Contents

Abstract

Over the years, Artificial Intelligence has revolutionized Healthcare in many areas such as :-

a) Disease Diagnosis

b) Surgical Robots

c) Maximizing Hospital Efficiency

AI Healthcare market is expected to reach around $45.2 billion USD by 2026 from the current valuation of $4.9 billion USD. Deep learning has proven to be superior in the detection of diseases from X-rays, MRI (Magnetic Resonance Imaging) and CT (Computed Tomography) scans which improves the speed and accuracy of diagnosis.

Brain Tumor is an abnormal mass of tissue in which cells grow and multiply uncontrollably and apparently unregulated by mechanisms that control cells. Without AI, Tumor classification and segmentation from brain is computed through tomography image data which in turn is a time-consuming process performed by the medical experts. Our main concentration is on the techniques which use image segmentation to detect brain tumors. Image segmentation is used to extract the abnormal tumor portion in brain. Detecting and localizing brain tumors based on MRI scans would drastically reduce the cost and time of cancer diagnosis and help in the early detection of tumors which would essentially prove to be a life saver.

1. Introduction

Detection of brain tumor is very common fatality in current scenario of healthcare society. MRI-based medical image analysis for brain tumor studies is gaining attention in recent times due to an increased need for efficient and objective evaluation of large amounts of data. While the pioneering approaches applying automated methods for the analysis of brain tumor images date back almost two decades, the current methods are becoming more mature and coming closer to routine clinical application.

This report aims to provide a comprehensive overview by providing a brief introduction to brain tumors and their imaging of brain tumors. Then, we review the state of the art in segmentation and modeling related to tumor-bearing brain images. The objective of segmentation is to outline the tumor including its sub compartments and surrounding tissues while the main challenge in modeling is the handling of morphological changes caused by the tumor.

Medical imaging is performed in various modalities, such as MRI scans, CT scans, ultrasound etc. Segmentation is typically performed manually by expert physicians as a part of treatment planning and diagnosis. Due to the increasing amount of available data and the complexity of features of interest, it is becoming essential to develop automated segmentation methods to assist and speed-up image understanding tasks.

2. Hardware / Software Requirements

Hardware :-

1) Laptop / PC
2) Processor – Intel(R) Core (TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
3) RAM – 8.00 GB
4) Program Install Support – Install and Uninstall
5) System Requirements – No special requirements
6) OS Support – Windows Vista, Windows XP, Windows7 (x32 and x64), Windows 8 (x32 and x64), Windows 10 (x32 and x64) Windows 11, Linux, Ubuntu 20.0.4, macOS.

Software :-

1) VS Code – Text Editor
2) Jupyter Notebook – Live Code Notebook

3. Existing Method

Sophisticated imaging techniques can pinpoint brain tumors. Diagnostic tools include computed tomography (CT or CAT scan) and magnetic resonance imaging (MRI). Other MRI sequences can help the surgeon plan the resection of the tumor based on the location of the normal nerve pathways of the brain. Intraoperative MRI also is used during surgery to guide tissue biopsies and tumor removal. Magnetic resonance spectroscopy (MRS) is used to examine the tumor's chemical profile and determine the nature of the lesions seen on the MRI. Positron emission tomography (PET scan) can help detect recurring brain tumors.

Magnetic Resonance Imaging (MRI) :-

An MRI uses magnetic fields, not x-rays, to produce detailed images of the body. MRI can be used to measure the tumor's size. A special dye called a contrast medium is given before the scan to create a clearer picture. This dye can be injected into a patient's vein or given as a pill or liquid to swallow.

CT Scan :-

A CT scan takes pictures of the inside of the body using x-rays taken from different angles. A computer combines these pictures into a detailed, 3-dimensional image that shows any abnormalities or tumors. A CT scan can help find bleeding and enlargement of the fluid-filled spaces in the brain, called ventricles. Changes to bone in the skull can also be seen on a CT scan, and it can be used to measure a tumor's size.

Sometimes the only way to make a definitive diagnosis of a brain tumor is through a biopsy. The neurosurgeon performs the biopsy and the pathologist makes the final diagnosis, determining whether the tumor appears benign or malignant, and grading it accordingly.

3.1. Drawbacks / Limitations of the Existing Method

1) It is a slow and time-consuming process. The present demands require faster and accurate approaches in diagnosing brain tumors.

2) It is really costly and hence not affordable for a large chunk of the population.

3) It delays the start of the patient's chemotherapy procedure since the time required to diagnose tumor is more.

4) The identification of tumors through MRI and CT Scans is complex and requires expertise in the medical field. Segmentation is typically performed manually by expert physicians as a part of treatment planning and diagnosis. Due to the increasing amount of available data and the

complexity of features of interest, it is becoming essential to develop automated segmentation methods to assist and speed-up image understanding tasks.

5) The precious life of someone's loved one is compromised while handling these issues.

4. Proposed Model

We've proposed two Deep Learning CNN (Convolutional Neural Network) models :-

1) **ResNet (Residual Network) Classifier Model** – To confirm the possibility of a brain tumor.

2) **ResUNet Segmentation Model** – To detect the exact location of the tumor.

Though these are pre-trained Models, but we've apply the methodology of Transfer Learning to re-train them in order to perform the classification and localization of tumors through the Brain MRI scans which are fed to these Models. Transfer Learning is used since starting from a pretrained model can drastically reduce the computational time required if the training is performed from scratch.

Keywords :-

1) **CNNs :-**
   A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that use deep learning to perform both generative and descriptive tasks, often using machine vison that includes image and video recognition etc.

2) **Transfer Learning :-**
   Transfer learning is a machine learning technique in which a model that has been trained to perform a specific task is being re-used (re-trained) as a starting point for another task.

3) **ResNet and ResUNet :-**
   ResNet is a CNN used for Image Recognition.
   ResUNet is a CNN used for Biomedical Image Segmentation.

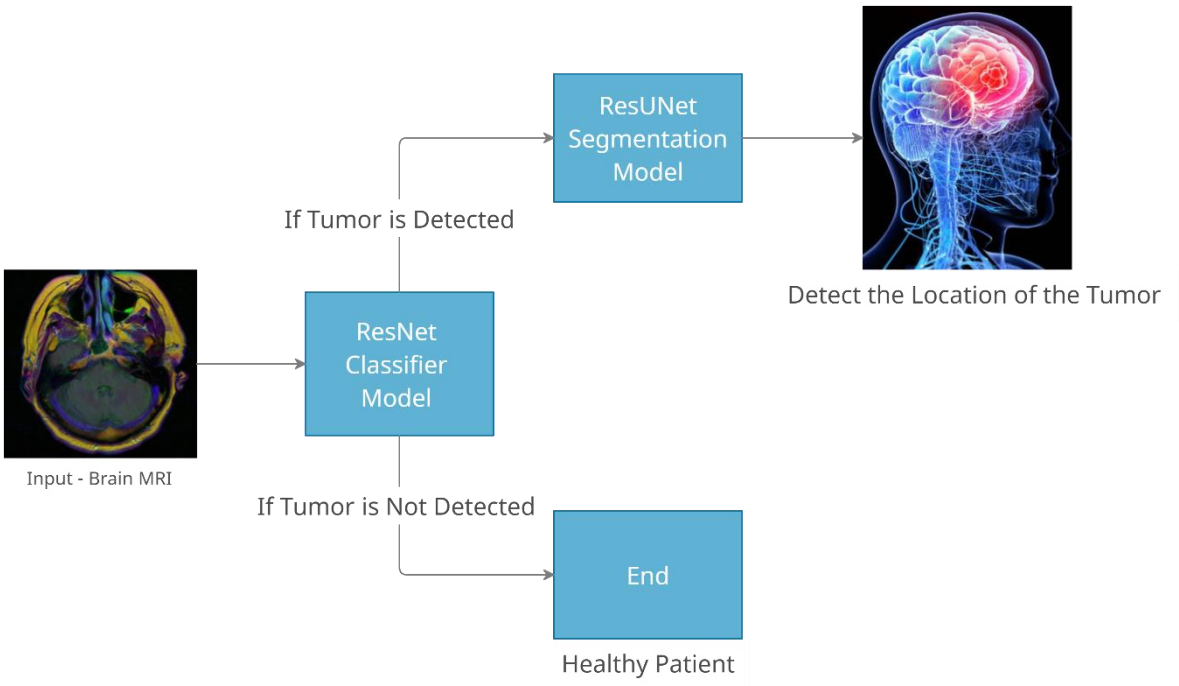
Dataset :-

Link – https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation

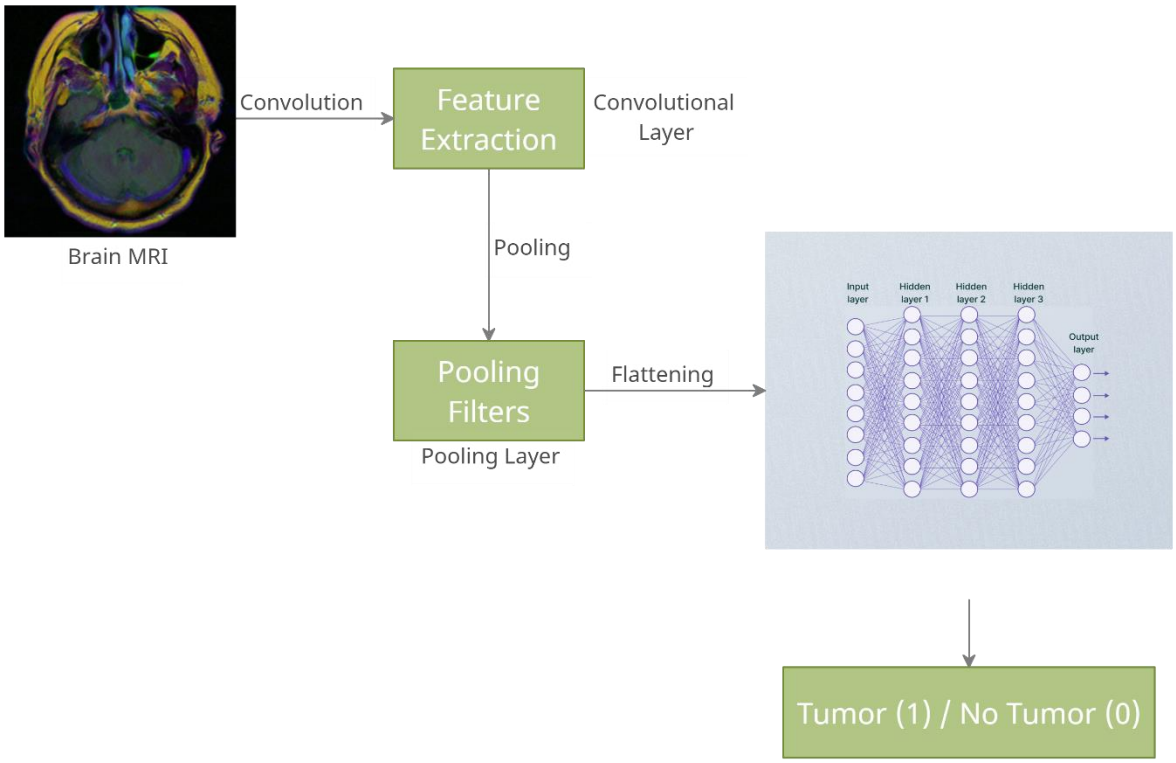
This dataset contains the Brain MRI Images along with their segmentation masks. The images were obtained from The Cancer Imaging Archive (TCIA).

4.1. Design

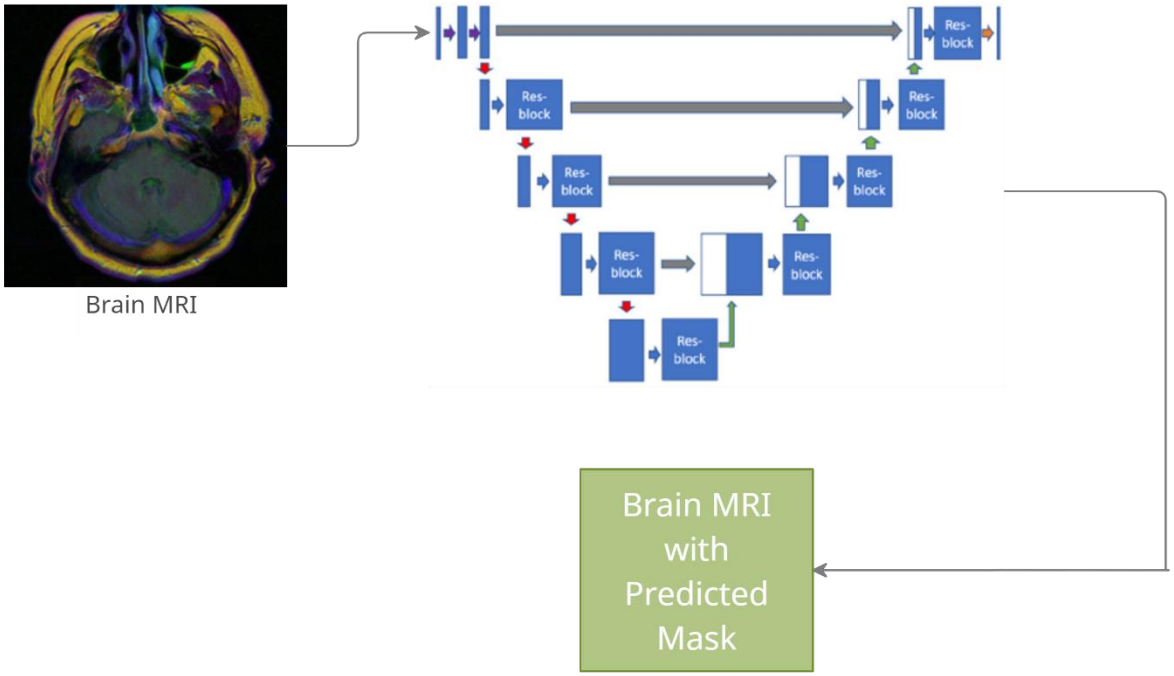
Architecture Diagram



ResNet Classifier Model



ResUNet Segmentation Model



4.2. Module Wise Description

1) **Importing the Libraries and the Dataset.**
   We make use of the following libraries :-
   a) **pandas** – Data analytics, preprocessing and manipulation.
   b) **numpy** – Operations on multi-dimensional arrays.
   c) **matplotlib** – Graphical plotting.
   d) **cv2** – Computer vision, machine learning and image processing.
   e) **skimage** – Image processing.
   f) **tenserflow** – Training ans inference of Deep Neural Networks.
   g) **keras** – Interface for the TensorFlow library.
   h) **random** – Generating random numbers.

   The dataset contains 4 features including Patient ID, Path to the Image, Path to the Mask and Mask itself. It has around 3929 rows.

2) **Dataset Visualization**
   We use matplotlib to visualize different aspects of the dataset for the sake of undestanding, namely :-
   a) A bar chart representing the number of images with the two different masks.
   b) A random Brain MRI.
   c) A random Mask.
   d) MRIs along with their masks.
   e) Masks on top of their corresponding MRI.

3) **Training the ResNet Classifier Model.**
   The Dataset is split into training-validation (80 %) and testing (20 %). Then the pre-trained ResNet Model is downloaded and the methodology of Tranfer Learning is used to add our own classification head to this base model. The training data from the split dataset is fed into the model and it is compiled.

4) **Assessing the performance of the Trained ResNet Model.**
   Weights are loaded into the model and the predit method is called onto the testing data to obtain the output classes (0 or 1) for each Brain MRI and the results are stored in an array. The trained ResNet model gives an accuray of around 99 %.

5) **Building the ResUNet Segmentation Model.**
   The output of the ResNet model is used to fetch the MRIs with class of 1 (Tumor detected) and create a dataframe for those. This dataframe is split again into training (80 %) and testing (20 %) data. The ResUNet model is built by adding more layers to the network.

6) **Training the ResUNet Segmentation Model.**
Finally, the weights are loaded into the newly built model and the predict method called onto the testing data to generate the Predicted Masks for each MRI. A new dataframe is created for the predicted result which contains the Image Path, the Predicted Mask, a column representing the presense of mask for each MRI. This dataframe is merged with the original dataset and the Brain MRI along with their original as well as Predicted Mask are plotted side by side and on top of each other.

4.3. Implementation

1) VS Code was used as the Text Editor and Jupyter Notebook was used to implement the entire project in Python.

2) The Libraries used as well as their functions are already mentioned in the Module Description.

3) The dataset is visualized to a get proper understanding about the Models to be implemented.

4) The dataset is split into training and testing data. Image Generators are created. The ResNet50 Model is downloaded and re-trained using the training dataset. Our own classification heads heads are added to the network and the Model is compiled. Weights are loaded into the network and the predict method is called upon the testing data to detect whether tumor exists or not. The output is in the form of 0's and 1's. (1 = Tumor Detected and 0 = Tumor doesn't exist). The accuracy of this model comes out to be 99.1 % precisely.

5) The output obtained from the ResNet Model is split again into training and testing data. The ResUNet Model is built with some additional layers being added to the previous network. The training data is fed to the newly built model.

6) The ResUNet Model is compiled and the predict method is called upon the testing data to generate our own AI Predicted Masks for each Brain MRI. A dataframe is created for the predicted results and it is merged with the original dataset. The final output is plotted.

5. Results and Discussion

Output from the ResNet Classifier Model

```
array(['1', '0', '1', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '1', '0', '0', '0', '0', '0', '0', '1', '1', '1', '1', '0',
       '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '1', '0',
       '0', '0', '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1',
       '0', '0', '1', '1', '0', '1', '0', '0', '1', '1', '1', '1', '1',
       '1', '0', '0', '0', '1', '0', '0', '1', '1', '1', '0', '0', '0',
       '1', '0', '0', '1', '1', '0', '1', '0', '0', '0', '1', '0', '0',
       '0', '0', '0', '0', '0', '0', '1', '0', '0', '1', '0', '0', '0',
       '0', '1', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0',
       '0', '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0',
       '1', '0', '0', '0', '0', '0', '1', '1', '0', '1', '0', '0', '1',
       '0', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '1',
       '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '1', '0', '1',
       '1', '0', '0', '0', '0', '0', '0', '0', '1', '1', '0', '0', '0',
       '0', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '1', '0',
       '1', '0', '0', '1', '0', '0', '0', '0', '0', '1', '1', '0', '1',
       '0', '0', '1', '1', '0', '0', '0', '1', '0', '0', '1', '0', '0',
       '0', '1', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1',
       '0', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0',
       '1', '1', '0', '0', '0', '1', '0', '0', '1', '1', '0', '0', '0',
       '0', '0', '0', '1', '0', '0', '0', '1', '0', '0', '1', '0', '1',
       '0', '1', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '1',
       '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0', '0', '1',
       '1', '1', '1', '0', '1', '0', '1', '0', '0', '1', '1', '0', '1',
       '0', '0', '1', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0',
       '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',
       '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0',
       '0', '1', '1', '0', '1', '1', '1', '0', '1', '1', '0', '0', '0',
       '1', '1', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0', '1',
       '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0',
       '0', '0', '1', '0'], dtype='<U1')
```

This model predicts the possibility of tumor in each Brain MRI and stores the results in an array.

1 = Tumor exists

0 = Tumor doesn't exist

The accuracy of this model is close to 99 %.

```
# Obtaining the Accuracy of the Model
original = np.asarray(test['mask'])[:len(predict)]
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(original, predict)
accuracy * 100
```

✓ 0.3s

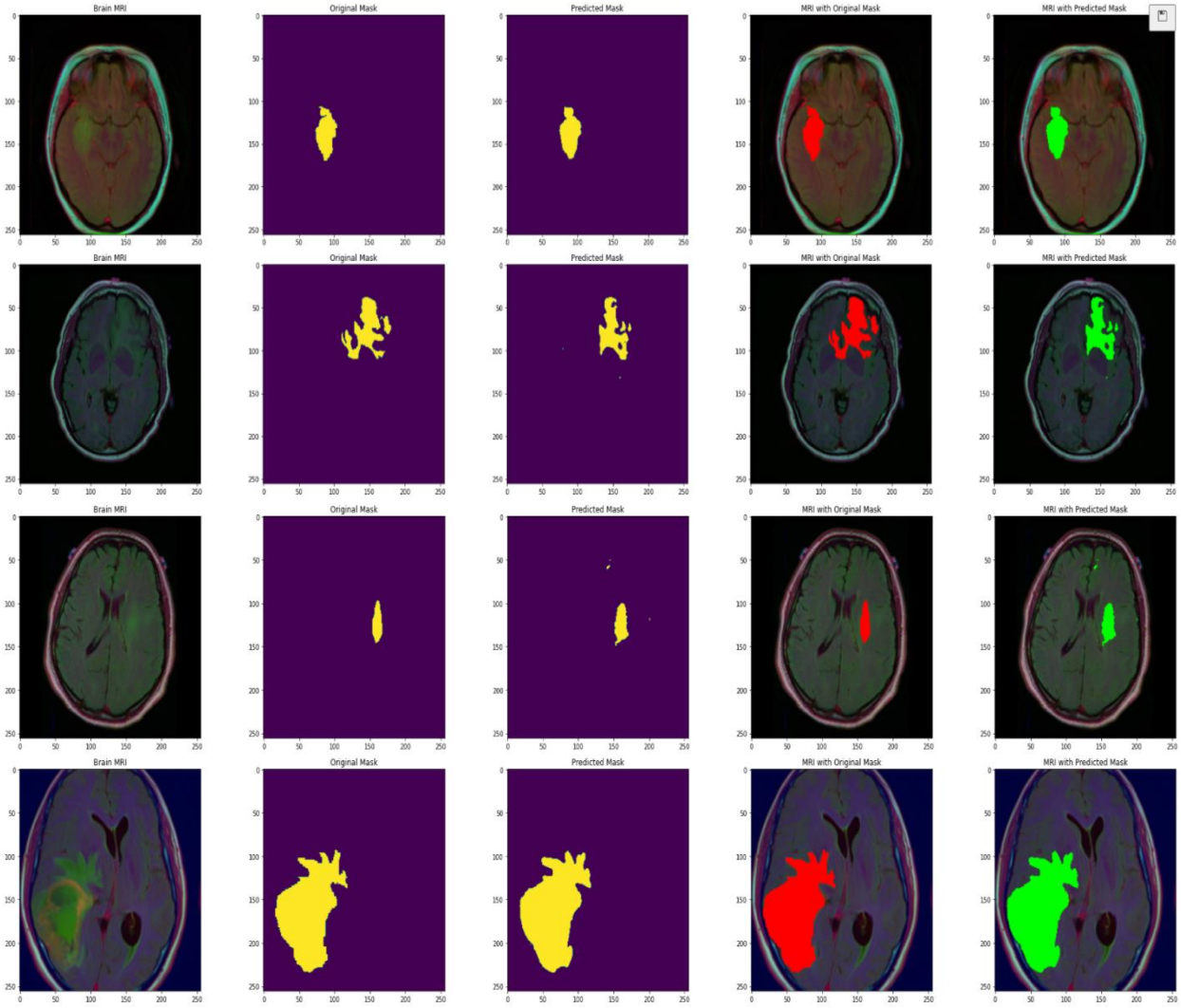
```
99.10714285714286
```

Accuracy Comparion of the ResNet Model with Different Activation Functions

Activation Functions	Accuracy of the ResNet Model
swish and softmax	99.1 %
mish and softmax	97.3 %
relu and sigmoid	97.8 %

Since the accuracy for the **swish-softmax** pair turns out to be the highest, we choose them as the activation functions for our network.

Output from the ResUNet Segmentation Model



This model predicts the segmentation mask for each brain MRI and plots them on top of each other. Turns out, the masks predicted by our implemented model are more accurate in terms of the proper localization of the tumor as compared to the original mask.

6. Conclusion

We were able to build and train two Models for the project, namely :-

a) ResNet Classifier Model – To detect the presence of brain tumor.
b) ResUNet Segmentation Model – To localize the tumor that was detected in the ResNet Model.

The results and accuracy of these models even outperform the traditional methods by a large margin. Since the present demands require faster and accurate approaches in diagnosing brain tumors, our approach can provide quick and non-time-consuming results within seconds. It is also affordable for the common man. The patient's chemotherapy procedure can start as early as possible.

Finally, the most important and ultimate objective is achieved i.e. **A Human Life is Saved!**

Limitations

There are no as such limitations in our approach since it gives positive results at all fronts. The only limitation that perhaps one can think of is that the patient has to still undergo the MRI Scan in order to obtain the images to be fed into these models but it isn't a major one.

References

- Classification Using Deep Learning Neural Networks for Brain Tumors
  ⇨ Heba Mohsen, El-Sayed A. El-Dahshan, El-Sayed M. El-Horbaty and Abdel-Badeeh M. Salem

- Classification of Brain Tumors from MRI Images Using a Convolutional Neural Network
  ⇨ Milica M. Badža and Marko Č. Barjaktarović

- Recent Advancement in Cancer Detection using Machine Learning: Systematic Survey of Decades, Comparisons and Challenges
  ⇨ Tanzila Saba

- A Survey of MRI-Based Brain Tumor Segmentation Methods
  ⇨ Jin Liu, Min Li, Jianxin Wang, Fangxiang Wu, Tianming Liu and Yi Pan

- Classification of Brain Tumors Using Deep Features Extracted Using CNN
  ⇨ Shaik Basheera1 and M. Satya Sai Ram

- Deep Residual Learning for Image Recognition
  ⇨ Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun

- Convolutional Neural Networks for Multi-Class Brain Disease Detection Using MRI Images
  ⇨ Muhammed Taloa, Ozal Yildirima, Ulas Baran Baloglub, Galip Aydinc, U Rajendra Acharya

- Brain Tumor Classification Using ResNet-101 Based Squeeze and Excitation Deep Neural Network
  ⇨ Palash Ghoshal, Lokesh Nandanwar and Swati Kanchan

- Brain Tumor Classification in MRI Image Using Convolutional Neural Network
  ⇨ Hassan Ali Khan, Wu Jue, Muhammad Mushtaq and Muhammad Umer Mushtaq

- An Efficient Brain Tumor Image Segmentation Based on Deep Residual Networks (ResNets)
  ⇨ Lamia H. Shehab, Omar M. Fahmy, Safa M. Gasser and Mohamed S. El-Mahallawy

- Automated Brain Tumor Classification Using Various Deep Learning Models: A Comparative Study
  ⇨ Alaa Ahmed Abbood, Qahtan Makki Shallal and Mohammed A. Fadhel

- Detection Of Tumors on Brain MRI Images Using The Hybrid Convolutional Neural Network Architecture
  ⇨ Ahmet Çinar and Muhammed Yildirim

- Deep Residual Networks (ResNet, ResNet50) – Guide in 2021
  ⇨ Gaudenz Boesch

- Understanding and Coding a ResNet in Keras
  ⇨ Priya Dwivedi

- Implementing a ResNet in Keras (6.3)
  ⇨ Jeff Heaton

- Transfer Learning Using Keras(ResNet-50)| Complete Python Tutorial|
  ⇨ Nachiketa Hebbar

- Deep Learning using Keras - Complete & Compact Dummies Guide
  ⇨ Abhilash Nelson

- 5 Advanced CNN Architectures

- Deep Learning Essentials

- Modern Convolutional Neural Networks

Appendix: Sample Code

Importing the Libraries

```
# Data analysis, preprocessing and manipulation
import pandas as pd

# Operations on multidimensional array objects
import numpy as np

# Graphical plotting
import matplotlib.pyplot as plt
%matplotlib inline

# Computer vision, machine learning and image processing
import cv2

# Image preprocessing
from skimage import io

# Training and Inference of Deep Neural Networks
import tensorflow as tf

# Interface for the TensorFlow library
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import *
from tensorflow.keras.models import Model

# Generating random numbers
import random
```

Importing the Dataset

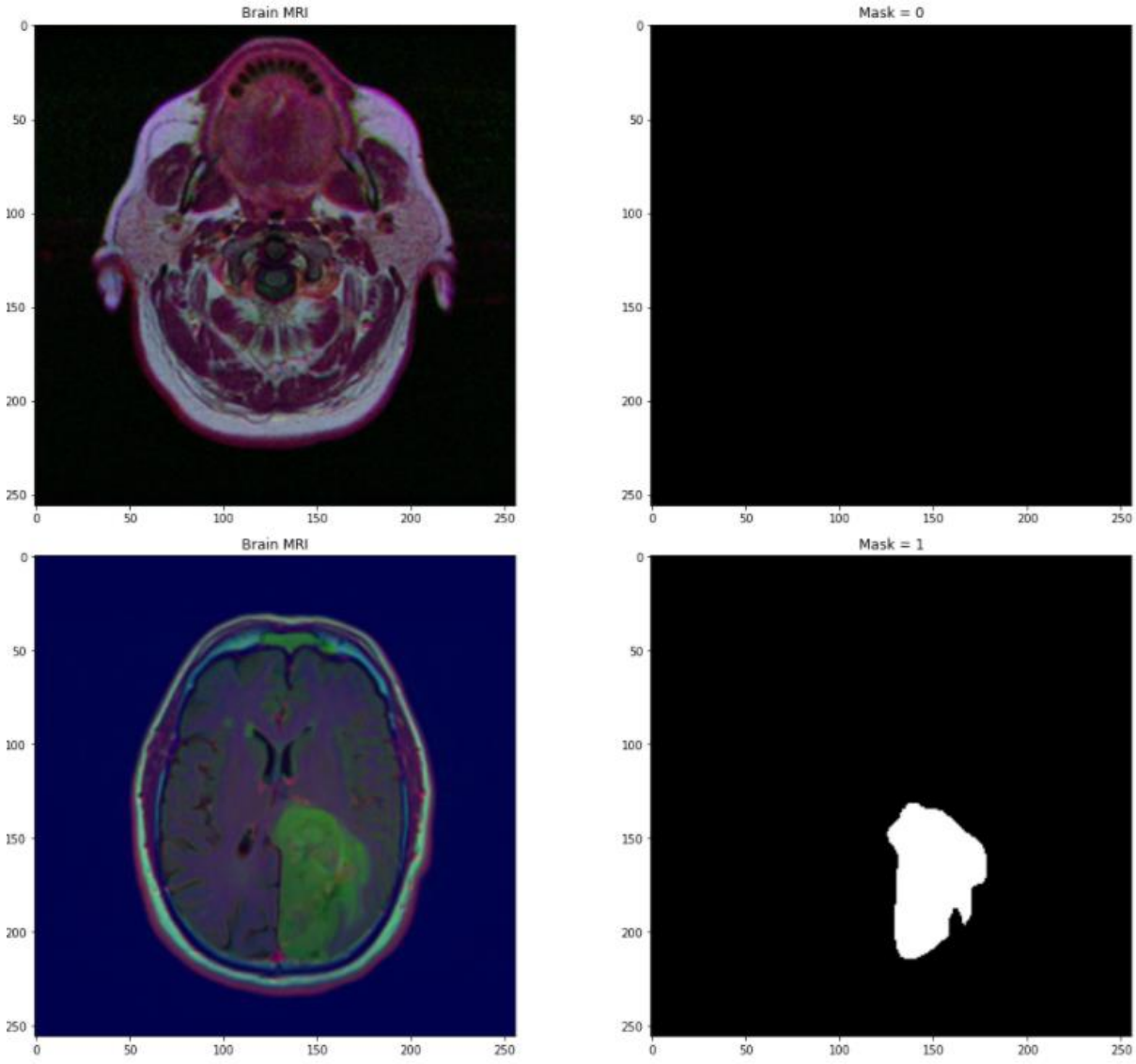
```
%cd Dataset
# Reading the CSV Data File
dataset = pd.read_csv("data_mask.csv")
dataset
```

	patient_id	image_path	mask_path	mask
0	TCGA_CS_5395_19981004	TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1.tif	TCGA_CS_5395_19981004/TCGA_CS_5395_19981004_1_...	0
1	TCGA_CS_5395_19981004	TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1.tif	TCGA_CS_4944_20010208/TCGA_CS_4944_20010208_1_...	0
2	TCGA_CS_5395_19981004	TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1.tif	TCGA_CS_4941_19960909/TCGA_CS_4941_19960909_1_...	0
3	TCGA_CS_5395_19981004	TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1.tif	TCGA_CS_4943_20000902/TCGA_CS_4943_20000902_1_...	0
4	TCGA_CS_5395_19981004	TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1.tif	TCGA_CS_5396_20010302/TCGA_CS_5396_20010302_1_...	0
...	...	...	...	...

Visualizing the MRIs and the Masks

```
import random
fig, axs = plt.subplots(5,2, figsize=(16,32))
count = 0
for x in range(5):
  i = random.randint(0, len(dataset))
  axs[count][0].title.set_text("Brain MRI")
  axs[count][0].imshow(cv2.imread(dataset.image_path[i]))
  axs[count][1].title.set_text("Mask = " + str(dataset['mask'][i]))
  axs[count][1].imshow(cv2.imread(dataset.mask_path[i]))
  count += 1

fig.tight_layout()
```



Splitting the Dataset into Training and Testing Data

```
# Training and Validation Dataset Size = 80%
# Testing Dataset Size = 20%
from sklearn.model_selection import train_test_split
train, test = train_test_split(trainingDataset, test_size = 0.20)
```

Creating the Image and Data Generators

```
# Creating an Image Generator
from keras_preprocessing.image import ImageDataGenerator

# Creating a Data Generator which scales the data from 0 to 1 and makes
validation split of 0.15
dataGenerator = ImageDataGenerator(rescale=1./255., validation_split =
0.20)
trainGenerator = dataGenerator.flow_from_dataframe(
dataframe = train,
directory = './',
x_col = 'image_path',
y_col = 'mask',
subset = "training",
batch_size = 16,
shuffle = True,
class_mode = "categorical",
target_size = (256,256))


validGenerator = dataGenerator.flow_from_dataframe(
dataframe = train,
directory = './',
x_col = 'image_path',
y_col = 'mask',
subset = "validation",
batch_size = 16,
shuffle = True,
class_mode = "categorical",
target_size = (256,256))

# Creating a Data Generator for the Test Images
testDatagen = ImageDataGenerator(rescale=1./255.)
testGenerator = testDatagen.flow_from_dataframe(
dataframe = test,
directory = './',
x_col = 'image_path',
y_col = 'mask',
```

```
batch_size = 16,
shuffle = False,
class_mode = 'categorical',
target_size = (256,256))
```

ResNet Model

```
# Downloading ResNet50 Base Model
baseModel = ResNet50(weights = 'imagenet', include_top = False,
input_tensor = Input(shape=(256, 256, 3)))
baseModel.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 262, 262, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 128, 128, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 128, 128, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 128, 128, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 130, 130, 64  0           ['conv1_relu[0][0]']
                                )

 pool1_pool (MaxPooling2D)      (None, 64, 64, 64)   0           ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2D)   (None, 64, 64, 64)   4160        ['pool1_pool[0][0]']

show more (open the raw output data in a text editor) ...

==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
```

Adding Classification Head to the Base Model

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size = (4,4))(headModel)
headModel = Flatten(name= 'flatten')(headModel)
headModel = Dense(256, activation = "swish")(headModel)
headModel = Dropout(0.3)(headModel)#
headModel = Dense(256, activation = "swish")(headModel)
headModel = Dropout(0.3)(headModel)
headModel = Dense(2, activation = 'softmax')(headModel)

model = Model(inputs = baseModel.input, outputs = headModel)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3       0
                             )]

 conv1_pad (ZeroPadding2D)   (None, 262, 262, 3)       0

 conv1_conv (Conv2D)         (None, 128, 128, 64       9472
                             )

 conv1_bn (BatchNormalization)  (None, 128, 128, 64    256
                             )

 conv1_relu (Activation)     (None, 128, 128, 64       0
                             )

 pool1_pad (ZeroPadding2D)   (None, 130, 130, 64       0
                             )

 pool1_pool (MaxPooling2D)   (None, 64, 64, 64)        0

 conv2_block1_1_conv (Conv2D)   (None, 64, 64, 64)     4160

 average_pooling2d (AveragePool  (None, 2, 2, 2048)    0
 ing2D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 256)               2097408

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 2)                 514

=================================================================
Total params: 25,751,426
Trainable params: 25,698,306
Non-trainable params: 53,120
```

Adding Weights to the Network and Making Prediction

```
with open('resnet-50-MRI.json', 'r') as json_file:
    json_savedModel= json_file.read()
model = tf.keras.models.model_from_json(json_savedModel)
model.load_weights('weights.hdf5')
model.compile(loss = 'categorical_crossentropy', optimizer='adam',
metrics= ["accuracy"])

testPredict = model.predict(testGenerator, steps = testGenerator.n //
16, verbose =1)

# Obtaining the Predicted Class from the Model Prediction
predict = []
for i in testPredict:
  predict.append(str(np.argmax(i)))
predict = np.asarray(predict)
predict
```

```
array(['0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '1',
       '1', '0', '0', '1', '0', '0', '0', '0', '1', '0', '1', '0', '1',
       '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1',
       '1', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '0', '0', '1', '1', '0', '1', '0', '0', '0', '0', '0', '0',
       '1', '0', '1', '1', '0', '0', '0', '0', '1', '0', '0', '0', '1',
       '1', '1', '0', '0', '0', '0', '1', '1', '0', '1', '1', '0', '0',
       '1', '0', '1', '0', '0', '1', '0', '0', '1', '0', '0', '0', '0',
       '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1', '1',
       '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '0',
       '1', '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0',
       '0', '0', '0', '1', '1', '1', '1', '0', '1', '0', '1', '0', '0',
       '1', '1', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1',
       '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '0', '1',
       '0', '0', '0', '0', '1', '1', '0', '0', '0', '0', '0', '0', '0',
       '1', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '0', '0',
       '0', '1', '0', '0', '1', '1', '1', '1', '0', '0', '0', '1', '0',
       '1', '0', '1', '1', '1', '1', '0', '1', '0', '1', '0', '0', '0',
       '0', '0', '1', '1', '1', '1', '0', '1', '0', '1', '0', '1', '0',
       '0', '0', '0', '1', '1', '1', '0', '0', '0', '0', '0', '1', '0',
       '0', '0', '0', '1', '0', '0', '1', '1', '0', '0', '0', '1', '1',
       '0', '0', '1', '0', '1', '0', '0', '0', '1', '1', '0', '0', '0',
       '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '1',
       '0', '1', '0', '0', '0', '0', '1', '0', '1', '1', '0', '0', '0',
       '1', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '1', '0',
```

Obtaining the Accuracy of ResNet Model

```
# Obtaining the Accuracy of the Model
original = np.asarray(test['mask'])[:len(predict)]
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(original, predict)
accuracy * 100
```

✓ 0.3s

```
99.10714285714286
```

Splitting the Output Data Obtained from ResNet Model into Training and Testing

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(mask, test_size=0.20)
test, validation = train_test_split(validation, test_size=0.5)

# Creating separate lists for imageId and classId to pass into the
generator

trainingID = list(train.image_path)
trainingMask = list(train.mask_path)

validationID = list(validation.image_path)
validationMask= list(validation.mask_path)
```

Creating Image Generators

```
from utilities import DataGenerator
trainingGenerator = DataGenerator(trainingID,trainingMask)
validationGenerator = DataGenerator(validationID,validationMask)

def resblock(X, f):

  # Making a copy of the input
  copy = X

  # Main Path
  X = Conv2D(f, kernel_size = (1,1) ,strides = (1,1),kernel_initializer
="he_normal")(X)
  X = BatchNormalization()(X)
  X = Activation("swish")(X)
```

```
  X = Conv2D(f, kernel_size = (3,3), strides =(1,1), padding = "same",
kernel_initializer ="he_normal")(X)
  X = BatchNormalization()(X)

  # Shortest Path
  copy = Conv2D(f, kernel_size = (1,1), strides =(1,1),
kernel_initializer ="he_normal")(copy)
  copy = BatchNormalization()(copy)

  # Adding the output from main path and short path
  X = Add()([X, copy])
  X = Activation("swish")(X)

  return X

# Function to Upscale and Concatenate the passed values
def upsample_concat(x, skip):
  x = UpSampling2D((2,2))(x)
  merge = Concatenate()([x, skip])

  return merge

inputShape = (256,256,3)

# Input Tensor Shape
tensorShape = Input(inputShape)

# Stage 1
conv1 = Conv2D(16,3,activation= 'swish', padding = 'same',
kernel_initializer ='he_normal')(tensorShape)
conv1 = BatchNormalization()(conv1)
conv1 = Conv2D(16,3,activation= 'swish', padding = 'same',
kernel_initializer ='he_normal')(conv1)
conv1 = BatchNormalization()(conv1)
pool1 = MaxPool2D(pool_size = (2,2))(conv1)

# Stage 2
conv2 = resblock(pool1, 32)
pool2 = MaxPool2D(pool_size = (2,2))(conv2)

# Stage 3
conv3 = resblock(pool2, 64)
pool3 = MaxPool2D(pool_size = (2,2))(conv3)

# Stage 4
conv4 = resblock(pool3, 128)
```

```
pool4 = MaxPool2D(pool_size = (2,2))(conv4)

# Stage 5 (Bottle Neck)
conv5 = resblock(pool4, 256)

# Upscale Stage 1
up1 = upsample_concat(conv5, conv4)
up1 = resblock(up1, 128)

# Upscale Stage 2
up2 = upsample_concat(up1, conv3)
up2 = resblock(up2, 64)

# Upscale Stage 3
up3 = upsample_concat(up2, conv2)
up3 = resblock(up3, 32)

# Upscale Stage 4
up4 = upsample_concat(up3, conv1)
up4 = resblock(up4, 16)

# Final Output
output = Conv2D(1, (1,1), padding = "same", activation =
"softmax")(up4)

segmentationModel = Model(inputs = tensorShape, outputs = output )
```

Compiling the ResUNet Model and Adding Weights

```
from utilities import focal_tversky, Tversky

# Compiling the Model
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
segmentationModel.compile(optimizer = adam, loss = focal_tversky,
metrics = [tversky])

# Adding weights to the network
from utilities import focal_tversky, tversky

with open('ResUNet-MRI.json', 'r') as json_file:
    json_savedModel= json_file.read()

segmentationModel = tf.keras.models.model_from_json(json_savedModel)
segmentationModel.load_weights('weights_seg.hdf5')
adam = tf.keras.optimizers.Adam(lr = 0.05, epsilon = 0.1)
```

```
segmentationModel.compile(optimizer = adam, loss = focal_tversky,
metrics = [tversky])
```

Predicting and Creating a Dataframe for the Predicted Result

```
# Prediction
from utilities import prediction
imageID, mask, hasMask = prediction(test, model, segmentationModel)

# Creating a Data Frame for the Predicted Result
dataframePredicted = pd.DataFrame({'image_path':
imageID,'predicted_mask': mask,'has_mask': hasMask})
dataframePredicted

# Merging the Original Dataset with the Predicted Results
mergedDataset = test.merge(dataframePredicted, on = 'image_path')
mergedDataset.head()
```

	patient_id	image_path	mask_path	mask	predicted_mask	has_mask
0	TCGA_DU_A5TR_19970726	TCGA_HT_7608_19940304/TCGA_HT_7608_19940304_16...	TCGA_HT_7608_19940304/TCGA_HT_7608_19940304_16...	1	[[[[8.785064e-07], [3.153003e-06], [5.1284264e...	1
1	TCGA_DU_7010_19860307	TCGA_DU_6405_19851005/TCGA_DU_6405_19851005_48...	TCGA_DU_6405_19851005/TCGA_DU_6405_19851005_48...	1	[[[[3.9119607e-07], [1.4473533e-06], [2.817144...	1
2	TCGA_HT_8563_19981209	TCGA_DU_5872_19950223/TCGA_DU_5872_19950223_35...	TCGA_DU_5872_19950223/TCGA_DU_5872_19950223_35...	1	[[[[8.5909994e-07], [3.384495e-06], [8.407663e...	1
3	TCGA_HT_A5RC_19990831	TCGA_FG_5962_20000626/TCGA_FG_5962_20000626_37...	TCGA_FG_5962_20000626/TCGA_FG_5962_20000626_37...	1	[[[[4.4609774e-06], [8.6445725e-06], [1.172767...	1
4	TCGA_FG_5962_20000626	TCGA_HT_7692_19960724/TCGA_HT_7692_19960724_17...	TCGA_HT_7692_19960724/TCGA_HT_7692_19960724_17...	1	[[[[5.8435205e-07], [2.086806e-06], [4.2034367...	1

Final Result

```
count = 0
fig, axs = plt.subplots(10, 5, figsize=(30, 50))
for i in range(len(mergedDataset)):
  if mergedDataset['has_mask'][i] == 1 and count < 10:

    # Reading the images and converting them to RGB format
    img = io.imread(mergedDataset.image_path[i])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    axs[count][0].title.set_text("Brain MRI")
    axs[count][0].imshow(img)

    # Obtaining the Mask for the image
    mask = io.imread(mergedDataset.mask_path[i])
    axs[count][1].title.set_text("Original Mask")
    axs[count][1].imshow(mask)
```

```
    # Obtaining the Predicted Mask for the image
    predicted_mask =
np.asarray(mergedDataset.predicted_mask[i])[0].squeeze().round()
    axs[count][2].title.set_text("Predicted Mask")
    axs[count][2].imshow(predicted_mask)

    # Applying the Mask over the image
    img[mask == 255] = (255, 0, 0)
    axs[count][3].title.set_text("MRI with Original Mask")
    axs[count][3].imshow(img)

    img_ = io.imread(mergedDataset.image_path[i])
    img_ = cv2.cvtColor(img_, cv2.COLOR_BGR2RGB)
    img_[predicted_mask == 1] = (0, 255, 0)
    axs[count][4].title.set_text("MRI with Predicted Mask")
    axs[count][4].imshow(img_)
    count += 1

fig.tight_layout()
```