
RANDOM_SEED = 123
```

Figure 5. Screenshot of importing relevant packages and libraries

First, we import all the packages and libraries needed to set up the environment to get started. The mainly used libraries are TensorFlow, Scikit-Learn, Numpy, and Keras. These libraries include packages like cv2, os, shutil, itertools, imutils, sklearn label binarizer, train_test_split, sklearn metrics, Keras preprocessing, Keras models, Keras optimizers, plotly, etc. The packages perform various operations including mathematical operations, image processing functions like resizing, rotating, displaying, cropping; automatic file copying and directories, efficient looping iterators, handling iterators, splitting the dataset into the train,

validate and test subfolders, building confusion matrix and accuracy scores, data visualization toolbox, etc.

3.2. Data Description and Preprocessing

The image dataset used in the project is readily available on the internet. Basically, machine learning projects are open-source projects, so anyone can download the dataset and build classification models. Unfortunately, there is no information in the dataset description where these MRI images came from and so on.

3.2.1. Data Description

Initially, all the images are in one folder with “YES” and “NO” subfolders. Now, the data needs to be spread into Train, Validate and Test folders.

Figure 6. Screenshot of new folder hierarchy

The train dataset is the dataset that is used to train the model while building.

The validation dataset is the dataset that is used during the training of the model in order to estimate how well the model has been trained. This set of data is helpful in tuning the hyperparameters. Examples of parameters are number of hidden units in a neural network, weights in the neural network, learning rate of the model, support vectors, coefficients in the functions, etc. The parameters are tuned to improve the efficiency of the model by considering the validation accuracy produced at the end of every epoch.

The test dataset is the dataset that remains untouched for the whole training process. This dataset will be used to evaluate model performance.

80% of the data is split into the Train data that are 193 images from that 74 does not have a tumor and 119 has the tumor present. For the test data, I have split 5 in each subfolder i.e. yes/no. And the remaining data is split into the validate data i.e. 50 images from that 20 are “no” and 30 are “yes”. The histogram below shows the count of classes in each of the 3 datasets.

```
IMG_PATH = r"C:\Users\shant\Desktop\brain_tumor_dataset\\"
# split the data by train/val/test
for CLASS in os.listdir(IMG_PATH):

    if not CLASS.startswith('.'):
        IMG_NUM = len(os.listdir(IMG_PATH  + CLASS))
        for (n, FILE_NAME) in enumerate(os.listdir(IMG_PATH + CLASS)):
            img = IMG_PATH + CLASS + '\\' + FILE_NAME
            if n < 5:
                shutil.copy(img, 'TEST\\' + CLASS.upper() + '\\' + FILE_NAME)
            elif n < 0.8*IMG_NUM:
                shutil.copy(img, 'TRAIN\\'+ CLASS.upper() + '\\' + FILE_NAME)
            else:
                shutil.copy(img, 'VAL\\'+ CLASS.upper() + '\\' + FILE_NAME)
```

Figure 7. Screenshot of code distributing the images into 3 datasets

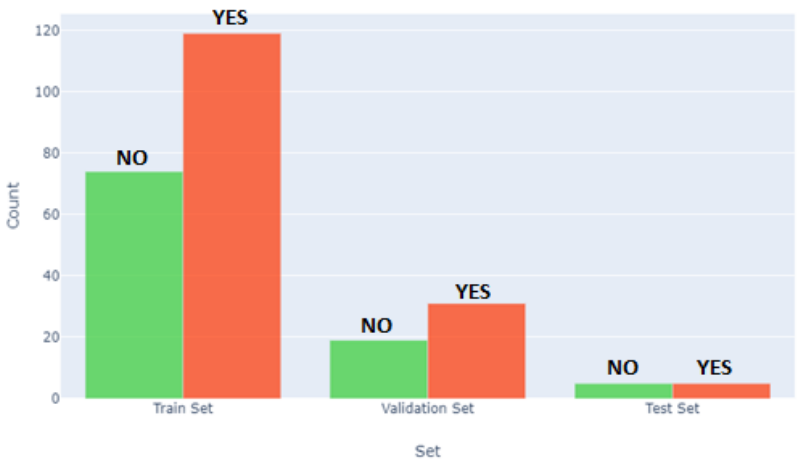


Figure 8. Count of classes in each data set

So, the data has been successfully divided into 3 datasets. Now, the next step is to plot a few of the samples from a specific dataset (e.g. Train dataset) to make sure the data is loaded

correctly into the workspace. By doing this, the dataset will be verified and we can proceed to the next process.

Here, out of 193 images from the train dataset, we plot 30 from each subfolder (Yes/ No). However, these images are not yet ready to be provided as input to the VGG-16 model. These are yet to be preprocessed through various steps.

3.2.2. Samples of Train Dataset

The images from the figures 9 & 10, are the samples of the brain MRI scans. The first set of images does not have any tumor and are uninfected. The second set of images is affected by a tumor that can be seen as a round lump inside the brain.

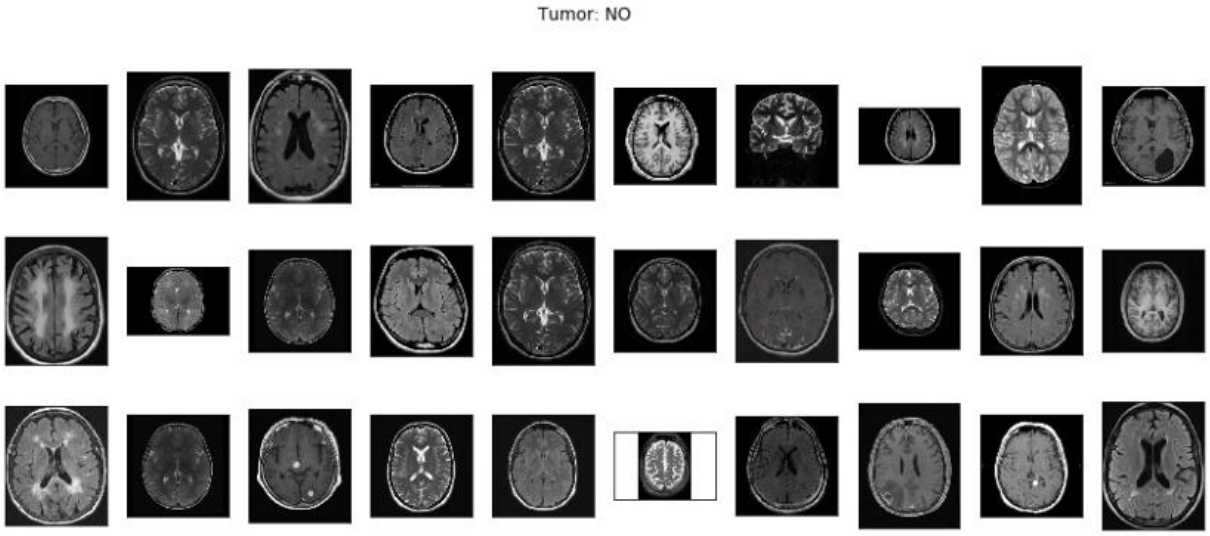


Figure 9. MRI scans without the tumor [3]

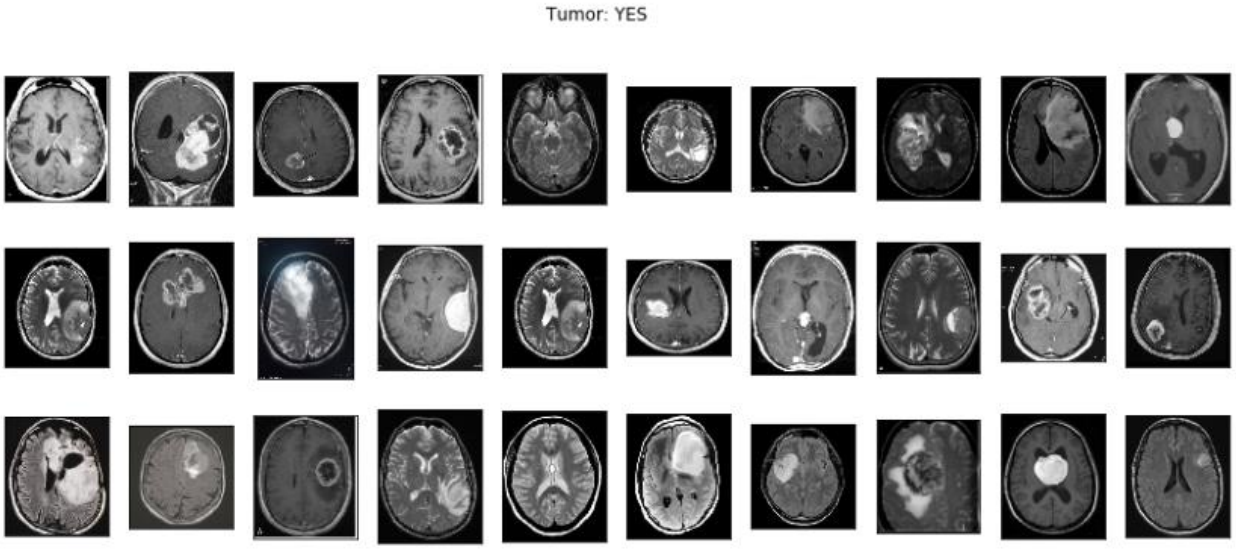


Figure 10. MRI scans with the tumor [3]

3.2.3. Data Preprocessing

The data has to be preprocessed before the model is built with the obtained dataset.

3.2.3.1. Image Ratio

The image size of the input layer of the VGG_16 model is (224, 224). But we can see from the figures 9 & 10, that the image sizes differ in terms of height and width including the difference in the size of black corners. Unfortunately, some images that are wide may look weird after resizing. Let's look at the ratio distributions (`ratio = width/height`).

```
RATIO_LIST = []
for set in (X_train, X_test, X_val):
    for img in set:
        RATIO_LIST.append(img.shape[1]/img.shape[0])

plt.hist(RATIO_LIST)
plt.title('Distribution of Image Ratios')
plt.xlabel('Ratio Value')
plt.ylabel('Count')
plt.show()
```

Figure 11. Screenshot of code for plotting the histogram

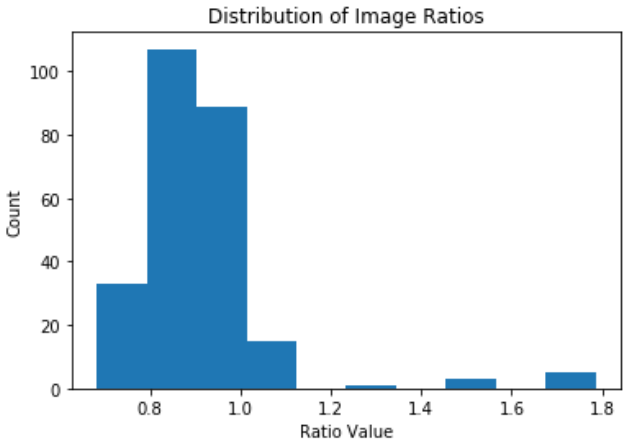
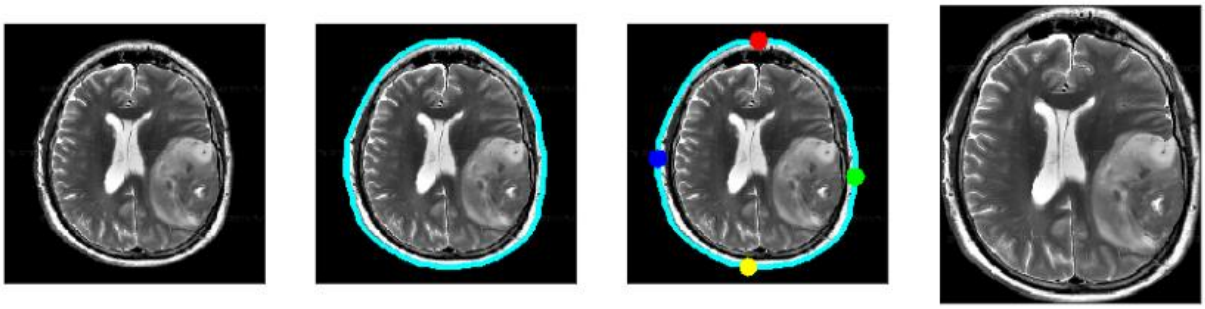


Figure 12. Histogram of image distributions

From the Figure 12, we can observe that the size of the images varies widely. Most of the images fall in the ratio value between the interval (0.7 - 1.1), which denotes that the images have more height. However, a few fall into the ratio values of (1.2 – 1.8), meaning the images are wider. The VGG_16 model only takes 224 by 224-pixel inputs. Thus, the data needs to be further preprocessed and that includes image cropping before providing it as an input to the model.

3.2.3.2. Cropping the MRI Scans

In order to normalize the data, first, the images need to be cropped. For this, I have used the technique mentioned in the "pyimagesearch" blog [5]. In general, it detects the relevant region from the image. Next, it finds the extreme North, East, West, and South (x, y) coordinates along the contour as described below.



Step 1 **Step 2** **Step 3** **Step 4**

Figure 13. Cropping of MRI scans

Step 1. First, we get the original image from the dataset.

Step 2. Next, we find the biggest contour of the brain.

Step 3. Third, we plot the extreme points on all four edges of the contour. Remember the contour is simply a Numpy array of (x, y) – coordinates.

```
Finding extreme points in contours with OpenCV                    Python
23 # determine the most extreme points along the contour
24 extLeft = tuple(c[c[:, :, 0].argmin()][0])
25 extRight = tuple(c[c[:, :, 0].argmax()][0])
26 extTop = tuple(c[c[:, :, 1].argmin()][0])
27 extBot = tuple(c[c[:, :, 1].argmax()][0])
```

Figure 14. Code for finding the extreme points along contour [5]

Step 4. Lastly, we crop the image based on those extreme points [5].

Next, the cropping must be applied to all the images present in each and every dataset. This makes the dataset ready that can be used to train and build the model.

3.3. Building the CNN Model

As we have cropped the images, these are now ready to be fed to the model. But before building the model, I want to increase the numer of data samples by using Data Augmentation because the obtained dataset is too small.

3.3.1. Data Augmentation

In order to make the most out of few data samples, we will augment the images through a number of random transformations like rotating through different angles, flipping, resizing, zooming in/out, etc. By doing this, the model will not see the exact same picture twice. This can be done by setting the parameters via an "ImageDataGenerator" class in Keras [9].

Next, apply this technique to each and every image in the train dataset. Data Augmentation helps to increase the number of images and produce a final dataset that we use to train the model.

```
Found 193 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
```

Figure 15. Screenshot showing the number of images to be augmented

3.3.2. Images of Data Augmentation Performed on a Sample Image (Demo)

This is a sample image which has a tumor. Let's perform Data augmentation on it to see what the technique produces.

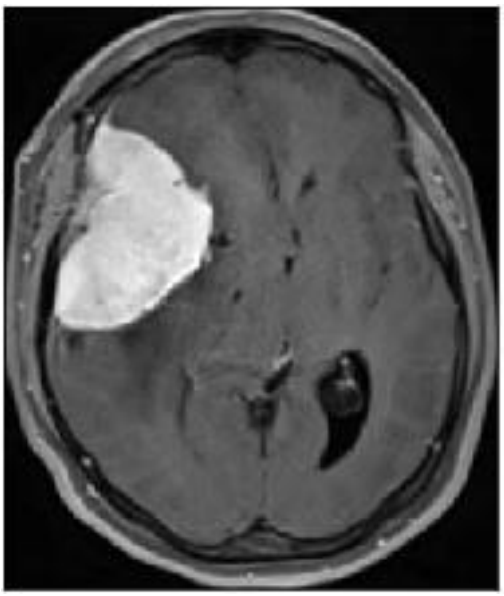


Figure 16. Sample original image

After augmenting the sample image, we obtain multiple augmented images that are shown in the figure 17.

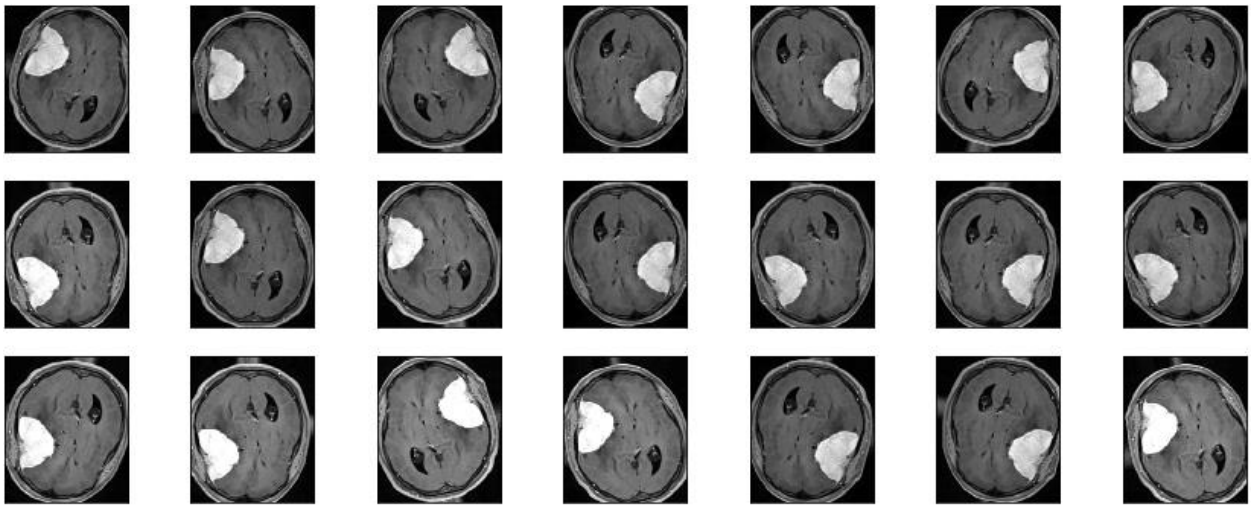


Figure 17. Augmented images

3.3.3. Model Building and Training

First, we load the pre-installed VGG_16 Convolutional neural network model package. This model is primarily used for image classification using deep learning. We can download the VGG-16 pre-trained package online. Once the model is downloaded, it has to be imported into the algorithm by specifying the path.

```
In [136]: # load base model
vgg16_weight_path = 'C:/Users/shant/Desktop/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
base_model = VGG16(
    weights=vgg16_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)
```

Figure 18. Screenshot of code showing the loading of the core VGG_16 model

Next, after importing the VGG_16 model, it has to be trained through multiple iterations (called Epochs in ML). Epochs are nothing but the rounds of optimization applied during the model training. Thus, through more rounds of optimizations i.e. epochs, the error in the training data will be highly reduced. However, there comes a point where the model might start to lose

performance due to over-fitting of the train dataset. If the validation error starts increasing, this might be a sign of overfitting. It is recommended to set as many epochs as possible and terminate the training by looking at the error rates.

An epoch usually consists of one full processing through the entire training dataset. It is basically a full iteration over the provided samples. This happens in the form of multiple steps. This can be best illustrated through an example. Let us suppose we have 1000 images in the train dataset and a batch size of 10 is used. Then every single epoch consists of 1000 images i.e. 10 images (batch size) in each step, accounting for (1000/10) steps in total or 100 steps.

```
In [138]: EPOCHS = 30
es = EarlyStopping(
    monitor='val_acc',
    mode='max',
    patience=6
)

history = model.fit_generator(
    train_generator,
    steps_per_epoch=50,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=25,
    callbacks=[es]
)
```

Figure 19. Screenshot showing the initiation of epochs

For this model, I standardized the number of epochs to 30 because I have observed that the learning rate degrades if the model is fed with too many iterations. I have also noticed a gradual dropping of the validation accuracy of the model during the 25th and 26th epochs, which is not at all a good sign. Furthermore, the model loses its effectiveness because of the over-fitting of the data. Due to this, the learning rate of the model decreases which results in poor accuracy. Therefore, it is always important to keep an eye on the accuracy of the results produced and the model loss after each epoch. We obtain the metrics like the training loss, the training accuracy,

the validation loss and the validation accuracy once the model has completed each epoch. In the end, we plot the results for all these metrics and draw a conclusion based on the performance graphs.

Figure 20 shows the results of every epoch in detail. Also, the estimated time taken for running is shown for each step in every epoch. Overall, my model did take nearly 5 hours to complete all the epochs and finally become trained. After making the conclusion that the model is built and ready, it can be tested on the test data to see the effectiveness of the model. This can be done by going through the prediction scores and the test accuracy.

```
Epoch 1/30
50/50 [==============================] - 408s 8s/step - loss: 4.3261 - accuracy: 0.6197 - val_loss: 1.8026 - val_accuracy: 0.6994
Epoch 2/30
50/50 [==============================] - 387s 8s/step - loss: 2.7577 - accuracy: 0.7057 - val_loss: 0.4047 - val_accuracy: 0.8038
Epoch 3/30
50/50 [==============================] - 387s 8s/step - loss: 2.2432 - accuracy: 0.7578 - val_loss: 0.9841 - val_accuracy: 0.8228
Epoch 4/30
50/50 [==============================] - 416s 8s/step - loss: 1.9812 - accuracy: 0.7788 - val_loss: 6.9697e-04 - val_accuracy: 0.7781
Epoch 5/30
50/50 [==============================] - 428s 9s/step - loss: 2.1627 - accuracy: 0.8127 - val_loss: 1.8352 - val_accuracy: 0.8386
Epoch 6/30
50/50 [==============================] - 419s 8s/step - loss: 1.9497 - accuracy: 0.8192 - val_loss: 0.5534 - val_accuracy: 0.8671
Epoch 7/30
50/50 [==============================] - 417s 8s/step - loss: 1.7216 - accuracy: 0.8366 - val_loss: 1.0879 - val_accuracy: 0.8196
Epoch 8/30
50/50 [==============================] - 413s 8s/step - loss: 1.5372 - accuracy: 0.8402 - val_loss: 1.1479e-05 - val_accuracy: 0.8609
Epoch 9/30
50/50 [==============================] - 412s 8s/step - loss: 1.0980 - accuracy: 0.8565 - val_loss: 0.8201 - val_accuracy: 0.8228
Epoch 10/30
50/50 [==============================] - 422s 8s/step - loss: 1.1407 - accuracy: 0.8554 - val_loss: 1.5184e-04 - val_accuracy: 0.8639
Epoch 11/30
50/50 [==============================] - 430s 9s/step - loss: 0.9201 - accuracy: 0.8791 - val_loss: 0.4948 - val_accuracy: 0.8513
Epoch 12/30
50/50 [==============================] - 420s 8s/step - loss: 1.2168 - accuracy: 0.8836 - val_loss: 0.0023 - val_accuracy: 0.8411
Epoch 13/30
50/50 [==============================] - 417s 8s/step - loss: 0.8111 - accuracy: 0.8913 - val_loss: 0.1366 - val_accuracy: 0.8259
Epoch 14/30
50/50 [==============================] - 418s 8s/step - loss: 0.9511 - accuracy: 0.8901 - val_loss: 0.0063 - val_accuracy: 0.8639
Epoch 15/30
50/50 [==============================] - 409s 8s/step - loss: 0.6694 - accuracy: 0.9075 - val_loss: 0.6717 - val_accuracy: 0.8291
.
.
.
.
Epoch 23/30
50/50 [==============================] - 406s 8s/step - loss: 0.3809 - accuracy: 0.9334 - val_loss: 0.7338 - val_accuracy: 0.8766
Epoch 24/30
50/50 [==============================] - 420s 8s/step - loss: 0.4033 - accuracy: 0.9385 - val_loss: 5.3089 - val_accuracy: 0.8411
Epoch 25/30
50/50 [==============================] - 422s 8s/step - loss: 0.4099 - accuracy: 0.9400 - val_loss: 0.6751 - val_accuracy: 0.8639
Epoch 26/30
50/50 [==============================] - 390s 8s/step - loss: 0.3138 - accuracy: 0.9422 - val_loss: 1.3493 - val_accuracy: 0.8513
Epoch 27/30
50/50 [==============================] - 388s 8s/step - loss: 0.3025 - accuracy: 0.9450 - val_loss: 0.7600 - val_accuracy: 0.8671
Epoch 28/30
50/50 [==============================] - 378s 8s/step - loss: 0.2506 - accuracy: 0.9475 - val_loss: 2.5771e-06 - val_accuracy: 0.8377
Epoch 29/30
50/50 [==============================] - 381s 8s/step - loss: 0.3181 - accuracy: 0.9527 - val_loss: 0.5917 - val_accuracy: 0.8449
Epoch 30/30
50/50 [==============================] - 390s 8s/step - loss: 0.3304 - accuracy: 0.9470 - val_loss: 0.8468 - val_accuracy: 0.8038
```

Figure 20. Screenshot showing the results of epochs

At the end of each epoch, the validation accuracy increases because the model runs through the same process every single time and the learning rate gradually increases. After running through 30 epochs, the model is built and ready to classify the images present in the test dataset.

However, by running the model through too many epochs, the learning rate gradually decreases because of the over-fitting of the data. The result of this over-fitting can be seen as the spikes in the Figure 22 at the epochs 16 and 24.

3.4. Plotting the Performance and Results

3.4.1. Model Performance

Two important metrics define the performance of a model i.e. Model Accuracy and Model Loss.

Model Accuracy: This is the primary metric used to evaluate classification models.

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- Accuracy gets better while Loss gets worse and vice-versa.

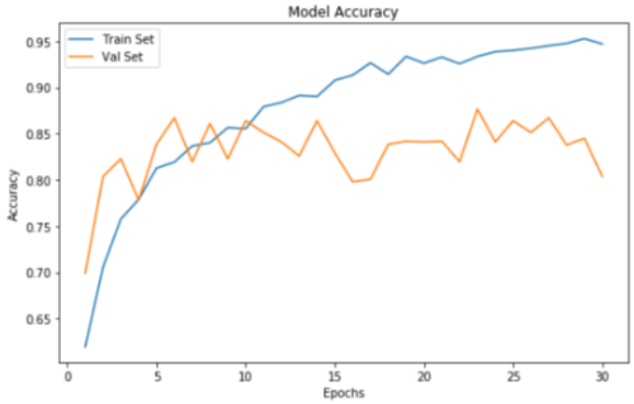


Figure 21. Model Accuracy

Model Loss: A machine learning algorithm can be optimized by using a Loss function. In fact, the machine learns through this loss function. If the predictions deviate too much from the actual results, the loss function will produce a very high number.

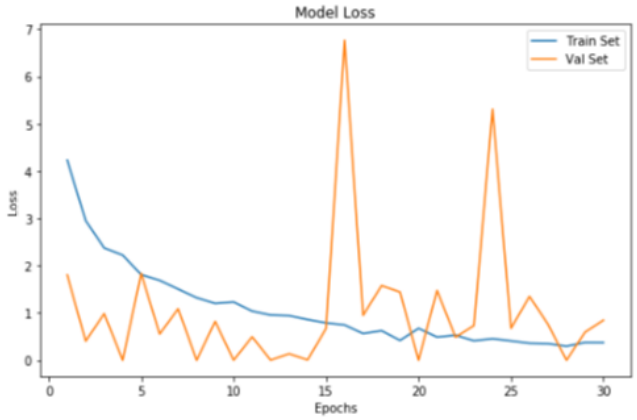


Figure 22. Model Loss

3.4.2. Accuracy When Tested on the Validate Dataset

During the process of model training i.e. running through 30 epochs, a validate dataset is used to train the model to tune the hyperparameters. After the model is built, the same validate dataset is provided as a test dataset to test the model's performance. The result produced is known as Validate Accuracy. I have achieved 88% accuracy on the validate dataset.

```
# validate on val set
predictions = model.predict(X_val_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_val, predictions)
print('Val Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_val, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
```

```
Val Accuracy = 0.88
```

Figure 23. Screenshot showing Accuracy on Validate dataset

3.4.3. Accuracy When Tested on the Test Dataset

I have achieved 100% accuracy on the test dataset and the implementation was successful. The results are highly satisfactory because the provided test dataset is small with 5

images in each i.e. yes/no. I have produced results by changing the test datasets multiple times and in most cases, I have achieved an average of 84% test accuracy. Therefore, I am concluding that I have achieved an acceptable prediction score. However, the confidence in the test accuracy can be gained when the model is provided with large test datasets.

```
# validate on test set
predictions = model.predict(X_test_prep)
predictions = [1 if x>0.5 else 0 for x in predictions]

accuracy = accuracy_score(y_test, predictions)
print('Test Accuracy = %.2f' % accuracy)

confusion_mtx = confusion_matrix(y_test, predictions)
cm = plot_confusion_matrix(confusion_mtx, classes = list(labels.items()), normalize=False)
```

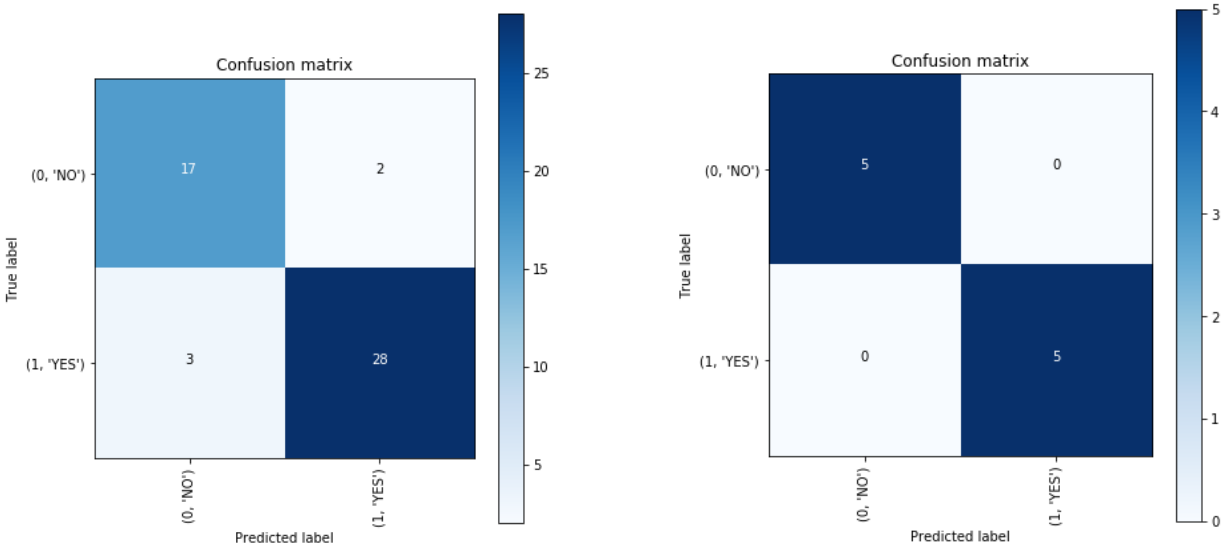
```
Test Accuracy = 1.00
```

Figure 24. Screenshot showing Accuracy on the Test dataset

3.4.4. Confusion Matrices

Finally, we plot the confusion matrix that helps to visualize the performance of the implemented algorithm. This is a table that usually describes the performance of the VGG_16 classification model.

From the validate set, we can observe that out of the 20 images that do not have a tumor, 17 are predicted correctly while 3 are classified as the presence of a tumor. Similarly, out of the 30 images that contain a tumor, 28 are predicted correctly while 2 are predicted as the absence of a tumor. This affects the accuracy of the prediction.



Validate set **Test set**

Figure 25. Confusion matrix

I have produced the test results by providing various sets of test data. I am able to achieve 100% accuracy once. Therefore, the average accuracy score is approximately 83%. From the figure 24, out of 10 images, all 5 were correctly predicted as infected and the remaining images were correctly predicted as not infected.

CHAPTER 4. CONCLUSION

In conclusion, this project proposes a work on brain cancer detection using MRI scans based on a machine learning algorithm with the combination of computer vision problems, that helps to automate the process of cropping the brain from MRI scans. This dataset of MRI scans consists of (i) different types of tumors having different sizes as well as positions and (ii) unaffected brain scans for training purposes. The CNN model used for this project is VGG_16. The existing works use the other CNN models like InceptionV3, ResNet50, FastAI, etc. Some even used the VGG_16 model but the results are not satisfactory because the dataset available is too small. The model I built is more like a prototype because it was trained using 243 samples of brain MRI scans. As the dataset is too small, I implemented a technique called Data Augmentation. This technique did generate approximately 7,776 (243 x 32) augmented images from the 243 images. Now the VGG_16 model has to be trained with these augmented images. After training the model, it is ready to be tested with the test dataset. I have achieved 84% Test Accuracy and 88% Validation Accuracy. The VGG_16 model is the most efficient and best known for its prediction accuracy. However, the predictions can be improved by implementing Ensemble Learning. This is going to be my future work.

CHAPTER 5. FUTURE WORK

This algorithm was built using one of the effective CNN models, VGG_16. This model is 16 layers deep and best suited for image classification problems. Different models (like Inception_V3, Resnet50, fastAI, etc.) produce different accuracy scores. We can also combine multiple models together to build a meta-model for better performance. A combination of different classifier models to produce a mega algorithm and training this built meta-model is a whole other level of work. This process is known as **Ensemble Learning**. "Stacking" is the most common Ensemble learning technique that combines multiple classification models via a meta-classifier. The base-level models use a complete training set to get trained and the outputs produced by these base models can be used to train the meta-model [6]. The prediction levels are much more accurate with the use of Ensemble Learning technique. However, the implementation of this meta-model requires a high-performance Graphics Processing Unit (GPU) in addition to CPU that demands high computational power, cost, and time. My future work lies in the implementation of Brain cancer detection using Ensemble learning.

REFERENCES

[1] Mayo Clinic. “Brain tumor.” Retrieved September 2019, from https://www.mayoclinic.org/diseases-conditions/brain-tumor/symptoms-causes/syc-20350084

[2] Healthline. “Different types of brain tumors” Retrieved September 2019, from: https://www.healthline.com/health/brain-tumor

[3] Chakrabarty, N. “Dataset for Brain tumor MRI images” Retrieved October 2019, from https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection

[4] Sharma, K., Kaur, A., Gujral, S., (October 2014). “Brain Tumor Detection based on Machine Learning Algorithms”, in *International Journal of Computer Applications* (0975 – 8887) Vol. 103, Issue 1.

[5] Rosebrock, A (2016, April 11). “Finding extreme points in contours with OpenCV” Retrieved October 2019, from https://www.pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/

[6] Smolyakov, V. (2017, August 22). “Ensemble Learning to Improve Machine Learning Results” Retrieved November 2019, from https://blog.statsbot.co/ensemble-learning-d1dcd548e936

[7] Keras Documentation. “Keras: The Python Deep Learning library” Retrieved October 2019, from https://keras.io/

[8] Heller, M. (2019, January 28). “What is Keras? The deep neural network API explained” *Infoworld*. Retrieved November 2019, from https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html

[9] The Keras Blog (2016, June 5). “Building powerful image classification models using very little data” Retrieved October 2019, from https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[10] Hassan, M.U. (2018, November 20). “VGG16 – Convolutional Network for Classification and Detection” Retrieved October 2019, from https://neurohive.io/en/popular-networks/vgg16/

[11] Wadhai, V.M., Chancalani, A., Kachwalla, M., Shinde, P., Katare, R., Agarwal, A. (April 2018). “Classification of Brain MRI images for Cancer Detection using Deep Learning”, in *International Journal of Advanced Research in Computer and Communication Engineering*, ISO 3297:2007 Certified, Vol. 7, Issue 4.