

BRAIN TUMOR IMAGE SEGMENTATION USING DEEP LEARNING APPROACH

BY

DARSHAN A/L SURESH

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2022

REPORT STATUS DECLARATION FORM

Title: Brain Tumor Image Segmentation using Deep Learning Approach

Academic Session: Jan 2022

I DARSHAN A/L SURESH
(CAPITAL LETTER)

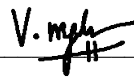
declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)



(Supervisor's signature)

Address:

6, Jalan Machang Bubok 4,

Taman Machang Bubok

14020 Bukit Mertajam, Penang

Mogana Vadiveloo

Supervisor's name

Date: 22/04/2022

Date: 22 April 2022

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 22/04/2022

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that Darshan A/L Suresh (ID No: 16ACB06423) has completed this final year project entitled “Brain Tumor Image Segmentation using Deep Learning Approach” under the supervision of Dr. Mogana Vadiveloo (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



(Darshan A/L Suresh)

DECLARATION OF ORIGINALITY

I declare that this report entitled “**BRAIN TUMOR IMAGE SEGMENTATION USING DEEP LEARNING APPROACH**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  _____

Name : DARSHAN A/L SURESH

Date : 22 April 2022

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. Mogana A/P Vadiveloo and my moderator, Dr. Tan Joi San who have given me the chance to involve myself in the Artificial Intelligence(AI) based project by using Deep Learning. Dr. Mogana have constantly supported and motivated me throughout the implementation of this project and allowed me to grow more responsible. Again, a million thanks to both my supervisor and moderator.

Besides that, I would like to also take this opportunity to thank my close friends Cheah Weng Hung, Yes bin Bah Hashim, Low Eng Soon, Chris Edward and my parents, Mr. Suresh and Mrs. Rajeswari for giving me immense support and motivation whenever I was having trouble or feeling down during the project implementation. Thank you for being there for me.

ABSTRACT

The role of computer vision in the field of biomedical sciences is crucial. In neurosurgical, Magnetic Resonance Images (MRI) scans are used to detect cancerous cell growth in brain called brain tumor. Application that can aid in providing automatic brain tumor segmentation is crucial as segmentation of the exact size and spatial location of these tumors are a time consuming task. In addition, the existing software for brain tumor segmentation are implemented using conventional image segmentation algorithms which are labor intensive and causes over or under segmented tumor regions. Deep learning algorithm is able to provide good tumor segmentation results compared to other conventional segmentation algorithms as it learns from the labeled brain MRIs to predict the location of tumor region and consequently segment the tumor. Therefore, in this work, deep learning approach is used to build uNet+segUnet ensemble model which automatically segments tumor regions. The proposed model is also deployed as a web application using Flask for public usage. The proposed model is developed with Keras by using “Brain MRI segmentation” dataset. 1373 images are used from the dataset where they are split as 1098 images for training, 137 images for validation and 138 images for testing and the proposed model managed to obtain 75.78% average tumor segmentation accuracy using Jaccard similarity index on the testing set. The proposed ensemble model was also tested on another dataset with 621 testing images where it achieved 66.75% testing accuracy by using the Jaccard similarity index.

Table of Contents

TITLE PAGE.....	i
REPORT STATUS DECLARATION FORM.....	ii
SUBMISSION OF FINAL YEAR PROJECT.....	iii
DECLARATION OF ORIGINALITY.....	iv
ACKNOWLEDGEMENTS.....	v
ABSTRACT.....	vi
TABLE OF CONTENTS.....	vii
LIST OF FIGURES.....	x
LIST OF TABLES.....	xii
LIST OF ABBREVIATIONS.....	xiii
CHAPTER 1 INTRODUCTION.....	1
1.1 Introduction.....	1
1.2 Problem Statement and Motivations.....	2
1.3 Project Scope.....	3
1.4 Project Objectives.....	4
1.5 Impact, Significance and Contributions.....	4
CHAPTER 2 LITERATURE REVIEW.....	5
2.0 Introduction.....	5
2.1 Review On Non-Deep Learning Based Brain Tumor Segmentation Application.....	5
2.1.1 MIPAV.....	5
2.1.2 ITK-SNAP.....	7
2.1.3 Olea Sphere 3.0.....	8
2.2 Review On Deep Learning Based Brain Tumor Segmentation Applications.....	8
2.2.1 DeepMIB.....	8
2.2.2 NiftyNet.....	10
2.3 Critical Remarks.....	11
CHAPTER 3 SYSTEM DESIGN.....	15
3.1 Flowchart of the overall flow of system.....	15
3.2 Activity diagram.....	17
3.3 Verification Plan.....	19
3.4 Chapter Summary.....	21
CHAPTER 4 METHODOLOGY.....	22
4.1 Methodology.....	22

4.2 Tools to Use.....	24
4.3 User Requirement.....	25
4.4 Evaluation Measure.....	26
4.5 Dataset.....	28
4.6 Model architecture.....	31
4.6.1 uNet based model architecture.....	31
4.6.2 segUnet based model architecture.....	32
4.6.3 Ensemble learning using segUnet and uNet.....	34
4.7 Issues and Challenges.....	34
4.8 Timeline.....	35
4.9 Chapter Summary.....	37
CHAPTER 5 SYSTEM IMPLEMENTATION.....	38
5.1 Implementation of proposed uNet+SegUnet model.....	38
5.1.1 Start.....	38
5.1.2 Data Preprocessing.....	38
5.1.3 Data loading.....	39
5.1.4 Data splitting.....	39
5.1.5 Data augmentation.....	40
5.1.6 Data loading in batches.....	41
5.1.7 Training parameters set up.....	41
5.1.8 uNet model implementation	42
5.1.9 segUnet model implementation.....	43
5.1.10 Experimental models and hyperparameter tuning.....	45
5.1.11 Deploy the built as a web application using python flask.....	49
5.1.12 Brain tumor segmentation produced from web application.....	51
5.2 Results of the uNet+segUnet ensemble model.....	54
5.3 Chapter summary.....	58
CHAPTER 6 CONCLUSION, NOVELTY AND FUTURE WORK.....	59
6.1 Conclusion.....	59
6.2 Novelty.....	59
6.3 Future work.....	59
REFERENCES.....	60
APPENDIX.....	
Appendix A.....	A-1

Appendix B.....	B-1
Appendix C1.....	C1-1
Appendix C2.....	C2-1
Appendix D.....	D-1
WEEKLY LOG.....	E-1
POSTER.....	E-7
PLAGIARISM CHECK RESULT.....	E-8
FYP2 CHECKLIST.....	E-12

LIST OF FIGURES

Figure	Title	Page
Figure 1.1	2D MRI brain tumor images. The tumor regions are labeled in green.	2
Figure 2.1	Over segmentation occurrence in MIPAV	6
Figure 2.2	Hyperparameter input requirement for mean shift segmentation	6
Figure 2.3	Semi-Manual segmentation results in ITK-SNAP	7
Figure 2.4	DeepMIB overall workflow	9
Figure 2.5	An example of multi-organ abdominal CT segmentation using NiftyNet	10
Figure 3.1	Flowchart of proposed system	15
Figure 3.2	Activity diagram	17
Figure 4.1	Iterative and Incremental model	22
Figure 4.2	FLAIR Brain MRI and the respective ground truth for patient number 31(slice number 30) in dataset[5]	28
Figure 4.3	FLAIR Brain MRI and the respective ground truth for patient number 21(slice number 19) in dataset[5]	29
Figure 4.4	FLAIR Brain MRI and the respective ground truth for patient number 3(slice number 100) in dataset[6]	30
Figure 4.5	Model architecture made based on uNet implementation	31
Figure 4.6	SegNet architecture by [9]	32
Figure 4.7	Original SegUnet model by [8]	33
Figure 4.8	Gantt chart of FYP1	35
Figure 4.9	Gantt chart of FYP2	37
Figure 5.1	Importing of necessary libraries	38
Figure 5.2	Loading data from dataset in drive and dataframe creation	39
Figure 5.3	Data splitting in 8:1:1 ratio	40

Figure 5.4	Data augmentations	40
Figure 5.5	Hyperparameter set up for model training	41
Figure 5.6	UNet model implementation from scratch	43
Figure 5.7	Defining classes for indices pooling	44
Figure 5.8	SegUnet model implementation from scratch	45
Figure 5.9	Default app route set up with flask	50
Figure 5.10	Both models are loaded and ready for segment	50
Figure 5.11	Set up of evaluation in web app	51
Figure 5.12	Homepage of web app hosted on local public IP	52
Figure 5.13	User chooses and upload a 2D Flair Brain MR image	52
Figure 5.14	Brain tumor segmentation shown on the UI	53
Figure 5.15	Jaccard score computed and displayed	53
Figure 5.16	Brain tumor segmentation by proposed model	54
Figure 5.17	Proposed model is able to segment the tumor	55
Figure 5.18	Results obtained from testing by database by [6]	56
Figure 5.19	Brain tumor segmentation on database[6] with high accuracy by proposed model	57
Figure 5.20	The proposed model unable to segment the tumor region accurately	57

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Summary of MIPAV	11
Table 2.2	Summary of ITK-SNAP	12
Table 2.3	Summary of Olea Sphere 3.0	13
Table 2.4	Summary of DeepMIB	13
Table 2.5	Summary of NiftyNet	14
Table 3.1	First verification plan	19
Table 3.2	Second verification plan	20
Table 3.3	Third verification plan	21
Table 4.1	Laptop specifications	24
Table 4.2	Activity Done in Final Year Project 1	36
Table 5.1	Summary of experimental model results	46
Table 5.2	Results obtained from testing dataset	55

LIST OF ABBREVIATIONS

<i>MRI</i>	Magnetic Resonance Imaging
<i>UI</i>	User Interface
<i>2D</i>	2-Dimensional
<i>3D</i>	3-Dimensional
<i>FLAIR</i>	Fluid Attenuated Inversion Recovery
<i>CSF</i>	Cerebrospinal Fluid
<i>ROI</i>	Region of Interest
<i>SLIC</i>	Simple Linear Iterative Clustering
<i>TIF</i>	Tag Image File
<i>MIPAV</i>	Medical Image Processing, Analysis, and Visualization
<i>VOI</i>	Volume of Interest
<i>CT</i>	Computed Tomography
<i>OS</i>	Operating System
<i>PACS</i>	Picture Archiving and Communication System
<i>CNN</i>	Convolutional Neural Network
<i>GPU</i>	Graphical Processing Unit
<i>IDE</i>	Integrated Development Environment
<i>API</i>	Application Programming Interface
<i>BCE</i>	Binary Cross Entropy
<i>RAM</i>	Rapid Access Memory

CHAPTER 1 INTRODUCTION

1.1 Introduction

Brain tumor is a deadly disease that stems from cancerous cell growth in the regions of brain that could become fatal without proper care and handling. According to [1], the average survival rate for all primary brain tumor patients is only 75.2%. This shows how dangerous this disease can be, especially when the tumor growth is not monitored. Due to increasing need for medical technology advancements, medical imaging field has been growing at a super rapid rate to accommodate the needs of image segmentation tools. Brain tumor Magnetic Resonance Image(MRI) is a type of medical images that are built through scanning the human brain using powerful magnets are commonly used for brain tumor segmentation.

Figure 1.1 below shows some examples of 2-Dimensional(2D) brain tumor Magnetic Resonance Image(MRI) scan and the tumorous cell growth labelled as green. There are many modalities in a brain MRI such as T1, T2 and Fluid Attenuated Inversion Recovery(FLAIR). In T1 modality, the high fat tissues, called white matter appears bright and the water-filled compartments called cerebrospinal fluid(CSF) appears dark and mainly used for anatomical demonstration[2]. On T2 modality, the CSF will appear brighter instead and the white matter will appear darker and this modality is useful for demonstrating most damaged tissues such as when doing tumor segmentation[2]. The scans below are of the Fluid Attenuated Inversion Recovery(FLAIR) modality. In this modality, the T2 effect is stronger due to the suppression of CSF in the brain and it brings the contrast between the gray and white matter to minimal[3]. With this modality, the cancerous tissue can be easily detected making it much easier to visualize and segment the overall structure of the tumor.

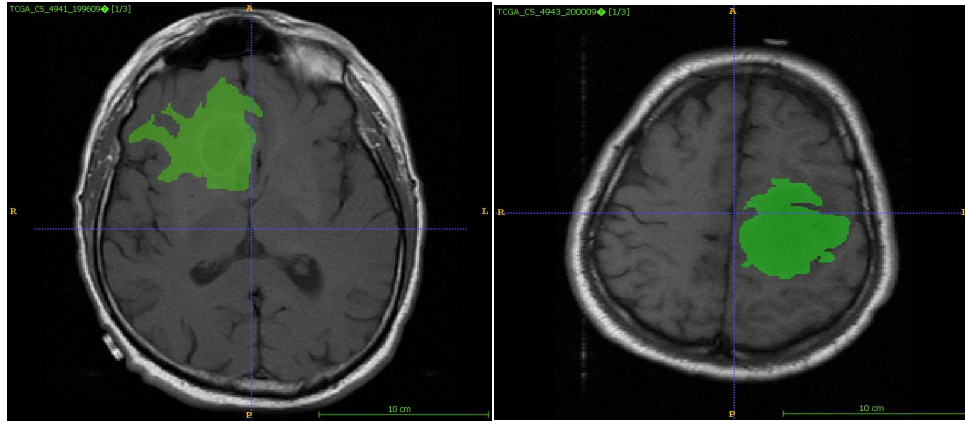


Figure 1.1 2D MRI brain tumor images. The tumor regions are labeled in green.

Physicians are often required to examine the various structure of tumor region growth, manually. This is a time consuming process. In order to circumvent this problem, image segmentation has been used in medical imaging processes to aid the physicians to detect the tumor region. This is done by extracting the tumor region as region of interest (ROI) through a semiautomatic or automatic approaches[4].

1.2 Problem Statement and Motivation

There are many existing software that are used for research and medical analysis to perform brain tumor segmentation. These software are MIPAV, ITK-SNAP and Olea Sphere 3.0 which have their own weaknesses. They perform conventional tumor segmentation using automatic or interactive image segmentation approaches namely pixel wise segmentation techniques such as thresholding and region based segmentation techniques such as watershed and Simple Linear Iterative Clustering(SLIC) superpixels. These processes are labor intensive as they require human intervention and complicated as it commonly requires users to enter high number of hyperparameters. In addition, it may be complex for novice users or researchers to use these software. As for existing deep learning based software such as DeepMIB and NiftyNet, they are commonly used for general purpose biomedical images segmentation instead of dedicating for brain tumor segmentation and requires users to train models before using them.

As such, the issues with the existing software for brain tumor segmentation has motivated the initiation of this proposed project. With this project, the aim is to develop a web application that is able to perform automatic brain tumor segmentation using deep learning techniques. For this purpose, a uNet+segUnet ensemble model is proposed for the segmentation in this work. In addition, the proposed web application is built with Flask which acts as a lightweight platform with User Interface (UI) to allow users to upload brain MRI slices and view the segmented tumor region through the web application instead of downloading the application and running the code on local machine. The users are also allowed to upload a ground truth image for their uploaded MR image for evaluation using Jaccard similarity index.

1.3 Project Scope

The scope of this proposed system are as following :

- 1) 2-Dimensional(2D) MRI brain tumor images of Fluid attenuated inversion recovery (FLAIR) modality in Tag Image File format (.TIF) are used obtained from [5] and [6].
- 2) The development of an automatic brain tumor segmentation application to perform accurate tumor segmentation to some extent using deep learning approach and hyperparameter fine-tuning. The deep learning architectures used in this work are uNet[7] and SegUnet[8] which is based on hybridization of Unet[7] with SegNet[9].
- 3) The web application is deployed using Flask, which is a micro web framework written in Python that connects a web-based User Interface (UI) to the trained deep learning model.
- 4) The rules derivation in the brain tumor region segmentation identification is based on the common understanding (or expert's opinion) on the datasets. Expert's opinion is required in the segmentation identification of the tumor regions in the 2D brain images.

1.4 Project Objective

The main objective of this project is to develop a web application that is made solely for 2D brain tumor segmentation using deep learning approach. The following are sub-objectives to achieve this :-

1. To implement a model using deep learning architecture to automate 2D brain tumor segmentation through feature extraction(encoding) and tumor region prediction(decoding).
2. To improve the built deep learning model in (1) with ensemble learning by combining predictions from two model architectures, namely uNet[7] and SegUnet[8] which is a hybrid model produced by combining architecture of uNet[7] with SegNet[9].
3. To deploy the proposed uNet+segUnet ensemble model in (2) as a web application through Flask which provides a user interface(UI) and output the segmented tumor results through the application for further analysis.

1.5 Impact, Significance and Contribution

Many positive impacts could be derived from implementation of this project. The development of this brain tumor segmentation web application will not only benefit the medical sector to detect and keep track of the tumor growth but also for educational purposes of image processing such as in UTAR or any other institutions. This is because the proposed application is able to automatically segment the 2D brain MRIs accurately to some extent using the proposed deep learning architecture.

Other than that, this project also optimized the deep learning model used for the segmentation through extensive hyperparameter tuning and ensemble learning technique to provide automatic brain tumor segmentation that are as accurate as possible and requires minimal user intervention. This proposed model is deployed as a web application using Python Flask, where users shall be able to view the segmented tumor regions from the User Interface (UI) for further analysis. Since the web application is hosted on local public IP address, it can easily be deployed in organizations or institutions to provide the segmentation service.

CHAPTER 2 : LITERATURE REVIEW

2.0 Introduction

In this chapter, the existing imaging software that are used to segment tumor regions in brain MRIs are reviewed. Among the reviewed software, they are divided into two types as in Section 2.1 and 2.2. The first type are the applications that perform tumor segmentation using the conventional image segmentation algorithms. While the second type are the applications that use deep learning approach to perform the brain tumor image segmentation.

2.1 Review On Non Deep Learning Based Brain Tumor Segmentation Applications

2.1.1 MIPAV

MIPAV which stands for Medical Image Processing, Analysis, and Visualization is a free to use medical image segmentation software that is able to perform brain tumor image segmentation. In this application, brain tumors images that are sent are loaded into the software and users are given the ability to choose from built-in preprocessing and segmentation algorithms such as mean shift and watershed to perform brain tumor segmentation. These algorithms are known as region-based segmentation algorithms. According the official documentations of MIPAV, region based algorithms provided by MIPAV are vulnerable to over segmentation[10]. An example occurrence of over segmentation in MIPAV is as shown in Figure 2.1 when watershed algorithm is applied to perform the tumor segmentation. The tumor region of the brain MRI in Figure 2.1 is segmented into finer region known as over segmentation.

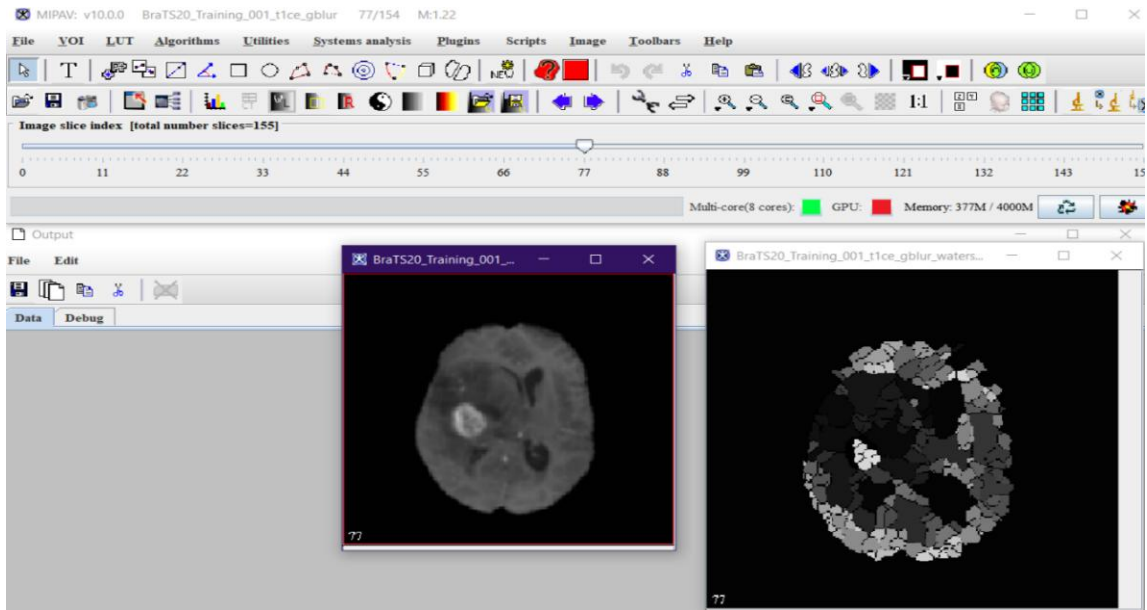


Figure 2.1 Over segmentation occurrence in MIPAV

In an attempt to address the issue, MIPAV software suggests users to specify the region of interest(ROI) or volume-of-interest(VOI) of the MRI[10]. Other ways proposed include modifying the hyper parameter input to adjust the region size to only segment the tumor. However, this often comes with the need of the user to manually specifying each and every region and also requiring the users to input a lot of hyper parameter to produce better segmentation as shown in Figure 2.2. Thus, for high resolution of MRIs to be segmented, this interactive method is labor intensive and time consuming.

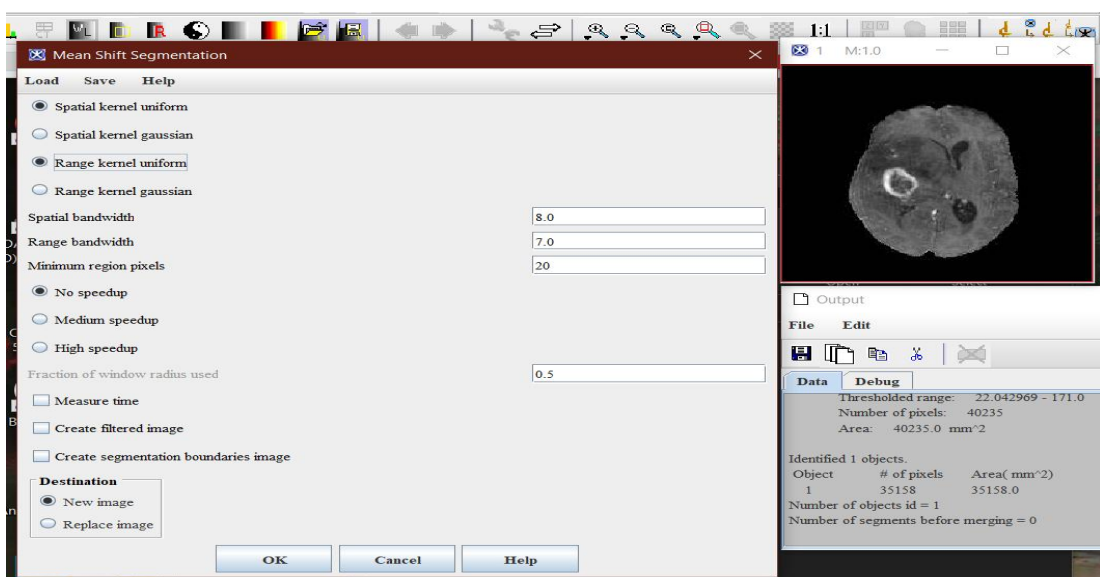


Figure 2.2 Hyperparameter input requirement for mean shift segmentation

2.1.2 ITK-SNAP

ITK-SNAP is another free and non-deep learning based medical image segmentation software that can be used for brain tumor segmentation. ITK-SNAP supports multiple image file formats namely generic ITK images like .TIF or medical volume data like .GIPL and DICOM and further provides free documentations to operate the software. ITK-SNAP provides a semi-automatic image segmentation option. In this context, the semi-automatic refers to the tumor segmentation performed partially automatic when the brain MRI is loaded into the software for the tumor segmentation. ITK-SNAP requires manual tweaking and setting up of region of interest(ROI) in the brain MRI before segmentation to ensure correct location is used to detect the tumor region.

In ITK-SNAP, active contour algorithm is used to perform the semi-automatic brain tumor segmentation. By using the semi-automatic active contour algorithm, the tumor regions has to be marked using a bounding box and it requires a cycle of pre-segmentation, initialization and evolution for segmentation.

ITK-SNAP also provides manual segmentation using polygon or paintbrush feature to draw polygon over the tumor region or paint over the tumor slice by slice using a series of left clicks[11]. This process is tedious and labor intensive especially if high number of MRIs to be segmented. Figure 2.3 shows the manual segmentation approach through paintbrush to segment the tumor region.

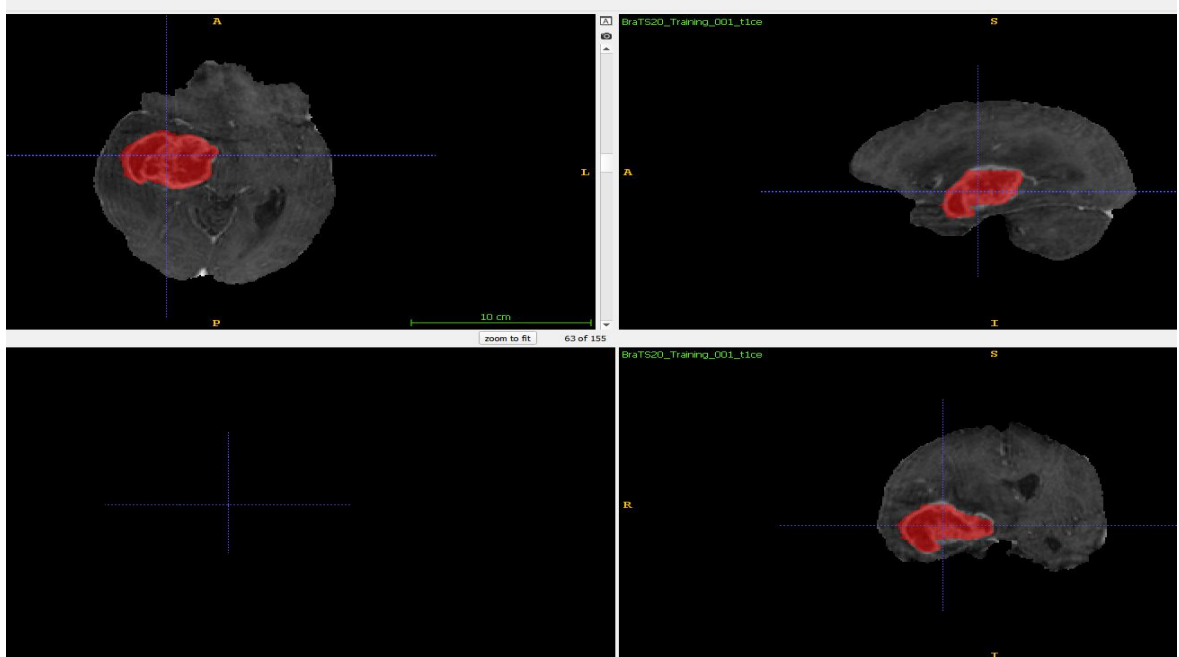


Figure 2.3 Semi-Manual segmentation results in ITK-SNAP

2.1.3 Olea Sphere 3.0

Olea Sphere 3.0 is a neurology application by Olea Medical that provides complete computer tomography(CT) and MRI solution. According to the user manual of Olea Sphere 3.0, this software is made for visualization of medical images, analysing and processing software packages called Picture Archiving and Communication System(PACS) while running on Windows, Macintosh or Linux operating system (OS)[12].

According to the user manual of the software, there are many features available in Olea Sphere 3.0 for the purpose of neurosurgery analysis such as tumors. Olea Sphere 3.0 provides vendor-neutral neurology solution in aiding fast tumor segmentation. This software can be used mainly to visualize and analyze images acquired through MRI such as brain tumor MRIs by using dynamic imaging data from the MRIs and showing properties of changes in contrast over time such as the tumor growth.

According to the official documentation, Olea Sphere 3.0 is said to be fast in segmenting benign and malignant differentiation in CT and MRI brain tumor analysis[12]. Other features of Olea Sphere 3.0 include automatic region of interest (ROI) defining through growing ROI feature called magic wand to aid in ROI comparison. However, Olea Sphere 3.0 is not an open source software nor does it implement a deep learning architecture for providing better segmentation results. Users are required to purchase the software in order to perform the brain tumor analysis and segmentation. In this case, users will not have much hands on experience on the software as it is locked behind a pay wall.

2.2 Review On Deep Learning Based Brain Tumor Segmentation Applications

2.2.1 DeepMIB

DeepMIB is a biological image segmentation open-source software which aimed for training deep learning networks. Its application involves segmentation for 2D and 3D images for usage in multiple biological fields whilst providing more options for freedom with the type of data used. It provides the deep learning network services through an open source multi-platform MATLAB code and provides a standalone application for Windows version.

DeepMIB provides 4 deep learning network architecture for the purpose of biomedical image segmentation. The 4 provided architectures are uNET, 3D uNet, 3D U-Net designed for anisotropic datasets and 2D SegNet[13]. The application accepts

both standard (.tif or .png) and microscopy(biomedical) image formats. Users are given the flexibility to adjust critical parameters such as input patch size, if convolution padding needed, number of output classes and the preferred depth of the network for tuning the application to work with different datasets. Data augmentation is applied to training images sent to the application such as reflection, rotation, scaling and shear[13]. Normalization settings is provided for the input layer through training parameters tweaking for any need of final fine-tuning in DeepMIB and network checkpoints are allowed to be stored after each iteration and training can be continued after those steps by loading back the checkpoint[13]. Figure 2.4 shows the overall UI of DeepMIB that includes the preprocessing, training, and predictions for a biomedical image.

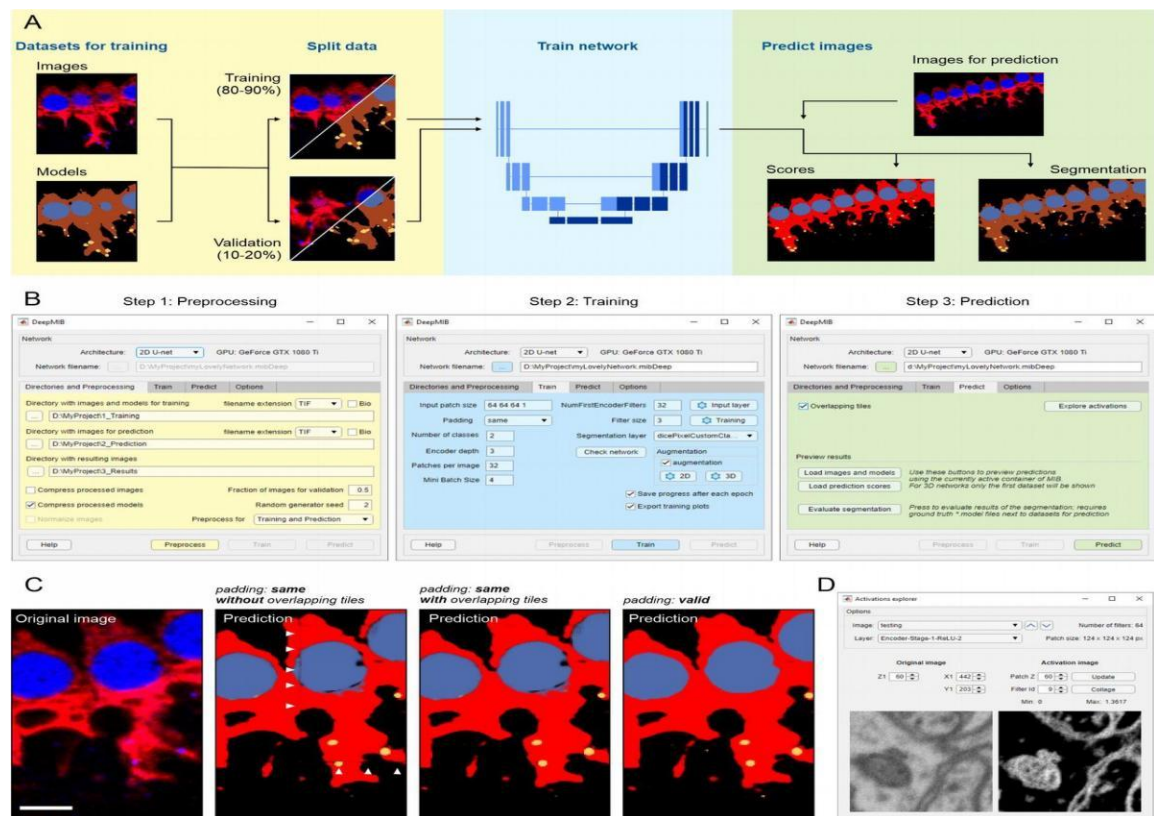


Figure 2.4 DeepMIB overall workflow

There are some shortcomings in this application such that the licensed MATLAB version being able to potentially give better perspective of the model and its results during testing as compared to the standalone application[13]. In addition, the application requires the end users to train the models, by tuning the critical parameters such as the depth of the network chosen. This may become complex and time consuming process to the users.

2.2.2 NiftyNet

NiftyNet is a TensorFlow based open source convolutional neural network(CNN) platform that is open source and free to use for public. It is made for medical image analysis and designed for networks and pre-trained models sharing [14]. NiftyNet is able to adapt the existing CNN architecture to user dataset and allows quick building of new solutions for user's own image analysis problems. NiftyNet provides an easily customizable network component interfaces, allows support for multiple data input types and provides multiple-GPU support for efficient training models.

In NiftyNet, 4 recent implementation of deep learning network architectures are performed, namely HighRes3DNet, 3D U-net, V-net and DeepMedic. Figure 2.5 below shows an overview of NiftyNet providing multi-organ abdominal CT segmentation[15].

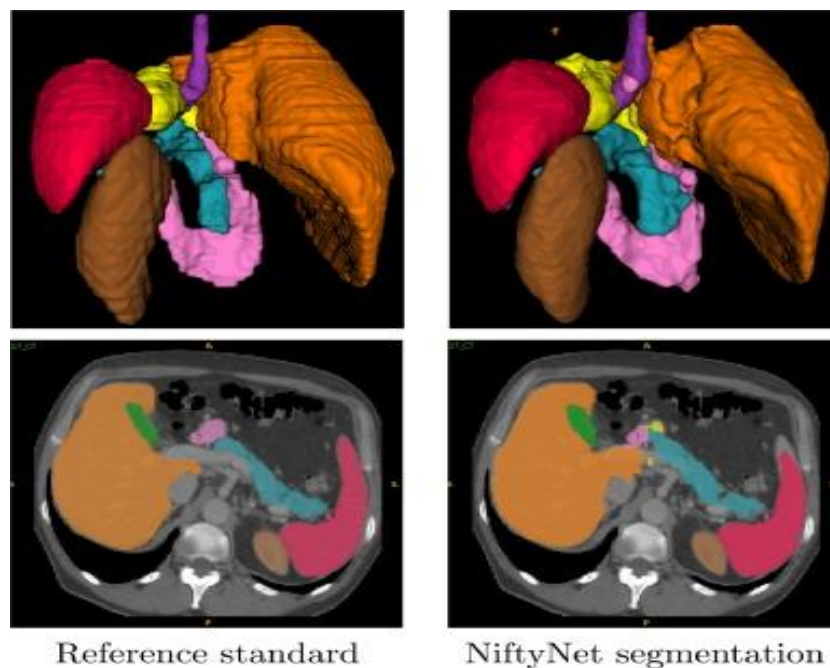


Figure 2.5 An example of multi-organ abdominal CT segmentation using NiftyNet

In NiftyNet, the users are required to train the model by themselves, and so it may not be a suitable application if users want to segment only a certain type of biomedical image such as brain MRI for tumor since they will have to understand how to implement the model and critical parameters to run the training. Furthermore, due to the nature of the application being made for 3-Dimensional(3D) images only[14], the application tends to struggle when working with 2-Dimensional(2D)

images such as .TIF format due to unavailability of 2D based CNN network architectures such as 2D uNet.

2.3 Critical Remarks

Table 2.1 Summary of MIPAV

Advantages	<ol style="list-style-type: none"> 1. MIPAV is free to use. 2. Provides multiple choices of conventional image segmentation algorithms such as watershed and mean shift region based segmentation algorithms.
Disadvantages	<ol style="list-style-type: none"> 1. Region based segmentation algorithms provided are prone to over segmented tumor images. 2. Requires complex hyperparameters adjustments to reduce over segmentation. Thus, it is time consuming and complicated for novice users 3. Too many features in MIPAV where novice users might have troubles learning to use the software. 4. No implementation of deep learning architecture for segmentation purposes.

Referring to Table 2.1, MIPAV is a software for tumor segmentation with high number of conventional image segmentation algorithms provided. Complex hyper parameter input is required which is difficult to be used by the novice users. Furthermore, using MIPAV for brain tumor segmentation is prone to oversegmentation problem. MIPAV also does not provide deep learning architecture for segmentation purposes.

Table 2.2 Summary of ITK-SNAP

Advantages	<ol style="list-style-type: none">1. Supports multiple image file formats2. Free to use3. Provides documentations to operate software
Disadvantages	<ol style="list-style-type: none">1. Complex parameter inputs and requires user intervention to perform the semi-automatic brain tumor segmentation.2. Longer segmentation runtime and if initial contour not provided, the semi automatic segmentation algorithm would not work.3. No implementation of deep learning architecture for segmentation purposes.

Referring to Table 2,2, ITK-SNAP is a simple and easy to use tool for tumor segmentation but requires complex parameter inputs and user intervention to perform semi-automatic brain tumor segmentation. Furthermore, the semi automatic segmentation algorithm in ITK-SNAP would not work without initial contour being provided. ITK-SNAP also does not provide deep learning architecture for segmentation purposes.

Table 2.3: Summary of Olea Sphere 3.0

Advantages	<ol style="list-style-type: none">1. Enable good segmentation algorithms that is able to provide fast segmentation results.2. Provide visualizations and analysis of the segmented tumor regions.
Disadvantages	<ol style="list-style-type: none">1. Not free to use / Paid software2. No implementation of deep learning architecture for segmentation purposes.

Referring to Table 2.3, Olea Sphere 3.0 can be used for tumor segmentation, however, it is not free.. Furthermore, Olea Sphere 3.0 do not have any implementation of deep learning architecture for segmentation purposes.

Table 2.4 Summary of DeepMIB

Advantages	<ol style="list-style-type: none">1. 4 CNN architecture are provided for end users to work with for biomedical image segmentation whilst using the application.2. Users are given the flexibility to adjust critical parameters.
Disadvantages	<ol style="list-style-type: none">1. Requires MATLAB license to use the application in MATLAB version.2. Complex to use as it is made for general purpose biomedical image segmentation, and not solely for brain tumor segmentation.

Referring to Table 2.4, DeepMIB is a software that uses deep learning approach for biomedical image segmentation. By using 4 different CNN network architecture to work with and with the ability to train the model from the user end point, it allows flexibility on suiting the application to needs. However, DeepMIB would require a MATLAB license to run the application in MATLAB version. In addition, the need for training of model by the user, adds to complexity in using the application.

Table 2.5 Summary of NiftyNet

Advantages	<ol style="list-style-type: none"> 1. Open-source and free 2. Provides 4 CNN architectures for biomedical image segmentation whilst using the application.
Disadvantages	<ol style="list-style-type: none"> 1. Complex to use as it is made for general purpose biomedical image segmentation, and not solely for brain tumor segmentation. 2. Does not work with 2D image formats for segmentation due to no implementation of 2D CNN architecture.

Referring to Table 2.5, NiftyNet uses 4 different CNN network architecture for biomedical image segmentation and allowing users to train the models for segmentation. Due to the application being general purpose, it is complex to use when used for brain tumor segmentation only. Besides that, due to no implementation of 2D CNN architecture, the application does not work properly for 2D brain tumor slice images.

CHAPTER 3 SYSTEM DESIGN

3.1 Flowchart of the Overall Flow of System

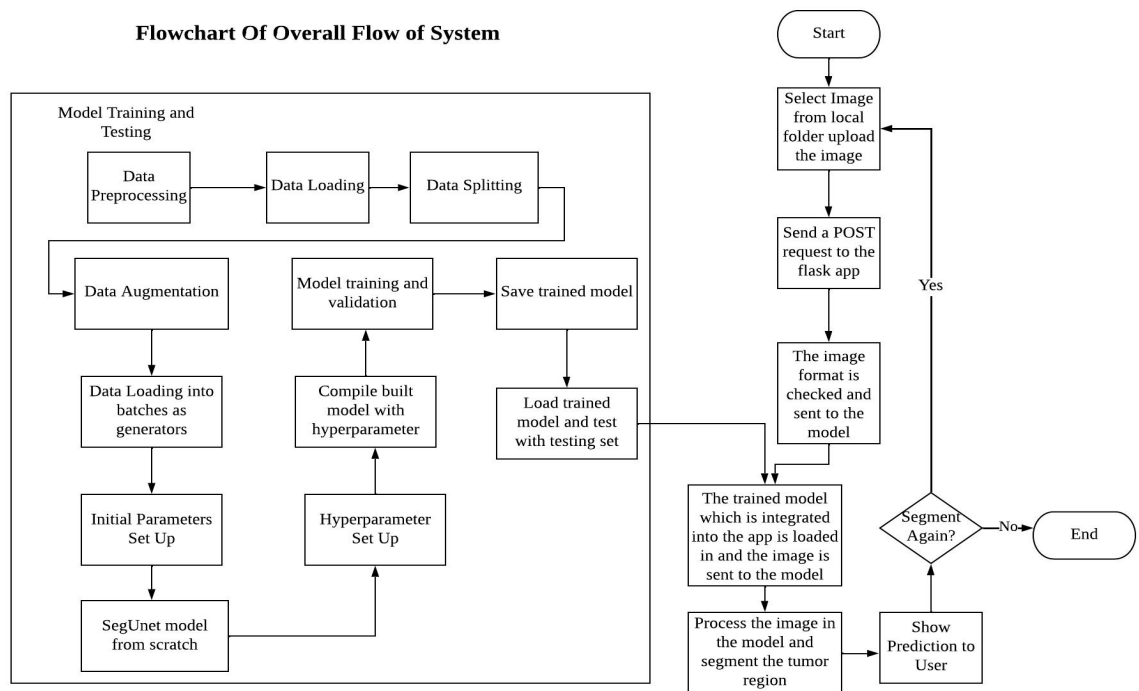


Figure 3.1 Flowchart of the proposed system

The Figure 3.1 above shows the overall flow of the web application and how the trained and tested uNet+segUnet ensemble model is integrated to perform brain tumor segmentation. At the web page, the users have to click the “upload” button to choose a 2D FLAIR brain MR image file from their local folder in the possible formats of “jpg”, “jpeg”, “tif” and “png”. A POST HTTP request is produced by the user to send the image to the model for segmentation. The uploaded image is then checked for format correctness and security through Flask and werkzeug.utils library and if there is no issue, the image will be passed on to the application.

The image is preprocessed in terms of the image size and are converted into tensors where each pixels are represented with 0 to 1 values instead of the usual 0 to 255 pixel intensities as deep learning models cannot be trained by using pixel intensities. Then, the image is sent to the trained uNet+segUnet ensemble model for testing through Google Colab. The uNet[7] and segUnet[8] model architectures are loaded in with their trained weights and are used to predict the

boundary of brain tumor region for the uploaded image and their combined output is used for the final segmentation output. Later the segmented brain tumor region will be displayed to the users on the UI. If the users have a ground truth for the uploaded MRI, they are allowed to do evaluation of the prediction through Jaccard score. This would be beneficial for novice users that are learning about how brain tumor segmentation or for researchers who are working on brain tumor segmentation. If users desire to segment another image, they can upload another brain tumor image on the homepage for another POST HTTP request for segmentation.

3.2 Activity Diagram



Figure 3.2 Activity Diagram of Proposed System

Figure 3.2 above shows the activity diagram used to show how the user, system and the deep learning model interact with each other to enable smooth user experience while using the web application. The user first opens their web application that is hosted under the public IP of their home or work network. This IP will be connected to port 5000 to enable the user to see the web

application. Upon connection, the users are shown the homepage of the system where the system presents the user with a menu to choose a 2D brain MR image for uploading. The user selects an MR image from their local folder and upload the image using the system. The system receives the uploaded image and check for format correctness(jpg, jpeg, png or tif formats are accepted).

If the image has no format issues, then it is stored in a static folder. Else, it will require the user to enter MR image that is in correct format. Then, the stored image is sent to the deep learning model for segmentation. This process occurs simultaneously with the model being loaded with its trained weights. As such, there is no delay between the model receiving the image and the model loading its trained weights. The segmentation process will take a few seconds of loading and after it is performed successfully, the segmentation result is sent to the static folder to be stored and is displayed on the screen through the folder. The users are then able to save the segmented image if desired and are allowed to segment another image by uploading another 2D brain tumor MR image into the application. The users are also able to do evaluation on the segmented tumor by uploading a ground truth image corresponding to the uploaded 2D brain tumor MR image. The Jaccard score is computed and displayed to the user from the application.

3.3 Verification Plan

To verify that the deep learning model is able to handle multiple situations where it may not usually work, verification plan is done and explained as follows :-

Table 3.1 First verification plan

Procedure Number	P1
Method	Testing
Applicable Requirements	Detect the brain image data that would not fit the data type required by the input for the deep learning model.
Purpose/Scope	To increase the robustness and error handling of system
Item Under Test	2D brain tumor MR image
Precautions	None
Limitations	None
Equipment/Facilities	Laptop
Data recording	None
Acceptance Criteria	The system should detect the brain image data that is not in the right input format, that is, if the image is not in 2-Dimensional(2D) format.
Procedure	1. Input to the web application any brain tumor MRI that is not in 2D format 2. When wrong input image format is loaded for segmentation, the deep learning model should detect the difference in input dimension and displays error.
Troubleshooting	Iterate this procedure
Post-Test Activities	None

Table 3.2 Second verification plan

Procedure Number	P2
Method	Analysis
Applicable Requirements	Analyse the suitable split of ratios for the training, validation and testing while ensuring no system performance degradation.
Purpose/Scope	To improve the accuracy of the trained model on validation and reduce overfitting
Item Under Test	Number of brain tumor images in dataset
Precautions	The number of training images should be much higher than validation and testing set.
Limitations	If the training images is less then validation and testing, the system may work with lower segmentation accuracy
Equipment/Facilities	Laptop
Data recording	None
Acceptance Criteria	Analyse the suitable ratio for the training, validation and testing set splitting.
Procedure	<ol style="list-style-type: none"> 1. Load in brain tumor images from the dataset 2. Split the models in different ratios that are commonly used in machine learning, such as 8:1:1, 7:2:1 or 6:2:2 [16] 3. Analyse the impact on the model's performance
Troubleshooting	Iterate this procedure
Post-Test Acitivities	None

Table 3.3 Third verification plan

Procedure Number	P3
Method	Testing
Applicable Requirements	Detect the brain image data that would not fit the image type required for uploading
Purpose/Scope	To increase the robustness and error handling of system
Item Under Test	2D brain tumor MR image/ 2D brain tumor ground truth image
Precautions	None
Limitations	None
Equipment/Facilities	Laptop
Data recording	None
Acceptance Criteria	The system should detect the chosen 2D brain tumor MRI or ground truth image that is not in the right input format, that is, if the image is not in .jpg, .jpeg, .png or .tif format.
Procedure	<ol style="list-style-type: none"> 1. Choose 2D brain tumor MRI or ground truth images that are not in .jpg, .jpeg, .png or .tif format for uploading in web application 2. When wrong input image format is loaded for segmentation, the backend in Flask should detect the wrong format and flash an error message.
Troubleshooting	Iterate this procedure
Post-Test Activities	None

3.4 Chapter Summary

Chapter 3 covers the web application flows from the user to the server side and the overall design of the proposed system. In addition to that, verification of the project is also carried out and presented in this chapter.

CHAPTER 4 METHODOLOGY

4.1 Methodology

The methodology chosen for the proposed brain tumor segmentation application is the Iterative and Incremental software development methodology. Figure 4.1 shows the general model of the methodology used[17].

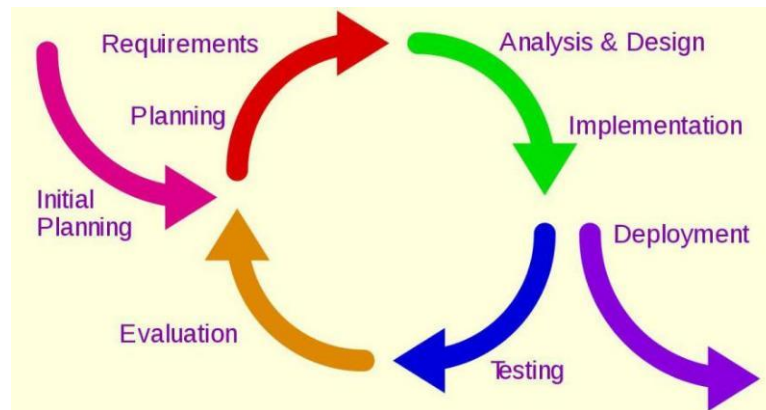


Figure 4.1 Iterative and Incremental Model

The simple idea of this methodology is that the development of the system using chosen deep learning architecture is done through repeated cycles (iterative) and in smaller portions at a time (incremental)[18]. This can be advantageous because prior obtained knowledge can be used for later parts of the project and problems can be fixed incrementally. This methodology is split into several phase for each cycle, namely planning, analysis, design, implementation, testing and evaluation and when all the cycles were enough to solve the problems, the actual deployment of the software is done[18]. This methodology is applied in two stages for this project. First is the development of the deep learning model through iterative and incremental technique and second is the development and deployment of the application implemented using the developed deep learning model.

In the planning phase, initial planning of the system is performed to know the project details and the outcome. In this phase, the required technologies for the proposed software development is understood. It is a

necessary phase to understand the materials needed, cost and the time constraint of the project before executing the project.

While, in the analysis phase, the system's requirements is analysed and understood. This is important as the requirements shape the later stages of the development. In this phase, information about the 2D dataset is understood and analysis of existing software is done to generate user requirements. Design phase refers to designs solutions with which the project result can apparently be achieved. In this phase, the deep learning network architecture, the User Interface(UI) design and all the tools and libraries needed to build the model are identified. This phase is followed by the implementation of the application according to the design specifications.

In the testing and evaluation phase the application is tested and evaluated for accuracy. This is important in order to make sure there are no bugs or problems when the application is being deployed. In the iterative and incremental methodology, the cycle is iterated to planning phase in order to improve the application system[18]. Once the built model provides expected results and is integrated into the application, final deployment will be done to allow users to use the application for the brain tumor segmentation purpose

4.2 Tools To Use

Hardware

Laptop

Table 4.1 Laptop specifications

Operating System	Windows 10 and above
Processor	2.30 GHz Intel Core i5
RAM	At least 12GB of RAM required, Smaller RAM will cause slower processing.
System Type	64 bit Operating System, x64 based processor
Graphical Processing Unit (GPU)	NVIDIA GeForce GTX 1050

Software

- Google Colaboratory

Google Colaboratory or *Colab* in short, is a web-based integrated development environment(IDE) that is invaluable for streamlining work and keeping track of the segmentation algorithm development progress. Jupyter Notebook in the Google Colab allows code and the results associated with it to be seen at the same time. Thus, it is convenient for errors checking and visualizing the model for the training, validation and testing processes..

- Python Language

For this project, most of the system is built on top of *Python* programming language. This is because *Python* provides many useful built-in libraries for image processing and machine learning using deep learning algorithm. Due to the easy to read nature of the language, building the model for training can be significantly simplified and easily visualized throughout the coding.

- Tensorflow/Keras

Keras is an Application Programming Interface (API) that is written in Python language. Due to the easy to read nature of Python language, *Keras* is a high-level API that is available as open-source library for building neural

networks. Keras is fast in carrying out experiments whilst building a deep neural network and runs on top of *Tensorflow*.

- *Flask*

Flask is used for the deployment of the deep learning model and to set up the proposed application for public usage. Flask is a micro web framework written in Python and does not require particular tools or libraries to build the web application. As such, it is lightweight and easy to integrate with the model proposed in this work. With Flask, segmentation requests from the user are sent to the proposed deep learning model, and the model sends the segmented image result to the user.

- *Pycharm*

Pycharm is an integrated development environment(IDE) that is used in computer programming that makes use of Python language. This IDE will be used when creating the web application using Flask that will provide a User Interface(UI) for public usage.

4.3 User Requirement

- The user should be able to choose a 2D FLAIR brain tumor MRI from their local directory for segmentation.
- The user should be able to upload the chosen MRI for segmentation.
- The user should be able to view their uploaded MRI after being sent for segmentation through the web application UI.
- The user should be able to view the segmented tumor region of their uploaded MRI through the web application UI.
- If available, the user should be able to upload a ground truth image for the uploaded MRI for Jaccard score computation indicating the accuracy of the segmentation.
- The user should be able to save the segmented tumor region image and restart the segmentation for another 2D FLAIR brain tumor MR image.

4.4 Evaluation Measure

For the brain tumor segmentation, the accuracy of the segmented tumor region is evaluated to indicate the performance of the proposed system. To this end, the deep learning model used in the proposed system in this work aims to provide higher accuracy and low loss values for the tumor segmentation in the validation set while ensuring the model does not overfit. Overfitting is when the model works too good in the training set, but the performance totally declines in the validation set which is a set of images that are “unseen” by the model. The deep learning model is evaluated based on its training and validation loss using binary-crossentropy(BCE) evaluation metrics[19]. BCE can be summarised as equation 3.1

$$L(\hat{y}.y) = (-y \log \hat{y}) - (1-y)\log(1-\hat{y}) \quad (3.1)$$

where \hat{y} is the predicted score and y is the actual label, being 0 for false and 1 for true. When the prediction score outputs a value between 0 to 1, it is compared with the threshold value of 0.5 where if the value is above 0.5, it is set to be a positive(true) class while below 0.5, it is set to be negative(false) class. BCE predicts how different is the prediction compared to the ground truth(mask) based on the threshold set which is 0.5. As the predictions in the training and validation differs more from the ground truth(mask), the loss increases. The convolution layers aim to reduce this loss through extracting the features and understanding the patterns or elements present in the input images. This process is iterated over for each samples in a batch(batch size is set to be 8 in this work). Every time the samples in a batch completes training and are validated, they will update the loss function and the model will do backpropagation, a process where the deep learning model adjusts its weights(w) and bias(b) to improve its accuracy in the next batch[20]. This is why loss function is an essential part of building a good deep learning model. Thus, the evaluation is done through BCE to ensure higher accuracy of model is obtained.

Since the initially set values of the hyperparameters can affect the proposed system’s performance greatly, experiments on different hyperparameter values are performed to fine tune the model to provide the best performance. This is because, there is not a single value fits for all model in hyperparameter. Only through experiments on the brain MRI, validation, and testing, the hyperparamter values will

be set. In order to test the accuracy of the brain tumor segmentation, the evaluation metrics used are Jaccard similarity index and also binary accuracy.

Jaccard similarity index[21] is shown in equation 3.2

$$J(\hat{y}, y) = |\hat{y} \cap y| / |\hat{y} \cup y| \quad (3.2)$$

where y represents ground truth mask and \hat{y} represent predicted tumor segmentation mask. $\hat{y} \cap y$ represents the intersection(overlap) between the two masks output while $\hat{y} \cup y$ represents the union(combination) of the two mask output. The overlap area produced by the masks are divided by the total sum of the area of both the masks to determine the accuracy of the predicted mask in comparison to the ground truth.

While binary accuracy[22] can be summarised as equation 3.3

$$\text{Binary Accuracy} = (TP + TN) / (TP + TN + FP + FN) \quad (3.3)$$

where True positive(TP) are the actual positive pixels predicted as positive, True Negative(TN) are the actual negative pixels predicted as negative, False Positive(FP) where the actual negative pixels are predicted as positive and False Negative(FN) where the actual positive pixels are predicted as negative. Binary here relates to the binary comparison between the ground truth mask and the segmented tumor mask. This accuracy metric is only used to determine the accuracy of the segmented brain tumor region using the proposed deep learning model during the training and validation phase of the model.

In this work, other than the evaluation metrics stated above, the hyperparameter that are used in the deep learning model are the learning rate, data augmentations, presence of dropout layers, loss function used, batch size and the number of filters set in the convolution block. Tuning on them are done to build multiple different models during model implementation to get a good model for the segmentation. Further details on these hyperparameter tuning and finalized hyperparameters are described in Chapter 5 Section 5.1.10.

4.5 Dataset

In this work, the dataset from Kaggle named “Brain MRI segmentation[5] is used for training, testing and validation purposes. In this dataset, the 2-dimensional (2D) brain tumor MRI with the FLAIR modality are provided for the aforementioned purposes. Moreover, in this dataset, 3929 brain MRI images are provided with its respective ground truth masks in the same directory. The images were obtained from The Cancer Imaging Archive (TCIA). The images are obtained from 110 patients from The Cancer Genome Atlas (TCGA) lower-grade glioma collection. The images are of size (256x256) and are in .TIF format. In this dataset, only 1373 images are used for training, testing, and validation as the remaining images do not show the presence of brain tumor regions.

Figure 4.2 and Figure 4.3 below shows 2D FLAIR brain MRI slice number 30 from patient number 31 and slice number 19 from patient number 21 and their respective ground truth images(mask). The tumor region is labeled in green coloured region in the MRI and as the white coloured region in the mask.

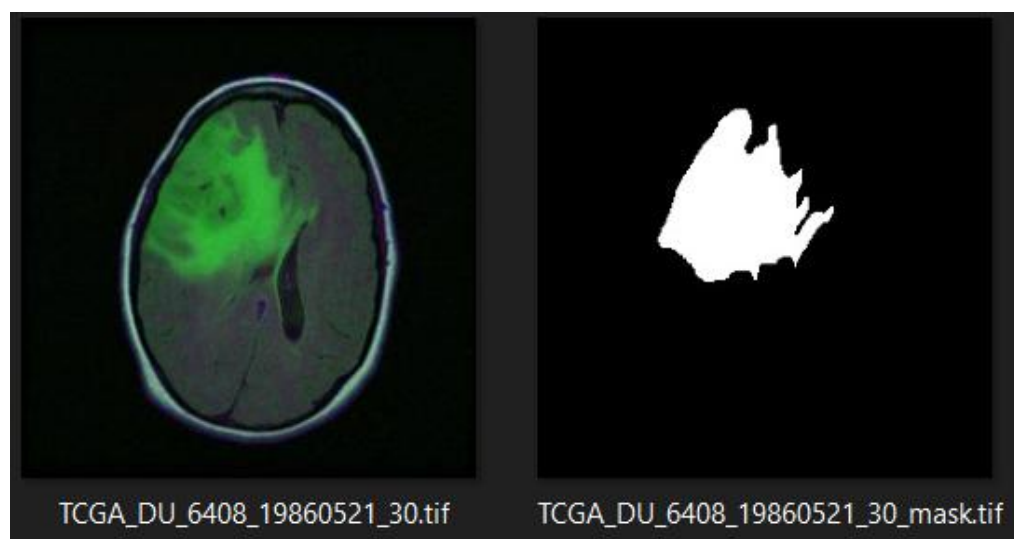


Figure 4.2 FLAIR Brain MRI and the respective ground truth for patient number 31(slice number 30) in dataset[5]

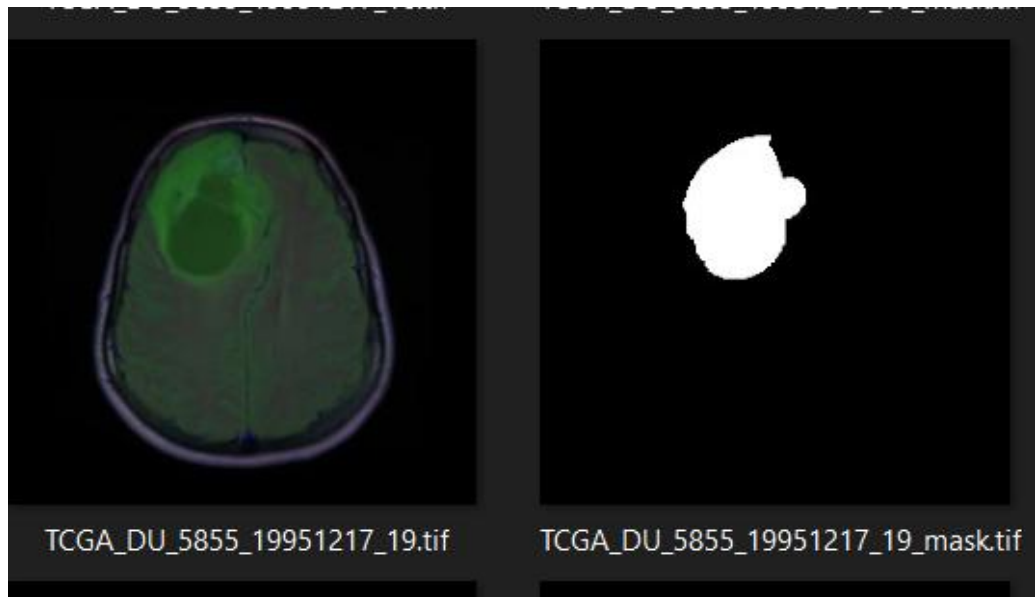


Figure 4.3 FLAIR Brain MRI and the respective ground truth for patient number 21(slice number 19) in dataset[5]

Depending on a single dataset for training, validation and testing is not a indicator for a robust and good deep learning model. And as such, in this work, another 2D brain tumor MRI dataset is used for testing the robustness of the model to segment images that do not belong to the dataset it was trained for. In order to achieve this, dataset named “Brain 2D MRI Images and Mask”[6] is used for testing of the model. This dataset contains 4715 (.h5) files with 2D brain MRI and their respective masks. However, similar to the dataset by Mateusz[5], this dataset consists of MRI slices with and without brain tumor. As such, manual preprocessing is required to extract only the slices that has tumor and its respective ground truth. In the end of preprocessing, 621 images were chosen for testing from this dataset on the proposed model. In addition, the MRI size in this dataset is (240x240) which requires resizing during testing to suit the (256x256) input size requirement by the proposed model. Figure 4.4 below shows FLAIR brain MRI and its mask for slice number 100 for patient number 3 in this dataset[6].

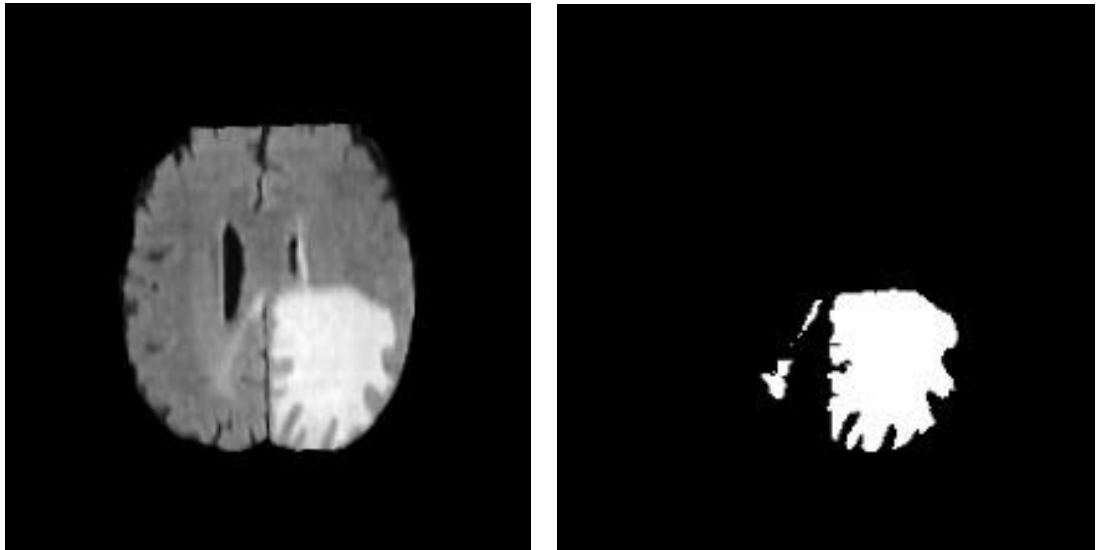


Figure 4.4 FLAIR Brain MRI and the respective ground truth for patient number 3(slice number 100) in dataset[6]

4.6 Model Architecture

4.6.1 uNet based model architecture

Ronneberger et.al[7] introduced a novel CNN based network architecture known as uNet architecture. It crops and concatenates patterns the model learnt from the encoding phase(feature extraction) to the decoding phase(predictions) as shown in Figure 4.5.

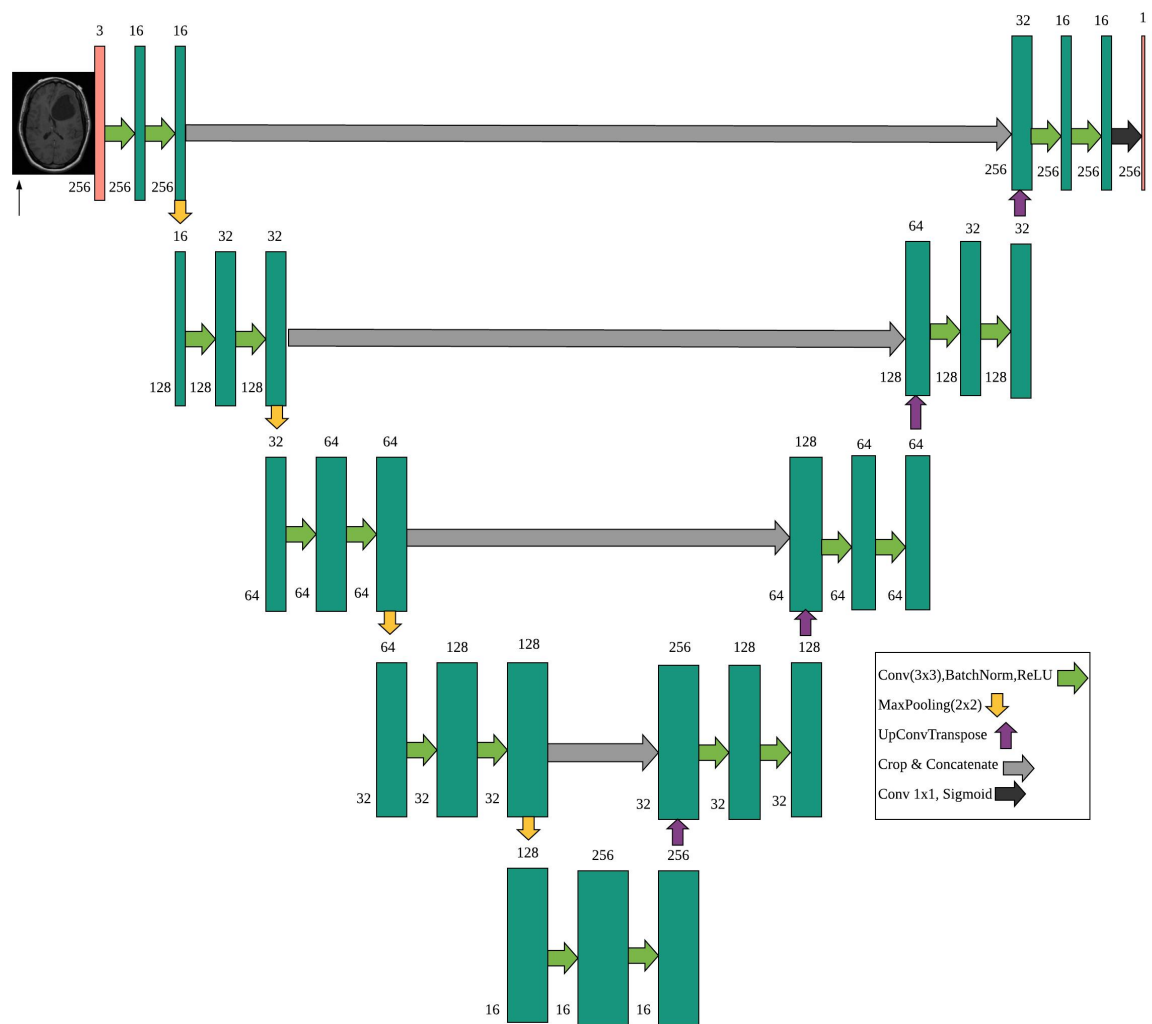


Figure 4.5 Model Architecture made based on original uNet implementation

This network is known as uNet is because of the 'U' like shape the architecture takes during implementation[23]. In uNet, the encoding phase(left side of the architecture which moves from up to down) looks similar to any typical Convolutional Neural Network(CNN). The convolution blocks are built by stacking two 3x3 convolutions(green arrow) and followed by a 2x2 max pooling(yellow arrow) to

downsample the image. Downsampling means reducing the dimension of the target data[23]. At each of the downsampling step, the number of channels is doubled from the initial number of channels set[23].

This downsampling process is then stopped when the model reaches the bottleneck region, which is the lowest part of the ‘U’ shape. After going through the bottleneck region, the data enter the decoding phase(right side of the architecture which moves from bottom to up). In this phase, 2x2 up-convolutions (purple arrow) are done to upsample the data and two 3x3 convolutions(green arrow) are done next. Upsampling means increasing the dimension of the target data[23]. The number of channels is reduced by half after each upsampling step.

Also in the decoding phase, concatenation(grey arrows) of feature maps from the encoding phase are done after every 2x2 up convolutions in order to provide information obtained from the encoding phase. This is very important as without this skip connections that links the encoding and decoding phase, a lot of information are washed away due to loss of border pixels after every convolution[23]. This uNet model will be used in ensemble learning with another model named segUnet which functions similarly to this uNet model, except it has added features of indices pooling.

4.6.2 SegUnet based model architecture

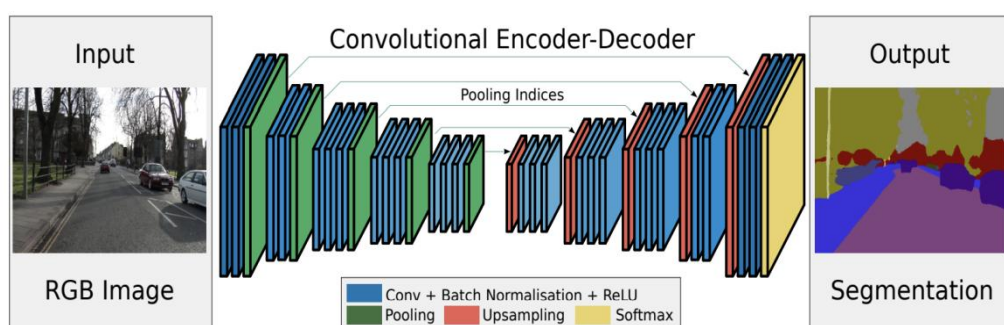


Figure 4.6 SegNet Architecture by [9]

To understand about segUnet[8], which is a hybrid model made by combining features of uNet[7] and SegNet[9], some information about SegNet will be explained first. SegNet is a semantic segmentation architecture that also has encoder and decoder phase similar to uNet and has a pixel wise classification layer to indicate the

tumor or non-tumor region using sigmoid activation[9]. Figure 4.6 above shows the architecture of SegNet introduced by Badrinarayanan et al in 2015[9].

The decoder phase functions to map the low-resolution encoder feature map and bring it to back to full resolution(256x256) which is very similar to how it functions in uNet architecture. However, the novelty of this architecture lies in the way the image is upsampled to bring it back to input resolution. Unlike uNet which uses entire feature map of encoder phase for concatenation in the decoder phase, SegNet uses pooling indices computed in the max-pooling to perform non-linear upsampling and thus uses much less memory compared to uNet. The indices pooling obtained from SegNet is combined with the skip connection that is present in uNet to produce SegUnet, a hybrid deep learning model first introduced by Daimary et al in 2019[8]. This makes the results produced by segUnet model very good and further hyperparameter tuning is done to finalize the segUnet model that will be used for ensemble learning with the uNet model. Figure 4.7 below shows how SegUnet model in the original paper by [8] is implemented.

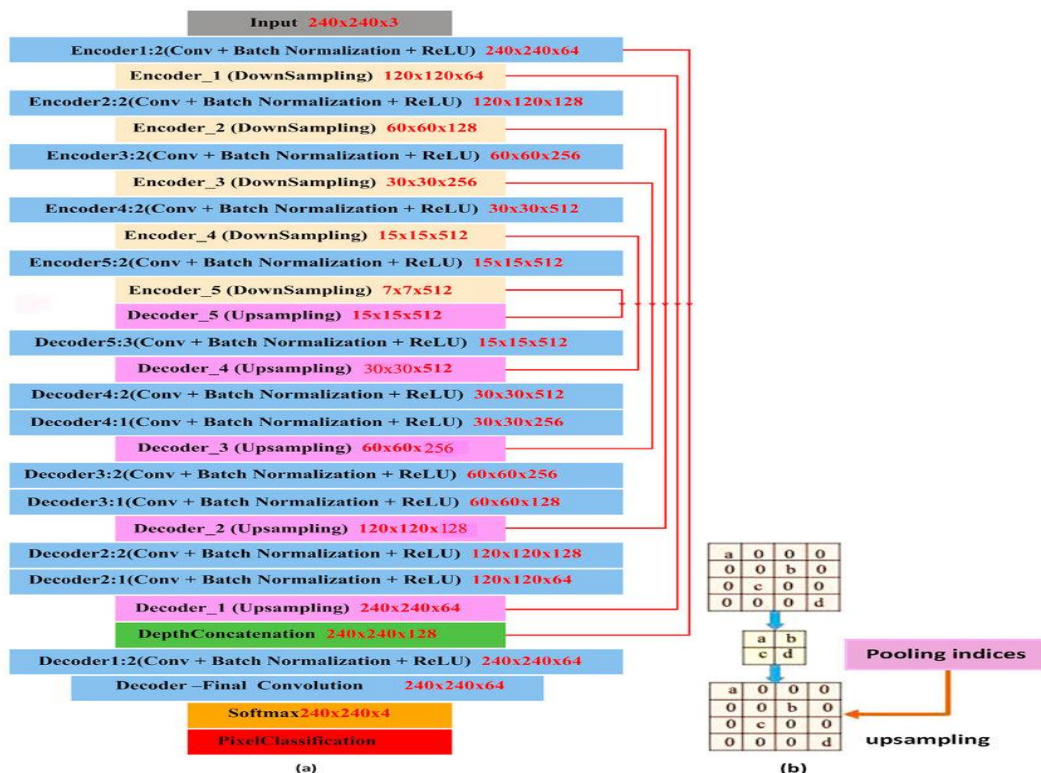


Figure 4.7 Original SegUnet model architecture by [8]

4.6.3 Ensemble learning using SegUnet and uNet

Ensemble learning is simply the combination of a two or more model architectures through a certain combination method that produces combined segmentation. In this work, an ensemble model using uNet(model 1) and segUnet(model2) is proposed for better 2D brain tumor segmentation accuracy.

4.7 Implementation Issues and Challenges

In developing the proposed system, there are some issues that were encountered. One of the biggest challenges is the model training. Training a CNN model requires a long time as the model takes time learning about the images sent to it and as such a powerful Graphical Processing Unit (GPU) is needed for processing. This issue is partially solved by training the model in the Google Colab instead of local machine. In Google Colab, GPU is provided but this does not solve the issue of high utilization of Rapid Access Memory(RAM) from the local machine. Few weeks were taken to switch to a machine with sufficient RAM to allow Google Colab to perform the training model. In addition, in Google Colab without PRO subscription, the usage limit is set to maximum 12 hours and maximum 90 minutes of inactivity. In this case, a constant checking has to be performed while preparing the training model with the chosen images to achieve the intended number of epochs.

Another issue was to acquire the suitable 2D brain MRI dataset. As the model training in this proposed work requires 2D MRI with FLAIR modality, therefore attaining to the suitable dataset with high number of training images was rather challenging. Another dataset is also needed for testing of the model robustness and to prove the ability of the model in segmenting images from other datasets. However, with the guidance and advices from supervisor, this issue was solved and the brain tumor MRI dataset from [5] and “Brain 2D MRI Images and Mask dataset by [6] were chosen.

4.8 Timeline

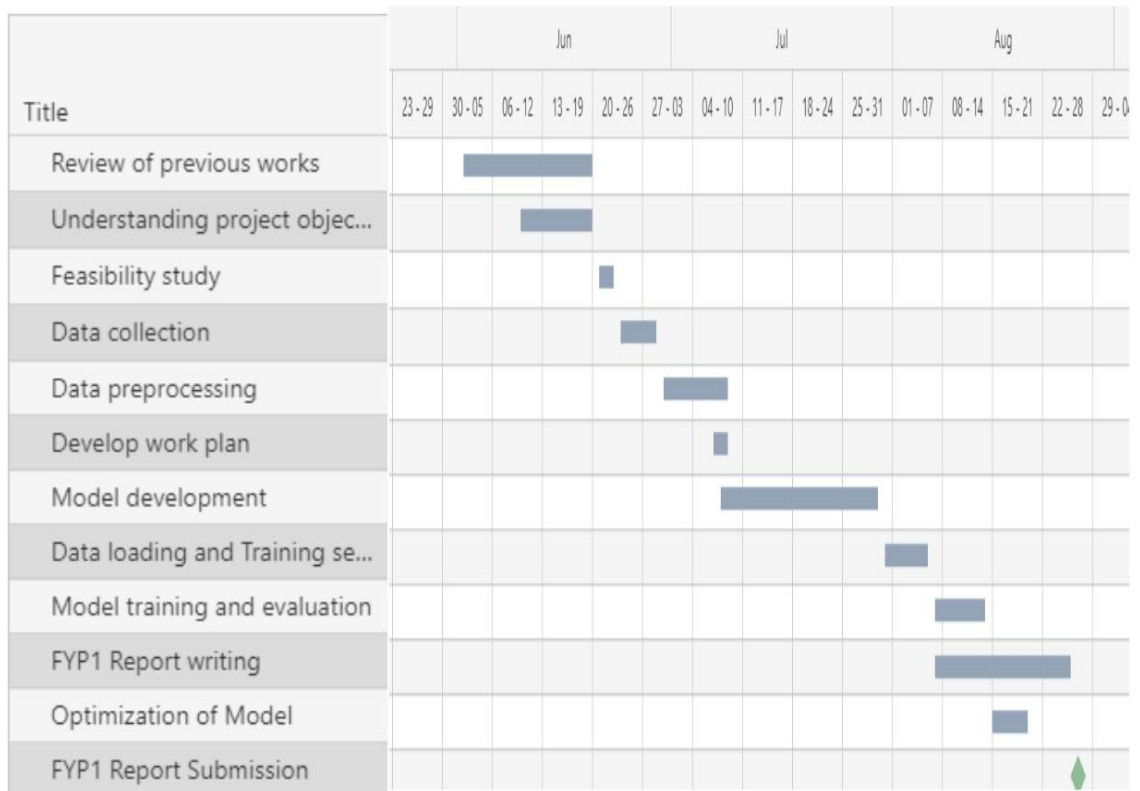


Figure 4.8 Gantt Chart of FYP1 Project Timeline

Table 4.2 Activity Done during Final Year Project 1

Title	Start Time <input type="checkbox"/>	End Time
Review of previous works	06/02/2021	06/20/2021
Understanding project objec...	06/10/2021	06/20/2021
Feasibility study	06/21/2021	06/23/2021
Data collection	06/24/2021	06/29/2021
Data preprocessing	06/30/2021	07/09/2021
Develop work plan	07/07/2021	07/09/2021
Model development	07/08/2021	07/30/2021
Data loading and Training se...	07/31/2021	08/06/2021
Model training and evaluation	08/07/2021	08/14/2021
FYP1 Report writing	08/07/2021	08/26/2021
Optimization of Model	08/15/2021	08/20/2021
FYP1 Report Submission	08/27/2021	08/27/2021

The Gantt chart and the activity table in Figure 4.8 and Table 4.2 above show the timeline and activities performed during the Final Year Project 1 (FYP1). The first activity was to perform, review on the existing works to derive the problem statements and propose the objectives. Later, the feasibility study are performed to derive the project scope for FYP1 which includes the computational requirements. Next the data collection and data preprocessing are performed. In addition, the work plan for proposed deep learning model are structured As per the work plan, the proposed deep learning model including the training, validating and testing are performed on the 2D brain tumor image dataset to segment the brain tumor region. This includes the optimization such as the code refactoring in order to ensure the code is clear, concise and readable. Finally, the activity is concluded with the FYP1 report writing.

Project Task	Project Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. Model Implementations														
1.1 Model research		■	■	■										
1.2 Coding the models		■	■	■	■									
2. Model Training														
2.1 Model Training					■	■	■							
2.2 Model Testing					■	■	■							
2.3 Hyperparameter Tuning					■	■	■							
2.4 Model Refinement					■	■	■							
3. Deployment														
3.1 Model deployment								■	■	■				
3.2 Web app design								■	■	■	■			
4. Report														
4.1 Report Writing								■	■	■	■	■		
4.2 Report submission													■	

Figure 4.9 Gantt Chart of FYP2

The Figure 4.9 above shows the timeline of activities conducted in FYP2. To improve model accuracy, research on other model architectures were done initially. This is followed by performing coding from scratch to then train the model. After training of models, the models are then tested. Hyperparameter tuning were conducted to build more varied models and refine them till the best model is produced. The model is then deployed through Flask starting from week 8 and its design started after the deployment is successful. The report writing for FYP2 started in week 9 and finally the FYP2 ended with the report submission and presentation.

4.9 Chapter Summary

Chapter 4 covered the methodology used in building the web application and the proposed deep learning model that is integrated into the web application. This chapter also explained the evaluation measures, datasets, tools and the model architecture used for building the proposed deep learning model and the web application and explained the challenges faced during the implementation of this project.

CHAPTER 5 : SYSTEM IMPLEMENTATION

5.1 Implementation of proposed uNet+SegUnet model

5.1.1 Importing of Necessary Libraries

```
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.models import Model, load_model, save_model
from tensorflow.keras.layers import Input, Activation, BatchNormalization, Dropout, Lambda, Conv2D, Conv2DTranspose, MaxPooling2D, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

from skimage.io import imread, imshow, concatenate_images
from sklearn.model_selection import train_test_split
from skimage.transform import resize
from skimage.morphology import label
from skimage.color import rgb2gray

import os
import random
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use("ggplot")
%matplotlib inline

import cv2
from tqdm import tqdm_notebook, trange
from glob import glob
from itertools import chain
```

Figure 5.1 Importing of Necessary Libraries

At the beginning of the model creation, all the necessary libraries for the model is first imported as shown in Figure 5.1. Most of the libraries are from tensorflow for model building and hyperparameter set ups, scikit-image, os and cv2 for data loading and splitting and for tumor segmentation visualizations.

5.1.2 Data Preprocessing

In the data preprocessing, processing is performed on the dataset by [5] before using it. Though the dataset is large where 3929 tumor images are provided with ground truth (masks), there are a lot of images that do not actually have tumor regions in it. As such, data preprocessing are performed on the dataset to remove all images and masks of the MRI that do not have tumor region. After the removal of those images, the dataset is left with 1373 images and their respective masks.

5.1.3 Data Loading

```
[ ] MRI_Location = "/content/drive/MyDrive/24August/kaggle_3m"

dirs = []
images = []
masks = []

# loading the file directory names, since the images are stored inside those directory.
# if image contains 'mask' its appened to masks[] and if not into the images[]
for dirname, _, filenames in os.walk(MRI_Location):
    for filename in filenames:
        if 'mask'in filename:
            dirs.append(dirname.replace(MRI_Location, ''))
            masks.append(filename)
            images.append(filename.replace('_mask', ''))

print("The number of images in MRI_Location is : " , len(images))
print("The number of mask images in MRI_Location is : " , len(masks))

The number of images in MRI_Location is : 1373
The number of mask images in MRI_Location is : 1373

[ ] #Storing the MRI Images and Masks into a 2D tabular data format
MRI_DF = pd.DataFrame({'directory':dirs, 'images': images, 'masks': masks})
```

Figure 5.2 Loading data from dataset in Drive and dataframe creation

Based on Figure 5.2 above, the total of 1373 images are then loaded into Google Drive and connected with the Google Colaboratory to initiate data loading. All 1373 tumor images and their respective mask paths are found from the MRI_Location path and are stored in pandas to be stored as a dataframe which is a 2D tabular data format. This MRI_DF dataframe is used to load in all the images from the paths into the Google Colab and sent for data splitting.

5.1.4 Data Splitting

After the loading of all the tumor images and their masks, the dataset by[5] are then to be split into training, validation and testing set. This is a crucial step to ensure the images from the dataset do not overlap. Knowing that the dataset is not in a large amount, namely only 1373 images are used, splitting of data is finalized to be in 80% training and 10% validation and 10% testing or 8:1:1 ratio, putting as much images into the training set as possible to ensure there is no lack of training samples for training. As such, after splitting, the images in each set are 1098 training, 137 validation and 138 testing images. Figure 5.3 below shows the data being split into the aforementioned ratios.

```

train , remaining = train_test_split(MRI_DF, train_size=0.8, random_state=21) # split 80% for training, 20% remaining
test_size = 0.5
valid, test = train_test_split(remaining, test_size=0.5, random_state=21) # split 50% of remaining 20% for 10% validation

print("Total Images before splitting : " , len(MRI_DF))
print("Training Images after splitting : " ,len(train))
print("Validation Images after splitting : " ,len(valid))
print("Testing Images after splitting : " ,len(test))

Total Images before splitting : 1373
Training Images after splitting : 1098
Validation Images after splitting : 137
Testing Images after splitting : 138

```

Figure 5.3 Data splitting in 8:1:1 ratio

5.1.5 Data Augmentation

Data augmentation is a crucial step in any machine learning approach as it solves the issue of model overfitting through introduction of difficult training samples and also artificially increases the number of training samples for the model to train from [19].

```

#In this model, I have finalized on using multiple data augmentation techniques
Augmentation_Collections = dict(rotation_range=0.2, shear_range=0.05,
                                zoom_range=0.05, horizontal_flip=True,
                                fill_mode='nearest')

```

Figure 5.4 Data augmentations are initialized

In this model, extensive data augmentations are used for the training set as the number of training images are quite low, namely: -

- Rotation in the range of 0.2 which rotates the image in clockwise or anticlockwise.
- Shear range of 0.05 which shifts a part of the image to one particular direction and another part to opposite direction.

- Zooming range of 0.05 which zooms into certain location of the image producing more augmented images.
- Horizontal flipping set to true which causes the images to be flipped horizontally, bringing tumor from left side to right side or vice versa.
- Fill mode nearest which replaces points outside the input boundaries with nearest pixels.

All the augmentation techniques are provided by the Tensorflow module called ImageDataGenerator and are defined as shown in Figure 5.4. Total of 5 augmentation techniques are used in the proposed model.

5.1.6 Data Loading in Batches as Generators

Using the ImageDataGenerator from Tensorflow module, the images in the MRI_DF dataframe is loaded as batches of 8 into their respective training, validation and testing sets. While loading them as generators, data augmentation mentioned are performed on the training images. This is not done on validation and testing images. The loading in batches is crucial as training the model without batches will be extremely slow and computationally expensive[17]. The batch size was chosen to be 8 after multiple batch size tuning of the uNet and SegUnet model and are shown in the Chapter 5 Section 5.1

5.1.7 Training parameters set up

```

model.compile(optimizer=tf.keras.optimizers.Adam(), loss='binary_crossentropy',
              metrics=['accuracy', tf.keras.metrics.MeanIoU(num_classes=2)])

callbacks = [
    EarlyStopping(patience=10, verbose=1),
    ReduceLROnPlateau(factor=0.1, patience=5, min_lr=0.00001, verbose=1),
    ModelCheckpoint('/content/drive/MyDrive/TrainedModelNewNewNew.h5',
                   verbose=1, save_best_only=True, save_weights_only=True)
]

```

Figure 5.5 Hyperparameter set up for model training

The parameters needed to train the model are set up in this step. Setting a proper training parameter is crucial to ensure the model gives the best possible output from training. For this model, the optimizer chosen is Adam which is a stochastic gradient descent method to slowly optimize the weights in the model to reduce the loss obtained after each epoch. The loss function chosen to work together with the optimizer is the binary-crossentropy, which is a loss function mainly used to find how much single sample prediction deviates from its actual labels for binary data such as this proposed model where the model labels each pixels as either tumor or non-tumor class. The initial learning rate is set to be 0.00001 and the model is allowed to automatically reduce the learning rate if the validation loss does not decrease after 5 epochs and to do an early stopping if the validation loss does not decrease for 10 epochs. The model checkpoint is the path and the name of the model weights that is being trained.

5.1.8 uNet model implementation from Scratch

The model for the brain tumor segmentation is built from scratch referring to the architecture by [7]. Although the model is based on the uNet architecture, the proposed network architecture differs from the original implementation of uNet by [7]. Unlike the original implementation of the uNet model architecture, the model implemented in this project differs in that the number of output channels after each convolution is set to be 16, 32, 64, 128 and 256 unlike the original implementation which uses 64, 128, 256, 512, 1024. The reason is that, for this proposed model, the amount of dataset is limited as aforementioned. As such this proposed model reduced the amount of output channels to reduce the amount of information the model learns from the dataset to prevent overfitting. This is further proved in hyperparameter tuning experiments done in Section 5.1.10 that the best number of output channels for this model is 16, 32, 64, 128 and 256.

The implemented model also keeps the spatial dimension of the input MRI after each convolution layers to be consistent through implementation of ‘same’ padding unlike the original implementation of uNet by [7]. The implemented model is set with ‘ReLU’ activation functions after each convolution except the final convolution in the decoding phase, using ‘sigmoid’ activation instead which is needed to label the pixel

values as either below 0.5 for non-tumor class and above 0.5 for tumor class. Similar to original implementation, skip connections are used to connect the encoding and decoding phase. The input brain tumor images of size (256x256) are down sampled after each pooling layer, reducing it in half to get a wider field of view of the model till the fourth convolution in the encoding phase. Then up convolutions transpose are used to bring back the spatial dimension of the images through the decoding phase increasing back the size of the image to (256x256) as shown in the figure. Figure 5.6 below shows a snippet of the model implementation.

```
# ENCODER PART OF UNET

# Similar implementation, but the amount of parameters in uNet is very large, and for 2D data like the
# dataset used in this project, it causes overfitting as the model learns too much info about training
# As such, I have decided to modify the number of output channels in uNet
#Here we are defining a convolution block creation function with filter size of 3x3
def createConvBlock(in_tensor, num_filters, kern_size = 3):
    input_tensors = in_tensor

    input_tensors = tf.keras.layers.Conv2D(filters=num_filters,kernel_size=(kern_size,kern_size),
        kernel_initializer='he_normal',strides=(1,1),padding='same')(input_tensors)

    input_tensors = tf.keras.layers.BatchNormalization()(input_tensors)

    input_tensors = tf.keras.layers.Activation('relu')(input_tensors)

    input_tensors = tf.keras.layers.Conv2D(filters=num_filters,kernel_size=(kern_size,kern_size),
        kernel_initializer='he_normal',strides=(1,1),padding='same')(input_tensors)

    input_tensors = tf.keras.layers.BatchNormalization()(input_tensors)

    input_tensors = tf.keras.layers.Activation('relu')(input_tensors)

    return input_tensors
```

Figure 5.6 uNet Model implementation from scratch

The coding of the proposed uNet model implementation is attached in the appendix section A1.

5.1.9 SegUnet Model implementation

The model for the SegUnet hybrid model also built from scratch referring to architecture by [8]. In order to build this model, there are two important classes that had to be used for doing the indices pooling using the pooling information for concatenation during the decoding phase. The classes were successfully defined with the documentations provided in Keras. Figure 5.7 below shows the classes needed to do the indices pooling.


```

class MaxPoolingWithArgmax2D(Layer):
    def __init__(self, pool_size=(2, 2), strides=(2, 2), padding='same', **kwargs):
        super(MaxPoolingWithArgmax2D, self).__init__(**kwargs)
        self.padding = padding
        self.pool_size = pool_size
        self.strides = strides

    def call(self, inputs, **kwargs):
        padding = self.padding
        pool_size = self.pool_size
        strides = self.strides
        if K.backend() == 'tensorflow':
            ksize = [1, pool_size[0], pool_size[1], 1]
            padding = padding.upper()
            strides = [1, strides[0], strides[1], 1]
            output, argmax = K.tf.nn.max_pool_with_argmax(inputs, ksize=ksize, strides=strides, padding=padding)
        else:
            errmsg = '{} backend is not supported for layer {}'.format(K.backend(), type(self).__name__)
            raise NotImplementedError(errmsg)
        argmax = K.cast(argmax, K.floatx())
        return [output, argmax]

    def compute_output_shape(self, input_shape):
        ratio = (1, 2, 2, 1)
        output_shape = [dim // ratio[idx] if dim is not None else None for idx, dim in enumerate(input_shape)]
        output_shape = tuple(output_shape)
        return [output_shape, output_shape]

    def compute_mask(self, inputs, mask=None):
        return 2 * [None]

class MaxUnpooling2D(Layer):
    def __init__(self, size=(2, 2), **kwargs):
        super(MaxUnpooling2D, self).__init__(**kwargs)
        self.size = size

    def call(self, inputs, output_shape=None):
        updates, mask = inputs[0], inputs[1]
        with K.tf.compat.v1.variable_scope(self.name):
            mask = K.cast(mask, 'int32')

```

Figure 5.7 Defining classes for indices pooling

Although the model is based on the SegUnet architecture proposed by [8], the proposed network architecture differs from the original implementation. Unlike the original implementation of the segUnet, this model that was implemented in this project differs in that the number of output channels after each convolution is set to be 16, 32, 64, 128 and 128 for unlike the original implementation which uses 64, 128, 256, 512, 512. As mentioned in Section 5.1.10, too many parameters would cause overfitting, as such this was changed in the model to prevent this from happening and proved in the experimental results obtained. In the research work by [8], the segUnet architecture was focused for multiclass classification, thus using ‘softmax’ activation. This was changed to be ‘sigmoid’ activation in the proposed model to facilitate binary classification of tumor or non tumor class. The input brain tumor images of size (256x256) are down sampled after each indices pooling layer (MaxPoolingWithArgmax2D), reducing it in half to get a wider field of view of the model till the fourth convolution in the encoding phase which proceeds to a bottleneck layer and are upsampled with MaxUnpooling2D and are concatenated with the indices pooling information during the encoding phase. Figure 5.8 below shows a snippet of the implementation of segUnet that was done from scratch in Google Colab.

```
Code + Text
from keras.layers.convolutional import Convolution2D
from keras.layers import Concatenate

def CreateSegUNet(input_shape, n_labels, kernel=3, pool_size=(2, 2), output_mode="sigmoid"):
    inputs = Input(shape=input_shape)

    # encoder phase, similar to uNet(left side)
    conv_1 = Convolution2D(16, (kernel, kernel), padding="same")(inputs)
    conv_1 = BatchNormalization()(conv_1)
    conv_1 = Activation("relu")(conv_1)
    conv_2 = Convolution2D(16, (kernel, kernel), padding="same")(conv_1)
    conv_2 = BatchNormalization()(conv_2)
    conv_2 = Activation("relu")(conv_2)

    pool_1, mask_1 = MaxPoolingWithArgmax2D(pool_size)(conv_2)

    conv_3 = Convolution2D(32, (kernel, kernel), padding="same")(pool_1)
    conv_3 = BatchNormalization()(conv_3)
    conv_3 = Activation("relu")(conv_3)
    conv_4 = Convolution2D(32, (kernel, kernel), padding="same")(conv_3)
    conv_4 = BatchNormalization()(conv_4)
    conv_4 = Activation("relu")(conv_4)

    pool_2, mask_2 = MaxPoolingWithArgmax2D(pool_size)(conv_4)

    conv_5 = Convolution2D(64, (kernel, kernel), padding="same")(pool_2)
    conv_5 = BatchNormalization()(conv_5)
    conv_5 = Activation("relu")(conv_5)
    conv_6 = Convolution2D(64, (kernel, kernel), padding="same")(conv_5)
    conv_6 = BatchNormalization()(conv_6)
    conv_6 = Activation("relu")(conv_6)
    conv_7 = Convolution2D(64, (kernel, kernel), padding="same")(conv_6)
    conv_7 = BatchNormalization()(conv_7)
    conv_7 = Activation("relu")(conv_7)

    pool_3, mask_3 = MaxPoolingWithArgmax2D(pool_size)(conv_7)

    conv_8 = Convolution2D(128, (kernel, kernel), padding="same")(pool_3)
    conv_8 = BatchNormalization()(conv_8)
    conv_8 = Activation("relu")(conv_8)
    conv_9 = Convolution2D(128, (kernel, kernel), padding="same")(conv_8)
    conv_9 = BatchNormalization()(conv_9)
```

Figure 5.8 SegUNet model implementation from scratch

The code of the proposed SegUNet model implementation is attached in the appendix section B1.

5.1.10 Experimental Models and Hyperparameter Tuning

Before concluding on using a uNet+segUnet ensemble model for this work, multiple deep learning models with hyperparameter tuning, different model architectures, hybridization of model architectures and ensemble learning were trained, validated and tested from scratch using the dataset by Mateusz[5]. For the purpose of these testings, 1373 images from the dataset by [5] were used where 1098 used for training, 137 for validation and 138 for testing. The results of all of the models with the different hyperparameter tuning and architecture are tabulated and shown in Table 5.1 below. The models chosen for uNet+segUnet ensemble model is model 1 and 12 and the proposed model is model 19, which are all in bold for easier reference.

Table 5.1 Summary of all experimental model results

No .	Architecture	Output Channels	Batch size	Dropout	Padding	Batch Norm before FC layer	Ensemble technique	Average Jaccard Score (%)
1.	uNet model	16,32,6 4..	8	0.3	Same	No	-	75.31
2.	uNet model	16,32,6 4..	4	0.3	Same	No	-	73.92
3.	uNet model	16,32,6 4..	16	0.3	Same	No	-	71.75
4.	uNet model	64,128, 256..	8	0.3	Same	No	-	69.87
5.	SegNet architecture	16,32,6 4..	8	0.3	Same	No	-	66.67
6.	MobileNet architecture	16,32,6 4..	8	0.3	Same	No	-	73.3
7.	Attention based uNet	16,32,6 4..	8	0.3	Same	No	-	69.77
8.	DenseUnet	16,32,6 4..	8	0.3	Same	No	-	70.93
9.	ResUnet model	16,32,6 4..	8	0.3	Same	No	-	63.69 %
10 .	SegUnet model	16,32,6 4..	8	No	Same	Yes	-	69.75
11 .	SegUnet model	16,32,6 4..	8	No	Valid	Yes	-	41.99
12 .	SegUnet model	16,32,6 4..	8	No	Same	No	-	74.61

13	SegUnet model	16,32,6 4..	8	0.3	Same	No	-	71.87
13	SegUnet model	64,128, 256..	8	0.3	Same	No	-	69.87
14	SegUnet model	16,32,6 4..	4	0.3	Same	No	-	71.9
15	SegUnet model	16,32,6 4..	16	0.3	Same	No	-	70.69
16	SegUnet model (3 conv blocks)	16,32,6 4..	8	0.3	Same	No	-	63.57
17	SegUnet model (5 conv blocks)	16,32,6 4..	8	0.3	Same	No	-	67.95
18	SegUnet+ uNet	16,32,6 4..	8	0.3	Same	No	Average	75.57
19	SegUnet+ uNet (proposed model)	16,32,6 4..	8	0.3	Same	No	Maximum	75.79

From the experiments conducted, it is clear that some hyperparameter tuning are really effective for the purpose of training a model for brain tumor segmentation. The finalized hyperparameter and specifications for the proposed model are described below.

1. Using Output channels with the doubles of 16, 32, 64... in the model.

It is evident from the Table 5.1 above that having a larger number of channels in the model causes performance degradation such as when using 64, 128, 256.. . This is caused by the limited number of dataset available for training which causes overfitting as there are a lot of trained filters.

2. Using batch size of 8.

Batch size is the number of samples sent for training for an epoch. After each batch, the filter weights and bias will be updated. For this project, batch size 8 is the most suitable through the experiments conducted.

3. Using same padding

Same padding is when the images are padded with zeros to prevent the size of the image from reducing when going through a convolution layer. This is important to prevent information in the corners of the image from being washed away when going through multiple convolution layers. In model 11, without same padding, the performance declined drastically.

4. Removal of batch normalization before final dense layer

Having batch normalization in the final layer causes the performance of the model to degrade as shown in model 10. Batch normalization is important for introducing stochasticity to the network, and the network will be more robust due to this. However, its impossible for the network to cope with this stochasticity right before the output.

5. Using dropout layers

Dropout layers prevents model overfitting by intentionally ignoring some neurons during model training. This is important as it causes the model to not be too good during training and validation but degrade during testing.

6. Using a Maximum() operator to combine the output of ensemble model

To produce model 18 and 19, the predictions produced by the two best individual models, that is model 1 and 12 are chosen and their output results are combined to produce an ensemble model. As seen by the results produced by model 18 and 19, having ensemble learning drastically improved the performance and surpassed any of

the individual model average accuracy using Jaccard similarity index. For the ensemble, model 1 which is a uNet model and model 12, a segUnet model were chosen with the specified hyperparameters .

Using average technique for ensemble of the model was able to improve the accuracy, however it is not the best for the predictions done by the model. Let's take an assumption that the uNet model detects a tumor pixel with a 0.5 confidence through model prediction and the segUnet model states that it is tumor with 0.4 confidence. In average ensemble system, an issue of equal model contribution could happen where $0.4+0.5/2$ produces 0.45, which falls below the threshold of 0.5 set for true prediction causing the pixel to be labeled as non-tumor. To resolve this issue, in this work, probability of the model with highest score of prediction for a single pixel is chosen through Maximum() operator which managed to improved the overall accuracy of the system compared to average.

The testing results obtained for 7 of these experimental models are attached in the appendix Section D.

5.1.11 Deploy the built models as a web application using Python Flask

After concluding on using an ensemble learning, both the uNet and SegUnet models are transferred over to Pycharm where the web application is set to be built. Before transferring the model from Google Colab to Pycharm and deploying it, it is important to set up the homepage where the web application would route to on default. Figure 5.9 below shows the `app.route("/")` done to allow the app homepage to receive any 'POST' request from users. This request is attached to upload button in the page and are used to receive the 2D brain MR images from user local folder.

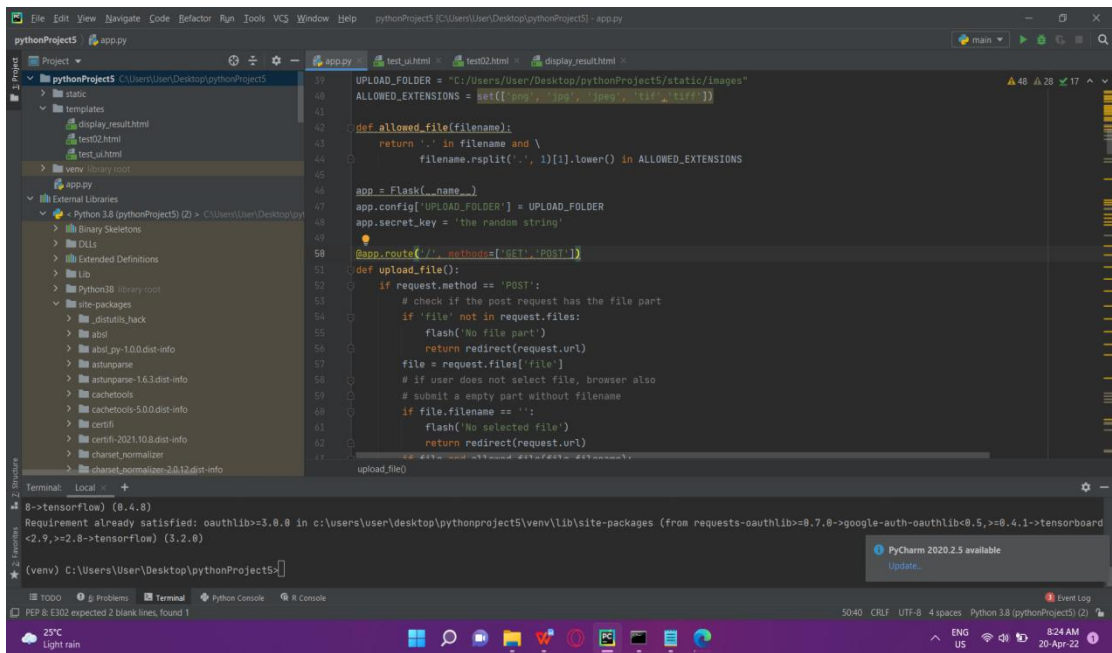


Figure 5.9 Default app route set up with Flask

Images of .jpg, .jpeg, .png and .tif formats are accepted, and if there is no issue with the format, the two models' architectures are loaded with their weights to carry out the segmentation. The models are set to predict the uploaded image individually, then combine their outputs through Maximum() layer to perform ensemble where their combined outputs are used for doing the segmentation as shown in Figure 5.10 below.

```

model2 = CreateSegUNet((256, 256, 3), 1)

# Load your trained model
model = load_model(MODEL_PATH)
model.load_weights('C:/Users/User/Desktop/FYP2/TrainedModelNewNewNew.h5')
model2.load_weights('C:/Users/User/Desktop/FYP2/FYP2_modified_SegUnet.h5')
# Necessary
# print('Model loaded. Start serving...')
img = cv2.imread("C:/Users/User/Desktop/pythonProject5/static/images/{}".format(filename))
img = cv2.resize(img, (256, 256))

# Preprocessing the image
img = img / 255 # to make the pixels to tensors 0 to 1
img = img[np.newaxis, :, :, :]

pred1 = model.predict(img)
pred2 = model2.predict(img)

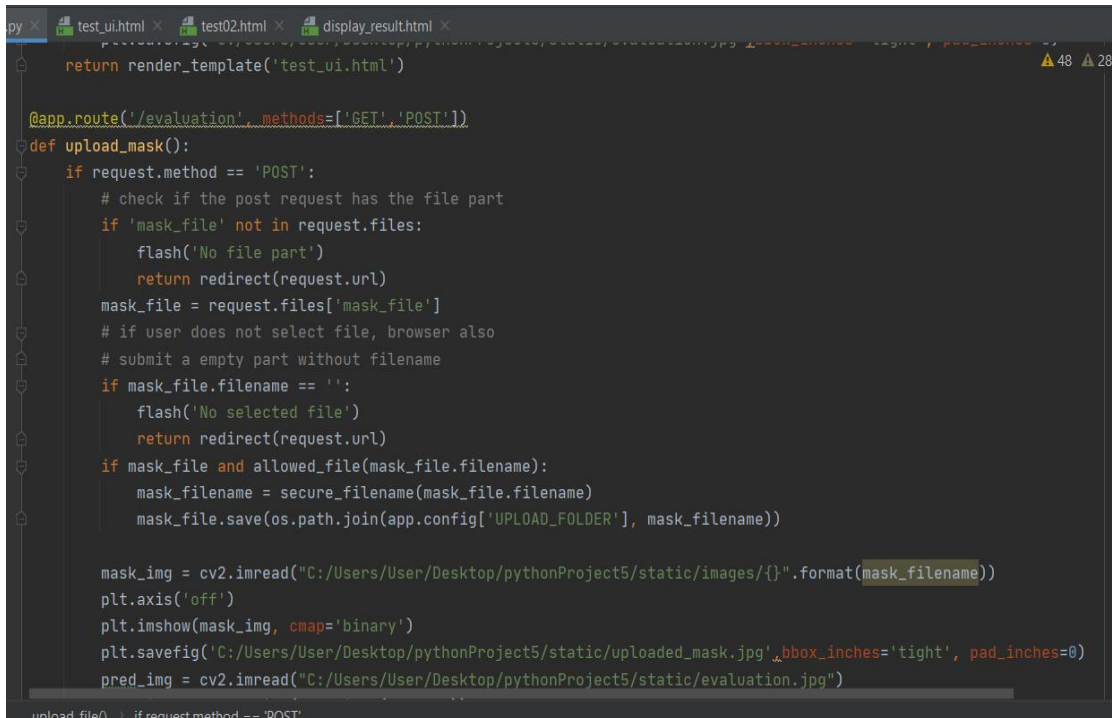
pred = Maximum()(pred1, pred2)

original_img = (np.squeeze(img))
plt.axis('off')
plt.imshow(original_img, cmap='gray')
plt.savefig('C:/Users/User/Desktop/pythonProject5/static/ori_output.jpg', bbox_inches="tight")

```

Figure 5.10 Both models are loaded in and ready for segmentation

In the web app, an evaluation function was added to allow for users to compare the predictions with ground truth image, if they have. Figure 5.11 below shows the implementation of another `app.route("/evaluation")` to carry out this function. User are requested to upload the ground truth image and the uploaded image are used to find Jaccard score and are displayed to the users.



```
return render_template('test_ui.html')

@app.route('/evaluation', methods=['GET', 'POST'])
def upload_mask():
    if request.method == 'POST':
        # check if the post request has the file part
        if 'mask_file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        mask_file = request.files['mask_file']
        # if user does not select file, browser also
        # submit a empty part without filename
        if mask_file.filename == '':
            flash('No selected file')
            return redirect(request.url)
        if mask_file and allowed_file(mask_file.filename):
            mask_filename = secure_filename(mask_file.filename)
            mask_file.save(os.path.join(app.config['UPLOAD_FOLDER'], mask_filename))

            mask_img = cv2.imread("C:/Users/User/Desktop/pythonProject5/static/images/{}".format(mask_filename))
            plt.axis('off')
            plt.imshow(mask_img, cmap='binary')
            plt.savefig('C:/Users/User/Desktop/pythonProject5/static/uploaded_mask.jpg', bbox_inches='tight', pad_inches=0)
            pred_img = cv2.imread("C:/Users/User/Desktop/pythonProject5/static/evaluation.jpg")
            upload_file() if request.method == 'POST'
```

Figure 5.11 Set up of evaluation page in web app

5.1.12 Brain tumor segmentation produced from the web application

Figure 5.12, Figure 5.13 and Figure 5.14 below shows how a user-uploaded 2D Flair brain tumor MRI is used to perform segmentation in real time and are displayed on the User Interface(UI) that is built on Flask. The users are presented with UI that requires minimal intervention to do the tumor segmentation.

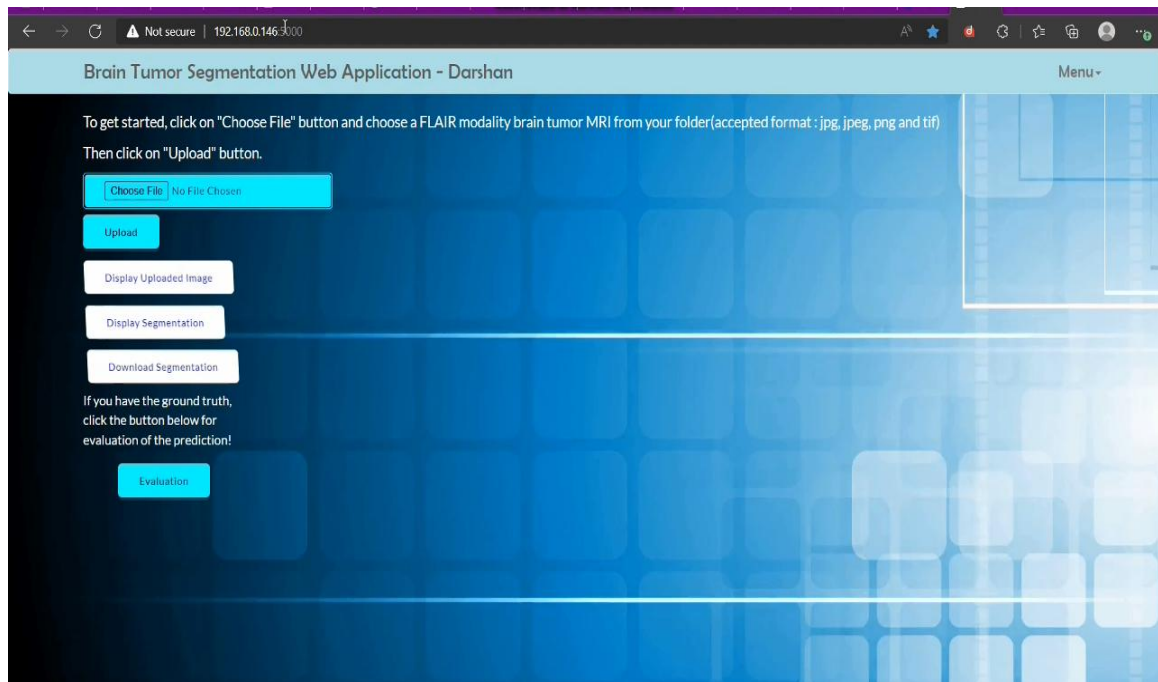


Figure 5.12 Homepage of the web application hosted on local public IP address

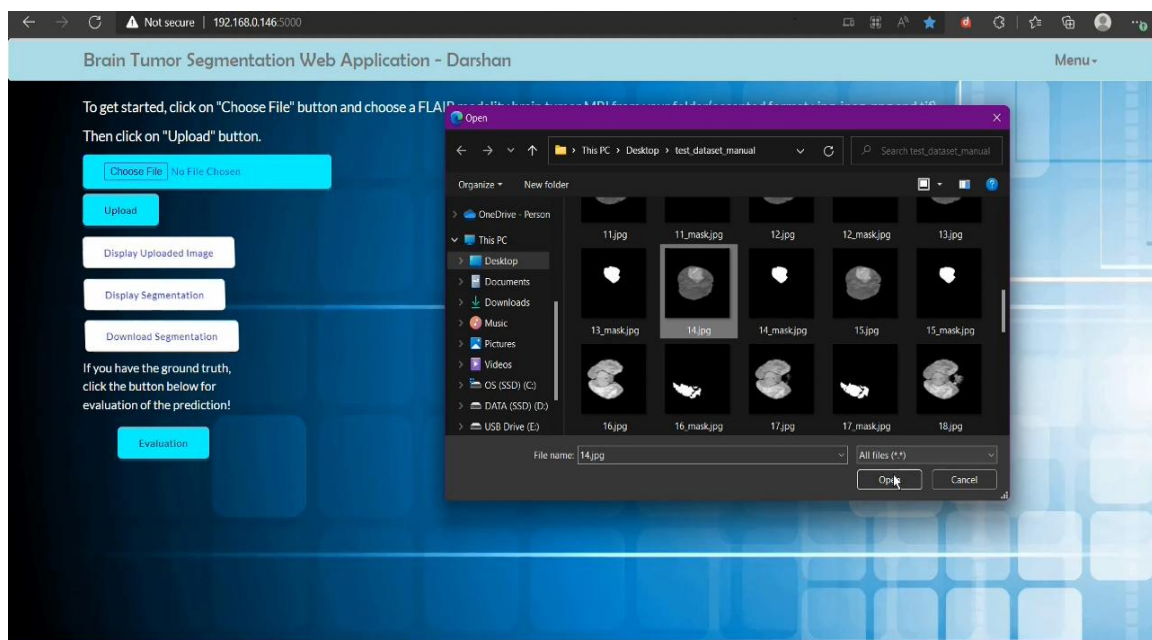


Figure 5.13 User chooses and uploads a 2D Flair Brain tumor MR image for segmentation

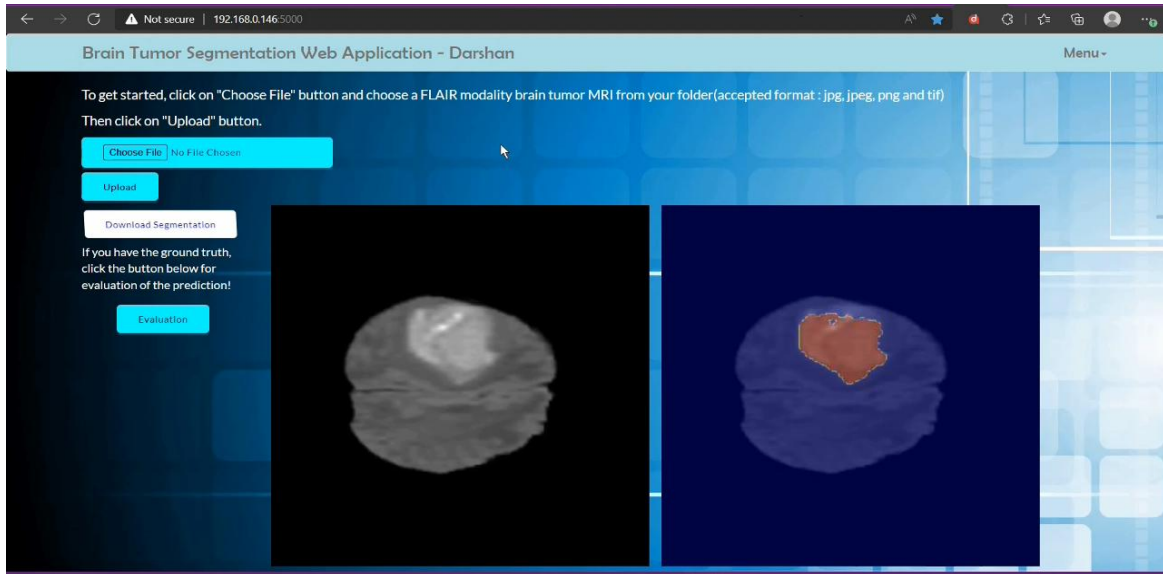


Figure 5.14 Brain Tumor Segmentation shown on the User Interface(UI)

If the users wish to do evaluation on the segmentation done by the proposed model, the users are allowed to upload a ground truth image of their uploaded MRI for the Jaccard score computation. The computed Jaccard score is then displayed on the UI for the users.

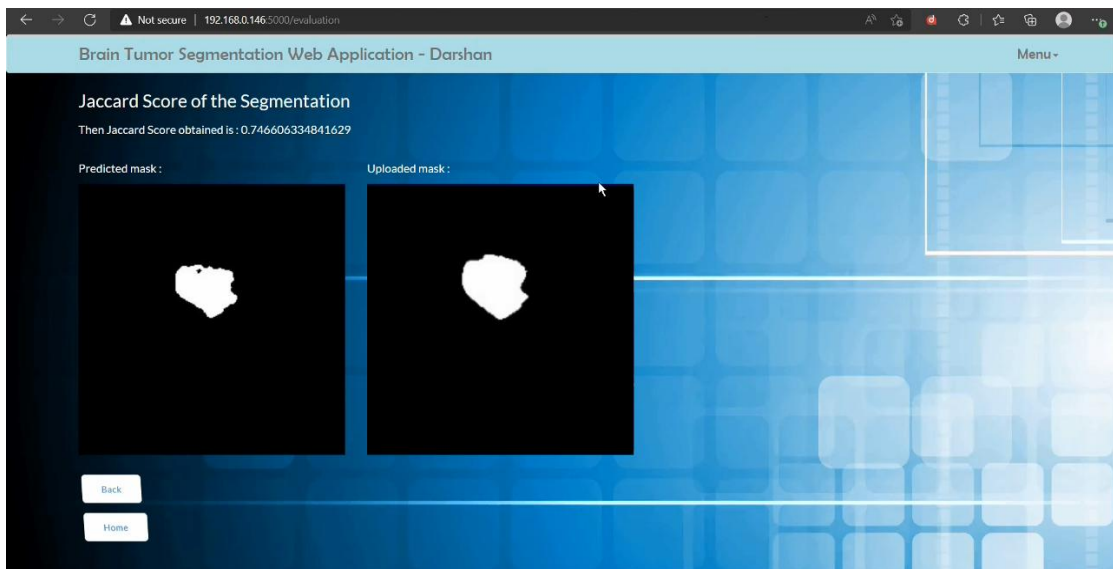


Figure 5.15 Jaccard score computed and displayed on User Interface(UI)

5.2 Results of the uNet+SeguNet ensemble model

The results obtained from segmentation of test images of slices number 1 till 3 for dataset[5] are shown in Figure 5.16 below. Further results on all the 138 testing images are attached in the appendix section C1.

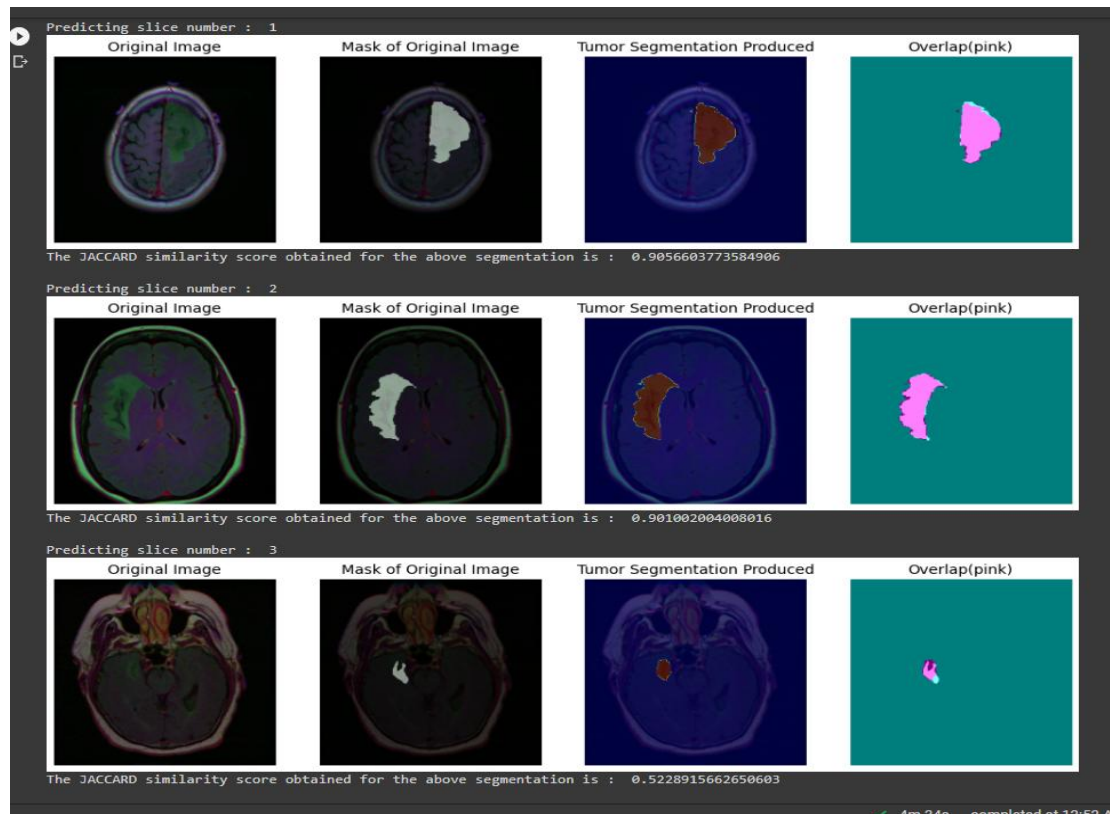


Figure 5.16 Brain tumor segmentation by the proposed uNet+SegUnet model ensemble

In the Figure 5.17 below, the image on top shows the individual uNet model while the bottom shows the segmentation result of the proposed model. From this figure, the results show that the uNet model is unable to segment the tumor region while the proposed model is able to perform the tumor delineation.

The uNet+segUnet ensemble model has obtained an average 75.79% accuracy for test dataset by using the Jaccard evaluation metric. This in comparison to the best individual model produced during experiments where uNet model only obtained average 75.31% accuracy for the testing. By using individual model such as uNet for the segmentation, it is noted that due to small tumor sizes or ambiguous boundary, some tumor was not segmented.. Hence, the accuracy is lower. However, in this

proposed uNet+segUnet ensemble model, the tumor was still able to be segmented to some extent. This is because segUnet is able to achieve in segmenting small tumor regions with ambiguous boundary .

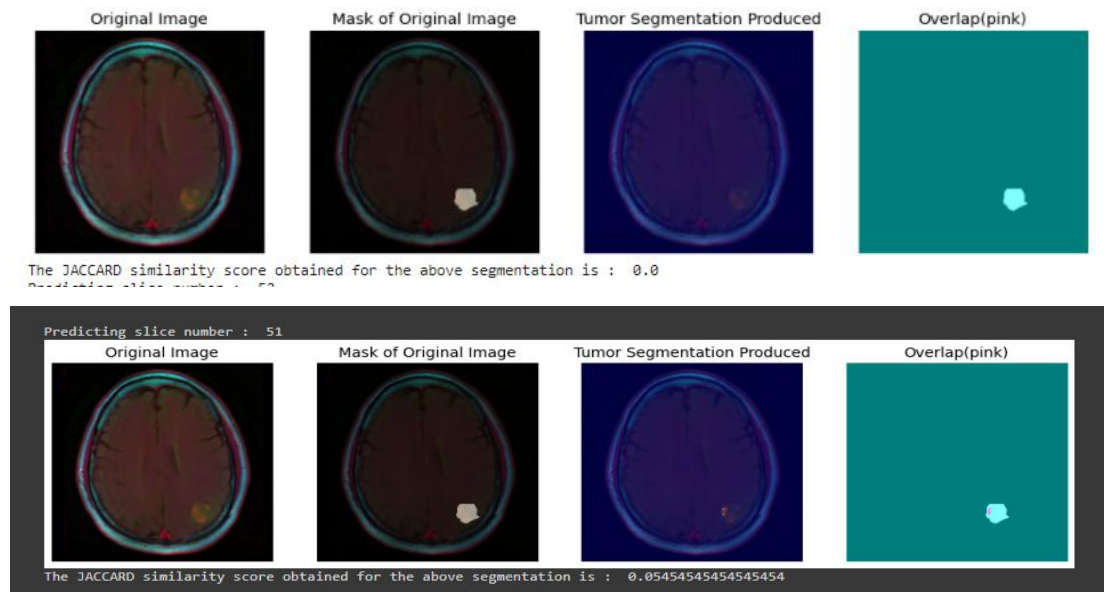


Figure 5.17 Proposed ensemble model is able to segment the tumor region

To evaluate the robustness of the proposed model another 2D Flair Brain Tumor MRI dataset [6] as mentioned in Chapter 4 Section 4.5 has been used for testing. This dataset was not used for training nor in the validation steps of the deep learning models but just for testing purposes. There are a total of 621 images from this dataset that were used for doing the testing. This dataset is used to test the individual uNet, segUnet, uNet+segUnet ensemble model with average() layer and the proposed uNet+segUnet with maximum() layer to compare the results. Table 5.2 below shows the results obtained on testing with dataset.

Table 5.2 Results obtained from testing dataset [6]

No.	Architecture	Jaccard Score (%)
1	uNet model	64.63
2	SegUnet model	52.83
3	uNet+segUnet model (average)	58.52
4	uNet+segUnet model(maximum) - proposed	66.75

As seen in Table 5.2 above, the proposed uNet+segUnet ensemble model with a Maximum() layer has drastically beat the individual models and the ensemble model with Average() layer when tested with the second dataset. The proposed model has performed the segmentation with a Jaccard score of 66.75%. More results obtained by this proposed model on the second dataset is attached in Appendix section C2. Figure 5.18 below shows the results obtained by the proposed model on this dataset.

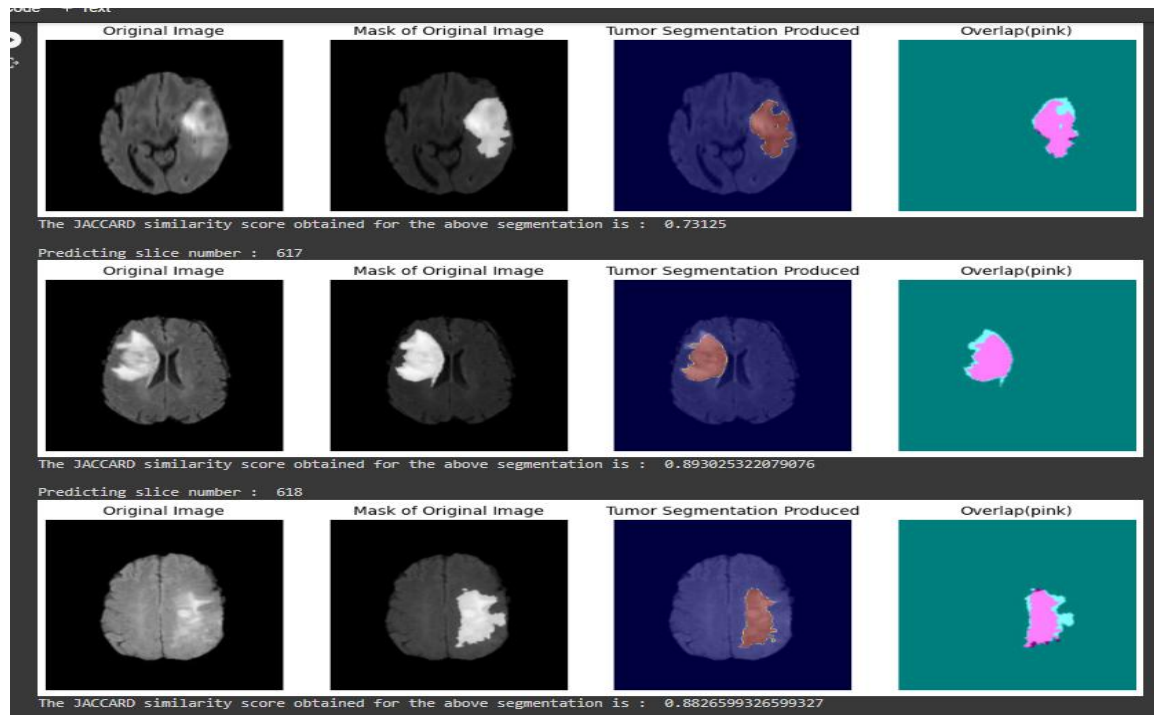


Figure 5.18 Results obtained from testing on dataset by [6]

It is to be noted that the images of the dataset [6] were rather blurry because manual splitting of mask and images from compressed .h5 carried out to prepare the dataset which may have caused the blurring during the processing. As such the model sometimes had issues with some of the image segmentation where the tumor was very similar to nearby tissues and caused a lower accuracy segmentation. A clearer dataset with more number of samples could see a definite improvement of accuracy when tested.

Figure 5.19 shows the segmentation produced by the proposed uNet+segUnet ensemble model with Maximum() layer by using the dataset from [6] where the proposed model is able to segment the tumor with high accuracy when the tumor size is large with less ambiguous boundary.

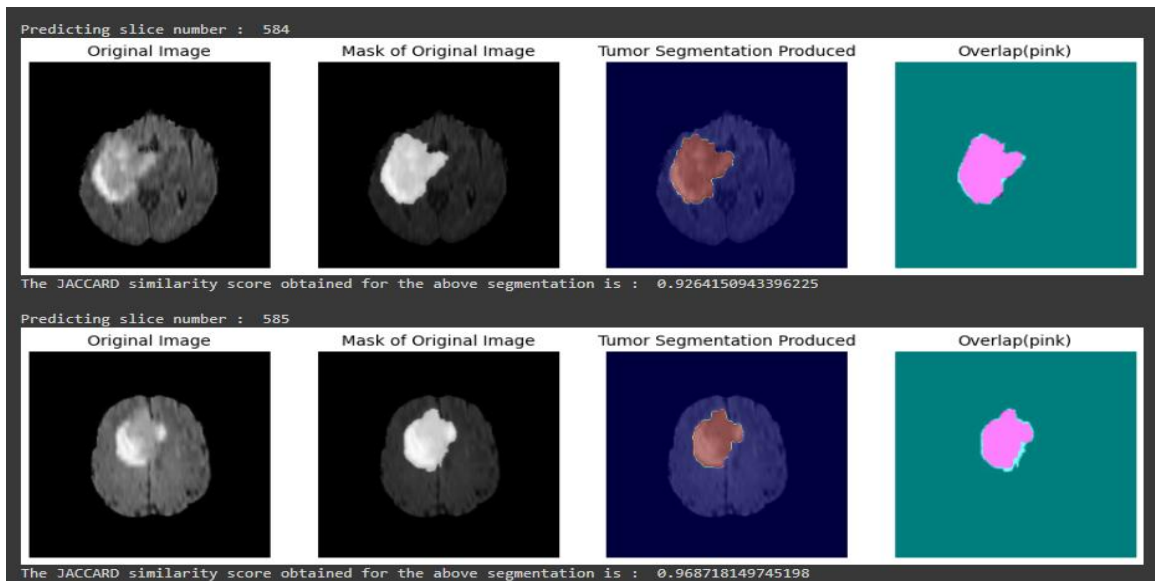


Figure 5.19 Brain tumor segmentation on dataset[6] with high accuracy by proposed model

However, in Figure 5.20 the proposed model was unable to segment the tumor region in high accuracy. This is very likely caused by the ambiguous boundary of the tumor or due to the images being slightly blurry causing the tumor regions to be merged with nearby tissues and be predicted as non-tumor class.

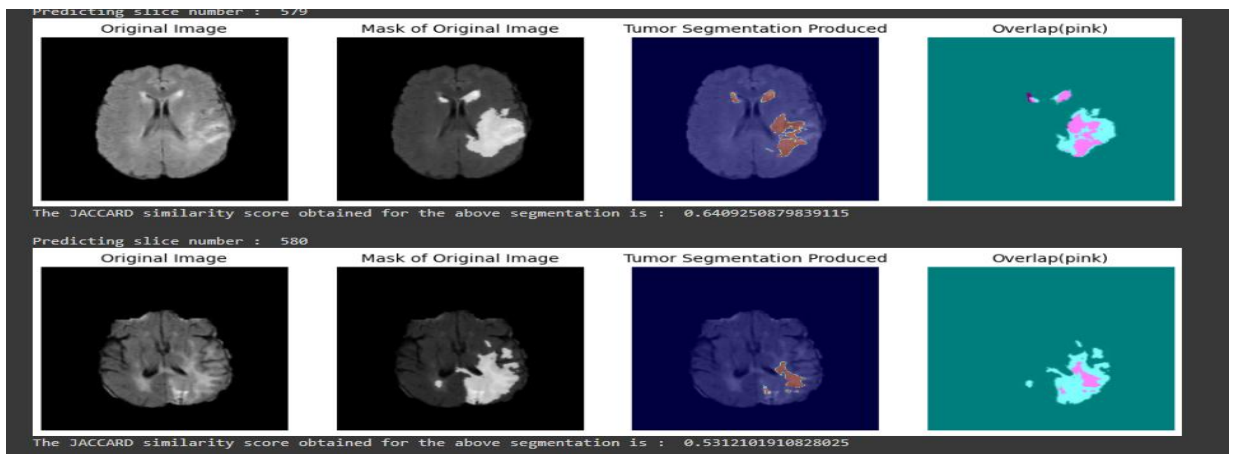


Figure 5.20 The proposed model unable to segment the tumor region accurately for the dataset by [6]

5.3 Chapter Summary

In this chapter, explanation on the implementation of proposed uNet+segUnet model and the deployment of the model in Flask is given. This chapter also covers the experimental models built and their respective results and also the testing results of the proposed uNet+segUnet model on two different 2D brain tumor datasets.

CHAPTER 6 : CONCLUSION, NOVELTY AND FUTURE WORK

6.1 Conclusion

In this work, a brain tumor segmentation using uNet+segUnet ensemble model is proposed. The proposed model is also deployed and hosted as a web application using Flask to provide a User Interface(UI) for usage. This model had achieved a 75.79% testing accuracy on the dataset provided by Mateusz[5] by using the Jaccard score evaluation metric. The proposed model is also tested with another dataset that has not been used for training and validation by Balakrishna[8] and managed to obtain 66.79% accuracy indicating the robustness of the proposed model to 2D FLAIR brain tumor MR images.

6.2 Novelty

In this work, an ensemble uNet+segUnet model making use of two model architectures, namely uNet by [6] and segUnet by [9] is proposed. Modifications in uNet and segUnet architectures are also proposed compared to their original implementation by [6] and [8] respectively. In this ensemble model, the usage of Maximum() layer for combining the prediction scores of every pixel by the model is proposed to allow the higher confidence model to do final decision if a pixel is tumor or non-tumor class which improved the overall accuracy of the ensemble model. The proposed model has also been deployed and hosted with Flask to provide User Interface(UI) for usage and provides an automatic 2D brain tumor segmentation. The web application also requires minimal user intervention, requires no training by the users and provides a satisfactory segmentation results which can be download and evaluated through the web application.

6.3 Future Work

There are more works that can be done to this project to improve it further. Mainly, the accuracy of the segmentation can be improved to higher values through implementation of transfer learning techniques or using a dataset with more number of training images and for longer epochs. The application will also be improved with implementation of 3-Dimensional(3D) segmentation models to allow users to use 3D MR images as well for brain tumor segmentation. Other than that, the design of the web application could be made more user-friendly and attractive in the future.

REFERENCES

1. "QUICK BRAIN TUMOR FACTS", 2021. Accessed on: 28 July 2021. [online]
Available : <https://braintumor.org/brain-tumor-information/brain-tumor-facts>

2. My-ms.org, MRI Basics, 2021 Accessed on: 22 July 2021. [online]
Available: https://my-ms.org/mri_basics.htm

3. B. Rohit, A. Suzie, H.B. Ralph, Benedict, Fluid-attenuated inversion recovery magnetic resonance imaging detects cortical and juxtacortical multiple sclerosis lesions, 2001. Accessed on: 1 August 2021.[online]
Available : <https://jamanetwork.com/journals/jamaneurology/fullarticle/779326>

4. G. Yanhui, S.A. Amira, Neutrosophic sets in dermoscopic medical image segmentation, 2021. Accessed on: 19 July 2021[online] Available:
<https://www.sciencedirect.com/topics/engineering/medical-image-segmentation>

5. B. Mateusz, Brain MRI Segmentation, 2019. Accessed on: 20 June 2021. [online]
Available : <https://www.kaggle.com/mateuszbudalgg-mri-segmentation>

6. K. Balakrishna, Brain 2D MRI Images and Mask, 2020, Accessed on: 27 February 2022 Available : <https://www.kaggle.com/datasets/balakrishcodes/brain-2d-mri-imgs-and-mask>

7. R. Olaf , F. Philipp , B. Thomas ,U-Net: Convolutional Networks for Biomedical Image Segmentation, 18 May 2015. Accessed on: 18 July 2021. [online] Available : <https://arxiv.org/abs/1505.04597>

8. D. Dinthisrang, B.B. Mayur, Amitab, K. Debdatta, Brain Tumor Segmentation from MRI Images using Hybrid CONvolution Neural Networks, January 2020. Accessed on: 27 January 2022. [online] Available: https://www.researchgate.net/publication/340699988_Brain_Tumor_Segmentation_from_MRI_Images_using_Hybrid_Convolutional_Neural_Networks
9. B. Vijay, K. Alex, C. Roberto, SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, 2 Nov 2015. Accessed on: 11 February 2022. [online] Available at: <https://arxiv.org/abs/1511.00561>
10. MIPAV, MIPAV User Guide, 2008. Accessed on : 26 Aug 2020. [online] Available: <https://mipav.cit.nih.gov/documentation/userguide/MIPAVUsersGuideVolume1.pdf>
11. ITK-SNAP, ITK-SNAP Home, 2018. Accessed on: 11 September 2020.[online] Available at : www.itksnap.org/pmwiki/pmwiki.php
12. Olea Medical, User Guide Olea Sphere V2.2 VITAL Revision 2.2, 2012. Accessed on: 10 June 2021.[online] Available: <http://92.103.51.217/vital/help/en/pdf/Olea-Sphere%20User%20Guide%20v2.2.pdf>
13. Belevich, I. and Jokitalo, E., DeepMIB: User-friendly and open-source software for training of deep learning network for biological image segmentation,2021. Accessed on: 27 June 2021. [online] Available: <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008374#sec002>

14. NiftyNet, NiftyNet Home, 2018. Accessed on: 12 June 2021.[online] Available : <https://niftynet.io/>
15. G. Eli, L. Wenqi, S. Carole, F. Lucas, I. Dzhoshkun, Shakir et al, NiftyNet: a deep-learning platform for medical imaging. Accessed on: 13 June 2021.[online] Available : <https://www.sciencedirect.com/science/article/pii/S0169260717311823>
16. B. Jason, Train-Test Split for Evaluating Machine Learning Algorithms, 26 August 2020. Accessed on: 21 June 2021.[online] Available: <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>
17. C. Ehtesham, B. Abhijit, M. H. Hasan, M. S. Rahim, Analysis of the Veracities of Industry Used Software Development Life Cycle Methodologies, June 2017. Accessed on: 27 July 2021.[online] Available: https://www.researchgate.net/publication/318779643_Analysis_of_the_Veracities_of_Industry_Used_Software_Development_Life_Cycle_Methodologies
18. V. Farcic, Software Development Models : Iterative and Incremental Development, 2014. Accessed on: 2 August 2021.[online] Available: <https://technologyconversations.com/2014/01/21/software-development-models-iterative-and-incremental-development>
19. Khan, Where did the Binary Cross-Entropy Loss Function come from?, 2021. Accessed on: 3 July 2021 [online] Available: <https://towardsdatascience.com/where-did-the-binary-cross-entropy-loss-function-come-from-ac3de349a715>

20. Medium, WHEN and WHY are batches used in machine learning ?, 2021. Accessed on: 13 July 2021 [online] Available: <https://medium.com/analytics-vidhya/when-and-why-are-batches-used-in-machine-learning-acda4eb00763#:~:text=Another%20reason%20for%20why%20you,decrease%20in%20speed%20of%20training>
21. “Jaccard Similarity Index”, 1912. Accessed on: 10 August 2021.[online] Available : <https://ase.tufts.edu/chemistry/walt/sepa/Activities/jaccardPractice.pdf>
22. C.E. Metz, Basic principles of ROC analysis, 1978. Accessed on: 8 August 2021. [online]Available: : <https://www.sciencedirect.com/science/article/abs/pii/S0001299878800142?via%3Dihub>

APPENDICES

APPENDIX A: uNet model implementation based on [7]

```
#####
# dataset used in this project, it causes overfitting as the model learns too much info about training
# As such, I have decided to modify the number of output channels in uNet
# Here we are defining a convolution block creation function with filter size of 3x3
def createConvBlock(in_tensor, num_filters, kern_size = 3):
    input_tensors = in_tensor

    input_tensors = tf.keras.layers.Conv2D(filters=num_filters, kernel_size=(kern_size, kern_size),
                                           kernel_initializer='he_normal', strides=(1,1), padding='same')(input_tensors)

    input_tensors = tf.keras.layers.BatchNormalization()(input_tensors)

    input_tensors = tf.keras.layers.Activation('relu')(input_tensors)

    input_tensors = tf.keras.layers.Conv2D(filters=num_filters, kernel_size=(kern_size, kern_size),
                                           kernel_initializer='he_normal', strides=(1,1), padding='same')(input_tensors)

    input_tensors = tf.keras.layers.BatchNormalization()(input_tensors)

    input_tensors = tf.keras.layers.Activation('relu')(input_tensors)

    return input_tensors

# Now we will create a block that will create the convolution block we defined earlier
def encodeBlock(inputs, num_filters=16, pool_size=(2,2), dropout=0.3):
    conv_output = createConvBlock(inputs, num_filters=num_filters)

    pooling = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_output)

    pooling = tf.keras.layers.Dropout(0.3)(pooling) # to increase the generalization ability of the model(reduce overfitting)

    return conv_output, pooling

def encoder(inputs):
#input images are with the size (256,256)
    conv_output1, pooling1 = encodeBlock(inputs, num_filters = 16, pool_size=(2,2), dropout=0.3) #output channel is 16
    #so shape now is : (16,128,128) after pooling with size (2,2)
    conv_output2, pooling2 = encodeBlock(pooling1, num_filters = 32, pool_size=(2,2), dropout=0.3) #output channel is 32
    #so shape now is : (32,64,64) after pooling with size (2,2)
    conv_output3, pooling3 = encodeBlock(pooling2, num_filters = 64, pool_size=(2,2), dropout=0.3) #output channel is 64
    #so shape now is : (64,32,32) after pooling with size (2,2)
    conv_output4, pooling4 = encodeBlock(pooling3, num_filters = 128, pool_size=(2,2), dropout=0.3) #output channel is 128
    #so shape now is (128,16,16) after pooling with size (2,2)

    return pooling4, (conv_output1, conv_output2, conv_output3, conv_output4)

def bottleneck(inputs):
    bottle_neck = createConvBlock(inputs, num_filters = 256)

[ ] def bottleneck(inputs):
    bottle_neck = createConvBlock(inputs, num_filters = 256)
    return bottle_neck

[ ] # DECODER PART OF UNET

def decodeBlock(inputs, conv_output, num_filters=16, kern_size=3, strides=2, dropout=0.3):

    upsampling = tf.keras.layers.Conv2DTranspose(num_filters, kern_size, strides=strides, padding='same')(inputs)
    #inputs are in parentheses after the upsampling layer using the functional API syntax
    concatenate = tf.keras.layers.concatenate(upsampling, conv_output) #skip connection between encoder and decoder
    concatenate = tf.keras.layers.Dropout(dropout)(concatenate) #prevents overfitting
    concatenate = createConvBlock(concatenate, num_filters, kern_size=3)

    return concatenate

def decoder(inputs, convolutions, output_channels):
#input convolutions are output_conv1, output_conv2, output_conv3, output_conv4
    output_conv1, output_conv2, output_conv3, output_conv4 = convolutions

    conv_output6 = decodeBlock(inputs, output_conv4, num_filters = 128, kern_size=(3,3), strides =(2,2), dropout = 0.3 )

    conv_output7 = decodeBlock(conv_output6, output_conv3, num_filters = 64, kern_size=(3,3), strides =(2,2), dropout = 0.3 )

    conv_output8 = decodeBlock(conv_output7, output_conv2, num_filters = 32, kern_size=(3,3), strides =(2,2), dropout = 0.3 )

    conv_output9 = decodeBlock(conv_output8, output_conv1, num_filters = 16, kern_size=(3,3), strides =(2,2), dropout = 0.3 )
    #after upsampling, shape is now (16,256,256)

    outputs = tf.keras.layers.Conv2D(output_channels, kernel_size=(1,1), activation='sigmoid')(conv_output9) #chosen sigmoid activation for binary pred

    return outputs

[ ] output_channels=1

[ ] def uNetModel():

    inputs = tf.keras.layers.Input(shape=(256,256,3))

    encoder_output, convolutions = encoder(inputs)

    bottlenecklayer = bottleneck(encoder_output)

    outputs = decoder(bottlenecklayer, convolutions, output_channels=output_channels)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    return model
```

Model Summary After building : -

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	
conv2d (Conv2D)	(None, 256, 256, 16)	448	input_1[0][0]
batch_normalization (BatchNormaliza)	(None, 256, 256, 16)	64	conv2d[0][0]
activation (Activation)	(None, 256, 256, 16)	0	batch_normalization[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	2320	activation[0][0]
batch_normalization_1 (BatchNor)	(None, 256, 256, 16)	64	conv2d_1[0][0]
activation_1 (Activation)	(None, 256, 256, 16)	0	batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	activation_1[0][0]
dropout (Dropout)	(None, 128, 128, 16)	0	max_pooling2d[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	dropout[0][0]
batch_normalization_2 (BatchNor)	(None, 128, 128, 32)	128	conv2d_2[0][0]
activation_2 (Activation)	(None, 128, 128, 32)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	9248	activation_2[0][0]
batch_normalization_3 (BatchNor)	(None, 128, 128, 32)	128	conv2d_3[0][0]

activation_3 (Activation) (None, 128, 128, 32) 0
batch_normalization_3[0][0]
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 32) 0
activation_3[0][0]
dropout_1 (Dropout) (None, 64, 64, 32) 0
max_pooling2d_1[0][0]
conv2d_4 (Conv2D) (None, 64, 64, 64) 18496
dropout_1[0][0]
batch_normalization_4 (BatchNor (None, 64, 64, 64) 256
conv2d_4[0][0]
activation_4 (Activation) (None, 64, 64, 64) 0
batch_normalization_4[0][0]
conv2d_5 (Conv2D) (None, 64, 64, 64) 36928
activation_4[0][0]
batch_normalization_5 (BatchNor (None, 64, 64, 64) 256
conv2d_5[0][0]
activation_5 (Activation) (None, 64, 64, 64) 0
batch_normalization_5[0][0]
max_pooling2d_2 (MaxPooling2D) (None, 32, 32, 64) 0
activation_5[0][0]
dropout_2 (Dropout) (None, 32, 32, 64) 0
max_pooling2d_2[0][0]
conv2d_6 (Conv2D) (None, 32, 32, 128) 73856
dropout_2[0][0]
batch_normalization_6 (BatchNor (None, 32, 32, 128) 512
conv2d_6[0][0]
activation_6 (Activation) (None, 32, 32, 128) 0
batch_normalization_6[0][0]
conv2d_7 (Conv2D) (None, 32, 32, 128) 147584
activation_6[0][0]
batch_normalization_7 (BatchNor (None, 32, 32,

128) 512 conv2d_7[0][0]

activation_7 (Activation) (None, 32, 32, 128) 0

batch_normalization_7[0][0]

max_pooling2d_3 (MaxPooling2D) (None, 16, 16, 128) 0

activation_7[0][0]

dropout_3 (Dropout) (None, 16, 16, 128) 0

max_pooling2d_3[0][0]

conv2d_8 (Conv2D) (None, 16, 16, 256) 295168

dropout_3[0][0]

batch_normalization_8 (BatchNor (None, 16, 16, 256) 1024

conv2d_8[0][0]

activation_8 (Activation) (None, 16, 16, 256) 0

batch_normalization_8[0][0]

conv2d_9 (Conv2D) (None, 16, 16, 256) 590080

activation_8[0][0]

batch_normalization_9 (BatchNor (None, 16, 16, 256) 1024

conv2d_9[0][0]

activation_9 (Activation) (None, 16, 16, 256) 0

batch_normalization_9[0][0]

conv2d_transpose (Conv2DTranspo (None, 32, 32, 128) 295040

activation_9[0][0]

concatenate (Concatenate) (None, 32, 32, 256) 0

conv2d_transpose[0][0] activation_7[0][0]

dropout_4 (Dropout) (None, 32, 32, 256) 0

concatenate[0][0]

conv2d_10 (Conv2D) (None, 32, 32, 128) 295040

dropout_4[0][0]

batch_normalization_10 (BatchNo (None, 32, 32, 128) 512

conv2d_10[0][0]

activation_10 (Activation) (None, 32, 32, 128) 0

batch_normalization_10[0][0]

conv2d_11 (Conv2D) (None, 32, 32, 128)	147584
activation_10[0][0]	
batch_normalization_11 (BatchNo (None, 32, 32, 128)	512
conv2d_11[0][0]	
activation_11 (Activation) (None, 32, 32, 128)	0
batch_normalization_11[0][0]	
conv2d_transpose_1 (Conv2DTrans (None, 64, 64, 64)	73792
activation_11[0][0]	
concatenate_1 (Concatenate) (None, 64, 64, 128)	0
conv2d_transpose_1[0][0]	
activation_5[0][0]	
dropout_5 (Dropout) (None, 64, 64, 128)	0
concatenate_1[0][0]	
conv2d_12 (Conv2D) (None, 64, 64, 64)	73792
dropout_5[0][0]	
batch_normalization_12 (BatchNo (None, 64, 64, 64)	256
conv2d_12[0][0]	
activation_12 (Activation) (None, 64, 64, 64)	0
batch_normalization_12[0][0]	
conv2d_13 (Conv2D) (None, 64, 64, 64)	36928
activation_12[0][0]	
batch_normalization_13 (BatchNo (None, 64, 64, 64)	256
conv2d_13[0][0]	
activation_13 (Activation) (None, 64, 64, 64)	0
batch_normalization_13[0][0]	
conv2d_transpose_2 (Conv2DTrans (None, 128, 128, 32)	18464
activation_13[0][0]	
concatenate_2 (Concatenate) (None, 128, 128, 64)	0
conv2d_transpose_2[0][0]	
activation_3[0][0]	
dropout_6 (Dropout) (None, 128, 128, 64)	0
concatenate_2[0][0]	
conv2d_14 (Conv2D) (None, 128, 128, 32)	18464
dropout_6[0][0]	

2,161,649 Non-trainable params: 2,944

Model training and validation:

Epoch 1/40

137/137 [=====] - 56s 390ms/step - loss: 0.0201 - accuracy: 0.9903 - mean_io_u_1: 0.4866 - val_loss: 0.0194 - val_accuracy: 0.9932 - val_mean_io_u_1: 0.4853

Epoch 00001: val_loss improved from inf to 0.01937, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 2/40

137/137 [=====] - 53s 386ms/step - loss: 0.0201 - accuracy: 0.9900 - mean_io_u_1: 0.4865 - val_loss: 0.0237 - val_accuracy: 0.9922 - val_mean_io_u_1: 0.4851

Epoch 00002: val_loss did not improve from 0.01937

Epoch 3/40

137/137 [=====] - 53s 388ms/step - loss: 0.0206 - accuracy: 0.9902 - mean_io_u_1: 0.4865 - val_loss: 0.0200 - val_accuracy: 0.9929 - val_mean_io_u_1: 0.4851

Epoch 00003: val_loss did not improve from 0.01937

Epoch 4/40

137/137 [=====] - 53s 387ms/step - loss: 0.0195 - accuracy: 0.9904 - mean_io_u_1: 0.4864 - val_loss: 0.0165 - val_accuracy: 0.9938 - val_mean_io_u_1: 0.4858

Epoch 00004: val_loss improved from 0.01937 to 0.01653, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 5/40

137/137 [=====] - 53s 386ms/step - loss: 0.0182 - accuracy: 0.9908 - mean_io_u_1: 0.4865 - val_loss: 0.0225 - val_accuracy: 0.9912 - val_mean_io_u_1: 0.4849

Epoch 00005: val_loss did not improve from 0.01653

Epoch 6/40

137/137 [=====] - 53s 387ms/step - loss: 0.0174 - accuracy: 0.9911 - mean_io_u_1: 0.4865 - val_loss: 0.0194 - val_accuracy: 0.9925 - val_mean_io_u_1: 0.4852

Epoch 00006: val_loss did not improve from 0.01653

Epoch 7/40

137/137 [=====] - 53s 385ms/step - loss: 0.0181 - accuracy: 0.9909 - mean_io_u_1: 0.4865 - val_loss: 0.0181 - val_accuracy: 0.9929 - val_mean_io_u_1: 0.4854

Epoch 00007: val_loss did not improve from 0.01653
Epoch 8/40
137/137 [=====] - 52s 383ms/step - loss: 0.0167 - accuracy: 0.9913 - mean_io_u_1: 0.4865 - val_loss: 0.0168 - val_accuracy: 0.9937 - val_mean_io_u_1: 0.4852

Epoch 00008: val_loss did not improve from 0.01653
Epoch 9/40
137/137 [=====] - 52s 384ms/step - loss: 0.0161 - accuracy: 0.9915 - mean_io_u_1: 0.4865 - val_loss: 0.0177 - val_accuracy: 0.9932 - val_mean_io_u_1: 0.4854

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00009: val_loss did not improve from 0.01653
Epoch 10/40
137/137 [=====] - 52s 382ms/step - loss: 0.0156 - accuracy: 0.9917 - mean_io_u_1: 0.4864 - val_loss: 0.0155 - val_accuracy: 0.9940 - val_mean_io_u_1: 0.4851

Epoch 00010: val_loss improved from 0.01653 to 0.01554, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5
Epoch 11/40
137/137 [=====] - 52s 382ms/step - loss: 0.0152 - accuracy: 0.9919 - mean_io_u_1: 0.4866 - val_loss: 0.0156 - val_accuracy: 0.9940 - val_mean_io_u_1: 0.4851

Epoch 00011: val_loss did not improve from 0.01554
Epoch 12/40
137/137 [=====] - 54s 391ms/step - loss: 0.0147 - accuracy: 0.9920 - mean_io_u_1: 0.4865 - val_loss: 0.0153 - val_accuracy: 0.9941 - val_mean_io_u_1: 0.4849

Epoch 00012: val_loss improved from 0.01554 to 0.01527, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5
Epoch 13/40
137/137 [=====] - 53s 385ms/step - loss: 0.0143 - accuracy: 0.9921 - mean_io_u_1: 0.4865 - val_loss: 0.0148 - val_accuracy: 0.9944 - val_mean_io_u_1: 0.4861

Epoch 00013: val_loss improved from 0.01527 to 0.01476, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5
Epoch 14/40
137/137 [=====] - 52s 383ms/step - loss: 0.0144 - accuracy: 0.9922 - mean_io_u_1: 0.4866 - val_loss: 0.0149 - val_accuracy: 0.9942 - val_mean_io_u_1: 0.4858

Epoch 00014: val_loss did not improve from 0.01476
Epoch 15/40

137/137 [=====] - 52s 384ms/step - loss: 0.0144 - accuracy: 0.9921 - mean_io_u_1: 0.4865 - val_loss: 0.0145 - val_accuracy: 0.9944 - val_mean_io_u_1: 0.4853

Epoch 00015: val_loss improved from 0.01476 to 0.01449, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 16/40

137/137 [=====] - 52s 382ms/step - loss: 0.0150 - accuracy: 0.9920 - mean_io_u_1: 0.4865 - val_loss: 0.0143 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4855

Epoch 00016: val_loss improved from 0.01449 to 0.01432, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 17/40

137/137 [=====] - 52s 383ms/step - loss: 0.0143 - accuracy: 0.9922 - mean_io_u_1: 0.4865 - val_loss: 0.0146 - val_accuracy: 0.9944 - val_mean_io_u_1: 0.4852

Epoch 00017: val_loss did not improve from 0.01432

Epoch 18/40

137/137 [=====] - 52s 383ms/step - loss: 0.0142 - accuracy: 0.9922 - mean_io_u_1: 0.4865 - val_loss: 0.0142 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4852

Epoch 00018: val_loss improved from 0.01432 to 0.01418, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 19/40

137/137 [=====] - 52s 383ms/step - loss: 0.0146 - accuracy: 0.9922 - mean_io_u_1: 0.4865 - val_loss: 0.0152 - val_accuracy: 0.9941 - val_mean_io_u_1: 0.4849

Epoch 00019: val_loss did not improve from 0.01418

Epoch 20/40

137/137 [=====] - 52s 383ms/step - loss: 0.0140 - accuracy: 0.9923 - mean_io_u_1: 0.4865 - val_loss: 0.0139 - val_accuracy: 0.9946 - val_mean_io_u_1: 0.4853

Epoch 00020: val_loss improved from 0.01418 to 0.01386, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 21/40

137/137 [=====] - 52s 382ms/step - loss: 0.0141 - accuracy: 0.9923 - mean_io_u_1: 0.4865 - val_loss: 0.0151 - val_accuracy: 0.9943 - val_mean_io_u_1: 0.4854

Epoch 00021: val_loss did not improve from 0.01386

Epoch 22/40

137/137 [=====] - 52s 382ms/step - loss: 0.0143 - accuracy: 0.9922 - mean_io_u_1: 0.4865 - val_loss: 0.0151 - val_accuracy: 0.9942 - val_mean_io_u_1: 0.4846

Epoch 00022: val_loss did not improve from 0.01386

Epoch 23/40

137/137 [=====] - 52s 381ms/step - loss: 0.0132 - accuracy: 0.9924 - mean_io_u_1: 0.4865 - val_loss: 0.0136 - val_accuracy: 0.9947 - val_mean_io_u_1: 0.4854

Epoch 00023: val_loss improved from 0.01386 to 0.01364, saving model to /content/drive/MyDrive/TrainedModelNewNewNew.h5

Epoch 24/40

137/137 [=====] - 52s 383ms/step - loss: 0.0138 - accuracy: 0.9923 - mean_io_u_1: 0.4865 - val_loss: 0.0143 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4851

Epoch 00024: val_loss did not improve from 0.01364

Epoch 25/40

137/137 [=====] - 53s 390ms/step - loss: 0.0136 - accuracy: 0.9923 - mean_io_u_1: 0.4866 - val_loss: 0.0142 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4854

Epoch 00025: val_loss did not improve from 0.01364

Epoch 26/40

137/137 [=====] - 52s 382ms/step - loss: 0.0137 - accuracy: 0.9924 - mean_io_u_1: 0.4865 - val_loss: 0.0143 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4852

Epoch 00026: val_loss did not improve from 0.01364

Epoch 27/40

137/137 [=====] - 52s 383ms/step - loss: 0.0140 - accuracy: 0.9923 - mean_io_u_1: 0.4865 - val_loss: 0.0142 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4853

Epoch 00027: val_loss did not improve from 0.01364

Epoch 28/40

137/137 [=====] - 52s 383ms/step - loss: 0.0133 - accuracy: 0.9924 - mean_io_u_1: 0.4865 - val_loss: 0.0139 - val_accuracy: 0.9946 - val_mean_io_u_1: 0.4852

Epoch 00028: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.

Epoch 00028: val_loss did not improve from 0.01364

Epoch 29/40

137/137 [=====] - 52s 384ms/step - loss: 0.0139 - accuracy: 0.9924 - mean_io_u_1: 0.4865 - val_loss: 0.0139 - val_accuracy: 0.9947 - val_mean_io_u_1: 0.4858

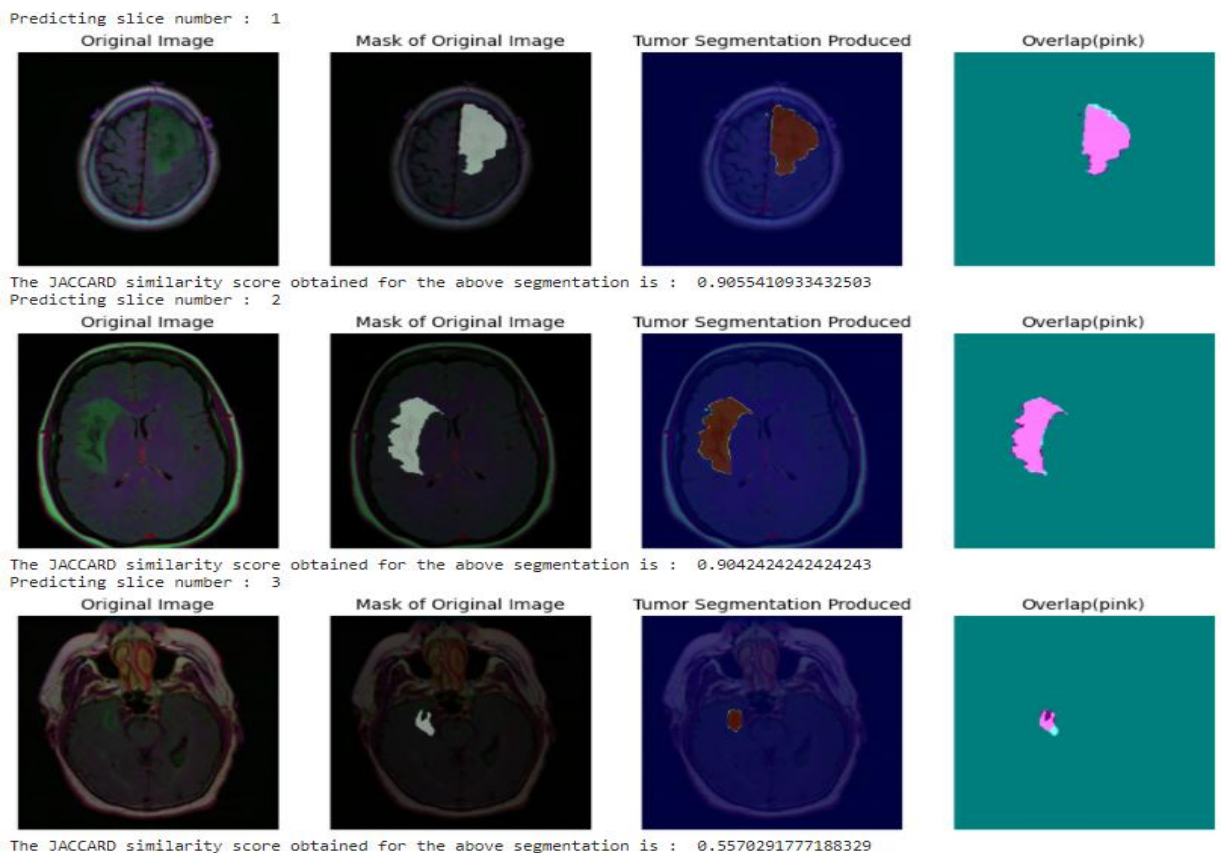
Epoch 00029: val_loss did not improve from 0.01364

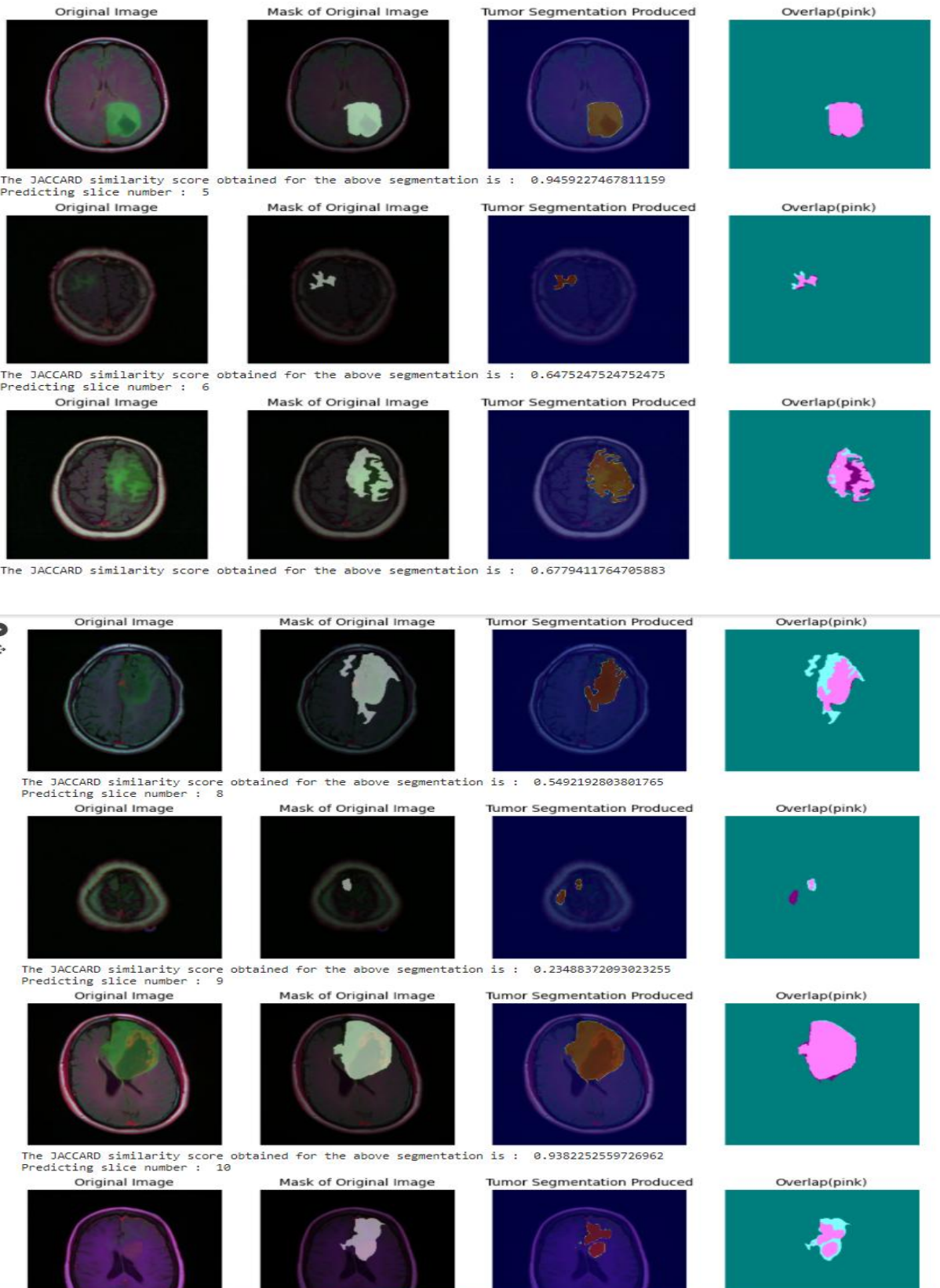
Epoch 30/40

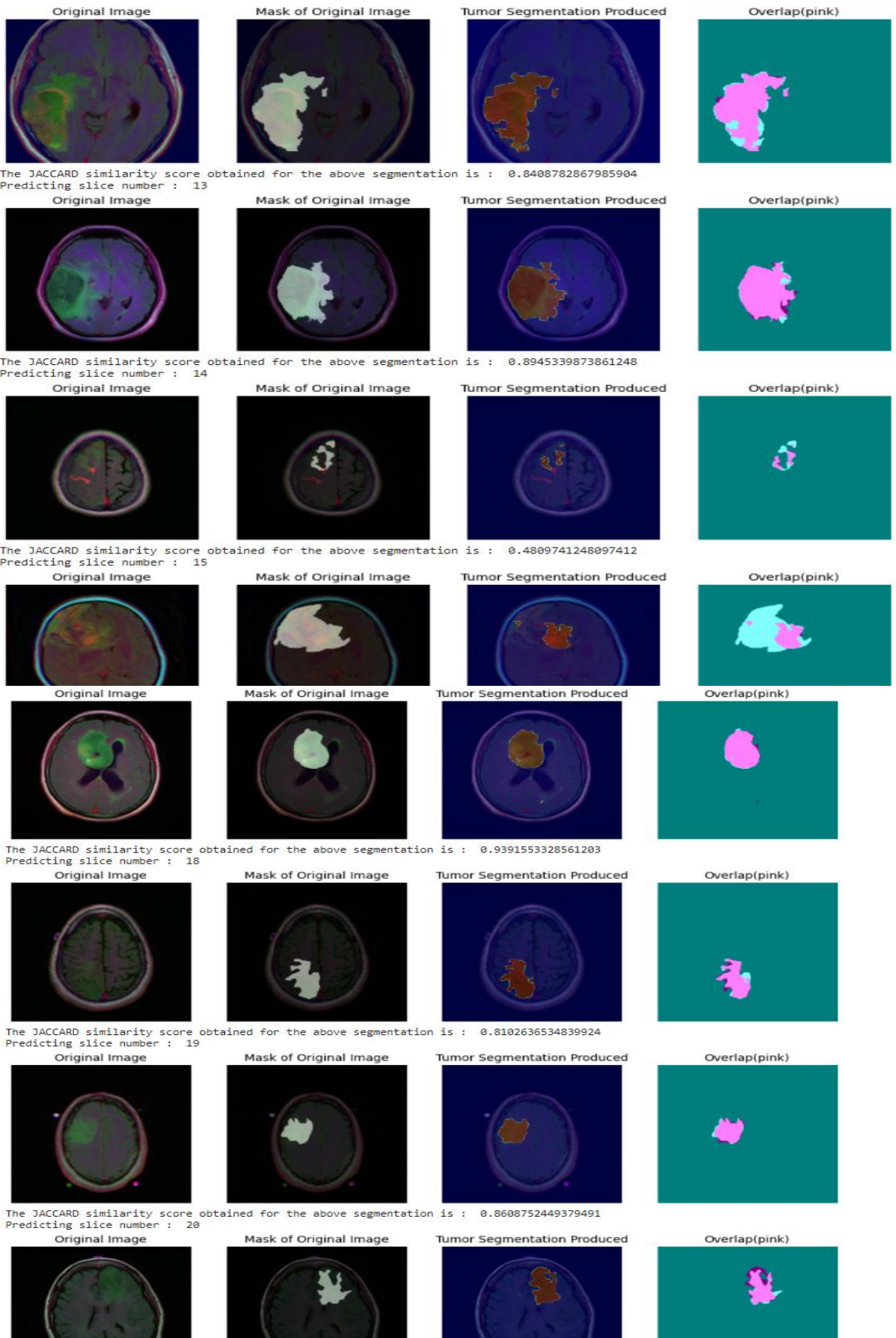
137/137 [=====] - 52s 382ms/step - loss: 0.0134 - accuracy: 0.9925 - mean_io_u_1: 0.4866 - val_loss: 0.0144 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4849

Epoch 00030: val_loss did not improve from 0.01364
 Epoch 31/40
 137/137 [=====] - 53s 389ms/step - loss: 0.0138 - accuracy: 0.9924 - mean_io_u_1: 0.4865 - val_loss: 0.0142 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4852
 Epoch 00031: val_loss did not improve from 0.01364
 Epoch 32/40
 137/137 [=====] - 53s 385ms/step - loss: 0.0132 - accuracy: 0.9925 - mean_io_u_1: 0.4866 - val_loss: 0.0144 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4852
 Epoch 00032: val_loss did not improve from 0.01364
 Epoch 33/40
 137/137 [=====] - 52s 384ms/step - loss: 0.0132 - accuracy: 0.9925 - mean_io_u_1: 0.4865 - val_loss: 0.0143 - val_accuracy: 0.9945 - val_mean_io_u_1: 0.4852
 Epoch 00033: ReduceLROnPlateau reducing learning rate to 1e-05.
 Epoch 00033: val_loss did not improve from 0.01364
 Epoch 00033: early stopping

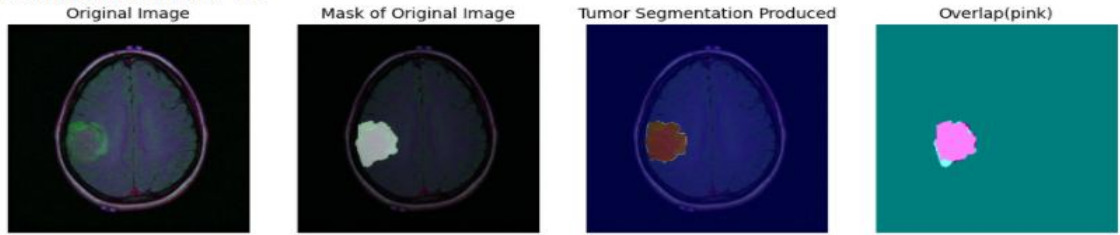
Tumor Segmentation Result Produced from the uNet Model :



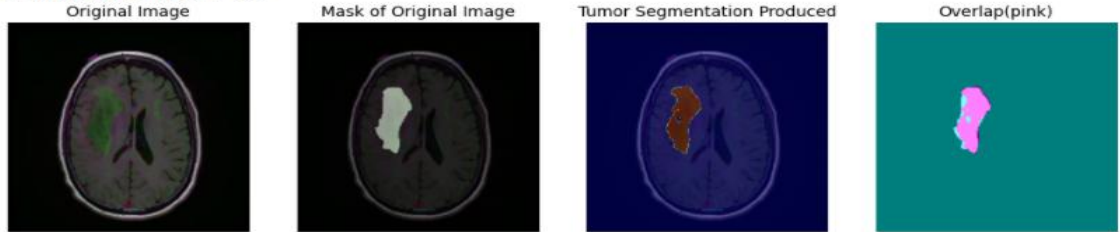




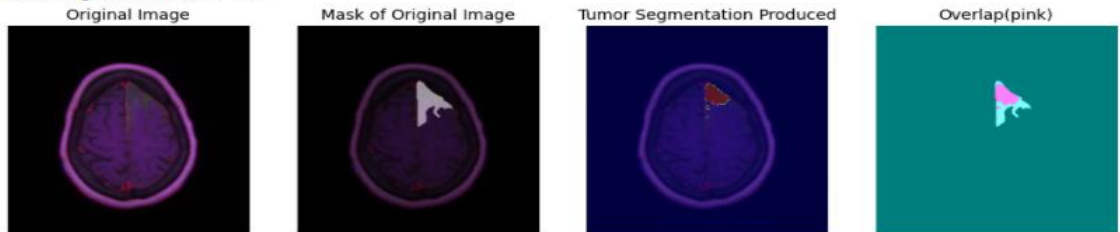
The JACCARD similarity score obtained for the above segmentation is : 0.7129353233830846
Predicting slice number : 21



The JACCARD similarity score obtained for the above segmentation is : 0.8782701548318206
Predicting slice number : 22

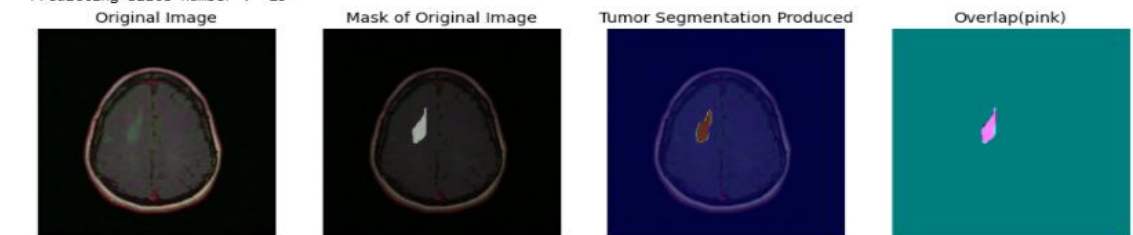


The JACCARD similarity score obtained for the above segmentation is : 0.8377239199157007
Predicting slice number : 23

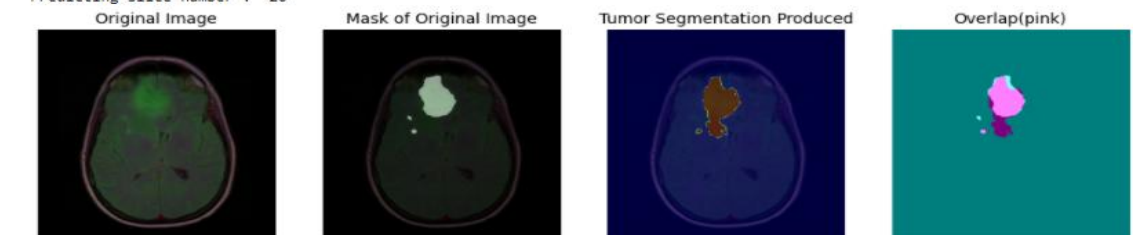


The JACCARD similarity score obtained for the above segmentation is : 0.4715359828141783
Predicting slice number : 24

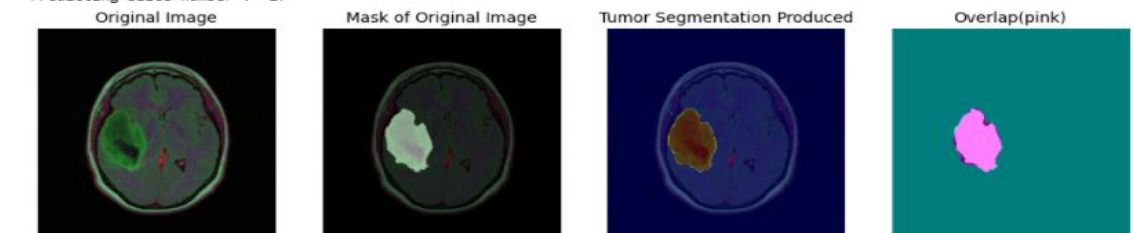
↳ Predicting slice number : 25



The JACCARD similarity score obtained for the above segmentation is : 0.8239795918367347
Predicting slice number : 26

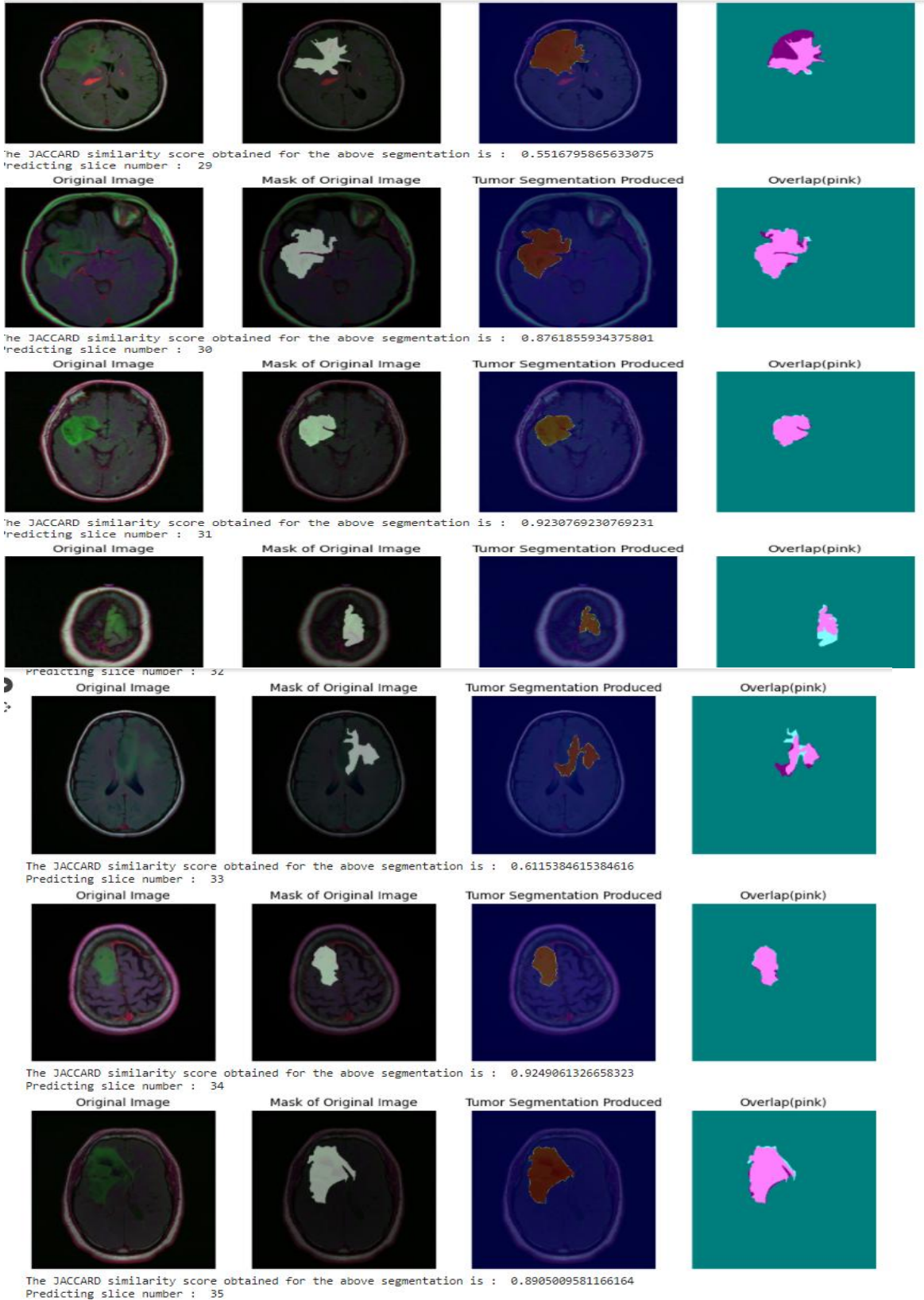


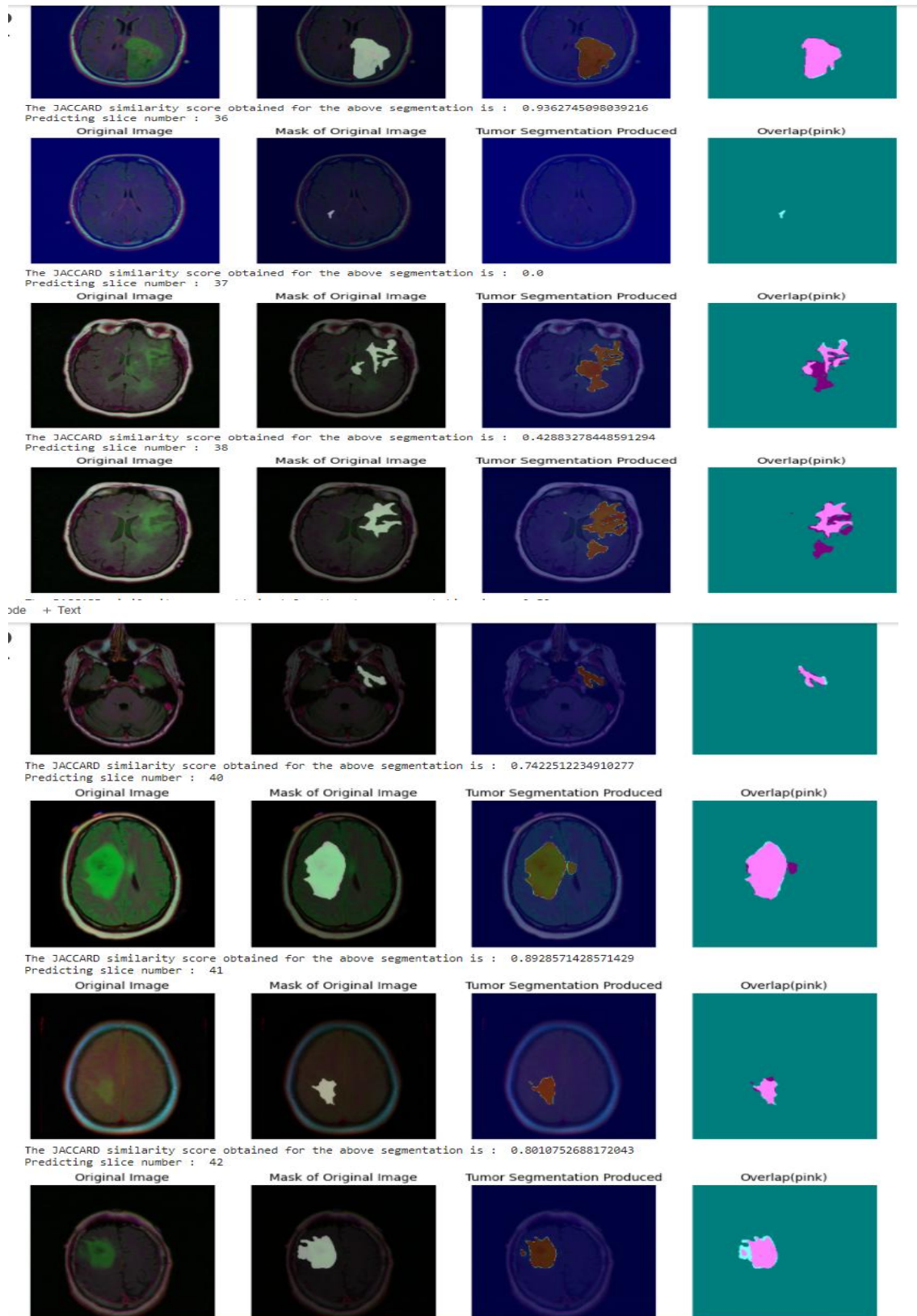
The JACCARD similarity score obtained for the above segmentation is : 0.6522427440633245
Predicting slice number : 27

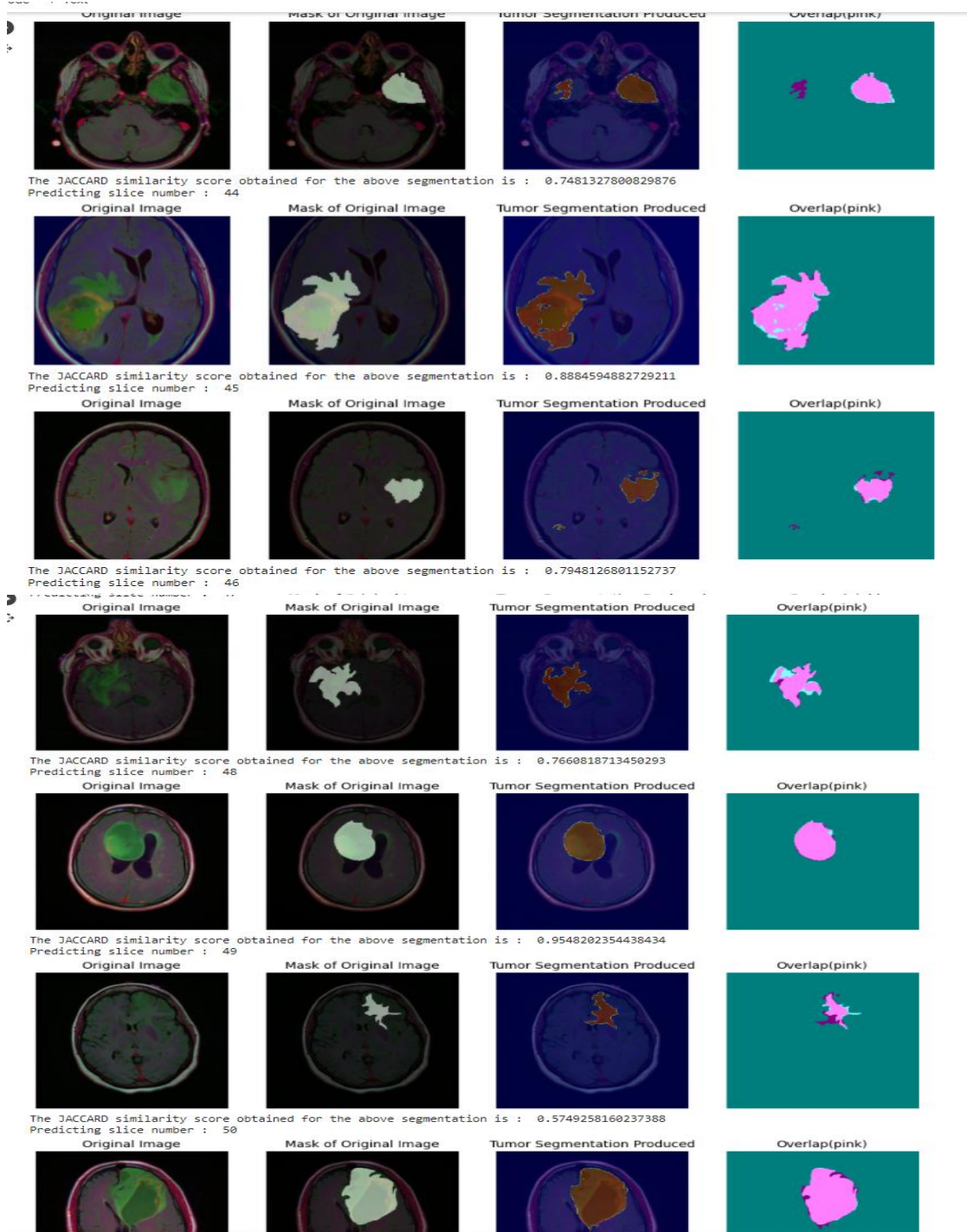


The JACCARD similarity score obtained for the above segmentation is : 0.9234414874225301
Predicting slice number : 28

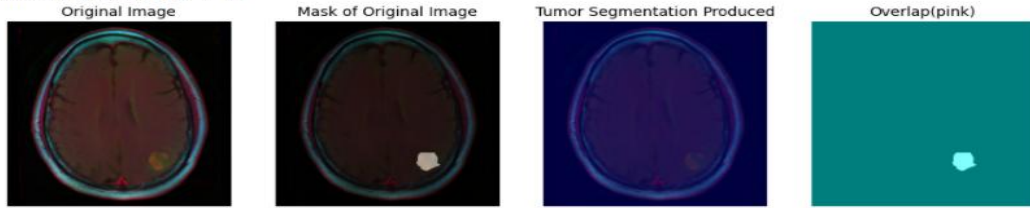
+ text



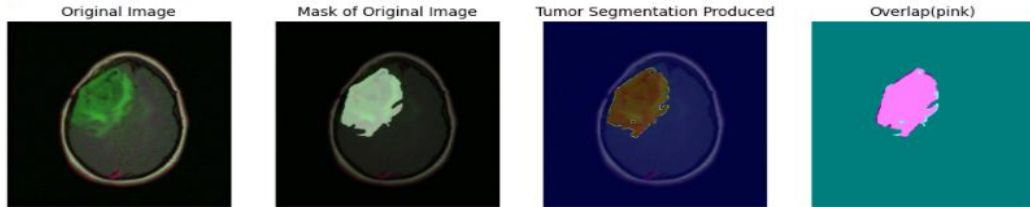




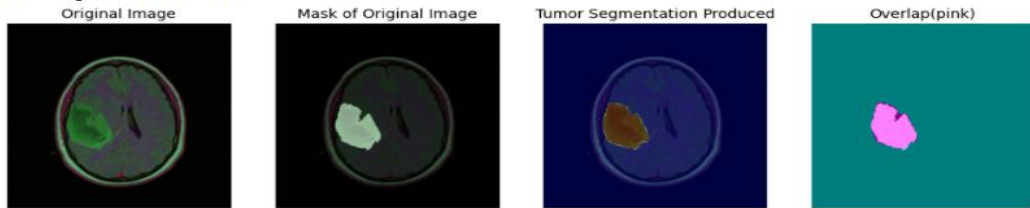
The JACCARD similarity score obtained for the above segmentation is : 0.9195518089029688
Predicting slice number : 51



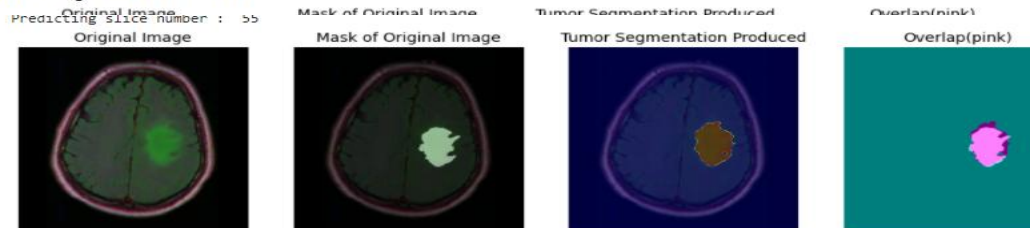
The JACCARD similarity score obtained for the above segmentation is : 0.0
Predicting slice number : 52



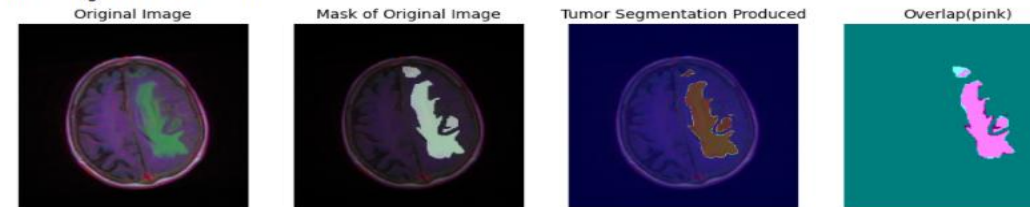
The JACCARD similarity score obtained for the above segmentation is : 0.9254857142857142
Predicting slice number : 53



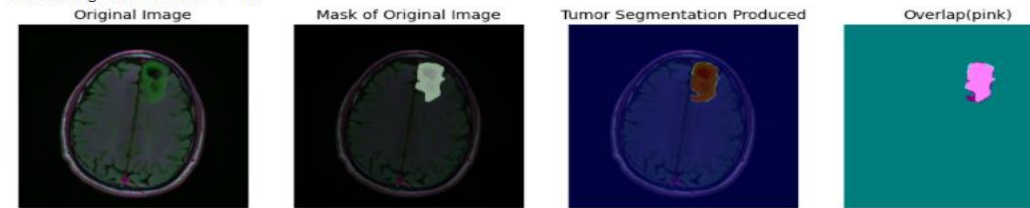
The JACCARD similarity score obtained for the above segmentation is : 0.9196237708422402
Predicting slice number : 54



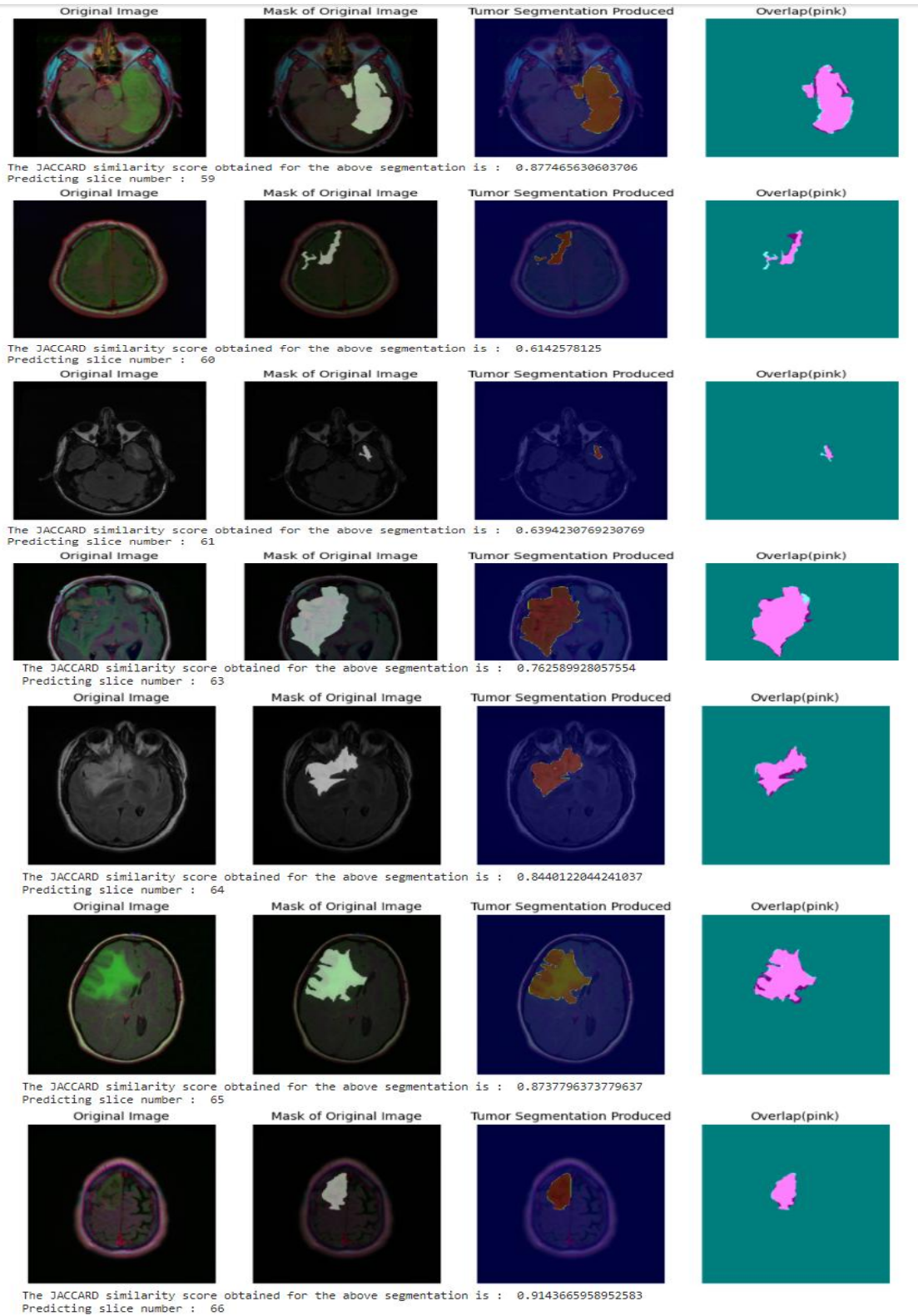
The JACCARD similarity score obtained for the above segmentation is : 0.7853483606557377
Predicting slice number : 55



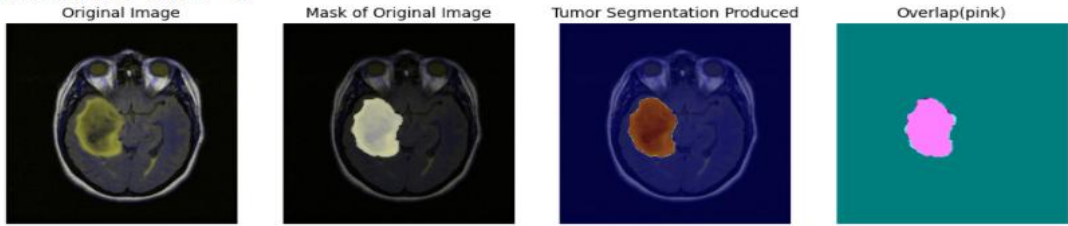
The JACCARD similarity score obtained for the above segmentation is : 0.8406015037593985
Predicting slice number : 57



The JACCARD similarity score obtained for the above segmentation is : 0.8650963597430407
Predicting slice number : 58

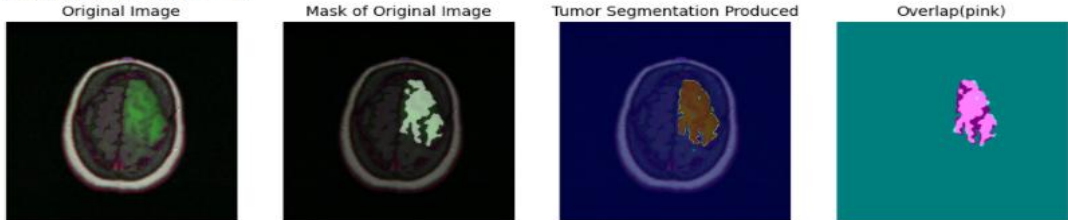


Predicting slice number : 66



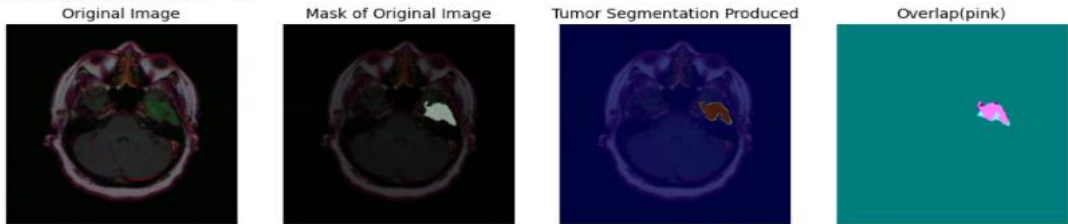
The JACCARD similarity score obtained for the above segmentation is : 0.9487424387137854

Predicting slice number : 67



The JACCARD similarity score obtained for the above segmentation is : 0.708139534883721

Predicting slice number : 68



The JACCARD similarity score obtained for the above segmentation is : 0.7701863354037267

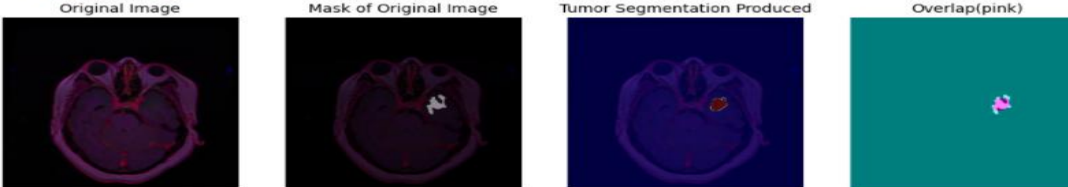
Predicting slice number : 69

ode + text



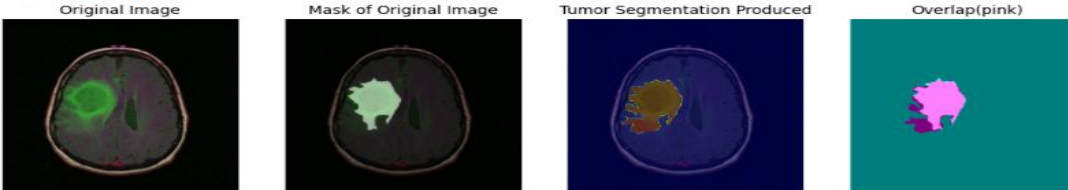
The JACCARD similarity score obtained for the above segmentation is : 0.6871961102106969

Predicting slice number : 71



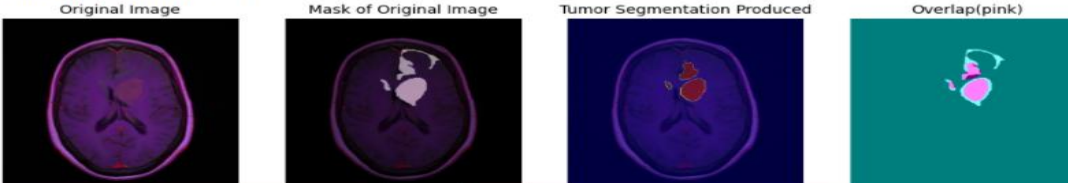
The JACCARD similarity score obtained for the above segmentation is : 0.5780346820809249

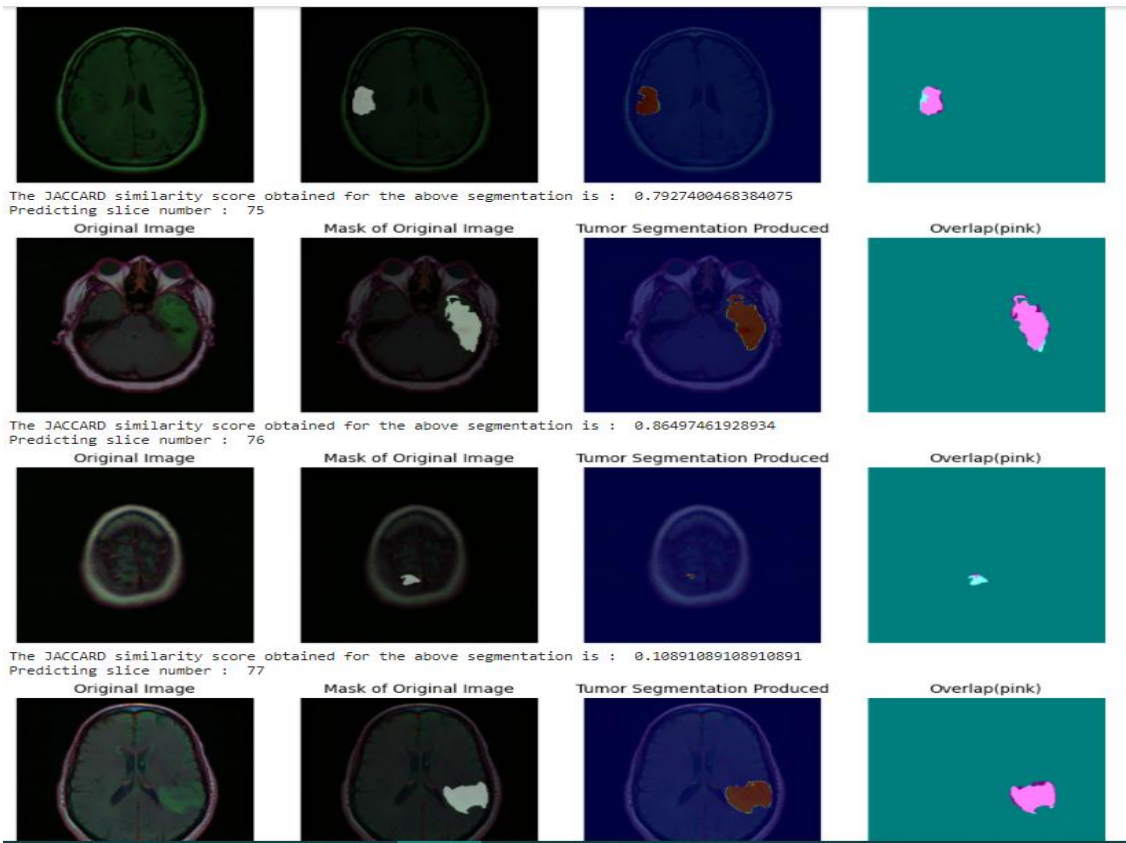
Predicting slice number : 72

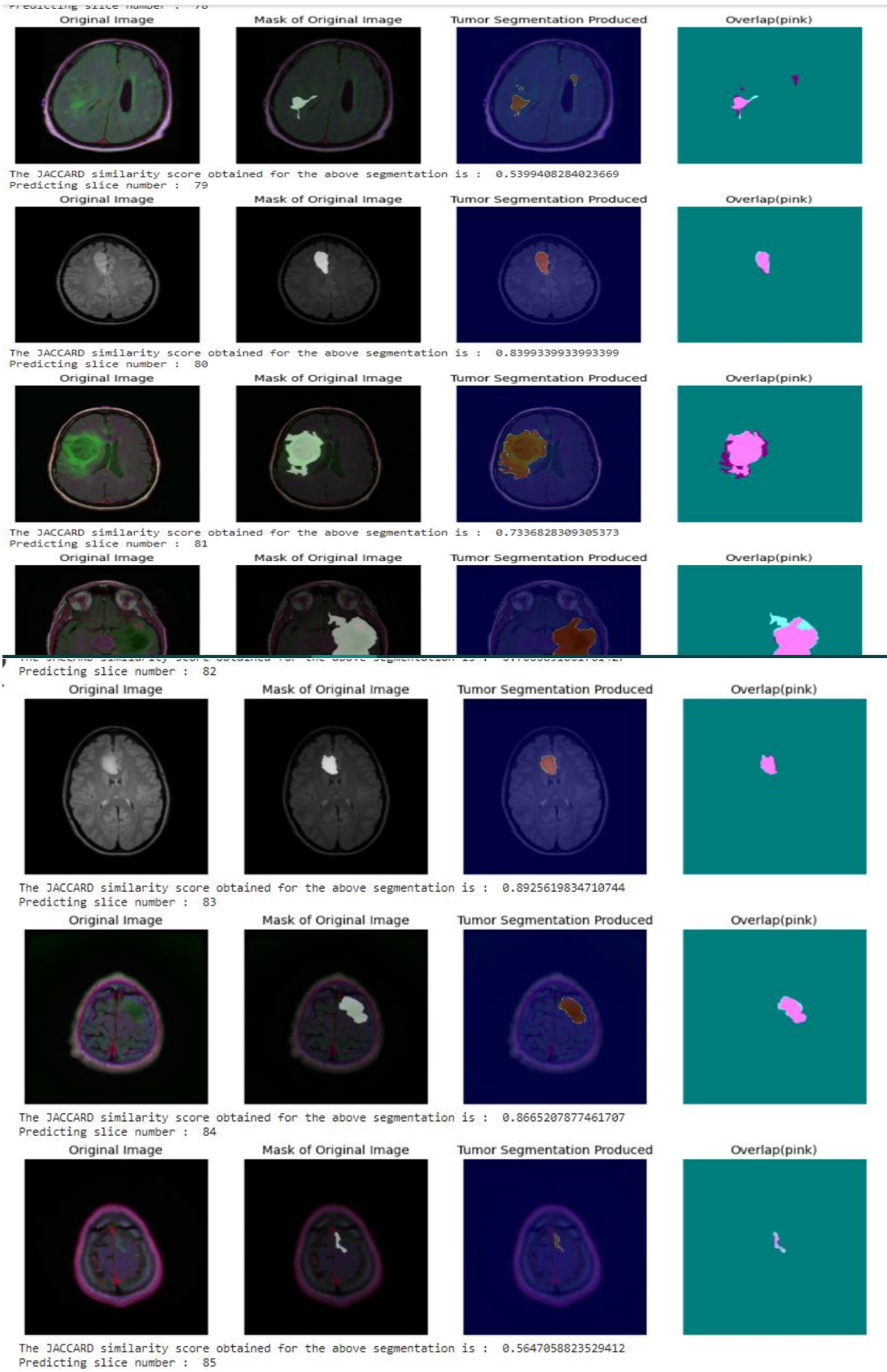


The JACCARD similarity score obtained for the above segmentation is : 0.7511606313834726

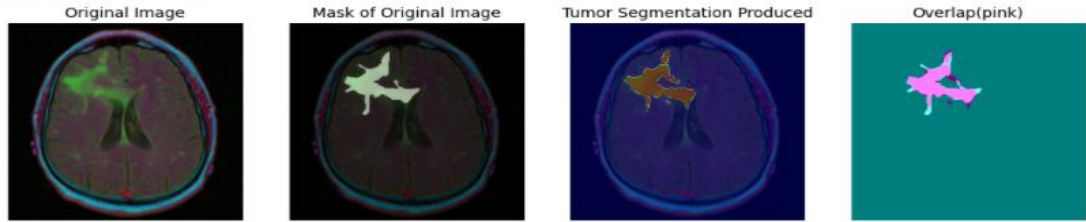
Predicting slice number : 73





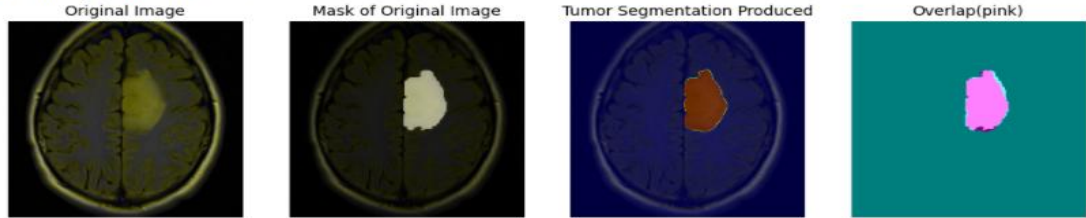


Predicting slice number : 90



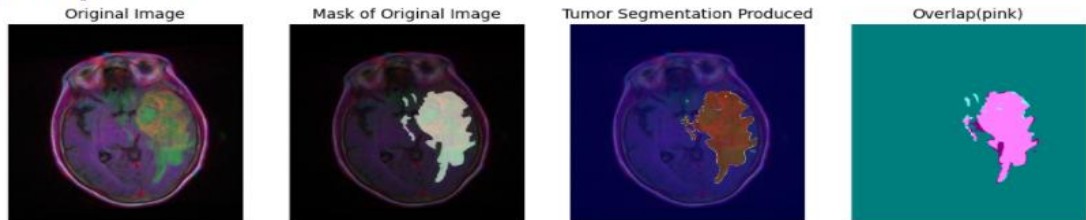
The JACCARD similarity score obtained for the above segmentation is : 0.7838214783821479

Predicting slice number : 91



The JACCARD similarity score obtained for the above segmentation is : 0.9031387107661154

Predicting slice number : 92

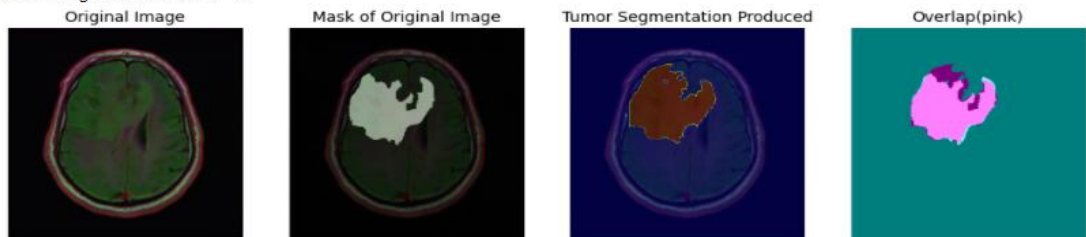


The JACCARD similarity score obtained for the above segmentation is : 0.8636363636363636

Predicting slice number : 93

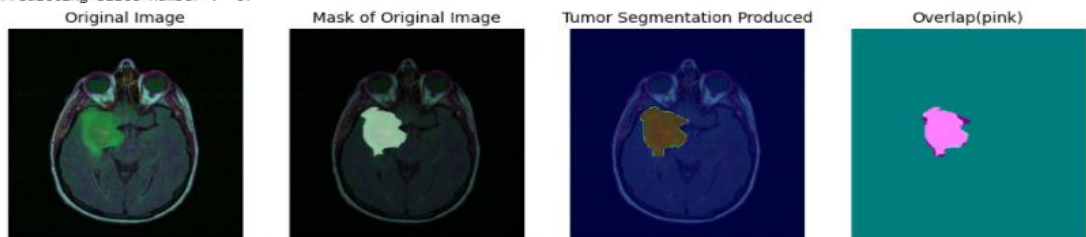
The JACCARD similarity score obtained for the above segmentation is : 0.8397584765443568

Predicting slice number : 86



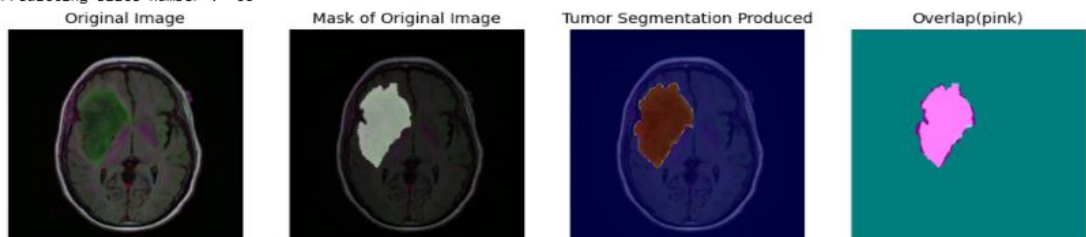
The JACCARD similarity score obtained for the above segmentation is : 0.8021665778724916

Predicting slice number : 87



The JACCARD similarity score obtained for the above segmentation is : 0.8970373967945605

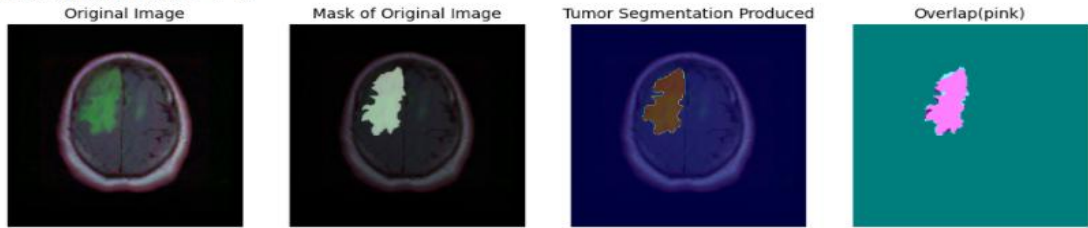
Predicting slice number : 88



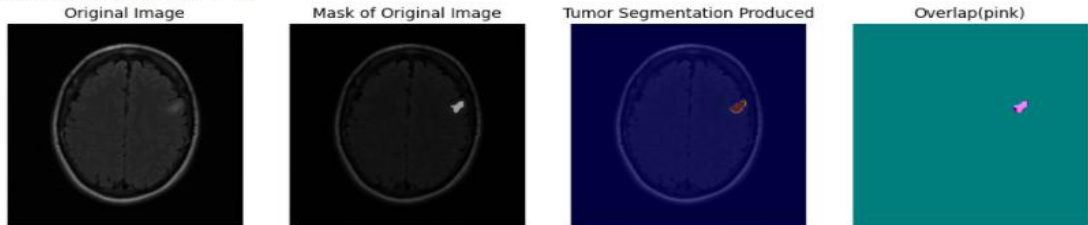
The JACCARD similarity score obtained for the above segmentation is : 0.8939800146234462

Predicting slice number : 89

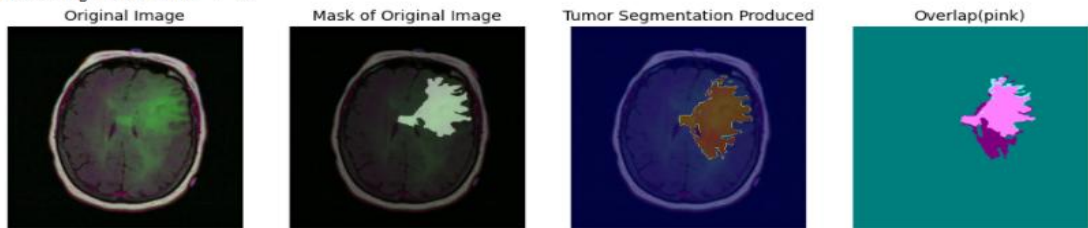
Predicting slice number : 93



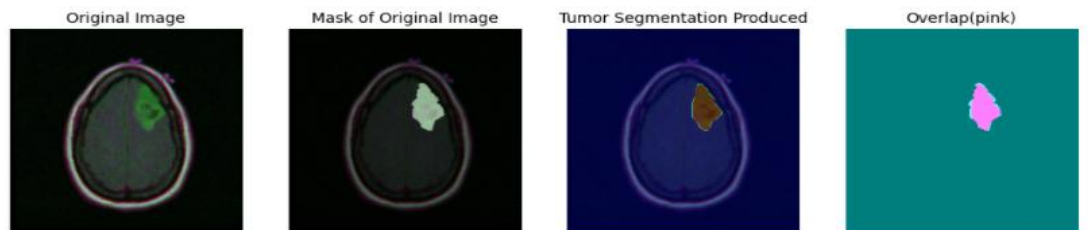
The JACCARD similarity score obtained for the above segmentation is : 0.8838323353293414
Predicting slice number : 94



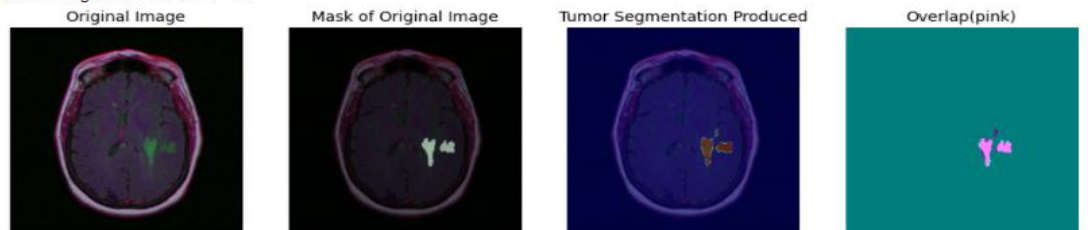
The JACCARD similarity score obtained for the above segmentation is : 0.7241379310344828
Predicting slice number : 95



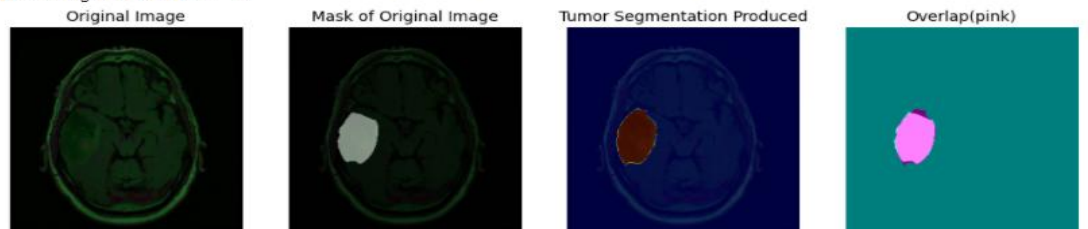
The JACCARD similarity score obtained for the above segmentation is : 0.6164316966456004
Predicting slice number : 96



The JACCARD similarity score obtained for the above segmentation is : 0.8847352024922118
Predicting slice number : 98



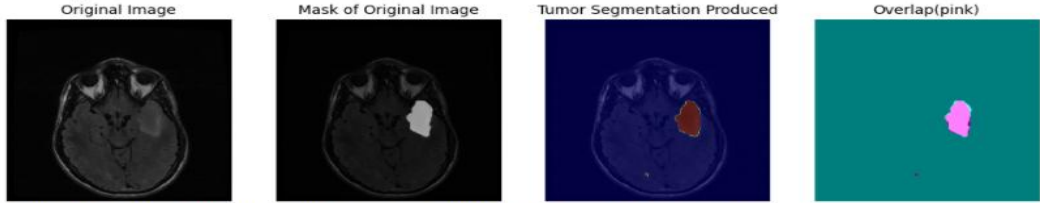
The JACCARD similarity score obtained for the above segmentation is : 0.7929759704251387
Predicting slice number : 99



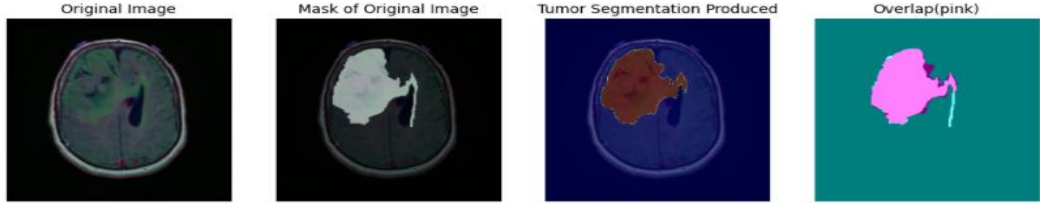
The JACCARD similarity score obtained for the above segmentation is : 0.9055289508053983
Predicting slice number : 100



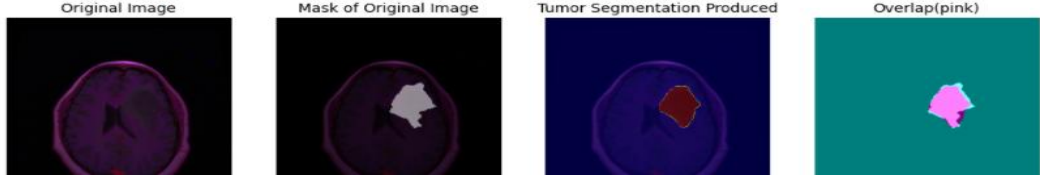
The JACCARD similarity score obtained for the above segmentation is : 0.9592070148684713
Predicting slice number : 101



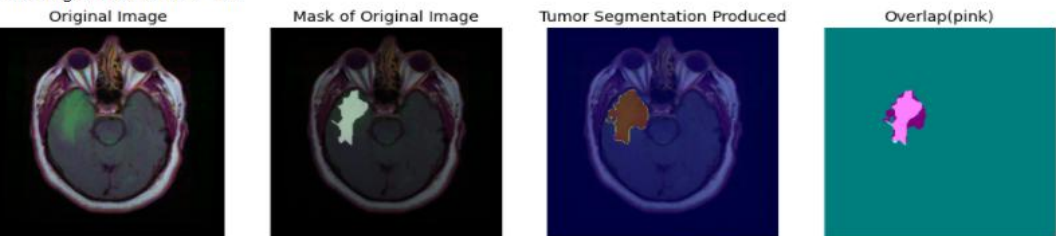
The JACCARD similarity score obtained for the above segmentation is : 0.8830935251798561
Predicting slice number : 102



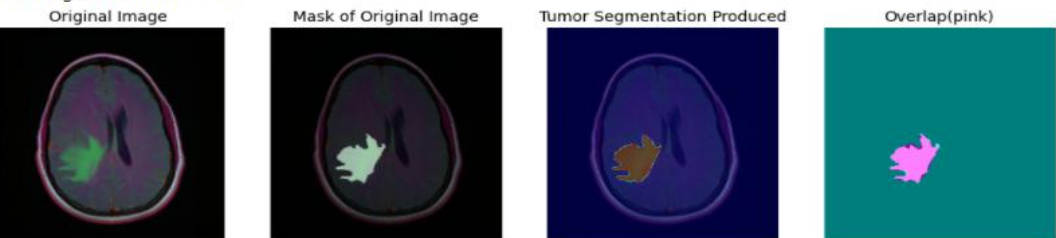
The JACCARD similarity score obtained for the above segmentation is : 0.893565539429125
Predicting slice number : 103



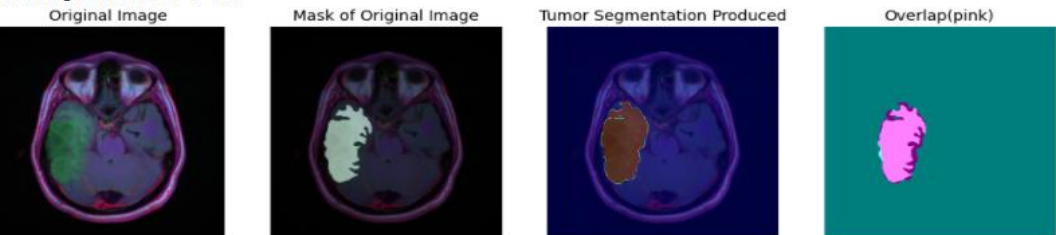
Predicting slice number : 105



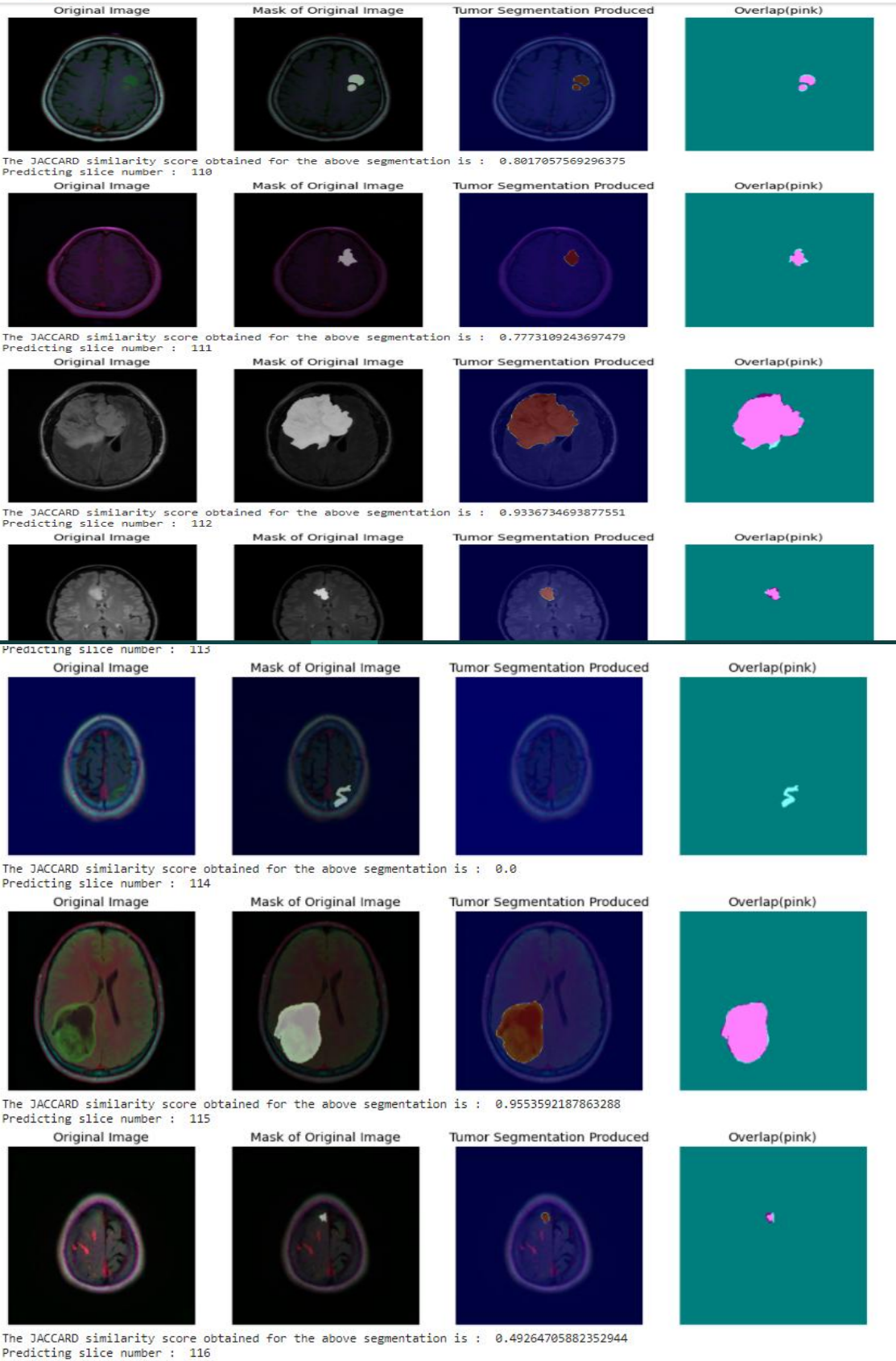
The JACCARD similarity score obtained for the above segmentation is : 0.656993615786419
Predicting slice number : 106

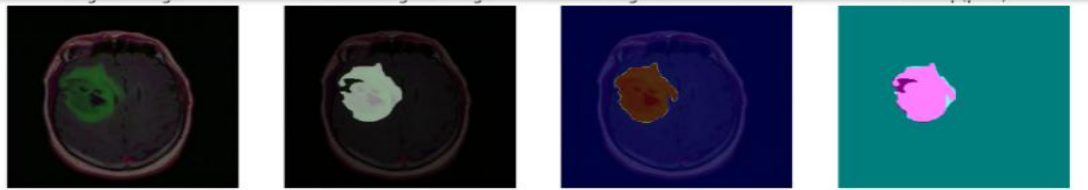


The JACCARD similarity score obtained for the above segmentation is : 0.9037463976945245
Predicting slice number : 107



The JACCARD similarity score obtained for the above segmentation is : 0.803409403354413





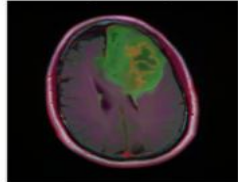
The JACCARD similarity score obtained for the above segmentation is : 0.8816152495610735
 Predicting slice number : 118

Original Image

Mask of Original Image

Tumor Segmentation Produced

Overlap(pink)



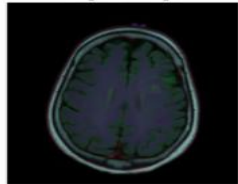
The JACCARD similarity score obtained for the above segmentation is : 0.9472759226713533
 Predicting slice number : 119

Original Image

Mask of Original Image

Tumor Segmentation Produced

Overlap(pink)



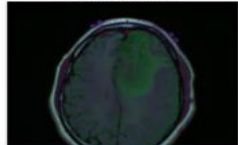
The JACCARD similarity score obtained for the above segmentation is : 0.35377358490566035
 Predicting slice number : 120

Original Image

Mask of Original Image

Tumor Segmentation Produced

Overlap(pink)



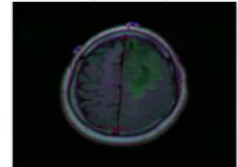
Predicting slice number : 121

Original Image

Mask of Original Image

Tumor Segmentation Produced

Overlap(pink)



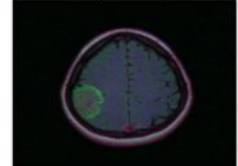
The JACCARD similarity score obtained for the above segmentation is : 0.8422266139657444
 Predicting slice number : 122

Original image

Mask of Original image

Tumor Segmentation Produced

Overlap(pink)



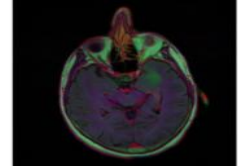
The JACCARD similarity score obtained for the above segmentation is : 0.9296987087517934
 Predicting slice number : 123

Original image

Mask of Original image

Tumor Segmentation Produced

Overlap(pink)



The JACCARD similarity score obtained for the above segmentation is : 0.6836461126005362
 Predicting slice number : 124

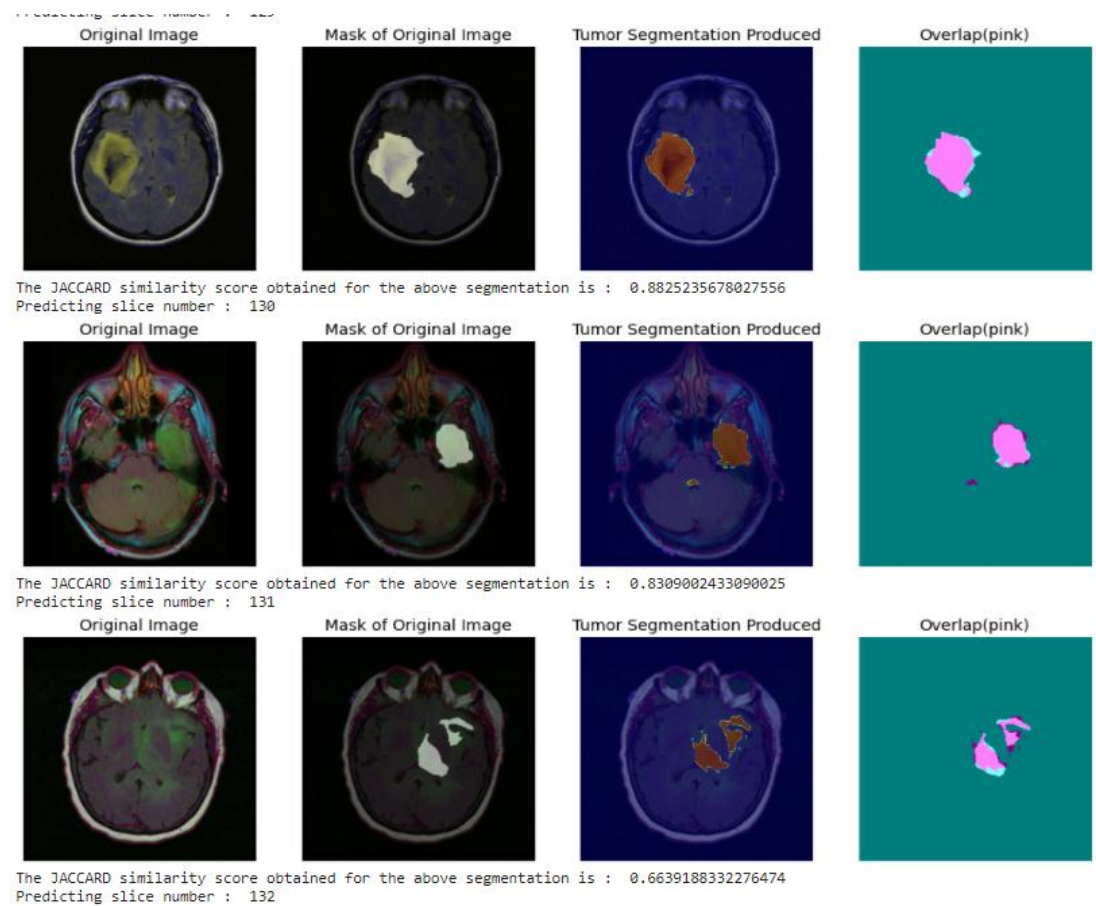
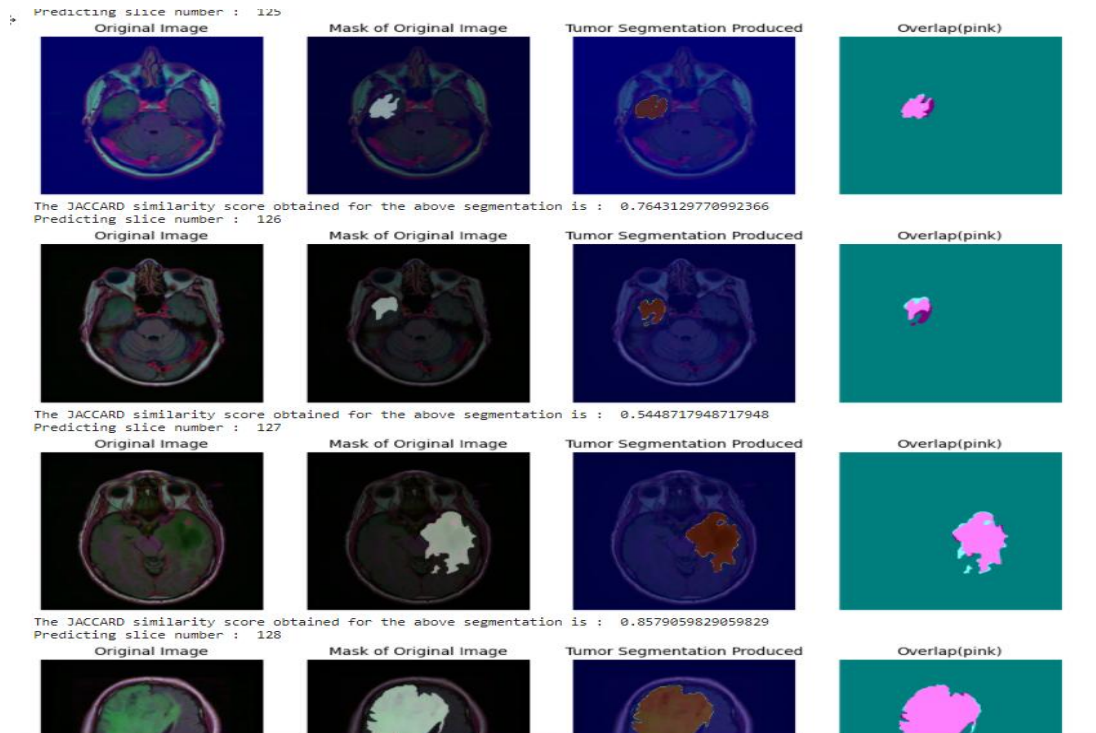
Original image

Mask of Original image

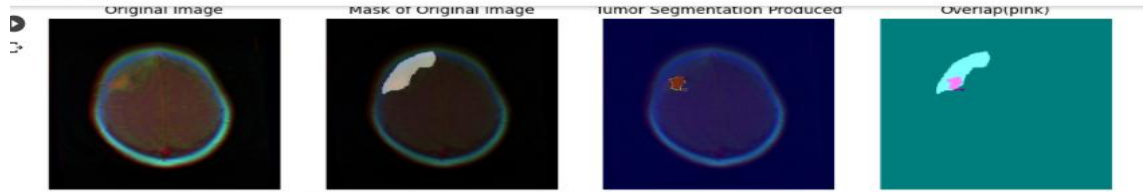
Tumor Segmentation Produced

Overlap(pink)

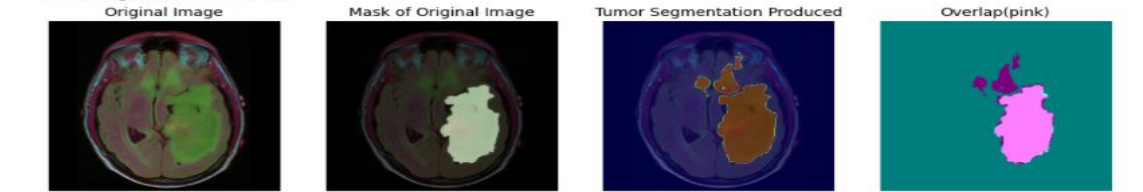




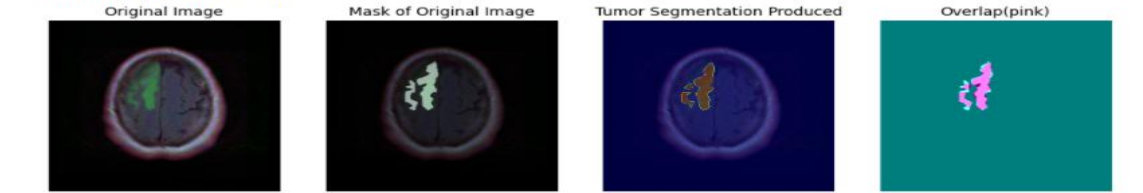
Code + Text



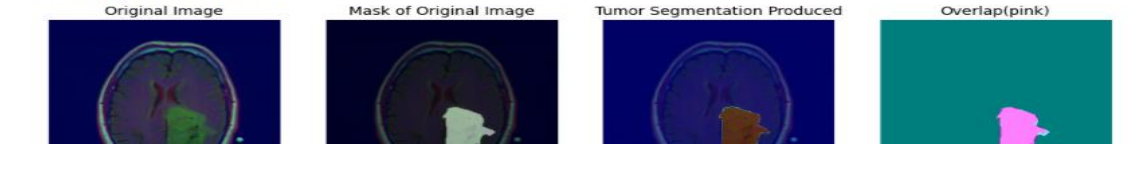
The JACCARD similarity score obtained for the above segmentation is : 0.1325898389095415
Predicting slice number : 133



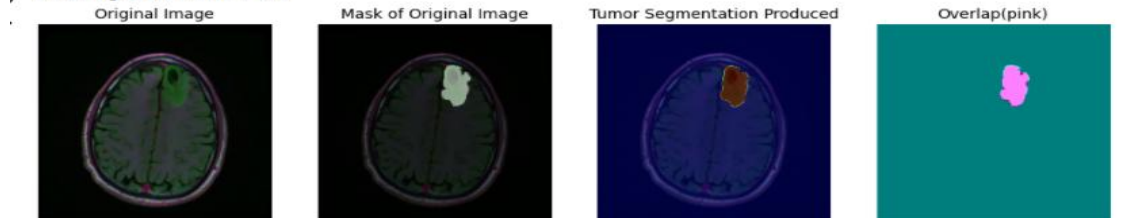
The JACCARD similarity score obtained for the above segmentation is : 0.7772511848341233
Predicting slice number : 134



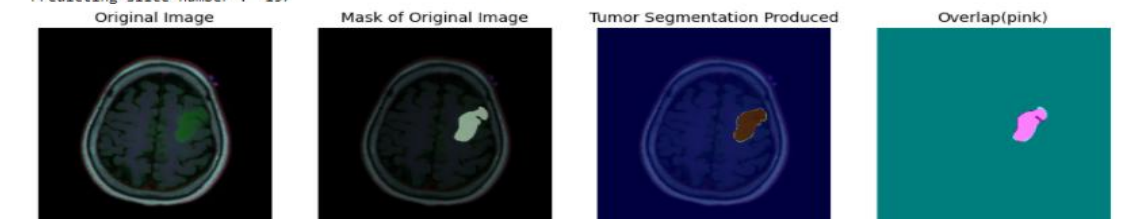
The JACCARD similarity score obtained for the above segmentation is : 0.7672209026128266
Predicting slice number : 135



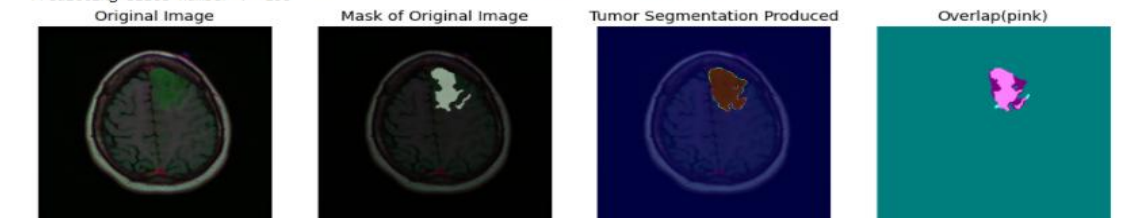
Predicting slice number : 136



The JACCARD similarity score obtained for the above segmentation is : 0.9278267842822775
Predicting slice number : 137



The JACCARD similarity score obtained for the above segmentation is : 0.8771929824561403
Predicting slice number : 138



The JACCARD similarity score obtained for the above segmentation is : 0.648936170212766
Average Testing Segmentation Accuracy : 0.7531473683472363

APPENDIX B: segUnet model implementation based on [8]

```
## Implemented by Darshan
from keras.layers import Concatenate
from keras.layers.convolutional import Convolution2D

def CreateSegUNet(input_shape, n_labels, kernel=3, pool_size=(2, 2), output_mode="sigmoid"):
    inputs = Input(shape=input_shape)

    # encoder phase, similar to uNet(left side)
    conv_1 = Convolution2D(16, (kernel, kernel), padding="same")(inputs)
    conv_1 = BatchNormalization()(conv_1)
    conv_1 = Activation("relu")(conv_1)
    conv_2 = Convolution2D(16, (kernel, kernel), padding="same")(conv_1)
    conv_2 = BatchNormalization()(conv_2)
    conv_2 = Activation("relu")(conv_2)
    ## conv block 1 -----encoder(2 layers)
    pool_1, mask_1 = MaxPoolingWithArgmax2D(pool_size)(conv_2)

    conv_3 = Convolution2D(32, (kernel, kernel), padding="same")(pool_1)
    conv_3 = BatchNormalization()(conv_3)
    conv_3 = Activation("relu")(conv_3)
    conv_4 = Convolution2D(32, (kernel, kernel), padding="same")(conv_3)
    conv_4 = BatchNormalization()(conv_4)
    conv_4 = Activation("relu")(conv_4)
    ## conv block 2 -----encoder(2 layers)
    pool_2, mask_2 = MaxPoolingWithArgmax2D(pool_size)(conv_4)

    conv_5 = Convolution2D(64, (kernel, kernel), padding="same")(pool_2)
    conv_5 = BatchNormalization()(conv_5)
    conv_5 = Activation("relu")(conv_5)
    conv_6 = Convolution2D(64, (kernel, kernel), padding="same")(conv_5)
    conv_6 = BatchNormalization()(conv_6)
    conv_6 = Activation("relu")(conv_6)
    conv_7 = Convolution2D(64, (kernel, kernel), padding="same")(conv_6)
    conv_7 = BatchNormalization()(conv_7)
    conv_7 = Activation("relu")(conv_7)
    ## conv block 3 -----encoder(3 layers)
    pool_3, mask_3 = MaxPoolingWithArgmax2D(pool_size)(conv_7)

    conv_8 = Convolution2D(128, (kernel, kernel), padding="same")(pool_3)
    conv_8 = BatchNormalization()(conv_8)
    conv_8 = Activation("relu")(conv_8)
    conv_9 = Convolution2D(128, (kernel, kernel), padding="same")(conv_8)
    conv_9 = BatchNormalization()(conv_9)
    conv_9 = Activation("relu")(conv_9)
    conv_10 = Convolution2D(128, (kernel, kernel), padding="same")(conv_9)
    conv_10 = BatchNormalization()(conv_10)
    conv_10 = Activation("relu")(conv_10)
    ## conv block 4 -----encoder(3 layers)
    pool_4, mask_4 = MaxPoolingWithArgmax2D(pool_size)(conv_10)

    conv_11 = Convolution2D(128, (kernel, kernel), padding="same")(pool_4)
    conv_11 = BatchNormalization()(conv_11)
    conv_11 = Activation("relu")(conv_11)
    conv_12 = Convolution2D(128, (kernel, kernel), padding="same")(conv_11)
    conv_12 = BatchNormalization()(conv_12)
    conv_12 = Activation("relu")(conv_12)
    conv_13 = Convolution2D(128, (kernel, kernel), padding="same")(conv_12)
    conv_13 = BatchNormalization()(conv_13)
    conv_13 = Activation("relu")(conv_13)
    ## conv block 5 -----encoder(3 layers)
    pool_5, mask_5 = MaxPoolingWithArgmax2D(pool_size)(conv_13)
    print("Build encoder done..")

    # between encoder and decoder - this is the bottleneck layer-----joins the encoder and decoder
    conv_14 = Convolution2D(128, (kernel, kernel), padding="same")(pool_5)
    conv_14 = BatchNormalization()(conv_14)
    conv_14 = Activation("relu")(conv_14)
    conv_15 = Convolution2D(128, (kernel, kernel), padding="same")(conv_14)
```

```

# between encoder and decoder - this is the bottleneck layer-----joins the encoder and decoder
conv_14 = Convolution2D(128, (kernel, kernel), padding="same")(pool_5)
conv_14 = BatchNormalization()(conv_14)
conv_14 = Activation("relu")(conv_14)
conv_15 = Convolution2D(128, (kernel, kernel), padding="same")(conv_14)
conv_15 = BatchNormalization()(conv_15)
conv_15 = Activation("relu")(conv_15)
conv_16 = Convolution2D(128, (kernel, kernel), padding="same")(conv_15)
conv_16 = BatchNormalization()(conv_16)
conv_16 = Activation("relu")(conv_16)

# decoder - right side, similar to uNet
unpool_1 = MaxUnpooling2D(pool_size)([conv_16, mask_5])
concat_1 = Concatenate()(unpool_1, conv_13)

conv_17 = Convolution2D(128, (kernel, kernel), padding="same")(concat_1)
conv_17 = BatchNormalization()(conv_17)
conv_17 = Activation("relu")(conv_17)
conv_18 = Convolution2D(128, (kernel, kernel), padding="same")(conv_17)
conv_18 = BatchNormalization()(conv_18)
conv_18 = Activation("relu")(conv_18)
conv_19 = Convolution2D(128, (kernel, kernel), padding="same")(conv_18)
conv_19 = BatchNormalization()(conv_19)
conv_19 = Activation("relu")(conv_19)
## deconv block 1 -----decoder(3 layers)
unpool_2 = MaxUnpooling2D(pool_size)([conv_19, mask_4])
concat_2 = Concatenate()(unpool_2, conv_10)

conv_20 = Convolution2D(128, (kernel, kernel), padding="same")(concat_2)
conv_20 = BatchNormalization()(conv_20)
conv_20 = Activation("relu")(conv_20)
conv_21 = Convolution2D(128, (kernel, kernel), padding="same")(conv_20)
conv_21 = BatchNormalization()(conv_21)
conv_21 = Activation("relu")(conv_21)
conv_22 = Convolution2D(64, (kernel, kernel), padding="same")(conv_21)
conv_22 = BatchNormalization()(conv_22)
conv_22 = Activation("relu")(conv_22)
## deconv block 2 -----decoder(3 layers)
unpool_3 = MaxUnpooling2D(pool_size)([conv_22, mask_3])
concat_3 = Concatenate()(unpool_3, conv_7)

```

```

conv_22 = BatchNormalization()(conv_22)
conv_22 = Activation("relu")(conv_22)
## deconv block 2 -----decoder(3 layers)
unpool_3 = MaxUnpooling2D(pool_size)([conv_22, mask_3])
concat_3 = Concatenate()(unpool_3, conv_7)

conv_23 = Convolution2D(64, (kernel, kernel), padding="same")(concat_3)
conv_23 = BatchNormalization()(conv_23)
conv_23 = Activation("relu")(conv_23)
conv_24 = Convolution2D(64, (kernel, kernel), padding="same")(conv_23)
conv_24 = BatchNormalization()(conv_24)
conv_24 = Activation("relu")(conv_24)
conv_25 = Convolution2D(32, (kernel, kernel), padding="same")(conv_24)
conv_25 = BatchNormalization()(conv_25)
conv_25 = Activation("relu")(conv_25)
## deconv block 3 -----decoder(3 layers)
unpool_4 = MaxUnpooling2D(pool_size)([conv_25, mask_2])
concat_4 = Concatenate()(unpool_4, conv_4)

conv_26 = Convolution2D(32, (kernel, kernel), padding="same")(concat_4)
conv_26 = BatchNormalization()(conv_26)
conv_26 = Activation("relu")(conv_26)
conv_27 = Convolution2D(16, (kernel, kernel), padding="same")(conv_26)
conv_27 = BatchNormalization()(conv_27)
conv_27 = Activation("relu")(conv_27)
## deconv block 4 -----decoder(2 layers)
unpool_5 = MaxUnpooling2D(pool_size)([conv_27, mask_1])
concat_5 = Concatenate()(unpool_5, conv_2)

conv_28 = Convolution2D(16, (kernel, kernel), padding="same")(concat_5)
conv_28 = BatchNormalization()(conv_28)
conv_28 = Activation("relu")(conv_28)

conv_29 = Convolution2D(n_labels, (1, 1), padding="same")(conv_28)
## deconv block 5 -----decoder(2 layers)
outputs = Activation(output_mode)(conv_29)
print("Build decoder done..")

segunet = Model(inputs=inputs, outputs=outputs, name="SegUNet")

```

Model after building:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
conv2d (Conv2D)	(None, 256, 256, 16)	448	['input_1[0][0]']
batch_normalization (BatchNormalization)	(None, 256, 256, 16)	64	['conv2d[0][0]']
activation (Activation)	(None, 256, 256, 16)	0	['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 256, 256, 16)	2320	['activation[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 256, 256, 16)	64	['conv2d_1[0][0]']
activation_1 (Activation)	(None, 256, 256, 16)	0	['batch_normalization_1[0][0]']
max_pooling_with_argmax2d (Max PoolingWithArgmax2D)	[(None, 128, 128, 1)], (None, 128, 128, 1)	0	['activation_1[0][0]']
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	['max_pooling_with_argmax2d[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 128, 128, 32)	128	['conv2d_2[0][0]']
activation_2 (Activation)	(None, 128, 128, 32)	0	['batch_normalization_2[0][0]']
conv2d_3 (Conv2D)	(None, 128, 128, 32)	9248	['activation_2[0][0]']

batch_normalization_3 (BatchNormalization)	(None, 128, 128, 32)	128	['conv2d_3[0][0]']
activation_3 (Activation)	(None, 128, 128, 32)	0	['batch_normalization_3[0][0]']
max_pooling_with_argmax2d_1 (MaxPoolingWithArgmax2D)	[(None, 64, 64, 32), (None, 64, 64, 32)]	0	['activation_3[0][0]']
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18496	['max_pooling_with_argmax2d_1[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 64, 64, 64)	256	['conv2d_4[0][0]']
activation_4 (Activation)	(None, 64, 64, 64)	0	['batch_normalization_4[0][0]']
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36928	['activation_4[0][0]']
batch_normalization_5 (BatchNormalization)	(None, 64, 64, 64)	256	['conv2d_5[0][0]']
activation_5 (Activation)	(None, 64, 64, 64)	0	['batch_normalization_5[0][0]']
conv2d_6 (Conv2D)	(None, 64, 64, 64)	36928	['activation_5[0][0]']
batch_normalization_6 (BatchNormalization)	(None, 64, 64, 64)	256	['conv2d_6[0][0]']
activation_6 (Activation)	(None, 64, 64, 64)	0	['batch_normalization_6[0][0]']
max_pooling_with_argmax2d_2 (MaxPoolingWithArgmax2D)	[(None, 32, 32, 64), (None, 32, 32, 64)]	0	['activation_6[0][0]']
conv2d_7 (Conv2D)	(None, 32, 32, 128)	73856	['max_pooling_with_argmax2d_2[0][0]']

batch_normalization_7 (BatchNormalization)	(None, 32, 32, 128)	512	['conv2d_7[0][0]']
activation_7 (Activation)	(None, 32, 32, 128)	0	['batch_normalization_7[0][0]']
conv2d_8 (Conv2D)	(None, 32, 32, 128)	147584	['activation_7[0][0]']
batch_normalization_8 (BatchNormalization)	(None, 32, 32, 128)	512	['conv2d_8[0][0]']
activation_8 (Activation)	(None, 32, 32, 128)	0	['batch_normalization_8[0][0]']
conv2d_9 (Conv2D)	(None, 32, 32, 128)	147584	['activation_8[0][0]']
batch_normalization_9 (BatchNormalization)	(None, 32, 32, 128)	512	['conv2d_9[0][0]']
activation_9 (Activation)	(None, 32, 32, 128)	0	['batch_normalization_9[0][0]']
max_pooling_with_argmax2d_3 (MaxPoolingWithArgmax2D)	[(None, 16, 16, 128), (None, 16, 16, 128)]	0	['activation_9[0][0]']
conv2d_10 (Conv2D)	(None, 16, 16, 128)	147584	['max_pooling_with_argmax2d_3[0][0]']
batch_normalization_10 (BatchNormalization)	(None, 16, 16, 128)	512	['conv2d_10[0][0]']
activation_10 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_10[0][0]']
conv2d_11 (Conv2D)	(None, 16, 16, 128)	147584	['activation_10[0][0]']

batch_normalization_11 (Batch Normalization)	(None, 16, 16, 128)	512	['conv2d_11[0][0]']
activation_11 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_11[0][0]']
conv2d_12 (Conv2D)	(None, 16, 16, 128)	147584	['activation_11[0][0]']
batch_normalization_12 (Batch Normalization)	(None, 16, 16, 128)	512	['conv2d_12[0][0]']
activation_12 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_12[0][0]']
max_pooling_with_argmax2d_4 (Max Pooling with Argmax2D)	[(None, 8, 8, 128), (None, 8, 8, 128)]	0	['activation_12[0][0]']
conv2d_13 (Conv2D)	(None, 8, 8, 128)	147584	['max_pooling_with_argmax2d_4[0][0]']
batch_normalization_13 (Batch Normalization)	(None, 8, 8, 128)	512	['conv2d_13[0][0]']
activation_13 (Activation)	(None, 8, 8, 128)	0	['batch_normalization_13[0][0]']
conv2d_14 (Conv2D)	(None, 8, 8, 128)	147584	['activation_13[0][0]']
batch_normalization_14 (Batch Normalization)	(None, 8, 8, 128)	512	['conv2d_14[0][0]']
activation_14 (Activation)	(None, 8, 8, 128)	0	['batch_normalization_14[0][0]']
conv2d_15 (Conv2D)	(None, 8, 8, 128)	147584	['activation_14[0][0]']
batch_normalization_15 (Batch Normalization)	(None, 8, 8, 128)	512	['conv2d_15[0][0]']
activation_15 (Activation)	(None, 8, 8, 128)	0	['batch_normalization_15[0][0]']
max_unpooling2d (Max Unpooling2D)	(None, 16, 16, 128)	0	['activation_15[0][0]', 'conv2d_15[0][0]']
concatenate (Concatenate)	(None, 16, 16, 256)	0	['max_unpooling2d[0][0]', 'activation_12[0][0]']
conv2d_16 (Conv2D)	(None, 16, 16, 128)	295040	['concatenate[0][0]']
batch_normalization_16 (Batch Normalization)	(None, 16, 16, 128)	512	['conv2d_16[0][0]']
activation_16 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_16[0][0]']
conv2d_17 (Conv2D)	(None, 16, 16, 128)	147584	['activation_16[0][0]']
batch_normalization_17 (Batch Normalization)	(None, 16, 16, 128)	512	['conv2d_17[0][0]']
activation_17 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_17[0][0]']
conv2d_18 (Conv2D)	(None, 16, 16, 128)	147584	['activation_17[0][0]']
batch_normalization_18 (Batch Normalization)	(None, 16, 16, 128)	512	['conv2d_18[0][0]']
activation_18 (Activation)	(None, 16, 16, 128)	0	['batch_normalization_18[0][0]']
max_unpooling2d_1 (Max Unpooling2D)	(None, 32, 32, 128)	0	['activation_18[0][0]', 'max_pooling_with_argmax2d_3[0][1]']
concatenate_1 (Concatenate)	(None, 32, 32, 256)	0	['max_unpooling2d_1[0][0]', 'activation_9[0][0]']
conv2d_19 (Conv2D)	(None, 32, 32, 128)	295040	['concatenate_1[0][0]']
batch_normalization_19 (Batch Normalization)	(None, 32, 32, 128)	512	['conv2d_19[0][0]']

activation_19 (Activation)	(None, 32, 32, 128)	0	['batch_normalization_19[0][0]']
conv2d_20 (Conv2D)	(None, 32, 32, 128)	147584	['activation_19[0][0]']
batch_normalization_20 (Batch Normalization)	(None, 32, 32, 128)	512	['conv2d_20[0][0]']
activation_20 (Activation)	(None, 32, 32, 128)	0	['batch_normalization_20[0][0]']
conv2d_21 (Conv2D)	(None, 32, 32, 64)	73792	['activation_20[0][0]']
batch_normalization_21 (Batch Normalization)	(None, 32, 32, 64)	256	['conv2d_21[0][0]']
activation_21 (Activation)	(None, 32, 32, 64)	0	['batch_normalization_21[0][0]']
max_unpooling2d_2 (MaxUnpooling2D)	(None, 64, 64, 64)	0	['activation_21[0][0]', 'max_pooling_with_argmax2d_2[0][1]']
concatenate_2 (Concatenate)	(None, 64, 64, 128)	0	['max_unpooling2d_2[0][0]', 'activation_6[0][0]']
conv2d_22 (Conv2D)	(None, 64, 64, 64)	73792	['concatenate_2[0][0]']
batch_normalization_22 (Batch Normalization)	(None, 64, 64, 64)	256	['conv2d_22[0][0]']
activation_22 (Activation)	(None, 64, 64, 64)	0	['batch_normalization_22[0][0]']
conv2d_23 (Conv2D)	(None, 64, 64, 64)	36928	['activation_22[0][0]']
batch_normalization_23 (Batch Normalization)	(None, 64, 64, 64)	256	['conv2d_23[0][0]']
activation_23 (Activation)	(None, 64, 64, 64)	0	['batch_normalization_23[0][0]']
conv2d_24 (Conv2D)	(None, 64, 64, 32)	18464	['activation_23[0][0]']

```

batch_normalization_24 (BatchN (None, 64, 64, 32) 128 ['conv2d_24[0][0]']
ormalization)

activation_24 (Activation) (None, 64, 64, 32) 0 ['batch_normalization_24[0][0]']

max_unpooling2d_3 (MaxUnpoolin (None, 128, 128, 32) 0 ['activation_24[0][0]',
g2D) ) ['max_pooling_with_argmax2d_1[0][
1]']

concatenate_3 (Concatenate) (None, 128, 128, 64) 0 ['max_unpooling2d_3[0][0]',
) ['activation_3[0][0]']

conv2d_25 (Conv2D) (None, 128, 128, 32) 18464 ['concatenate_3[0][0]']
)

batch_normalization_25 (BatchN (None, 128, 128, 32) 128 ['conv2d_25[0][0]']
ormalization)

activation_25 (Activation) (None, 128, 128, 32) 0 ['batch_normalization_25[0][0]']
)

conv2d_26 (Conv2D) (None, 128, 128, 16) 4624 ['activation_25[0][0]']
)

batch_normalization_26 (BatchN (None, 128, 128, 16) 64 ['conv2d_26[0][0]']
ormalization)

activation_26 (Activation) (None, 128, 128, 16) 0 ['batch_normalization_26[0][0]']
)

max_unpooling2d_4 (MaxUnpoolin (None, 256, 256, 16) 0 ['activation_26[0][0]',
g2D) ) ['max_pooling_with_argmax2d[0][1]
']

concatenate_4 (Concatenate) (None, 256, 256, 32) 0 ['max_unpooling2d_4[0][0]',
) ['activation_1[0][0]']

conv2d_27 (Conv2D) (None, 256, 256, 16) 4624 ['concatenate_4[0][0]']
)

batch_normalization_27 (BatchN (None, 256, 256, 16) 64 ['conv2d_27[0][0]']
ormalization)

activation_27 (Activation) (None, 256, 256, 16) 0 ['batch_normalization_27[0][0]']
)

conv2d_28 (Conv2D) (None, 256, 256, 1) 17 ['activation_27[0][0]']
)

activation_28 (Activation) (None, 256, 256, 1) 0 ['conv2d_28[0][0]']
)

=====
Total params: 2,636,545
Trainable params: 2,631,809
Non-trainable params: 4,736
=====

```

Model training and validation:

Epoch 1/40

138/137 [=====] - ETA: -3s - loss: 0.2362 - accuracy: 0.9576 - mean_io_u: 0.4865

Epoch 1: val_loss improved from inf to 0.29547, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 661s 5s/step - loss: 0.2362 - accuracy: 0.9576 - mean_io_u: 0.4865 - val_loss: 0.2955 - val_accuracy: 0.9460 - val_mean_io_u: 0.4850 - lr: 0.0010

Epoch 2/40

138/137 [=====] - ETA: 0s - loss: 0.1060 - accuracy: 0.9801 - mean_io_u: 0.4865

Epoch 2: val_loss improved from 0.29547 to 0.11021, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 376ms/step - loss: 0.1060 - accuracy: 0.9801 - mean_io_u: 0.4865 - val_loss: 0.1102 - val_accuracy: 0.9729 - val_mean_io_u: 0.4849 - lr: 0.0010

Epoch 3/40

138/137 [=====] - ETA: 0s - loss: 0.0773 - accuracy: 0.9802 - mean_io_u: 0.4865

Epoch 3: val_loss improved from 0.11021 to 0.09330, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 51s 374ms/step - loss: 0.0773 - accuracy: 0.9802 - mean_io_u: 0.4865 - val_loss: 0.0933 - val_accuracy: 0.9808 - val_mean_io_u: 0.4861 - lr: 0.0010

Epoch 4/40

138/137 [=====] - ETA: 0s - loss: 0.0599 - accuracy: 0.9823 - mean_io_u: 0.4865

Epoch 4: val_loss improved from 0.09330 to 0.07161, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 51s 375ms/step - loss: 0.0599 - accuracy: 0.9823 - mean_io_u: 0.4865 - val_loss: 0.0716 - val_accuracy: 0.9839 - val_mean_io_u: 0.4847 - lr: 0.0010

Epoch 5/40

138/137 [=====] - ETA: 0s - loss: 0.0502 - accuracy: 0.9835 - mean_io_u: 0.4865

Epoch 5: val_loss improved from 0.07161 to 0.07000, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 376ms/step - loss: 0.0502 - accuracy: 0.9835 - mean_io_u: 0.4865 - val_loss: 0.0700 - val_accuracy: 0.9748 - val_mean_io_u: 0.4848 - lr: 0.0010

Epoch 6/40

138/137 [=====] - ETA: 0s - loss: 0.0455 - accuracy: 0.9839 - mean_io_u: 0.4865

Epoch 6: val_loss improved from 0.07000 to 0.04653, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 53s 389ms/step - loss: 0.0455 - accuracy: 0.9839 - mean_io_u: 0.4865 - val_loss: 0.0465 - val_accuracy: 0.9865 - val_mean_io_u: 0.4868 - lr: 0.0010

Epoch 7/40

138/137 [=====] - ETA: 0s - loss: 0.0416 - accuracy: 0.9845 - mean_io_u: 0.4865

Epoch 7: val_loss improved from 0.04653 to 0.04195, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 379ms/step - loss: 0.0416 - accuracy: 0.9845 - mean_io_u: 0.4865 - val_loss: 0.0419 - val_accuracy: 0.9859 - val_mean_io_u: 0.4844 - lr: 0.0010

Epoch 8/40

138/137 [=====] - ETA: 0s - loss: 0.0378 - accuracy: 0.9854 - mean_io_u: 0.4865

Epoch 8: val_loss improved from 0.04195 to 0.04052, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 376ms/step - loss: 0.0378 - accuracy: 0.9854 - mean_io_u: 0.4865 - val_loss: 0.0405 - val_accuracy: 0.9853 - val_mean_io_u: 0.4854 - lr: 0.0010

Epoch 9/40

138/137 [=====] - ETA: 0s - loss: 0.0360 - accuracy: 0.9856 - mean_io_u: 0.4865

Epoch 9: val_loss improved from 0.04052 to 0.03799, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 53s 389ms/step - loss: 0.0360 - accuracy: 0.9856 - mean_io_u: 0.4865 - val_loss: 0.0380 - val_accuracy: 0.9858 - val_mean_io_u: 0.4855 - lr: 0.0010

Epoch 10/40

138/137 [=====] - ETA: 0s - loss: 0.0317 - accuracy: 0.9866 - mean_io_u: 0.4865

Epoch 10: val_loss did not improve from 0.03799

137/137 [=====] - 51s 373ms/step - loss: 0.0317 - accuracy: 0.9866 - mean_io_u: 0.4865 - val_loss: 0.0415 - val_accuracy: 0.9862 - val_mean_io_u: 0.4859 - lr: 0.0010

Epoch 11/40

138/137 [=====] - ETA: 0s - loss: 0.0310 - accuracy: 0.9867 - mean_io_u: 0.4865

Epoch 11: val_loss did not improve from 0.03799

137/137 [=====] - 53s 386ms/step - loss: 0.0310 - accuracy: 0.9867 - mean_io_u: 0.4865 - val_loss: 0.0403 - val_accuracy: 0.9848 - val_mean_io_u: 0.4852 - lr: 0.0010

Epoch 12/40

138/137 [=====] - ETA: 0s - loss: 0.0274 - accuracy: 0.9877 - mean_io_u: 0.4865

Epoch 12: val_loss improved from 0.03799 to 0.02815, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 376ms/step - loss: 0.0274 - accuracy: 0.9877 - mean_io_u: 0.4865 - val_loss: 0.0282 - val_accuracy: 0.9902 - val_mean_io_u: 0.4847 - lr: 0.0010

Epoch 13/40

138/137 [=====] - ETA: 0s - loss: 0.0268 - accuracy: 0.9880 - mean_io_u: 0.4865

Epoch 13: val_loss did not improve from 0.02815

137/137 [=====] - 51s 371ms/step - loss: 0.0268 - accuracy: 0.9880 - mean_io_u: 0.4865 - val_loss: 0.0448 - val_accuracy: 0.9859 - val_mean_io_u: 0.4843 - lr: 0.0010

Epoch 14/40

138/137 [=====] - ETA: 0s - loss: 0.0239 - accuracy: 0.9891 - mean_io_u: 0.4865

Epoch 14: val_loss did not improve from 0.02815

137/137 [=====] - 53s 385ms/step - loss: 0.0239 - accuracy: 0.9891 - mean_io_u: 0.4865 - val_loss: 0.0305 - val_accuracy: 0.9899 - val_mean_io_u: 0.4859 - lr: 0.0010

Epoch 15/40

138/137 [=====] - ETA: 0s - loss: 0.0250 - accuracy: 0.9886 - mean_io_u: 0.4865

Epoch 15: val_loss improved from 0.02815 to 0.02481, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 376ms/step - loss: 0.0250 - accuracy: 0.9886 - mean_io_u: 0.4865 - val_loss: 0.0248 - val_accuracy: 0.9913 - val_mean_io_u: 0.4855 - lr: 0.0010

Epoch 16/40

138/137 [=====] - ETA: 0s - loss: 0.0221 - accuracy: 0.9895 - mean_io_u: 0.4865

Epoch 16: val_loss did not improve from 0.02481

137/137 [=====] - 53s 389ms/step - loss: 0.0221 - accuracy: 0.9895 - mean_io_u: 0.4865 - val_loss: 0.0252 - val_accuracy: 0.9912 - val_mean_io_u: 0.4853 - lr: 0.0010

Epoch 17/40

138/137 [=====] - ETA: 0s - loss: 0.0235 - accuracy: 0.9894 - mean_io_u: 0.4865

Epoch 17: val_loss did not improve from 0.02481

137/137 [=====] - 53s 386ms/step - loss: 0.0235 - accuracy: 0.9894 - mean_io_u: 0.4865 - val_loss: 0.0251 - val_accuracy: 0.9914 - val_mean_io_u: 0.4849 - lr: 0.0010

Epoch 18/40

138/137 [=====] - ETA: 0s - loss: 0.0224 - accuracy: 0.9897 - mean_io_u: 0.4865

Epoch 18: val_loss improved from 0.02481 to 0.02368, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 54s 391ms/step - loss: 0.0224 - accuracy: 0.9897 - mean_io_u: 0.4865 - val_loss: 0.0237 - val_accuracy: 0.9914 - val_mean_io_u: 0.4854 - lr: 0.0010

Epoch 19/40

138/137 [=====] - ETA: 0s - loss: 0.0211 - accuracy: 0.9900 - mean_io_u: 0.4865

Epoch 19: val_loss did not improve from 0.02368

137/137 [=====] - 52s 380ms/step - loss: 0.0211 - accuracy: 0.9900 - mean_io_u: 0.4865 - val_loss: 0.0246 - val_accuracy: 0.9914 - val_mean_io_u: 0.4852 - lr: 0.0010

Epoch 20/40

138/137 [=====] - ETA: 0s - loss: 0.0221 - accuracy: 0.9895 - mean_io_u: 0.4865

Epoch 20: val_loss improved from 0.02368 to 0.02139, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 380ms/step - loss: 0.0221 - accuracy: 0.9895 - mean_io_u: 0.4865 - val_loss: 0.0214 - val_accuracy: 0.9925 - val_mean_io_u: 0.4850 - lr: 0.0010

Epoch 21/40

138/137 [=====] - ETA: 0s - loss: 0.0208 - accuracy: 0.9901 - mean_io_u: 0.4865

Epoch 21: val_loss did not improve from 0.02139

137/137 [=====] - 53s 386ms/step - loss: 0.0208 - accuracy: 0.9901 - mean_io_u: 0.4865 - val_loss: 0.0225 - val_accuracy: 0.9916 - val_mean_io_u: 0.4851 - lr: 0.0010

Epoch 22/40

138/137 [=====] - ETA: 0s - loss: 0.0205 - accuracy: 0.9900 - mean_io_u: 0.4865

Epoch 22: val_loss did not improve from 0.02139

137/137 [=====] - 54s 392ms/step - loss: 0.0205 - accuracy: 0.9900 - mean_io_u: 0.4865 - val_loss: 0.0223 - val_accuracy: 0.9926 - val_mean_io_u: 0.4858 - lr: 0.0010

Epoch 23/40

138/137 [=====] - ETA: 0s - loss: 0.0196 - accuracy: 0.9906 - mean_io_u: 0.4865

Epoch 23: val_loss improved from 0.02139 to 0.01911, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 379ms/step - loss: 0.0196 - accuracy: 0.9906 - mean_io_u: 0.4865 - val_loss: 0.0191 - val_accuracy: 0.9931 - val_mean_io_u: 0.4848 - lr: 0.0010

Epoch 24/40

138/137 [=====] - ETA: 0s - loss: 0.0187 - accuracy: 0.9907 - mean_io_u: 0.4865

Epoch 24: val_loss improved from 0.01911 to 0.01909, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 375ms/step - loss: 0.0187 - accuracy: 0.9907 - mean_io_u: 0.4865 - val_loss: 0.0191 - val_accuracy: 0.9928 - val_mean_io_u: 0.4850 - lr: 0.0010

Epoch 25/40

138/137 [=====] - ETA: 0s - loss: 0.0183 - accuracy: 0.9908 - mean_io_u: 0.4865

Epoch 25: val_loss did not improve from 0.01909

137/137 [=====] - 53s 386ms/step - loss: 0.0183 - accuracy: 0.9908 - mean_io_u: 0.4865 - val_loss: 0.0206 - val_accuracy: 0.9929 - val_mean_io_u: 0.4848 - lr: 0.0010

Epoch 26/40

138/137 [=====] - ETA: 0s - loss: 0.0177 - accuracy: 0.9910 - mean_io_u: 0.4865

Epoch 26: val_loss did not improve from 0.01909

137/137 [=====] - 51s 371ms/step - loss: 0.0177 - accuracy: 0.9910 - mean_io_u: 0.4865 - val_loss: 0.0201 - val_accuracy: 0.9925 - val_mean_io_u: 0.4870 - lr: 0.0010

Epoch 27/40

138/137 [=====] - ETA: 0s - loss: 0.0177 - accuracy: 0.9910 - mean_io_u: 0.4865

Epoch 27: val_loss did not improve from 0.01909

137/137 [=====] - 53s 389ms/step - loss: 0.0177 - accuracy: 0.9910 - mean_io_u: 0.4865 - val_loss: 0.0341 - val_accuracy: 0.9898 - val_mean_io_u: 0.4832 - lr: 0.0010

Epoch 28/40

138/137 [=====] - ETA: 0s - loss: 0.0181 - accuracy: 0.9911 - mean_io_u: 0.4865

Epoch 28: val_loss improved from 0.01909 to 0.01883, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 54s 397ms/step - loss: 0.0181 - accuracy: 0.9911 - mean_io_u: 0.4865 - val_loss: 0.0188 - val_accuracy: 0.9930 - val_mean_io_u: 0.4858 - lr: 0.0010

Epoch 29/40

138/137 [=====] - ETA: 0s - loss: 0.0169 - accuracy: 0.9913 - mean_io_u: 0.4865

Epoch 29: val_loss improved from 0.01883 to 0.01642, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 53s 384ms/step - loss: 0.0169 - accuracy: 0.9913 - mean_io_u: 0.4865 - val_loss: 0.0164 - val_accuracy: 0.9939 - val_mean_io_u: 0.4868 - lr: 0.0010

Epoch 30/40

138/137 [=====] - ETA: 0s - loss: 0.0161 - accuracy: 0.9916 - mean_io_u: 0.4865

Epoch 30: val_loss did not improve from 0.01642

137/137 [=====] - 52s 379ms/step - loss: 0.0161 - accuracy: 0.9916 - mean_io_u: 0.4865 - val_loss: 0.0174 - val_accuracy: 0.9938 - val_mean_io_u: 0.4844 - lr: 0.0010

Epoch 31/40

138/137 [=====] - ETA: 0s - loss: 0.0163 - accuracy: 0.9916 - mean_io_u: 0.4865

Epoch 31: val_loss did not improve from 0.01642

137/137 [=====] - 52s 377ms/step - loss: 0.0163 - accuracy: 0.9916 - mean_io_u: 0.4865 - val_loss: 0.0169 - val_accuracy: 0.9938 - val_mean_io_u: 0.4841 - lr: 0.0010

Epoch 32/40

138/137 [=====] - ETA: 0s - loss: 0.0168 - accuracy: 0.9913 - mean_io_u: 0.4865

Epoch 32: val_loss did not improve from 0.01642

137/137 [=====] - 54s 390ms/step - loss: 0.0168 - accuracy: 0.9913 - mean_io_u: 0.4865 - val_loss: 0.0173 - val_accuracy: 0.9939 - val_mean_io_u: 0.4865 - lr: 0.0010

Epoch 33/40

138/137 [=====] - ETA: 0s - loss: 0.0158 - accuracy: 0.9916 - mean_io_u: 0.4865

Epoch 33: val_loss did not improve from 0.01642

137/137 [=====] - 54s 392ms/step - loss: 0.0158 - accuracy: 0.9916 - mean_io_u: 0.4865 - val_loss: 0.0208 - val_accuracy: 0.9922 - val_mean_io_u: 0.4850 - lr: 0.0010

Epoch 34/40

138/137 [=====] - ETA: 0s - loss: 0.0152 - accuracy: 0.9918 - mean_io_u: 0.4865

Epoch 34: ReduceLROnPlateau reducing learning rate to 0.000100000000474974513.

Epoch 34: val_loss did not improve from 0.01642

137/137 [=====] - 54s 391ms/step - loss: 0.0152 - accuracy: 0.9918 - mean_io_u: 0.4865 - val_loss: 0.0218 - val_accuracy: 0.9924 - val_mean_io_u: 0.4851 - lr: 0.0010

Epoch 35/40

138/137 [=====] - ETA: 0s - loss: 0.0162 - accuracy: 0.9918 - mean_io_u: 0.4865

Epoch 35: val_loss improved from 0.01642 to 0.01537, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 53s 383ms/step - loss: 0.0162 - accuracy: 0.9918 - mean_io_u: 0.4865 - val_loss: 0.0154 - val_accuracy: 0.9943 - val_mean_io_u: 0.4850 - lr: 1.0000e-04

Epoch 36/40

138/137 [=====] - ETA: 0s - loss: 0.0129 - accuracy: 0.9925 - mean_io_u: 0.4865

Epoch 36: val_loss did not improve from 0.01537

137/137 [=====] - 54s 390ms/step - loss: 0.0129 - accuracy: 0.9925 - mean_io_u: 0.4865 - val_loss: 0.0154 - val_accuracy: 0.9942 - val_mean_io_u: 0.4851 - lr: 1.0000e-04

Epoch 37/40

138/137 [=====] - ETA: 0s - loss: 0.0129 - accuracy: 0.9926 - mean_io_u: 0.4865

Epoch 37: val_loss improved from 0.01537 to 0.01517, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 380ms/step - loss: 0.0129 - accuracy: 0.9926 - mean_io_u: 0.4865 - val_loss: 0.0152 - val_accuracy: 0.9943 - val_mean_io_u: 0.4852 - lr: 1.0000e-04

Epoch 38/40

138/137 [=====] - ETA: 0s - loss: 0.0131 - accuracy: 0.9925 - mean_io_u: 0.4865

Epoch 38: val_loss did not improve from 0.01517

137/137 [=====] - 51s 375ms/step - loss: 0.0131 - accuracy: 0.9925 - mean_io_u: 0.4865 - val_loss: 0.0158 - val_accuracy: 0.9941 - val_mean_io_u: 0.4849 - lr: 1.0000e-04

Epoch 39/40

138/137 [=====] - ETA: 0s - loss: 0.0131 - accuracy: 0.9926 - mean_io_u: 0.4865

Epoch 39: val_loss improved from 0.01517 to 0.01462, saving model to /content/drive/MyDrive/FYP2_modified_SegUnet.h5

137/137 [=====] - 52s 381ms/step - loss: 0.0131 - accuracy: 0.9926 - mean_io_u: 0.4865 - val_loss: 0.0146 - val_accuracy: 0.9945 - val_mean_io_u: 0.4851 - lr: 1.0000e-04

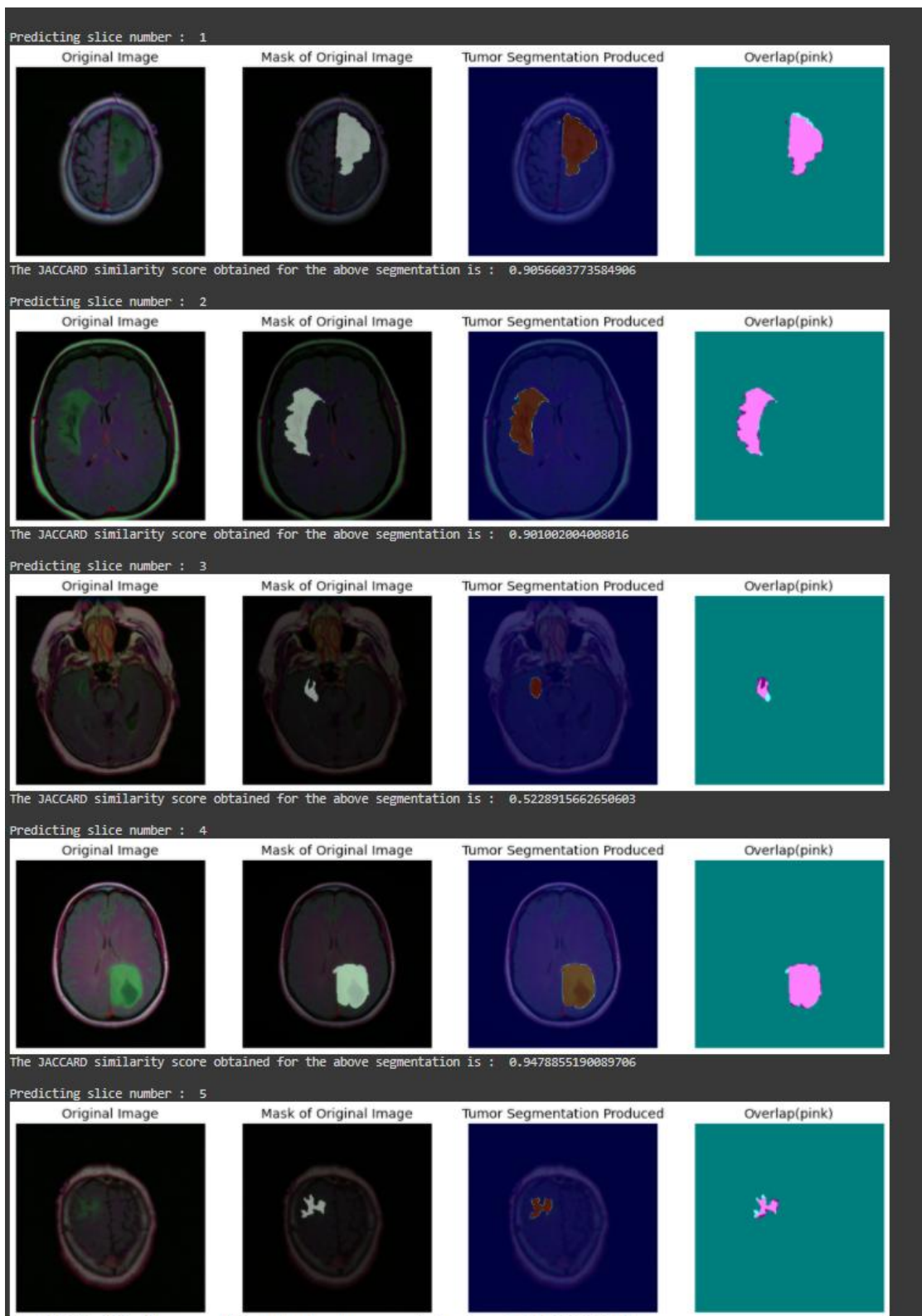
Epoch 40/40

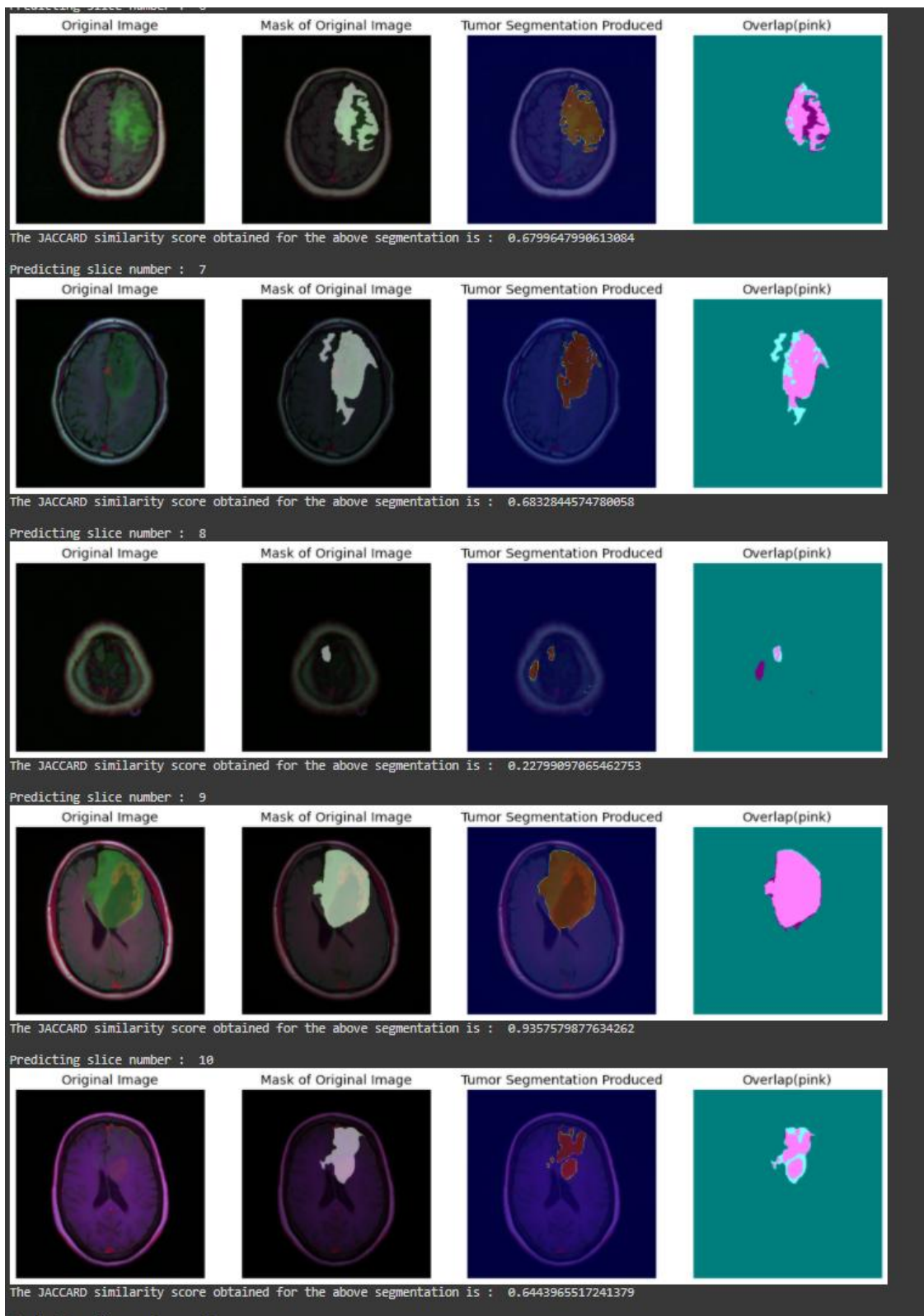
138/137 [=====] - ETA: 0s - loss: 0.0132 - accuracy: 0.9925 - mean_io_u: 0.4865

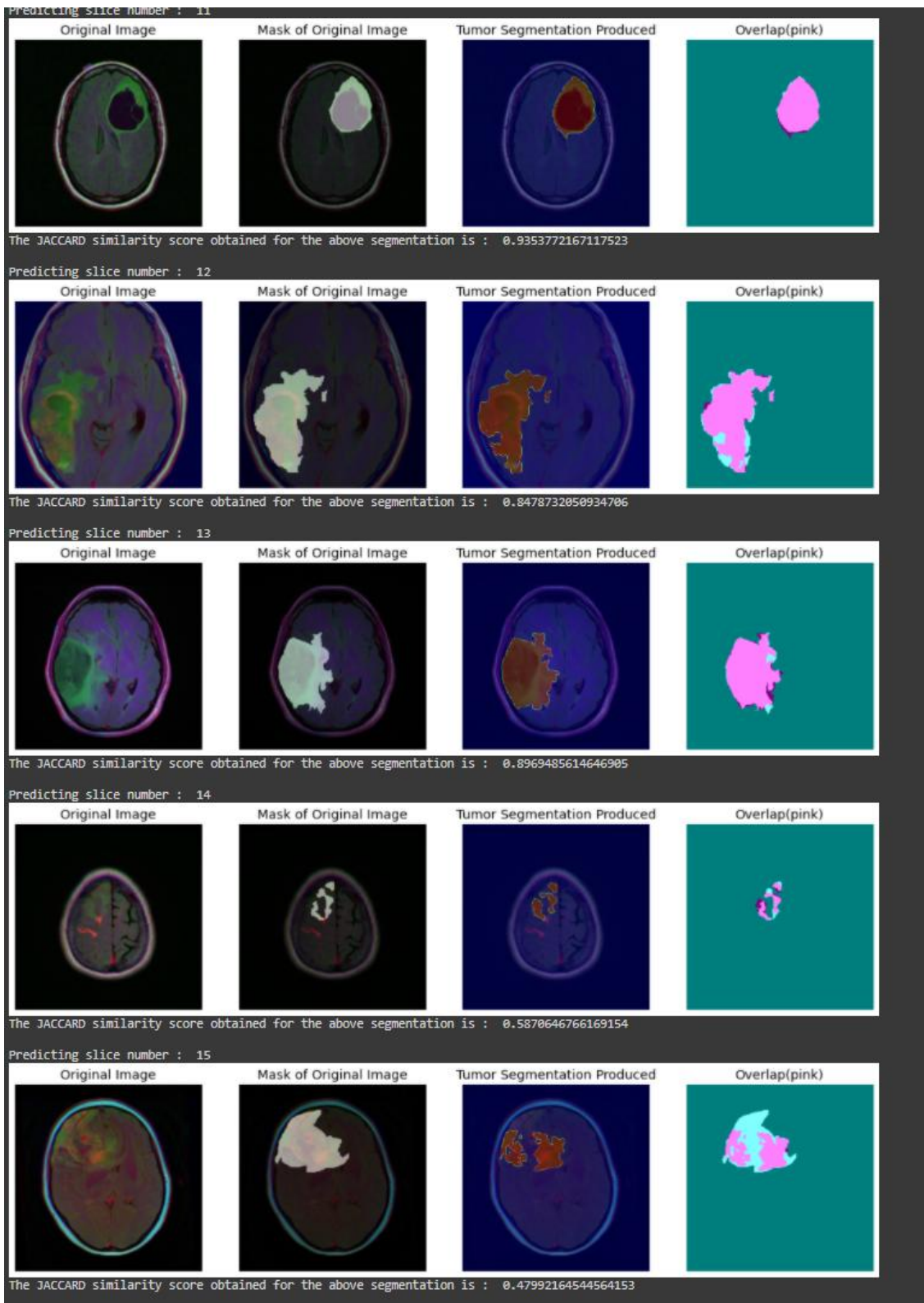
Epoch 40: val_loss did not improve from 0.01462

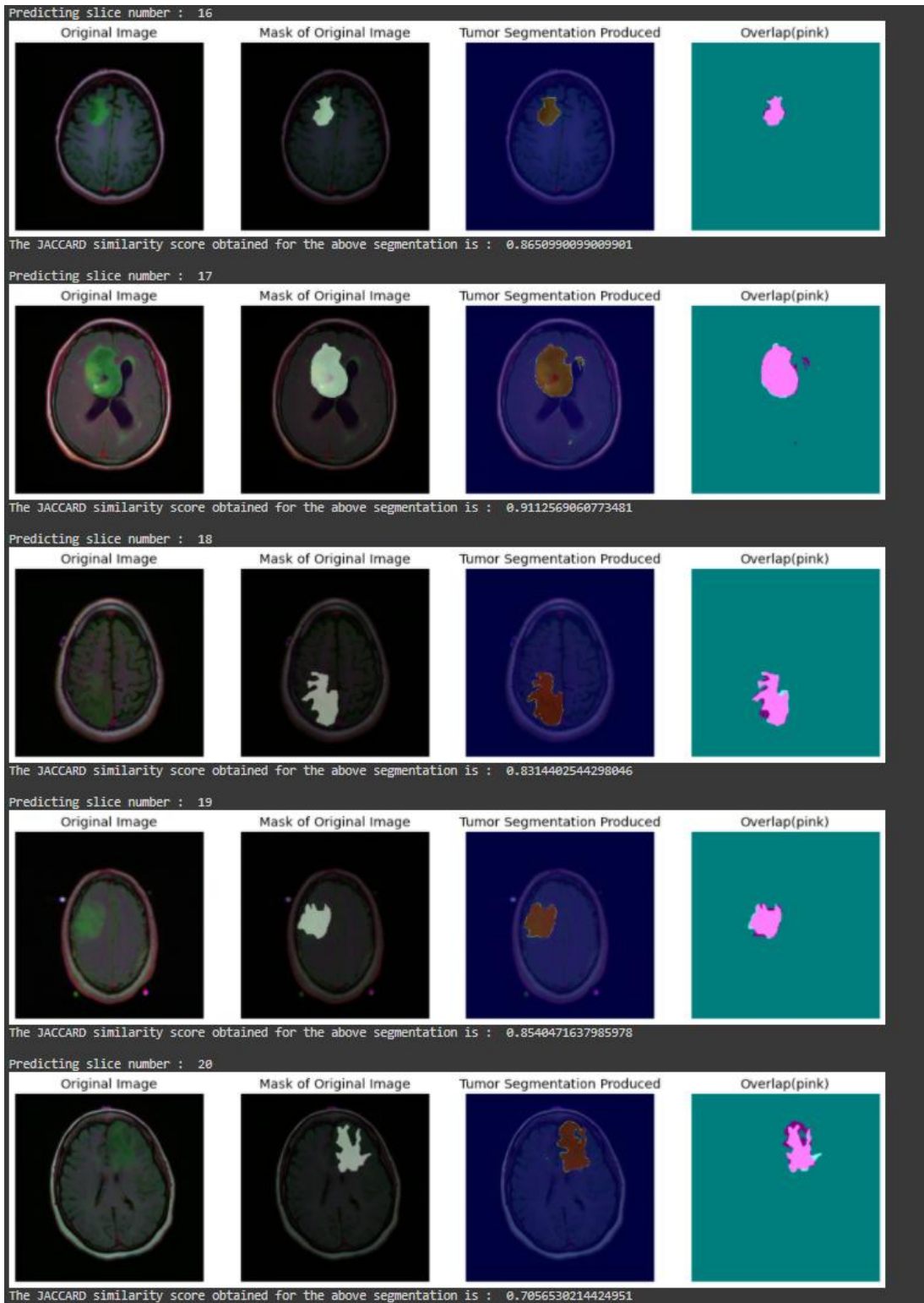
137/137 [=====] - 51s 375ms/step - loss: 0.0132 - accuracy: 0.9925 - mean_io_u: 0.4865 - val_loss: 0.0156 - val_accuracy: 0.9942 - val_mean_io_u: 0.4850 - lr: 1.0000e-04

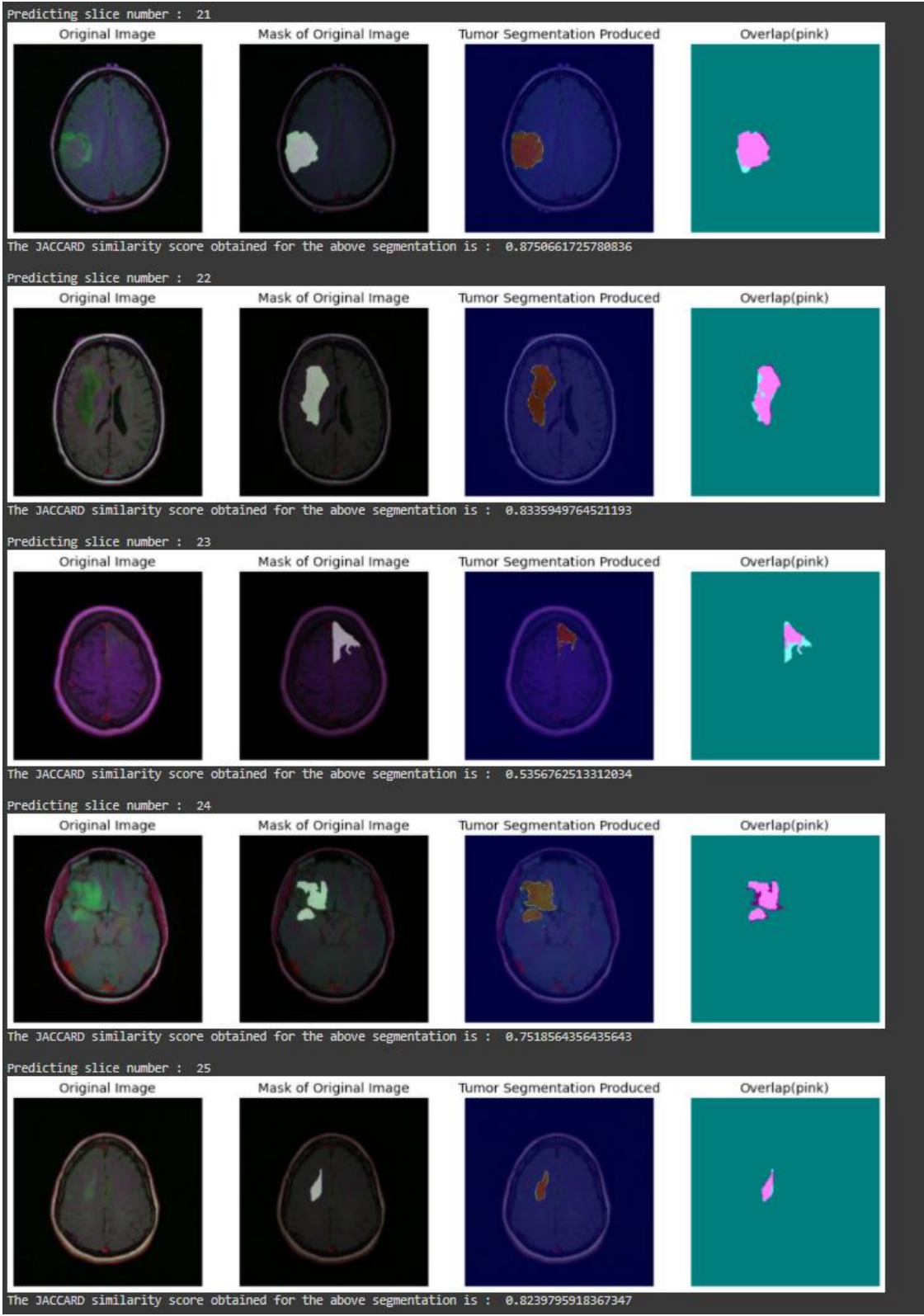
APPENDIX C1: Proposed model testing result on dataset[5]

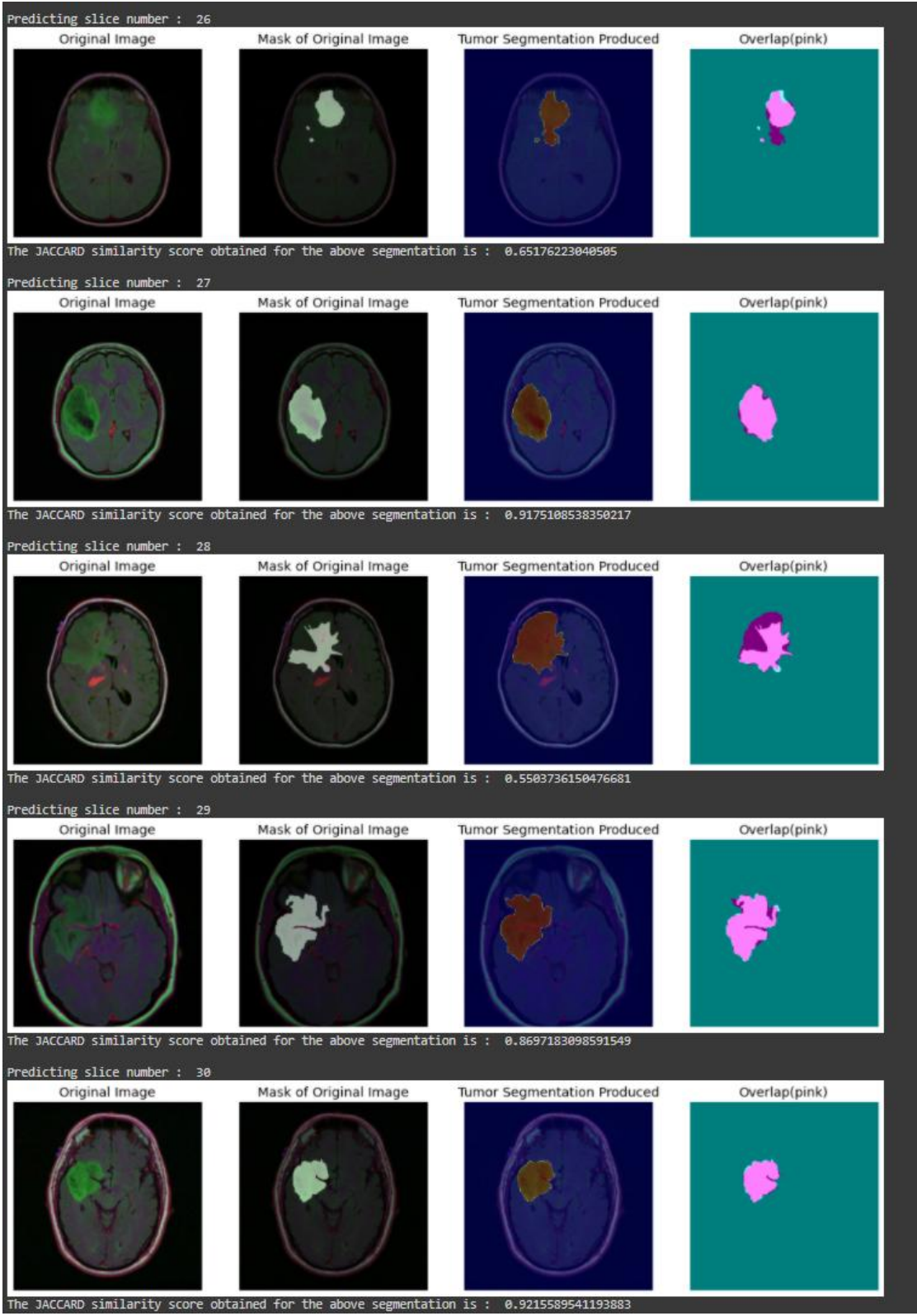


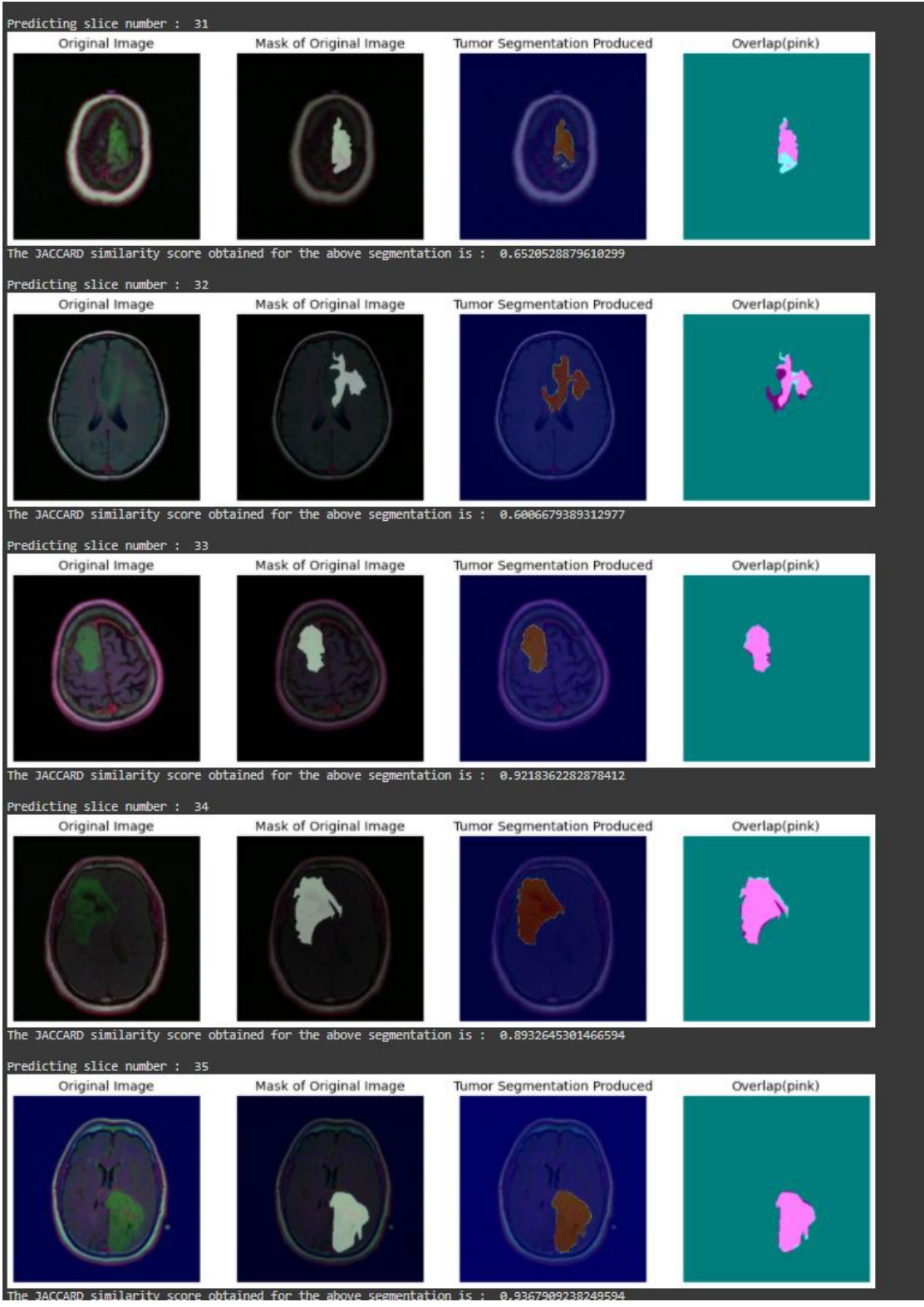


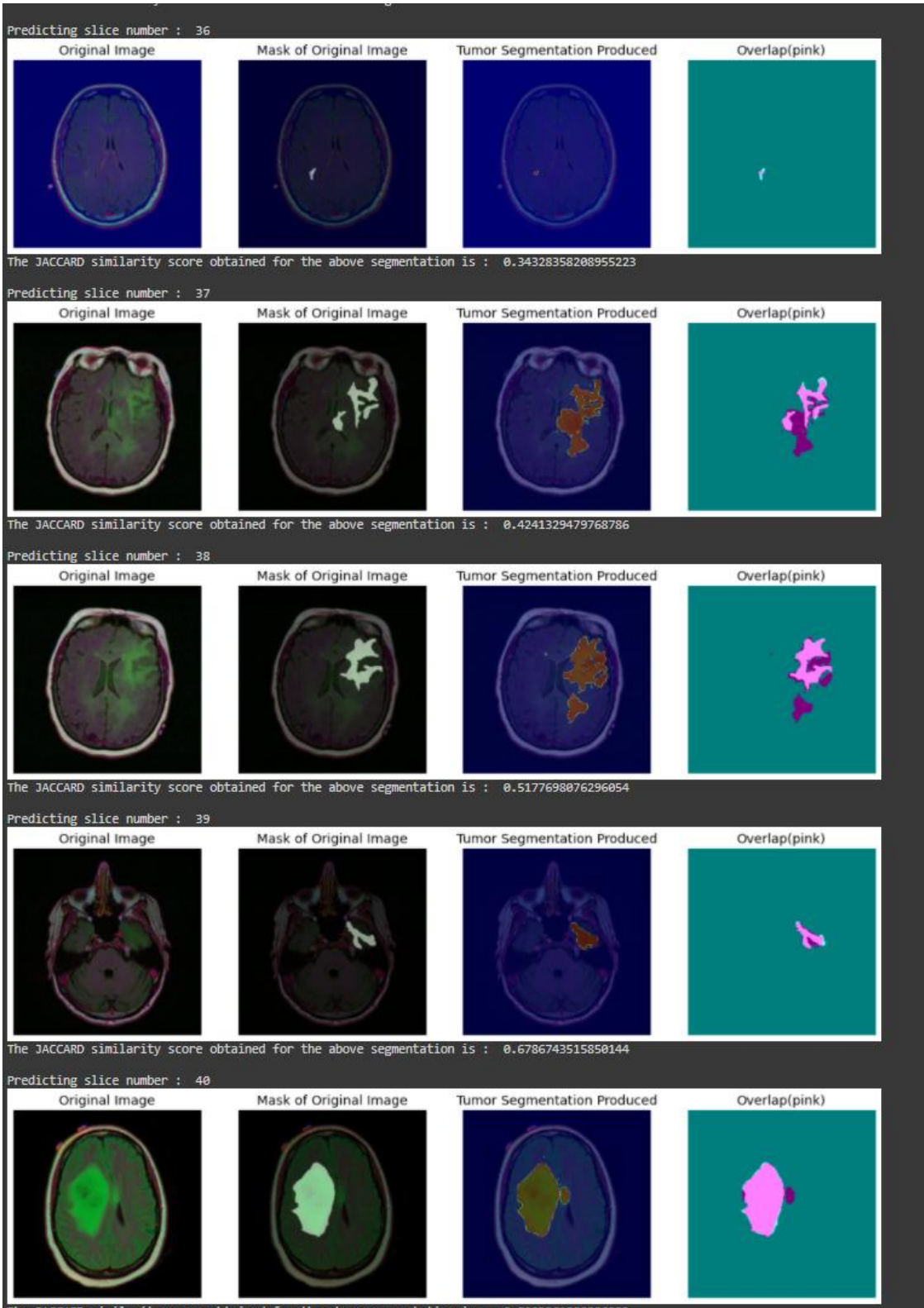


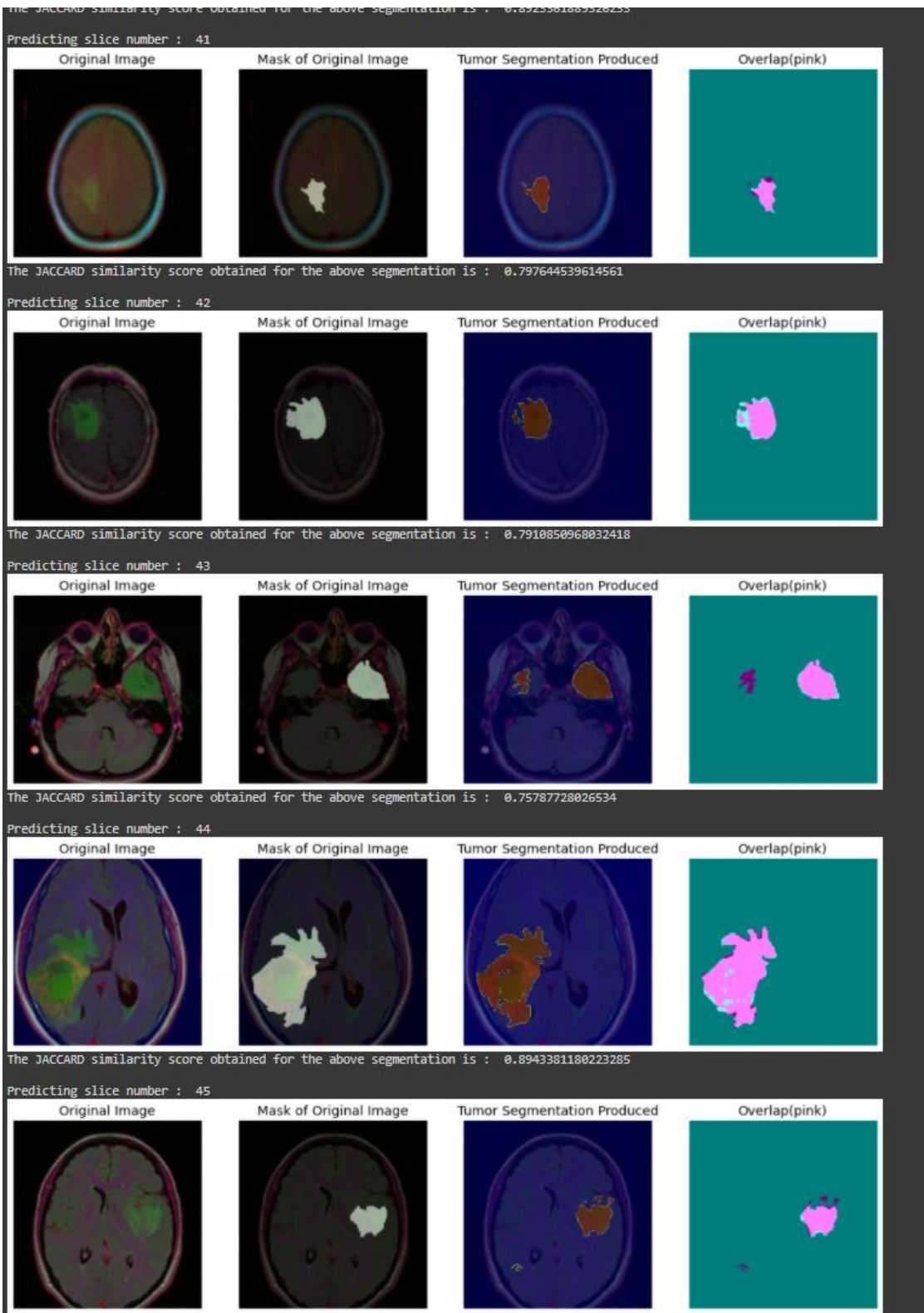


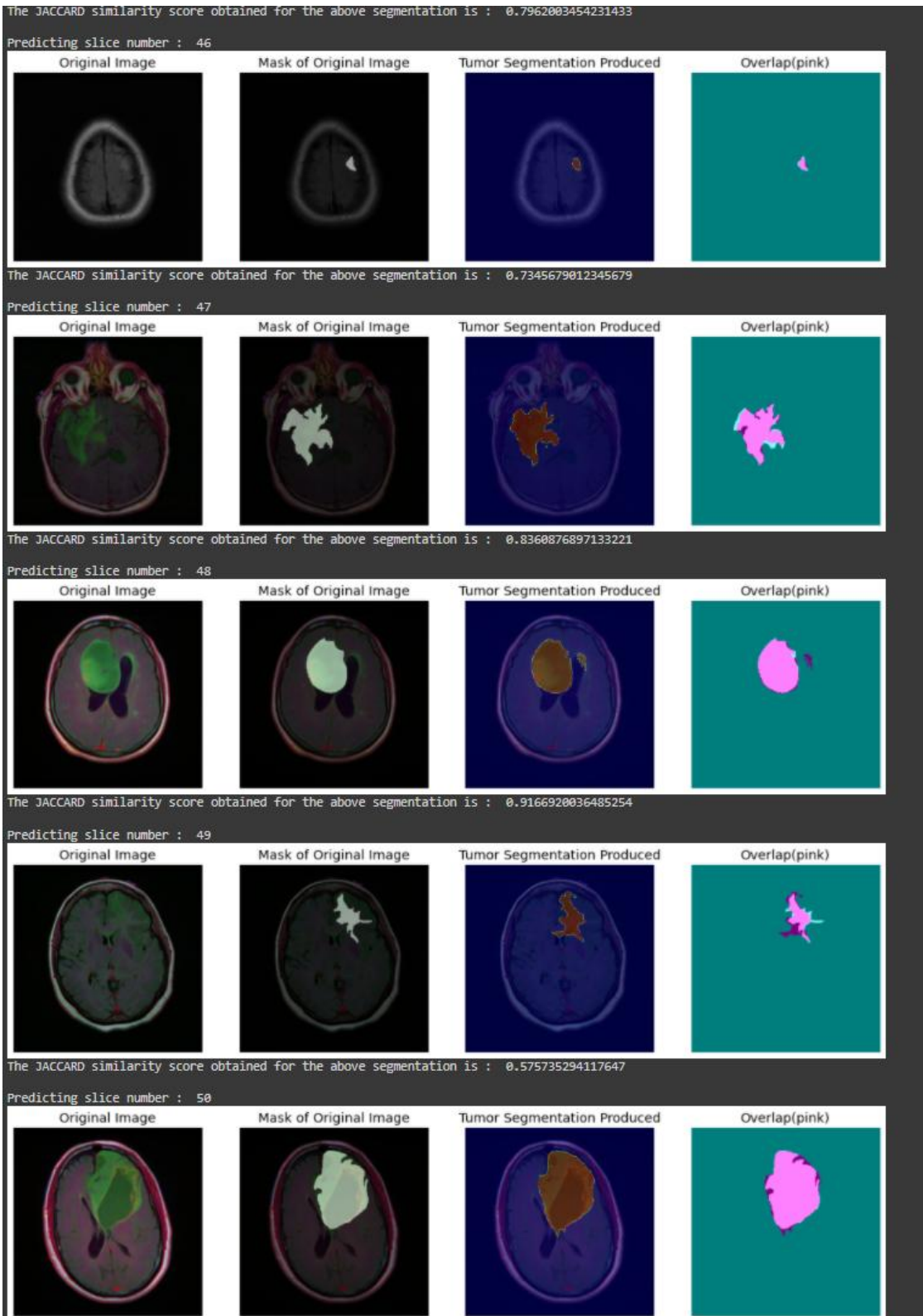


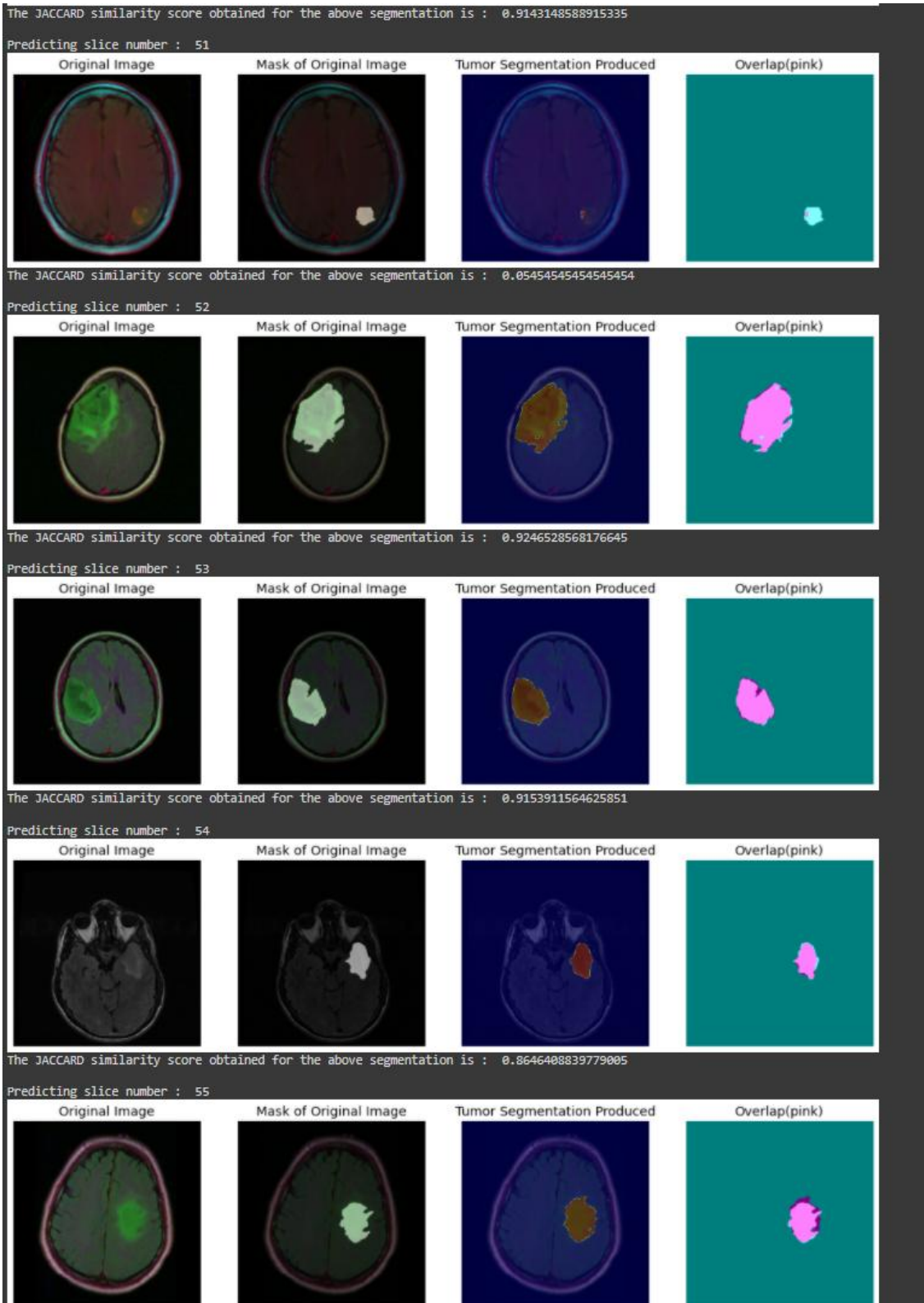


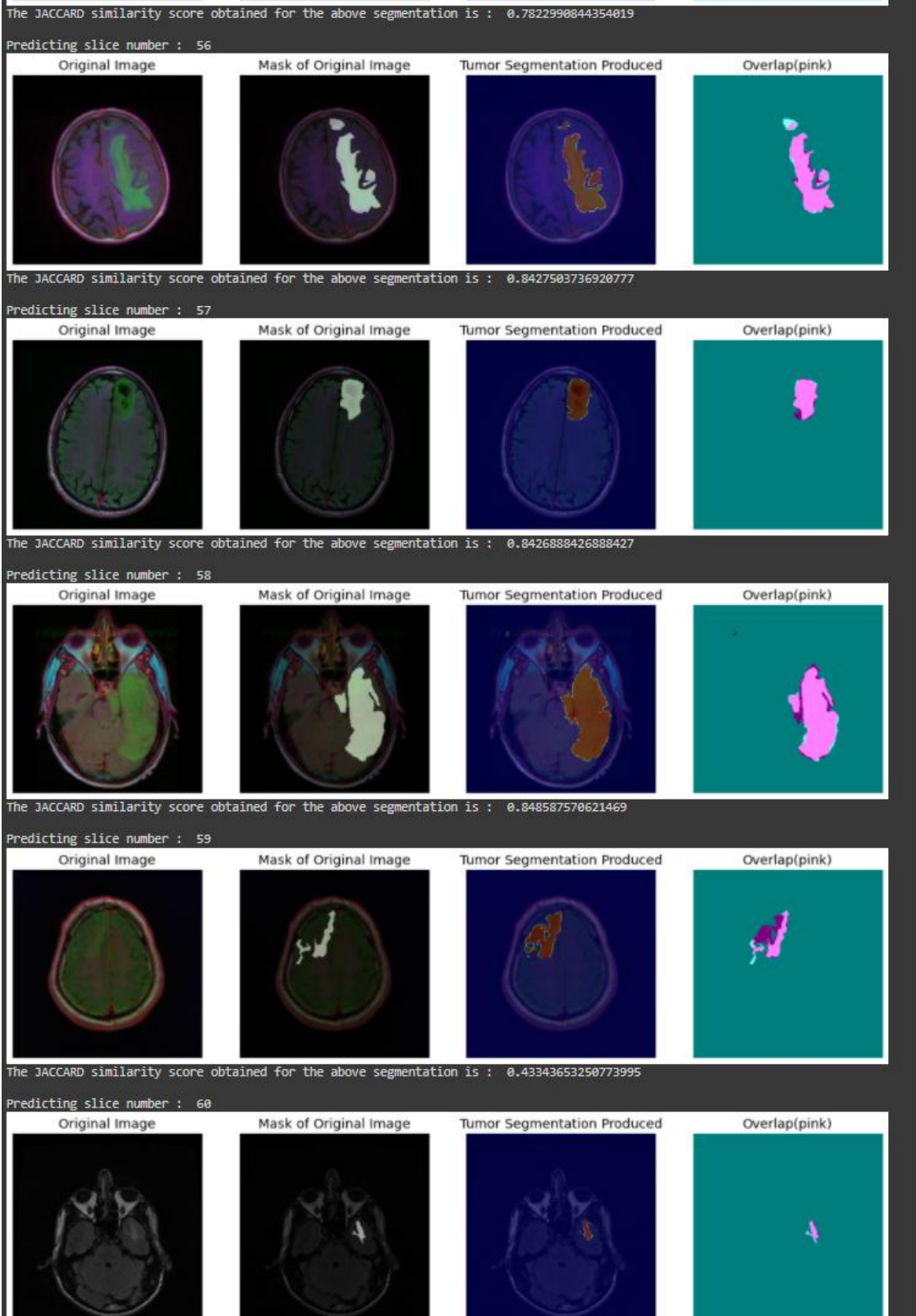


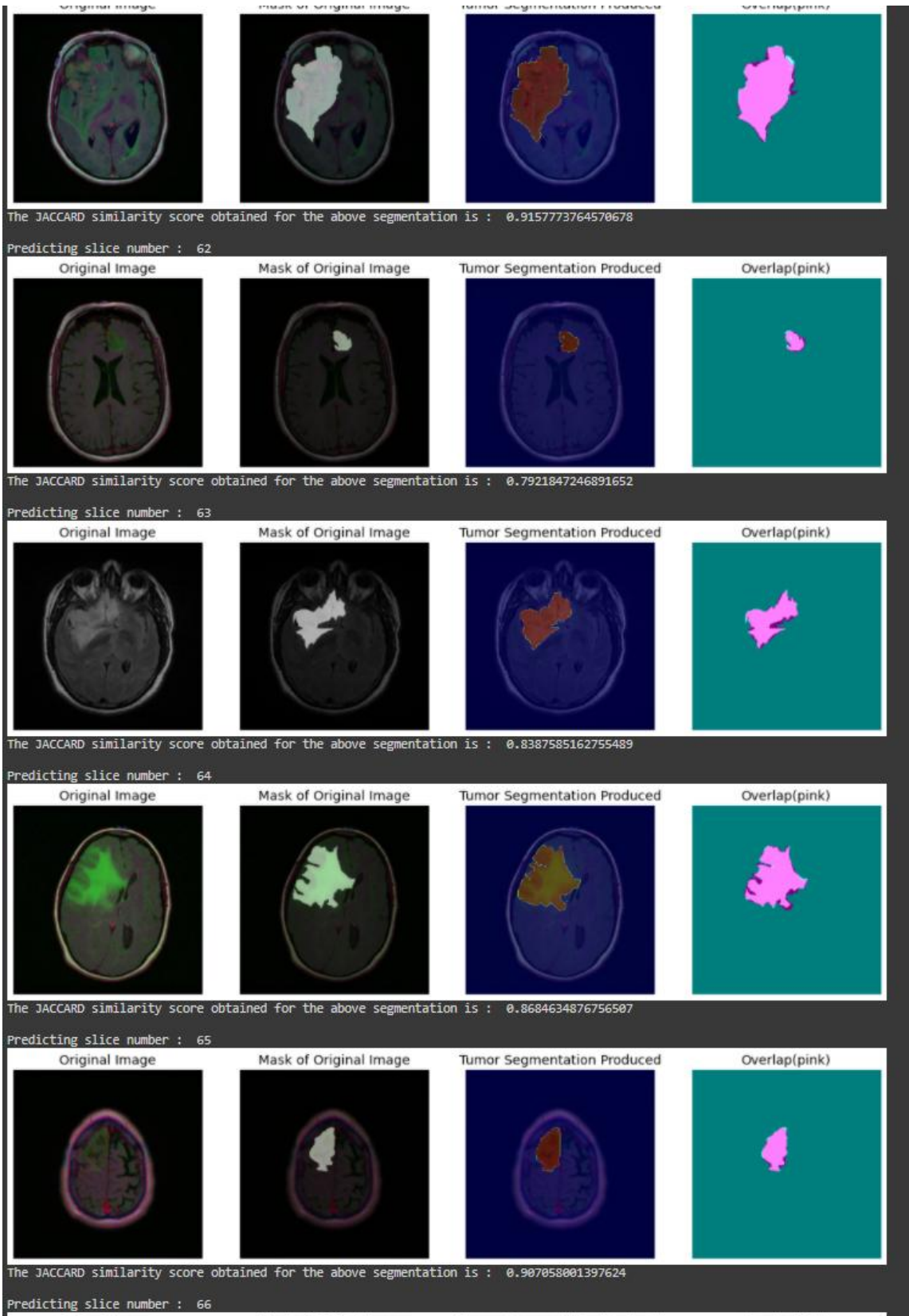


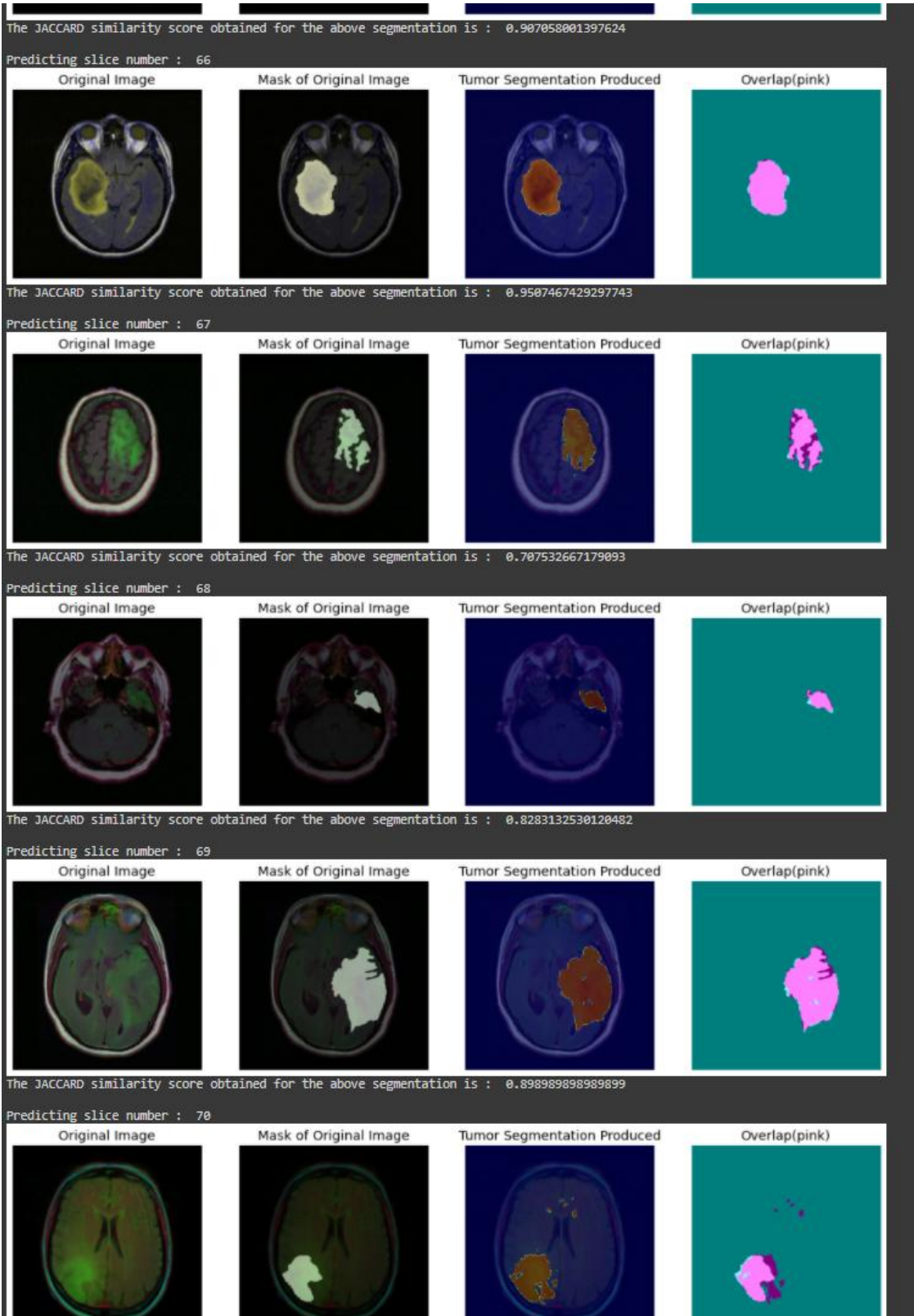


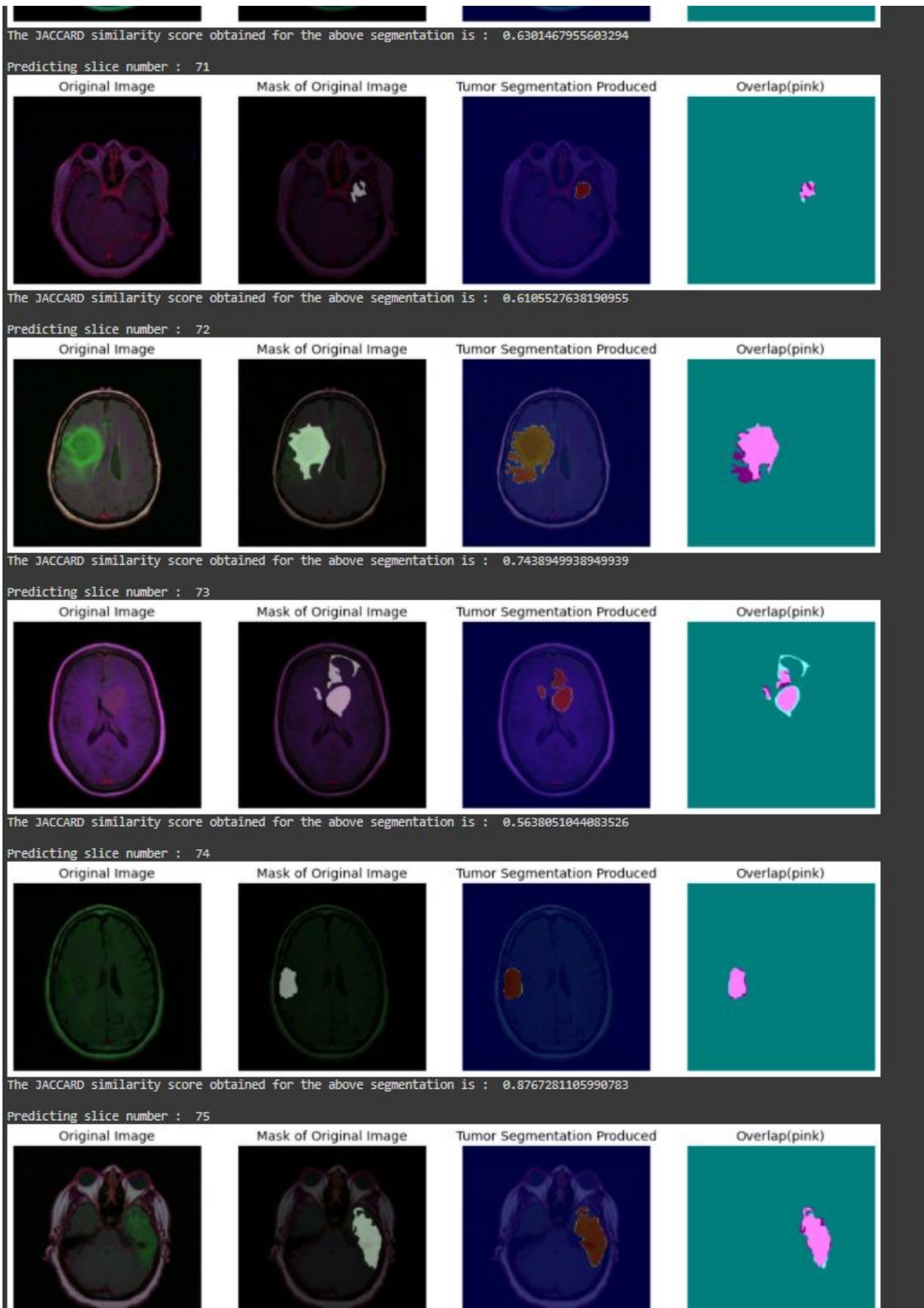


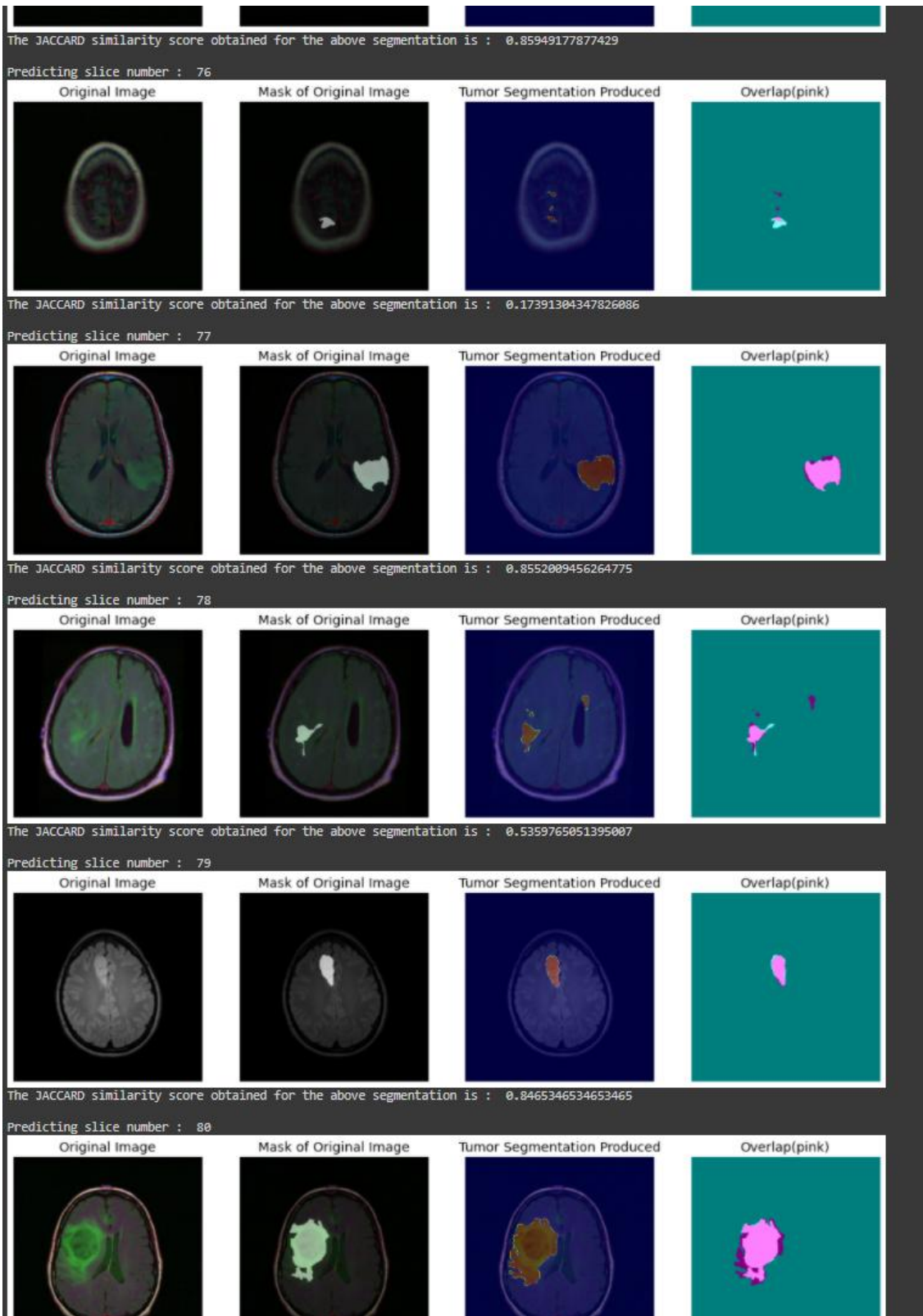


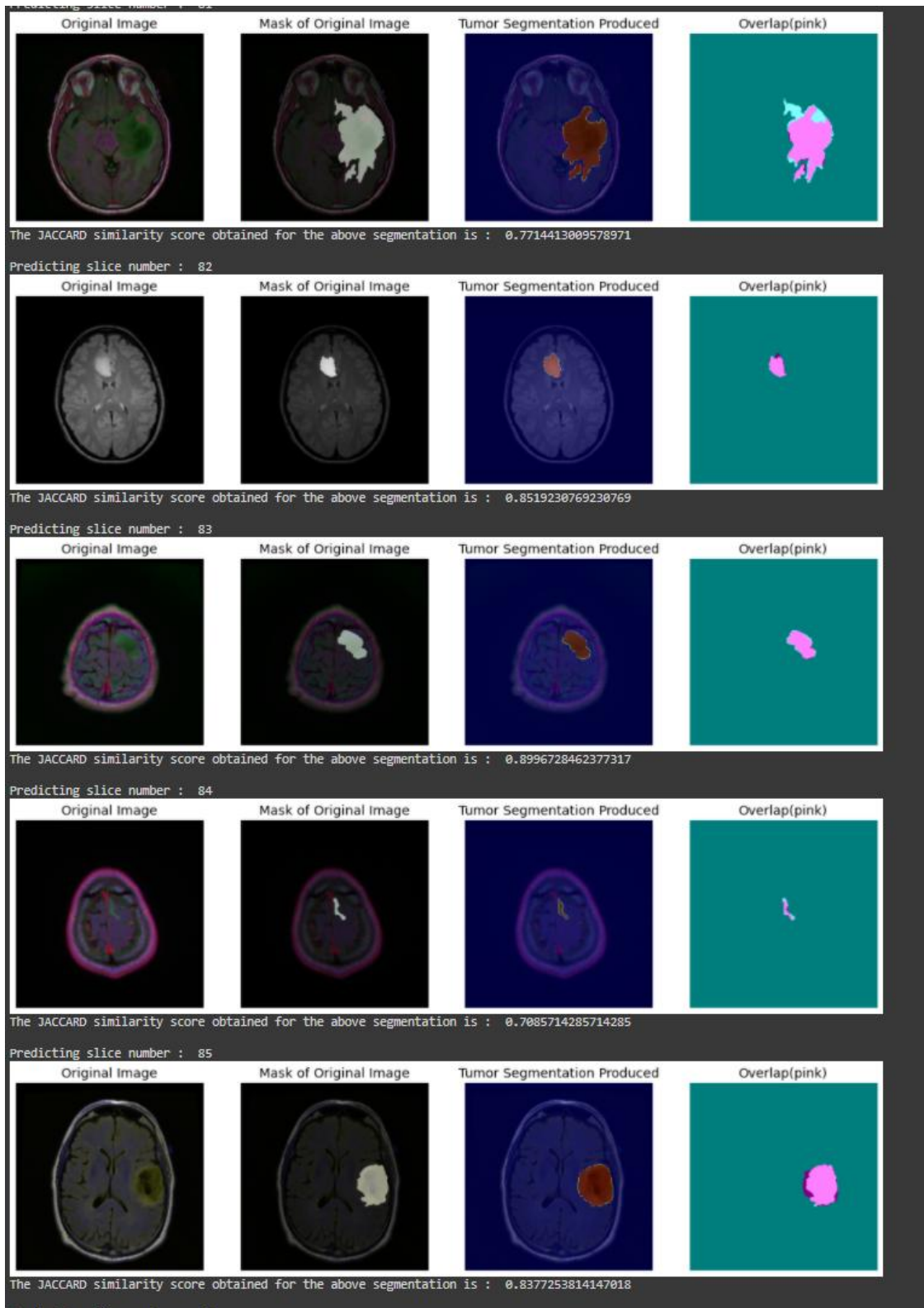


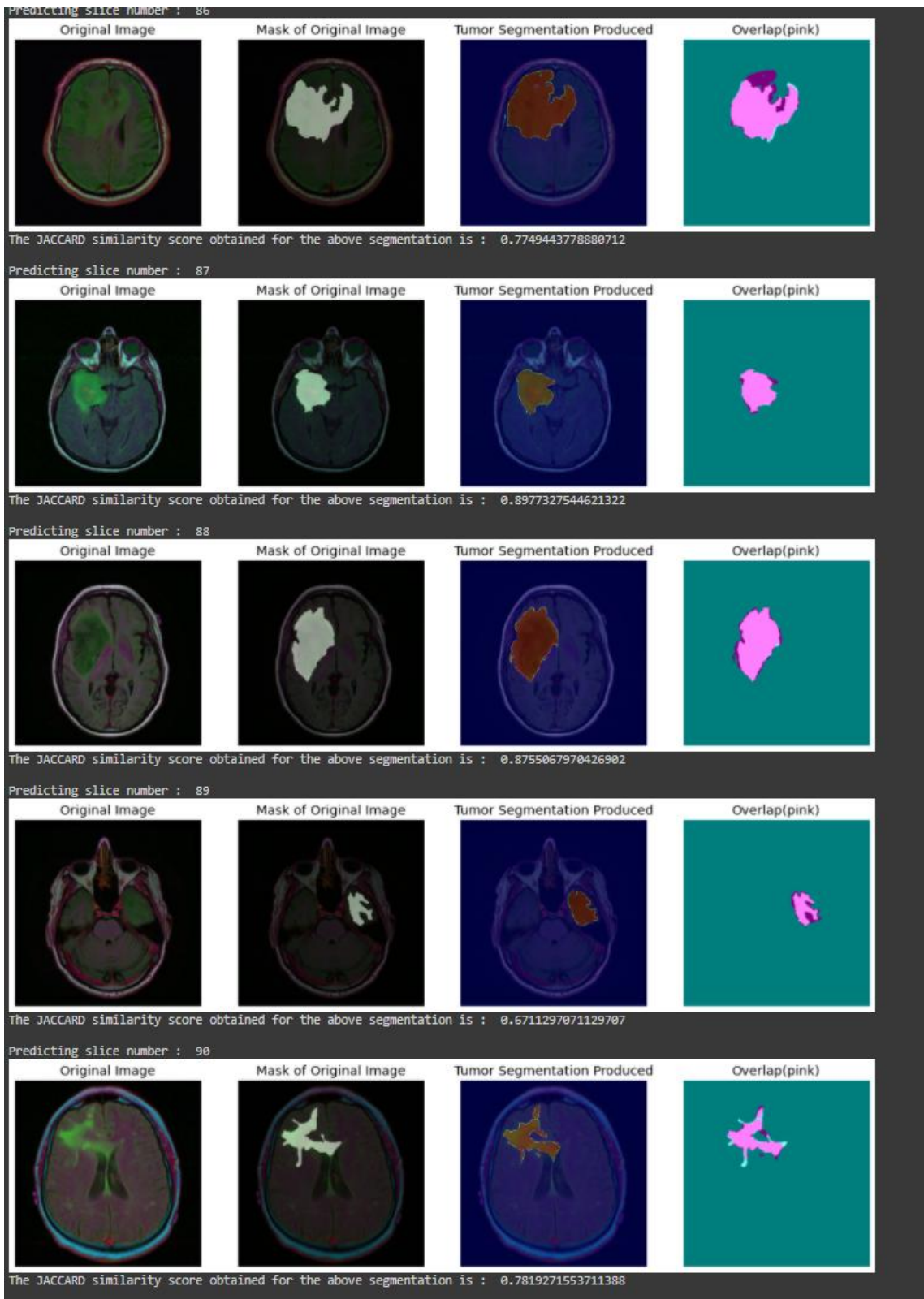


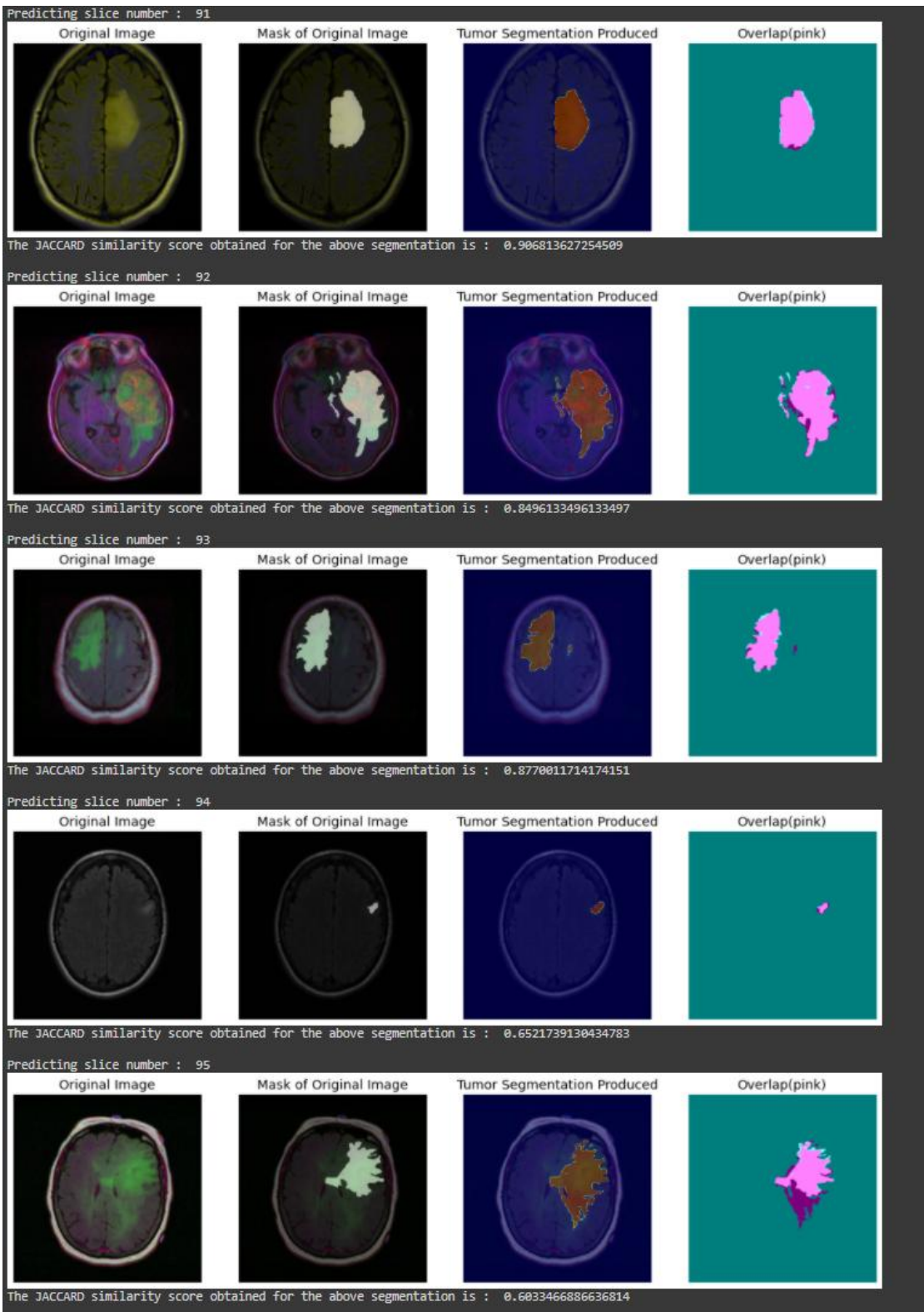


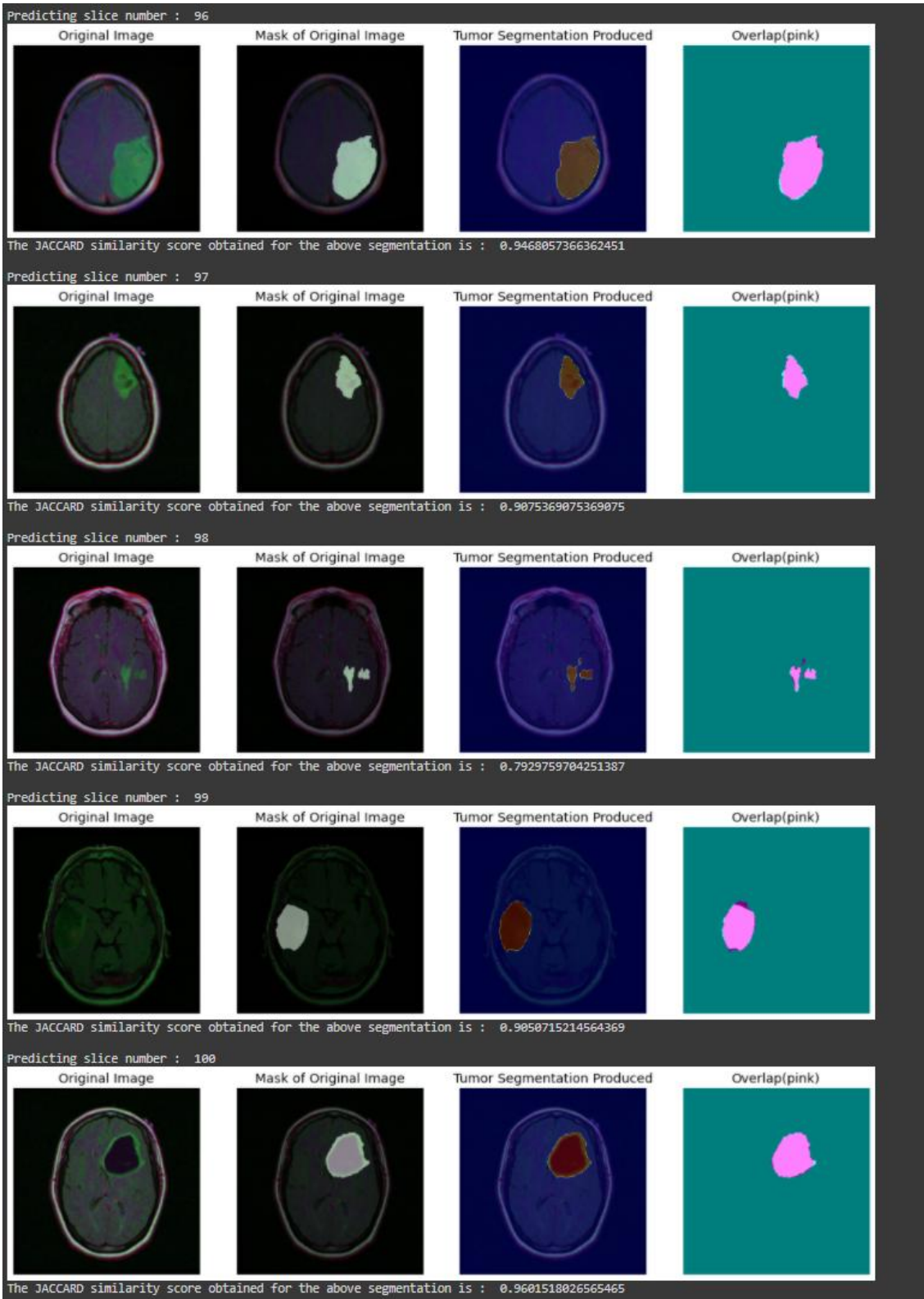


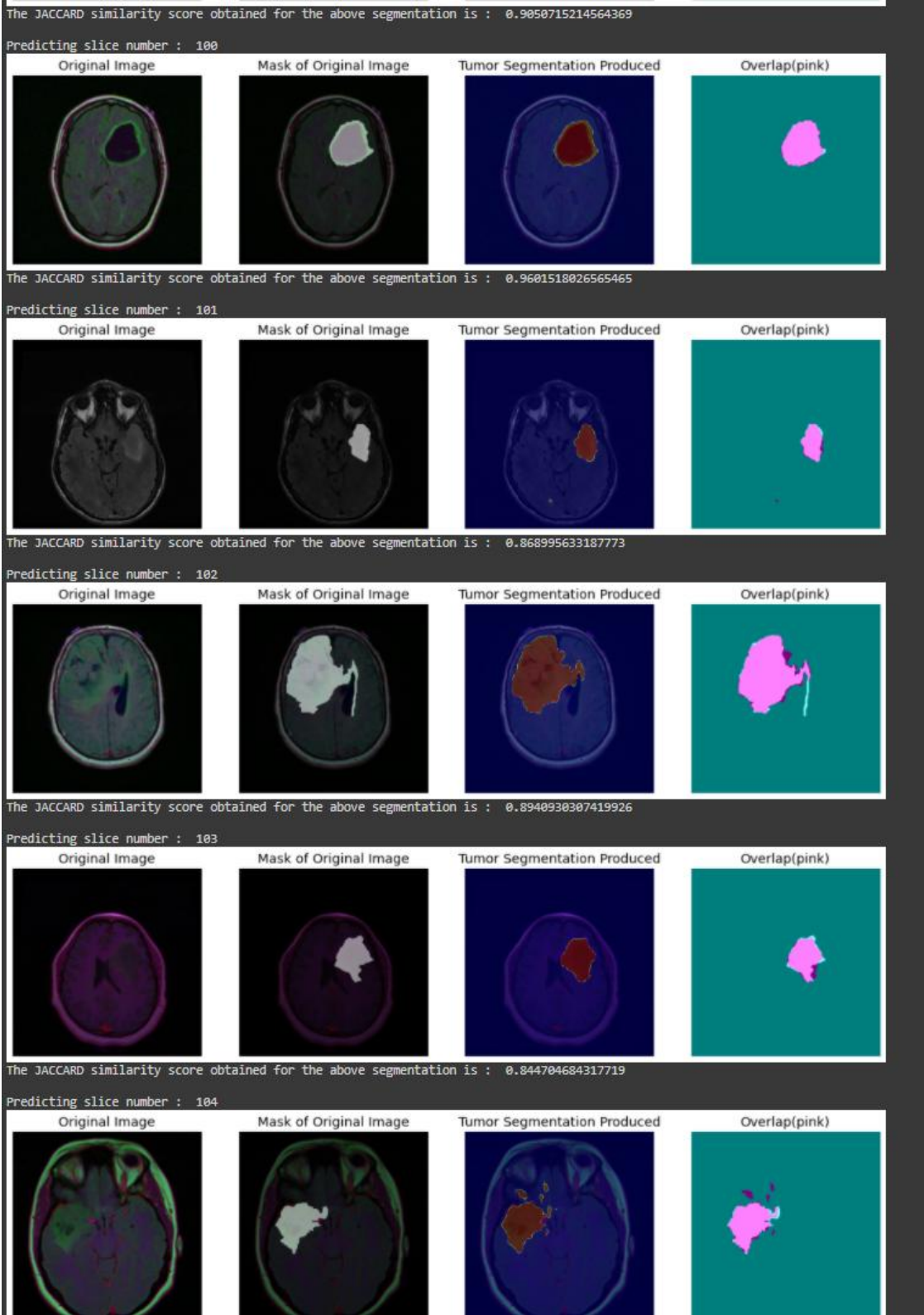


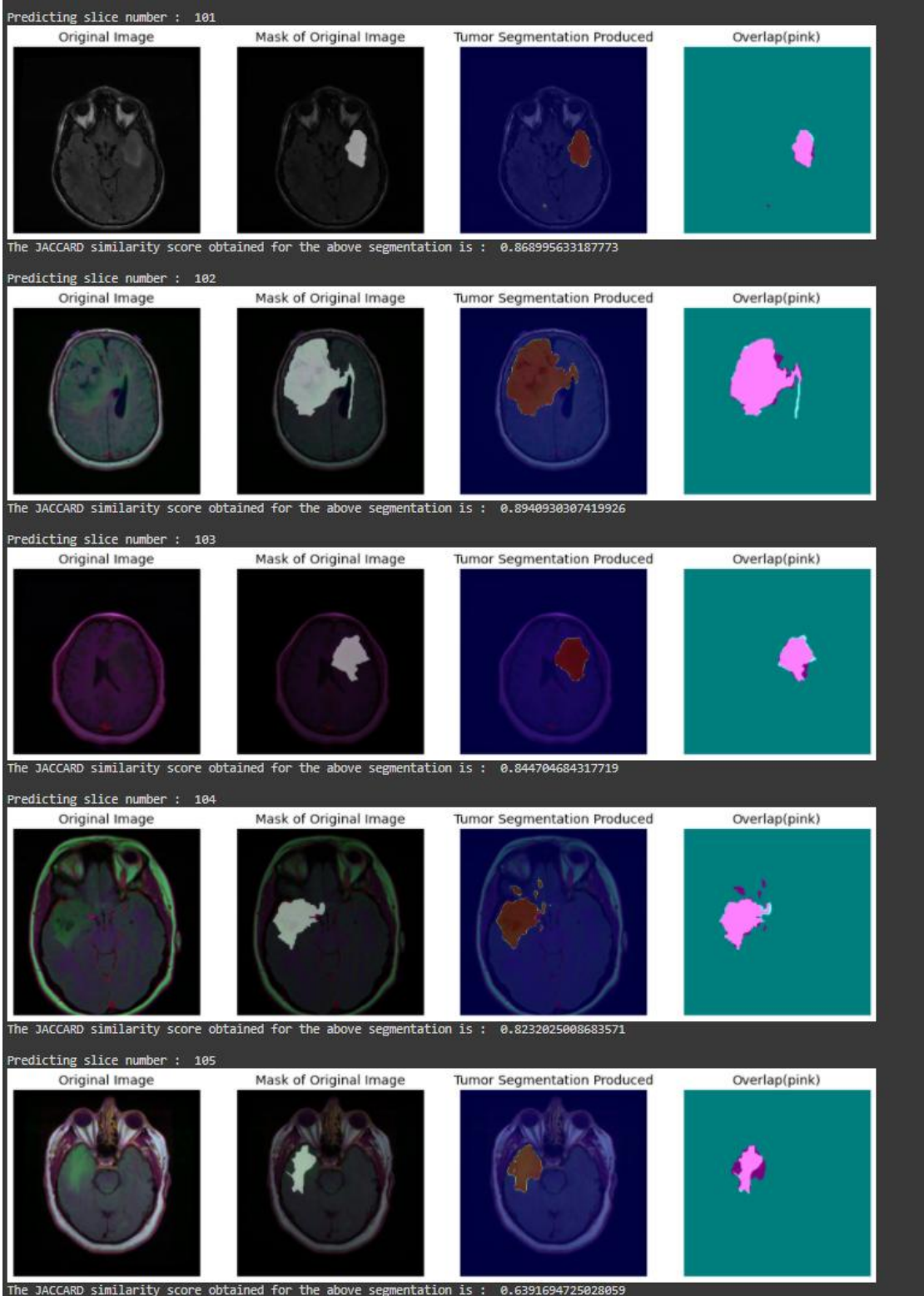


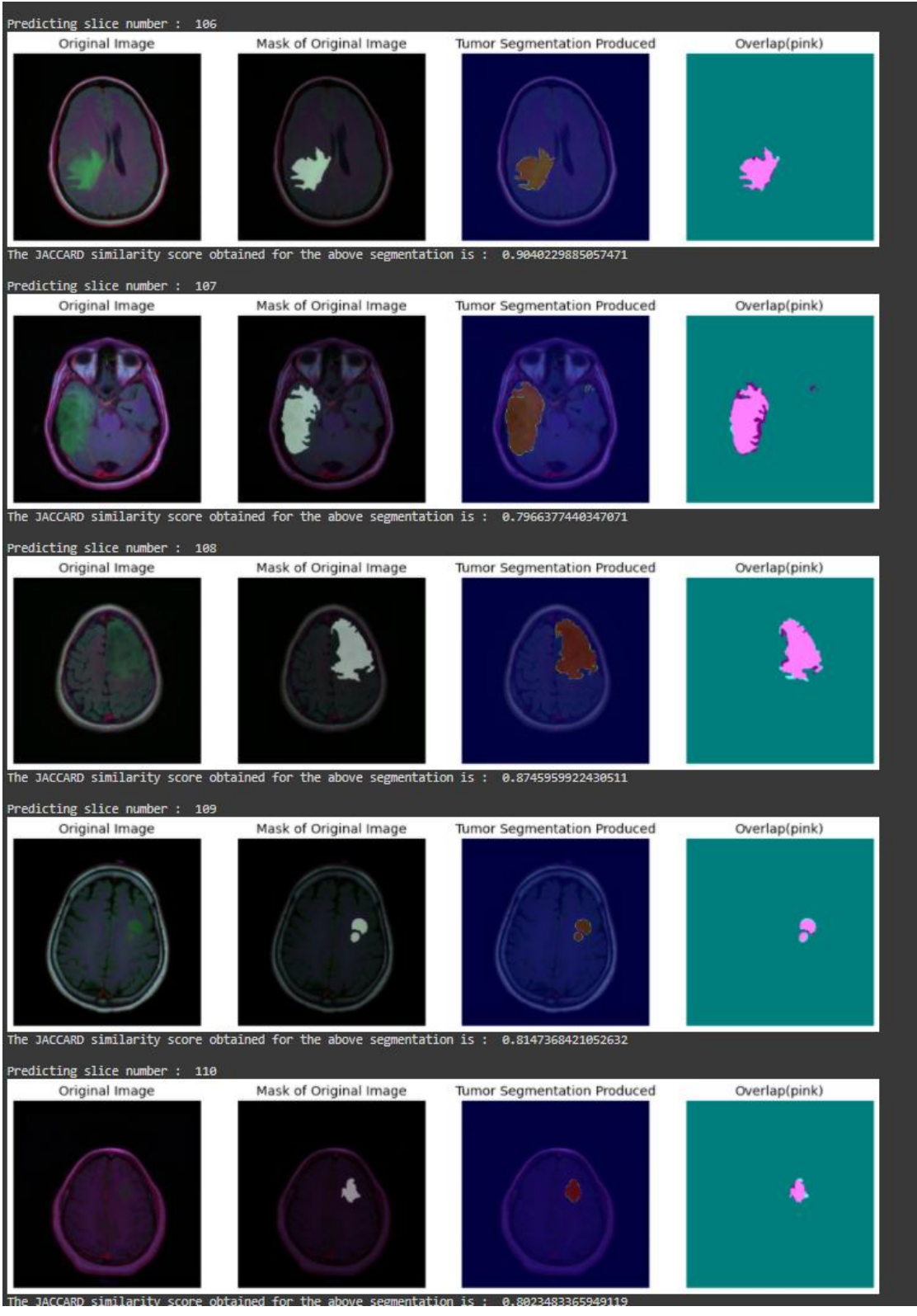


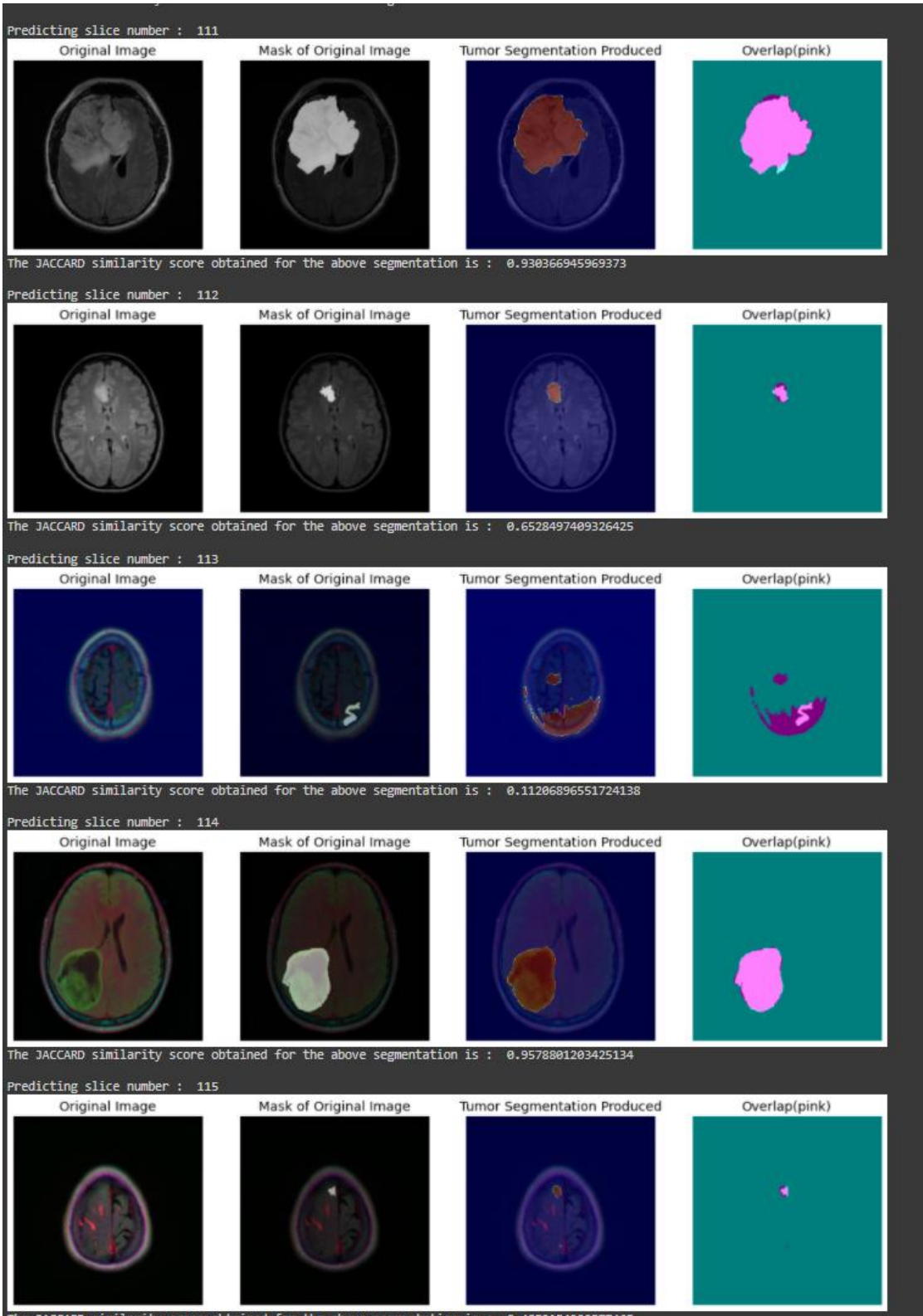


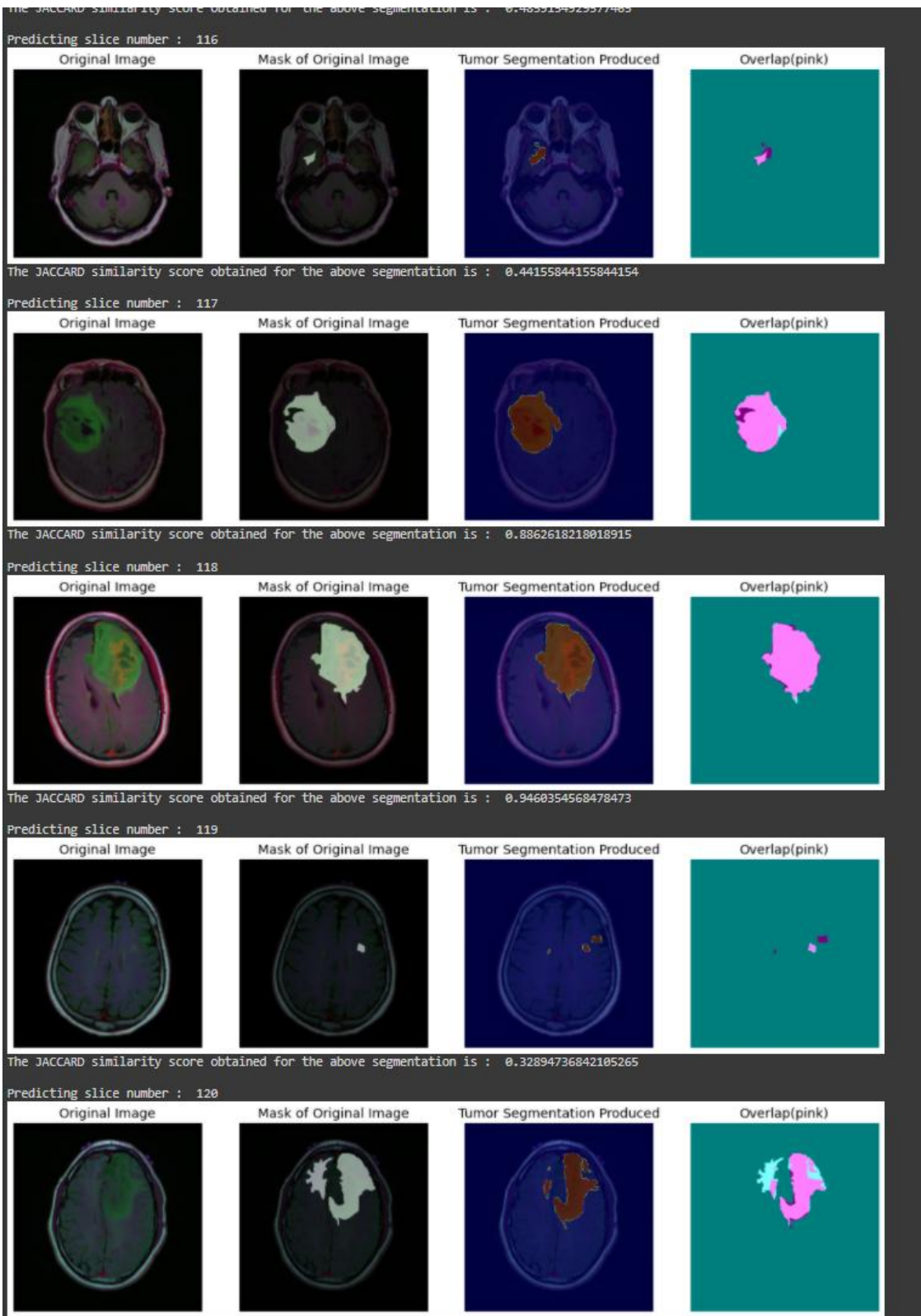


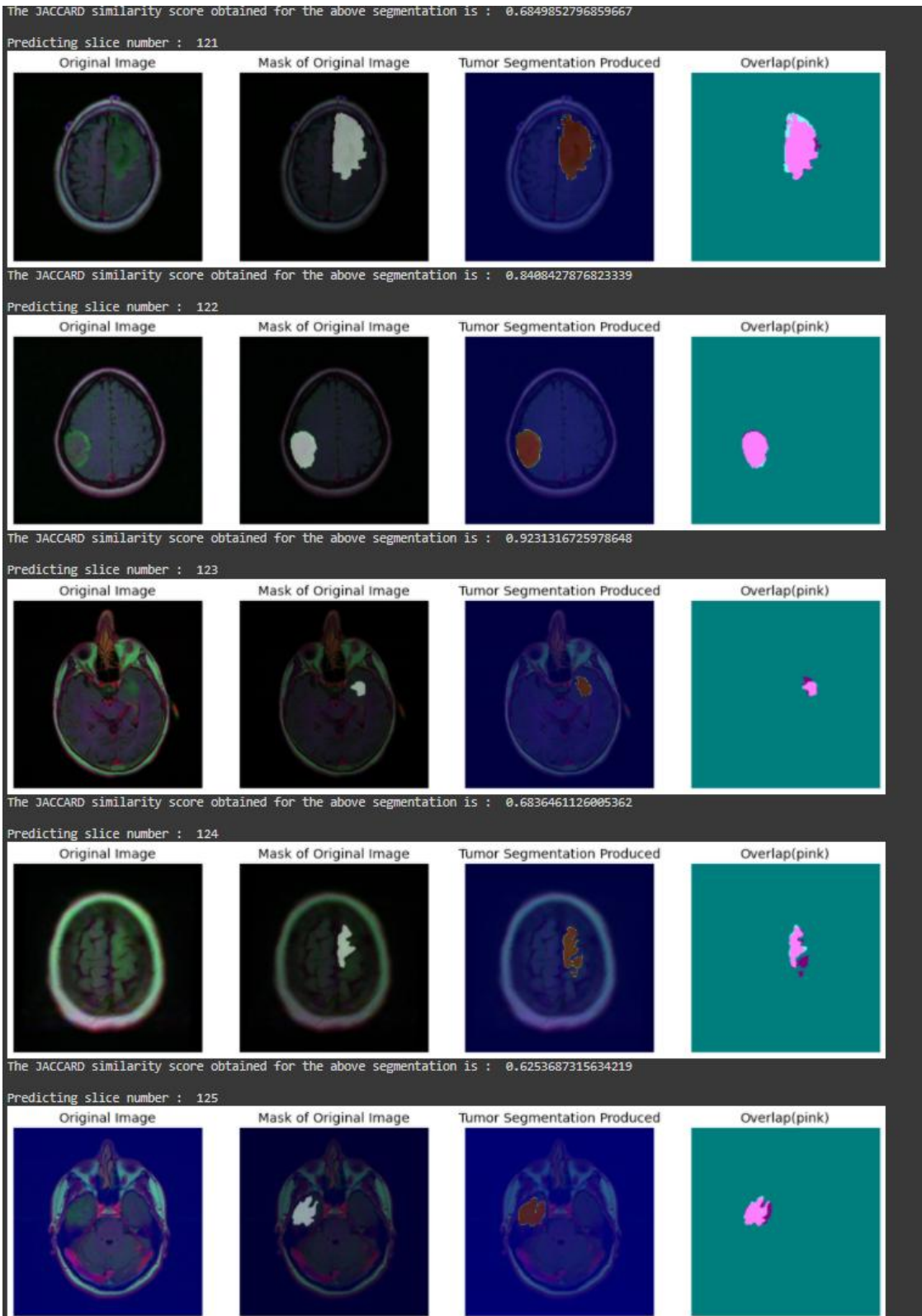






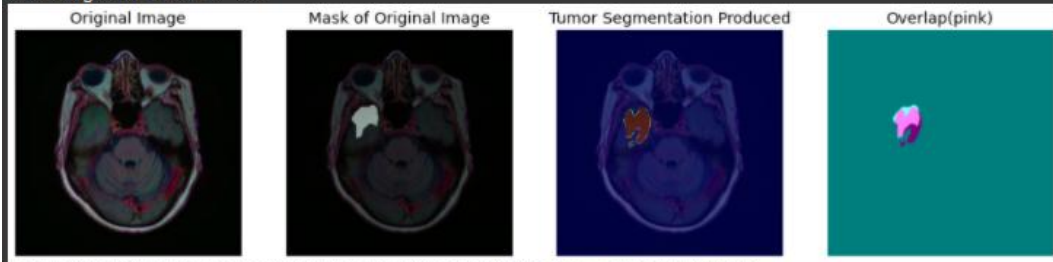






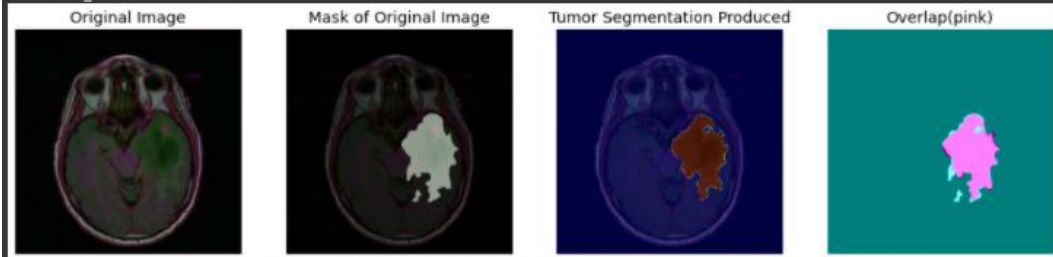
The JACCARD similarity score obtained for the above segmentation is : 0.7643129770992366

Predicting slice number : 126



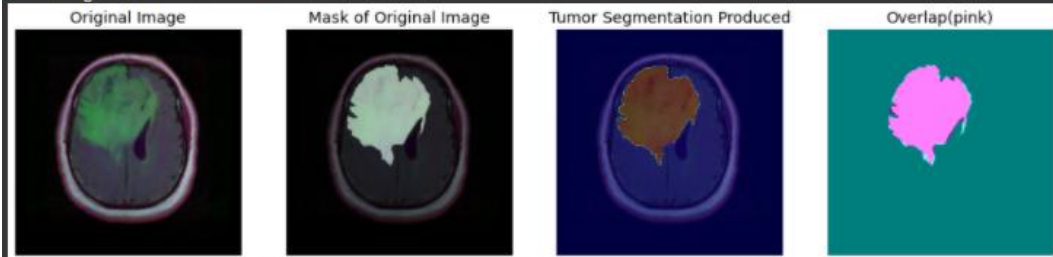
The JACCARD similarity score obtained for the above segmentation is : 0.5459401709401709

Predicting slice number : 127



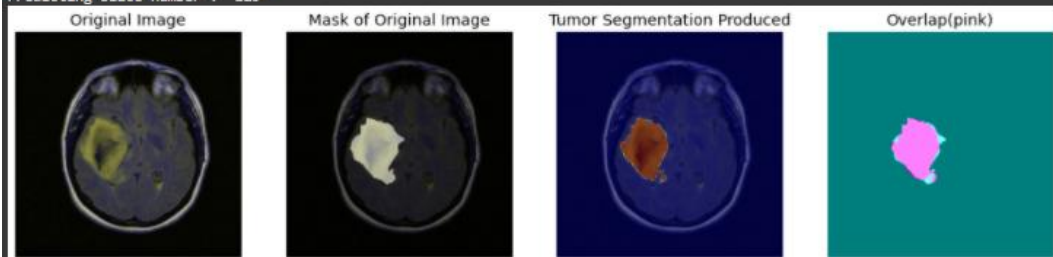
The JACCARD similarity score obtained for the above segmentation is : 0.8602863202545069

Predicting slice number : 128



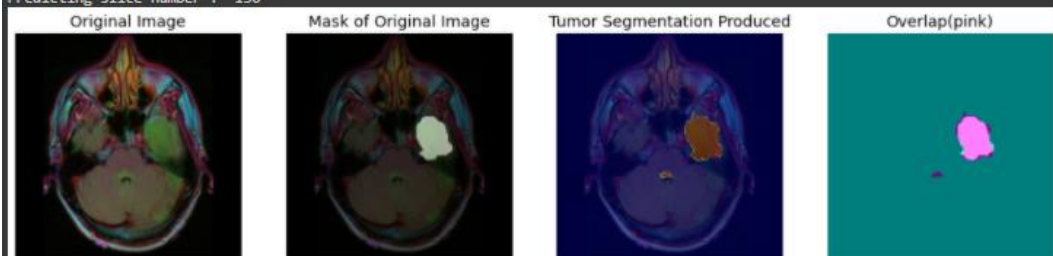
The JACCARD similarity score obtained for the above segmentation is : 0.9496248660235799

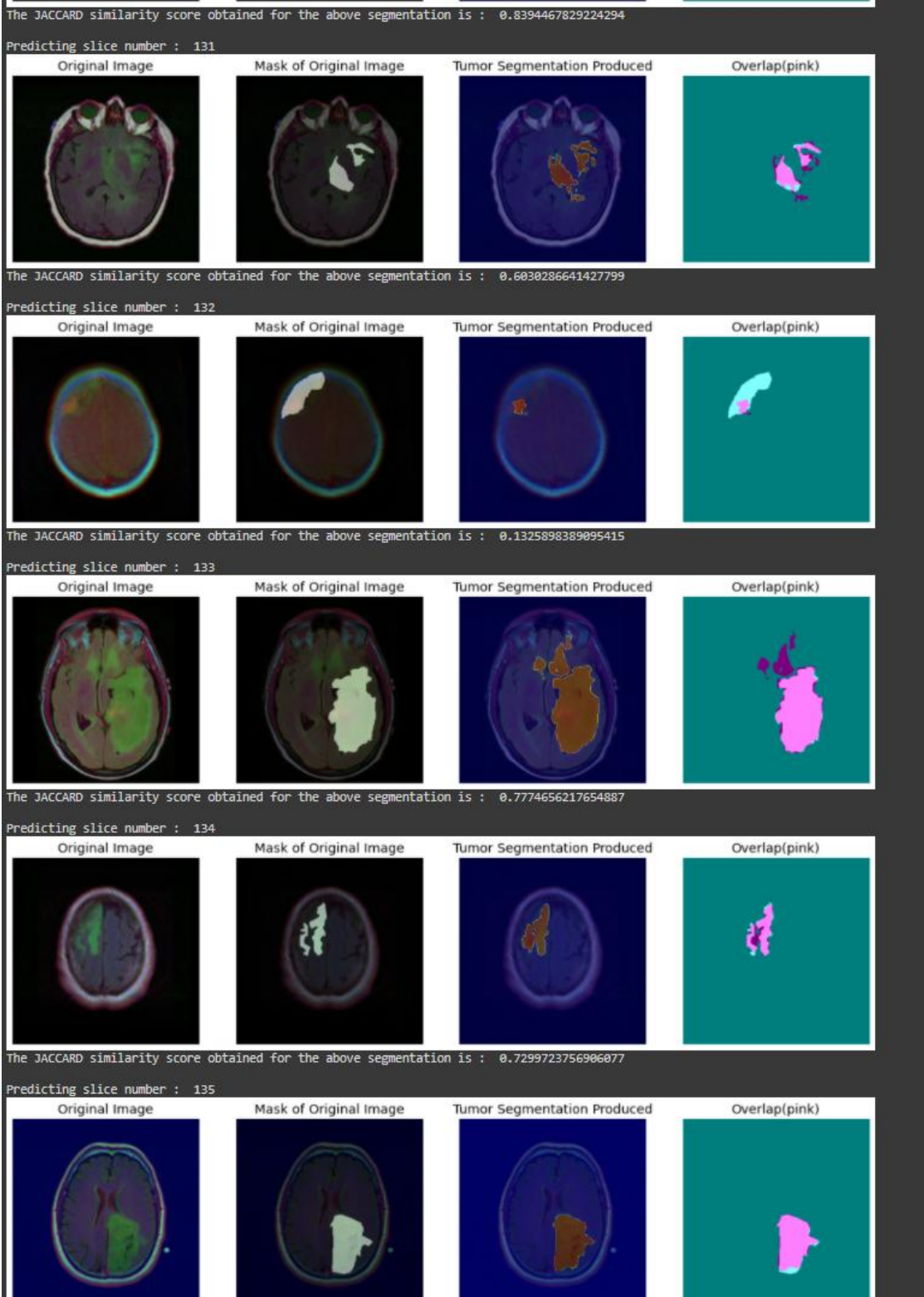
Predicting slice number : 129

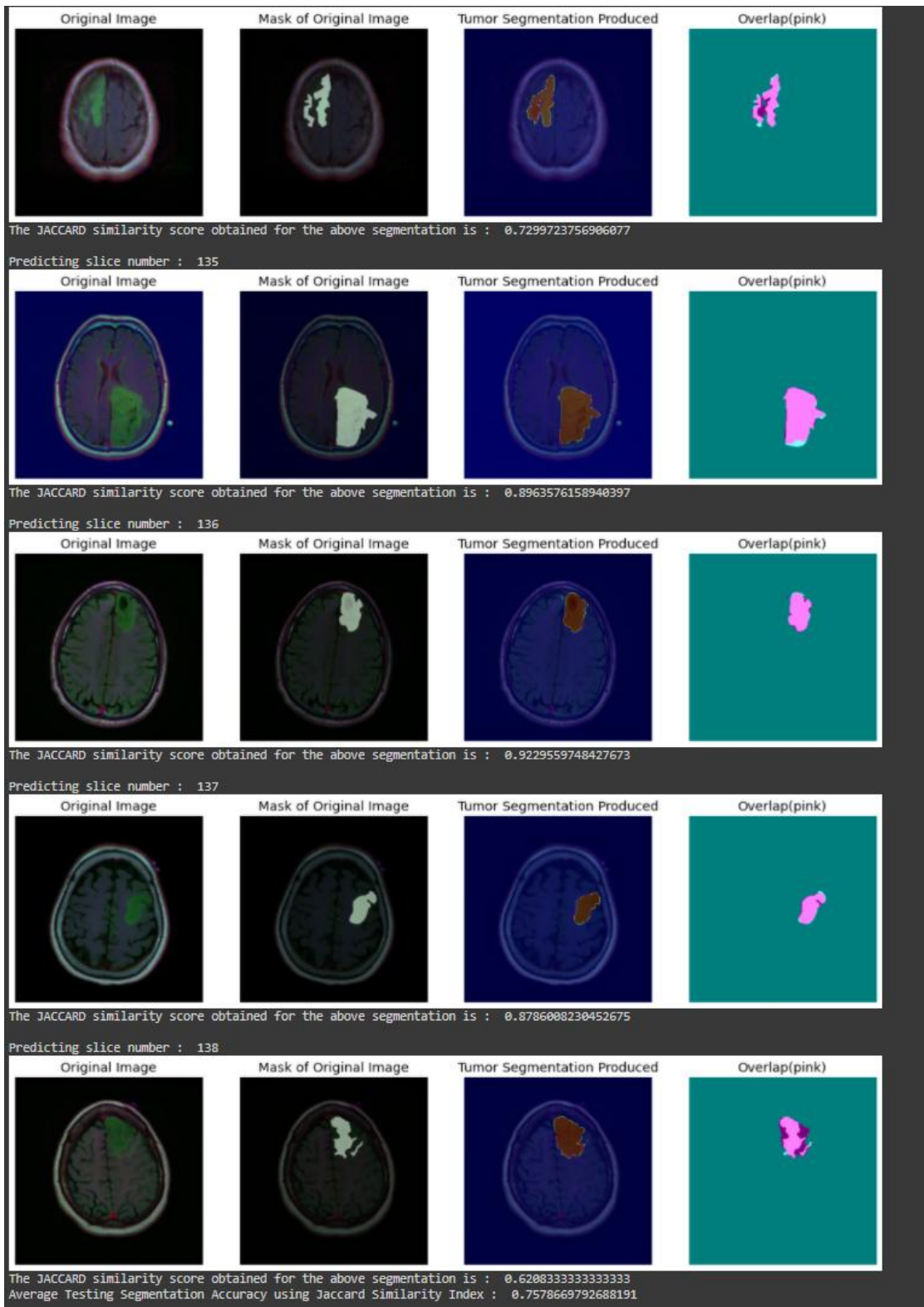


The JACCARD similarity score obtained for the above segmentation is : 0.8867992766726944

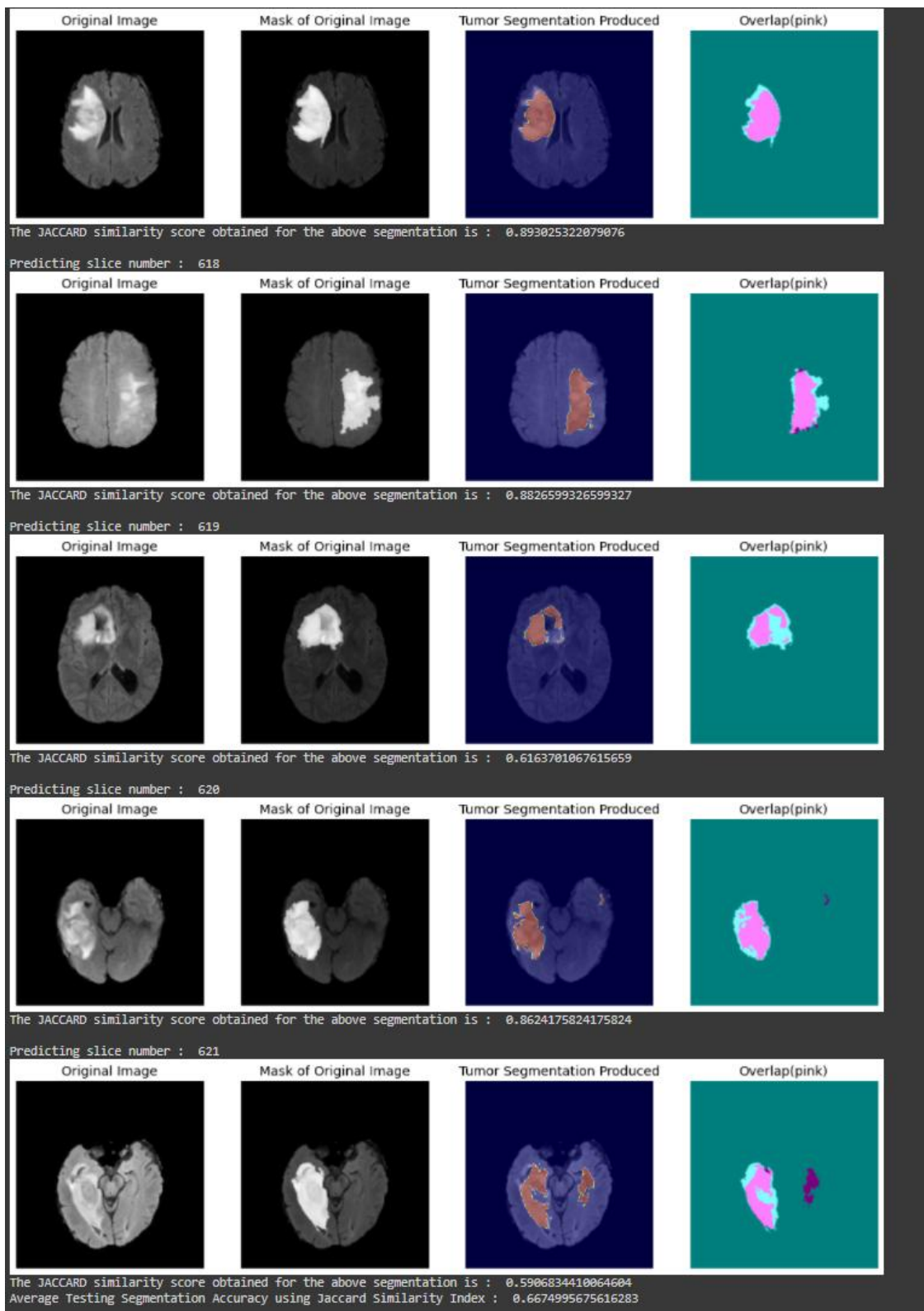
Predicting slice number : 130

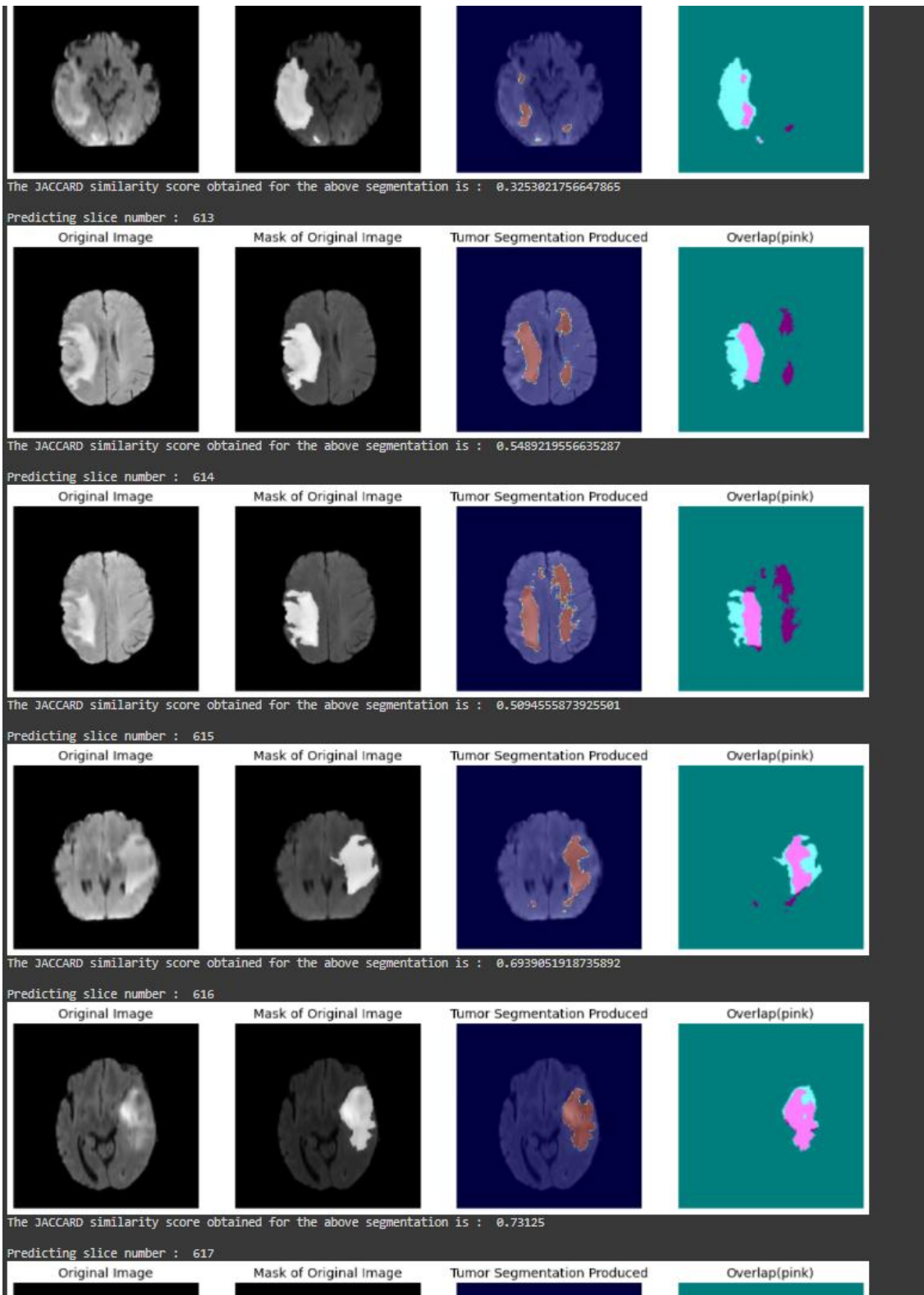


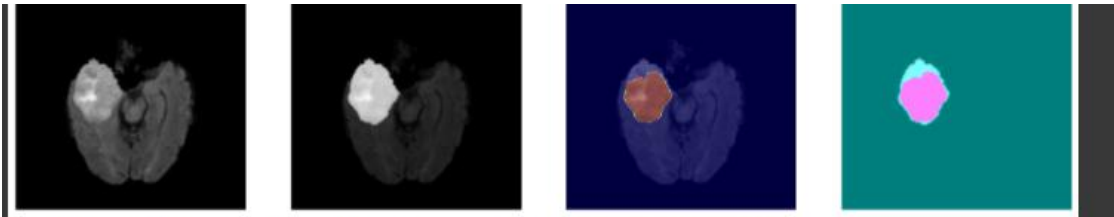




APPENDIX C2: Proposed model testing result on dataset[6]

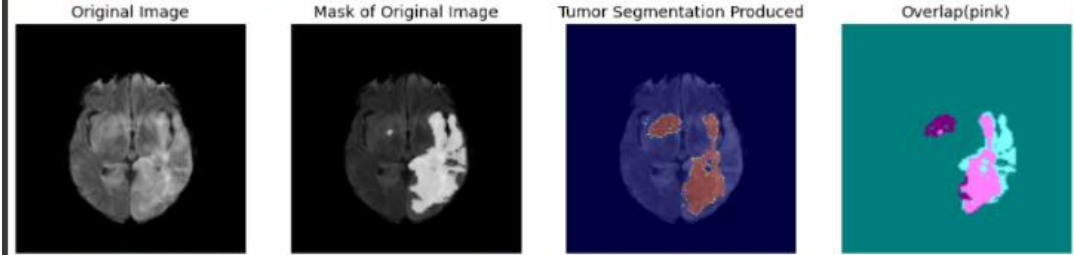






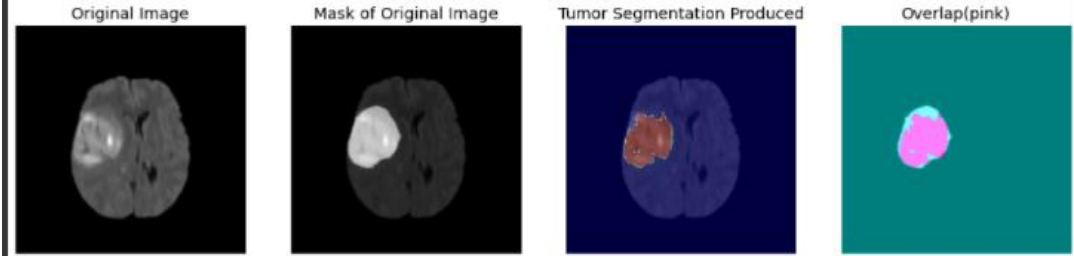
The JACCARD similarity score obtained for the above segmentation is : 0.9082405345211582

Predicting slice number : 604



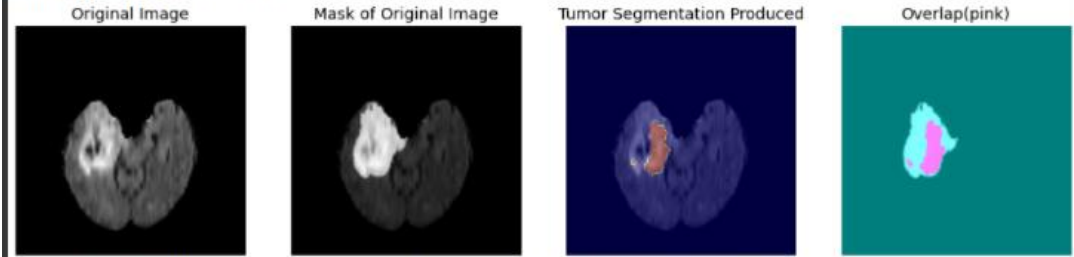
The JACCARD similarity score obtained for the above segmentation is : 0.6327153110047847

Predicting slice number : 605



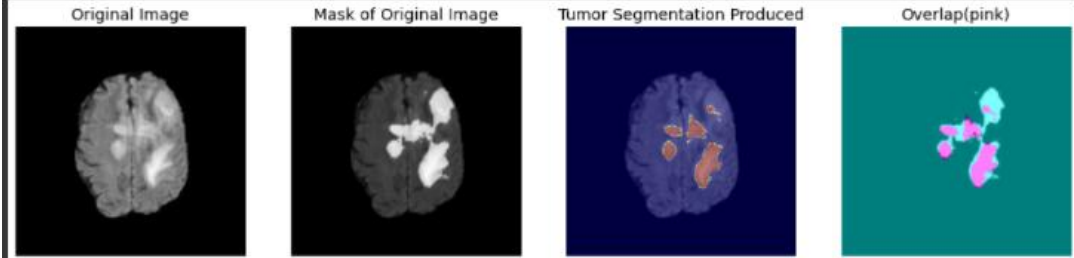
The JACCARD similarity score obtained for the above segmentation is : 0.8892700438771439

Predicting slice number : 606



The JACCARD similarity score obtained for the above segmentation is : 0.599047619047619

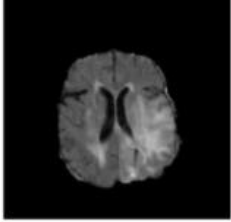
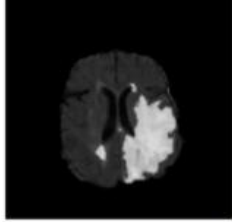
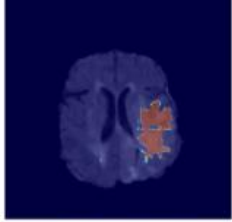

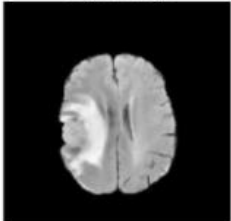
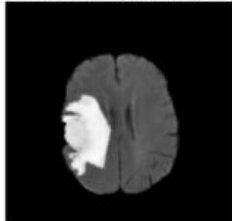
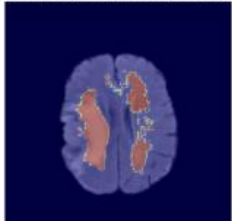

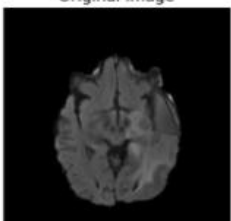
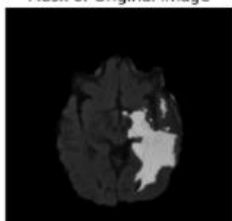
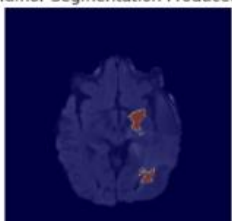

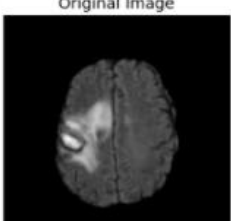
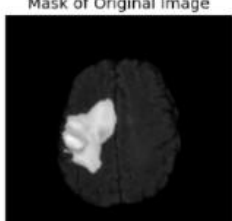
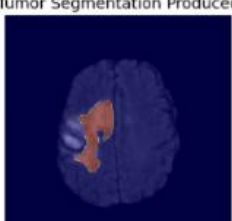

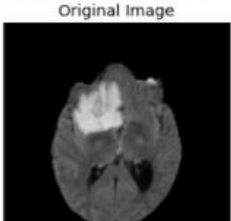

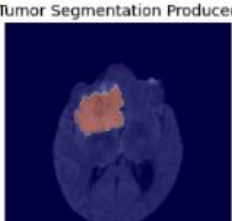

Predicting slice number : 607



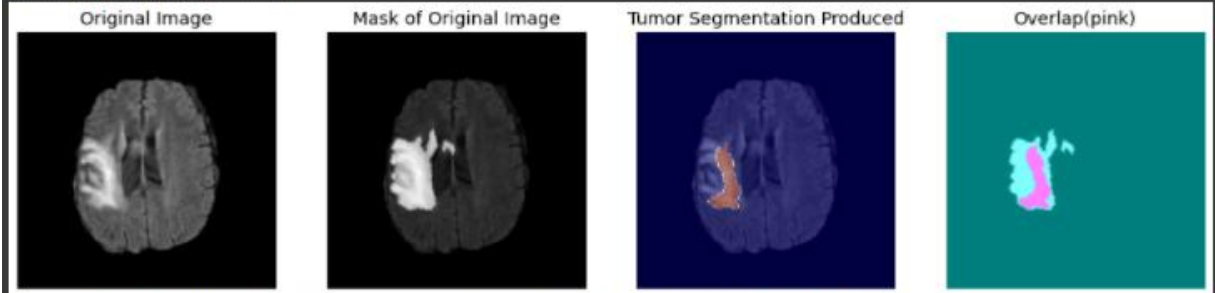
The JACCARD similarity score obtained for the above segmentation is : 0.5121476166068684

Predicting slice number : 608



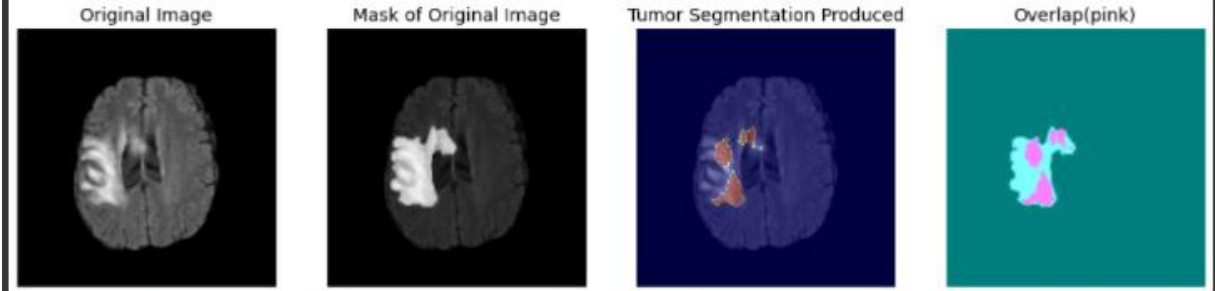
			
The JACCARD similarity score obtained for the above segmentation is : 0.6771817453963171			
Predicting slice number : 566			
Original Image	Mask of Original Image	Tumor Segmentation Produced	Overlap(pink)
			
The JACCARD similarity score obtained for the above segmentation is : 0.5133528265107212			
Predicting slice number : 567			
Original Image	Mask of Original Image	Tumor Segmentation Produced	Overlap(pink)
			
The JACCARD similarity score obtained for the above segmentation is : 0.24379562043795622			
Predicting slice number : 568			
Original Image	Mask of Original Image	Tumor Segmentation Produced	Overlap(pink)
			
The JACCARD similarity score obtained for the above segmentation is : 0.6991652754590985			
Predicting slice number : 569			
Original Image	Mask of Original Image	Tumor Segmentation Produced	Overlap(pink)
			
The JACCARD similarity score obtained for the above segmentation is : 0.9653579676674364			
Predicting slice number : 570			
Original Image	Mask of Original Image	Tumor Segmentation Produced	Overlap(pink)

Predicting slice number : 507



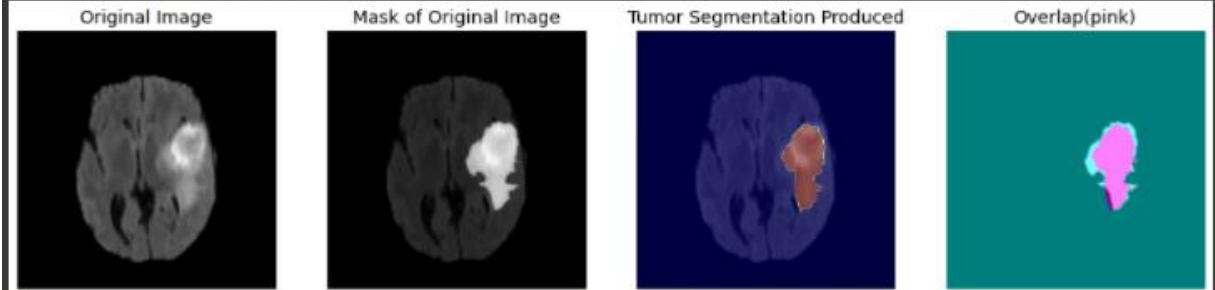
The JACCARD similarity score obtained for the above segmentation is : 0.643649373881932

Predicting slice number : 508



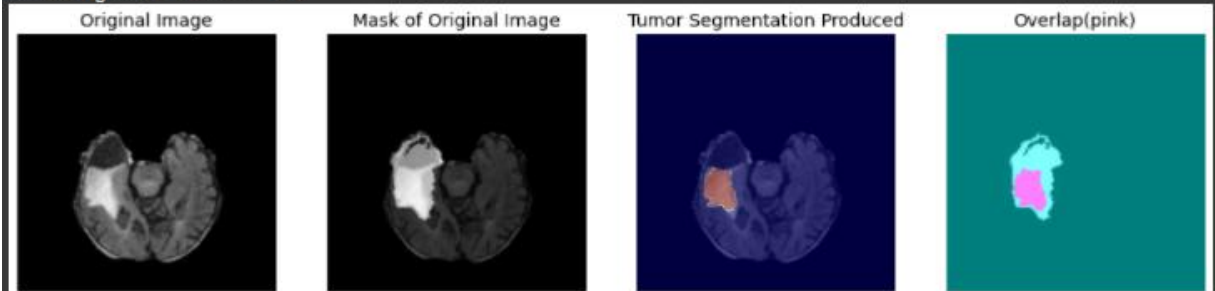
The JACCARD similarity score obtained for the above segmentation is : 0.6030418250950571

Predicting slice number : 509



The JACCARD similarity score obtained for the above segmentation is : 0.7526548672566371

Predicting slice number : 510

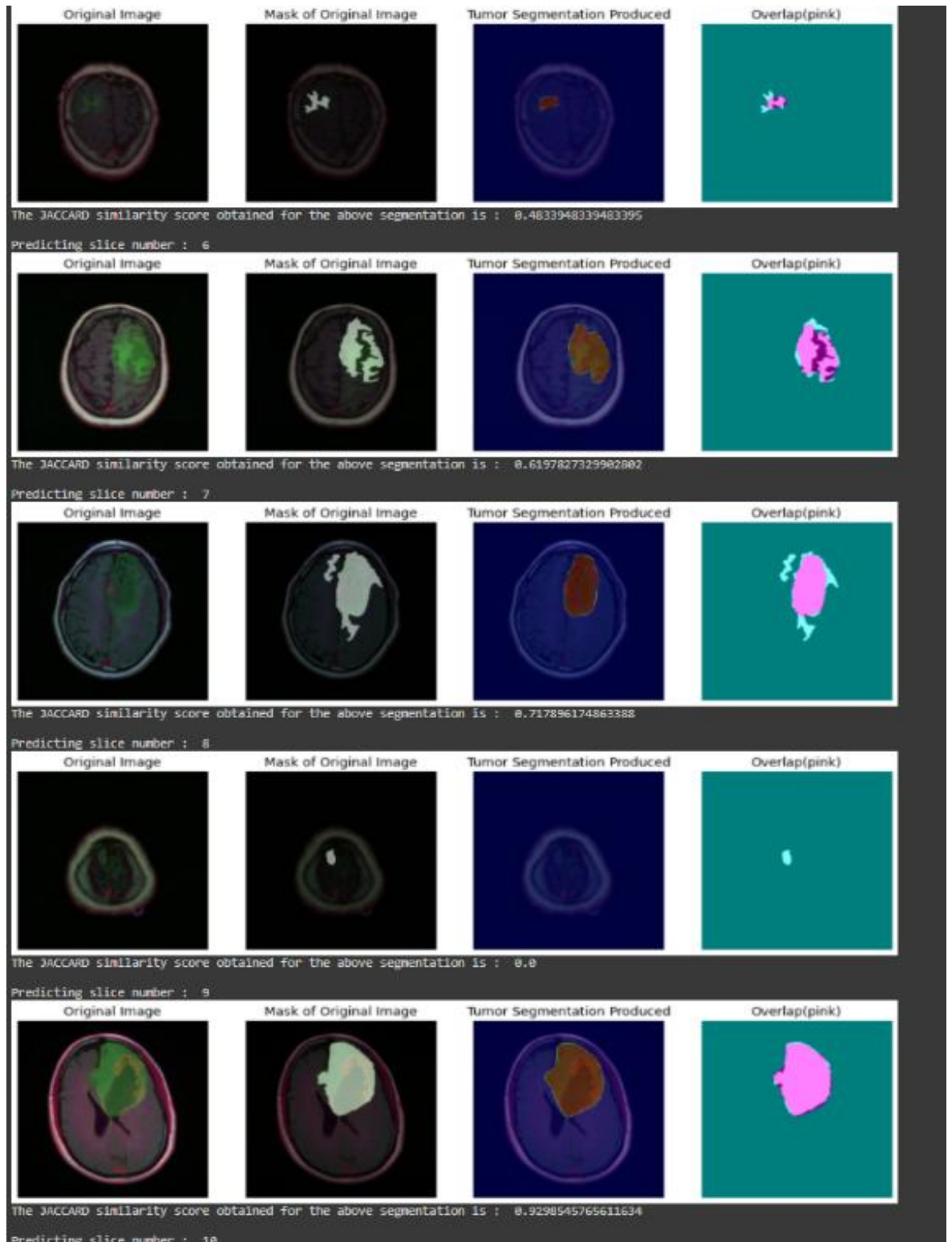


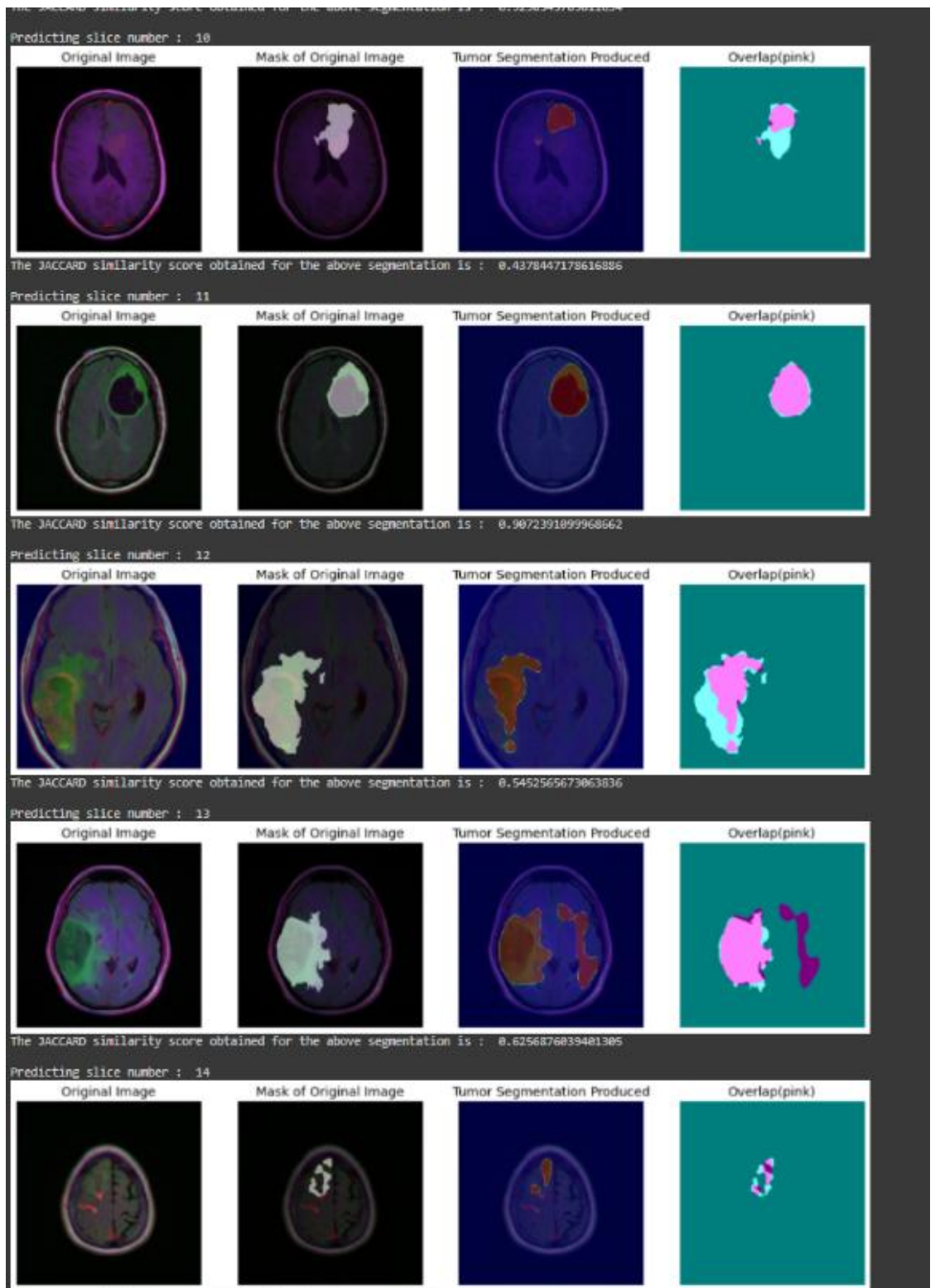
The JACCARD similarity score obtained for the above segmentation is : 0.618836565096953

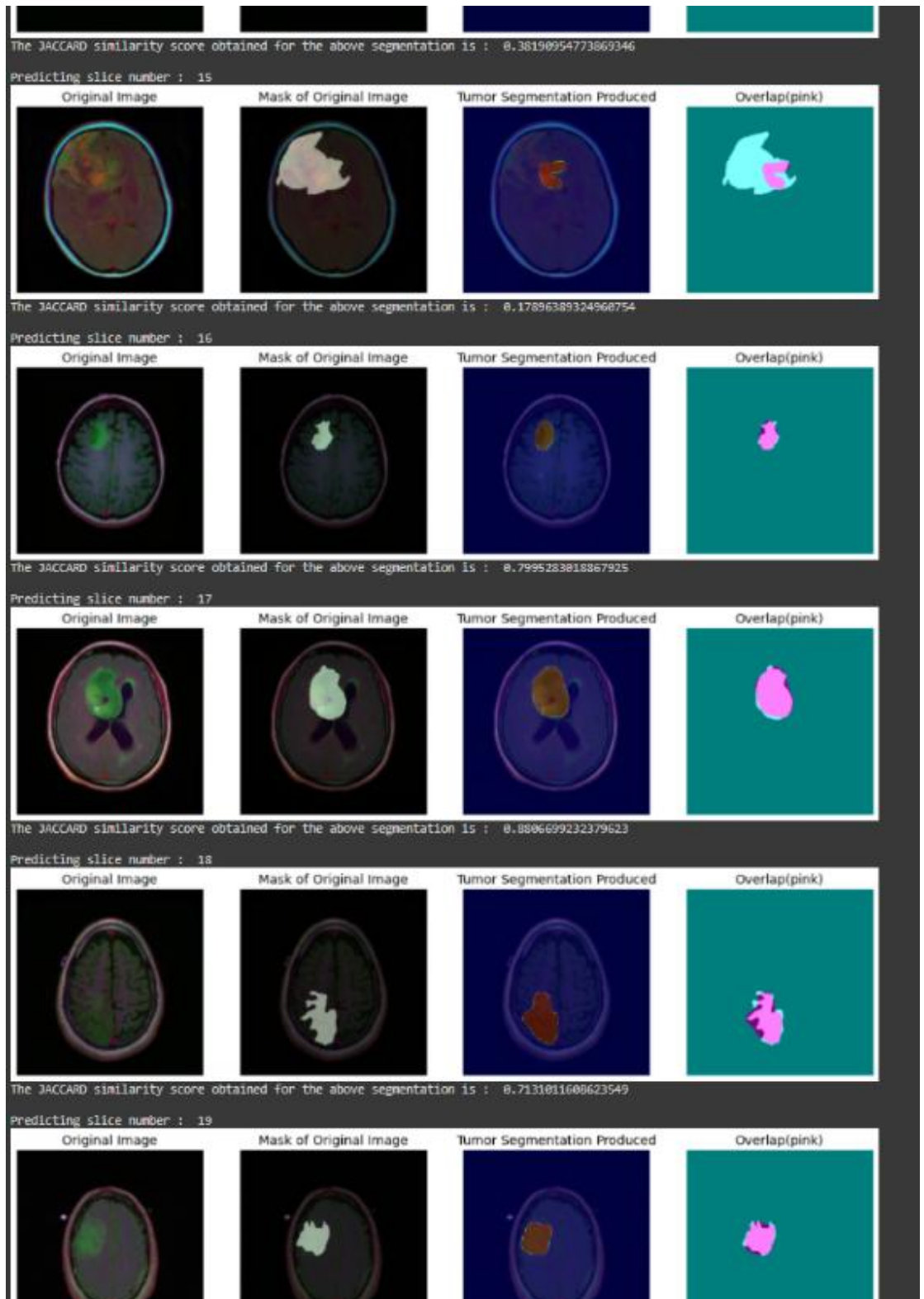
APPENDIX D: Experimental model results

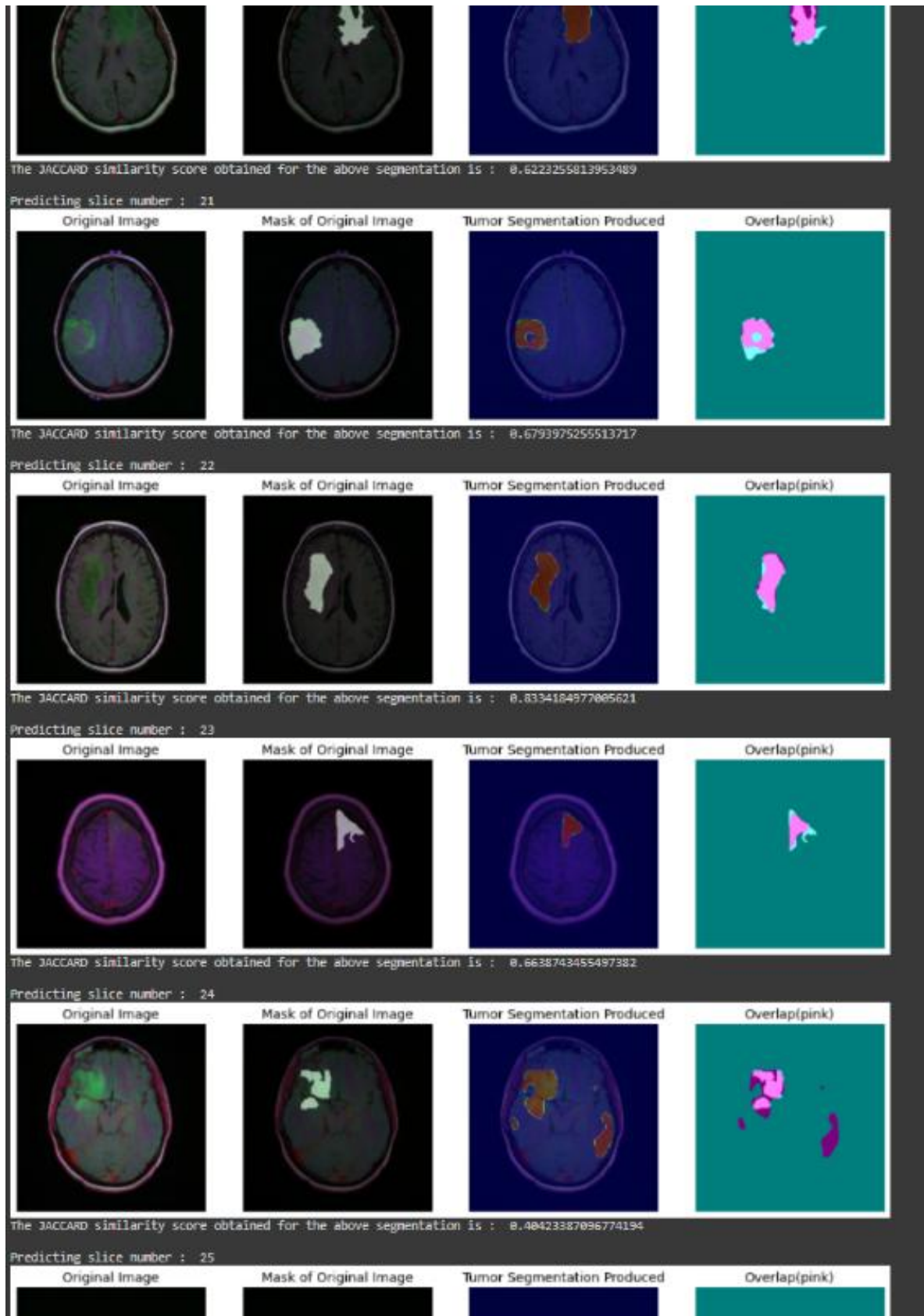
The experimental models is named based on Table 5.1

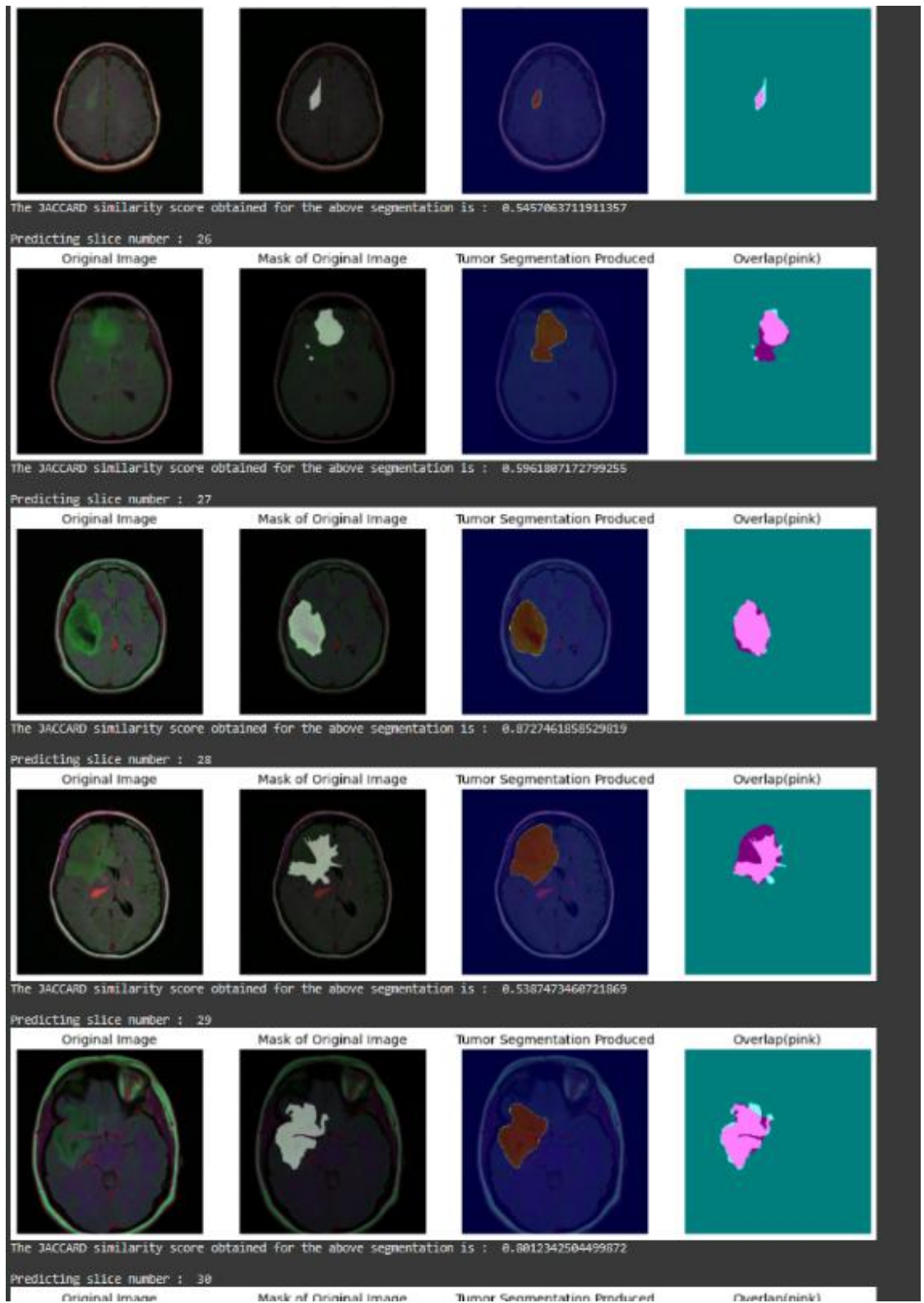
1. Experimental Model 5

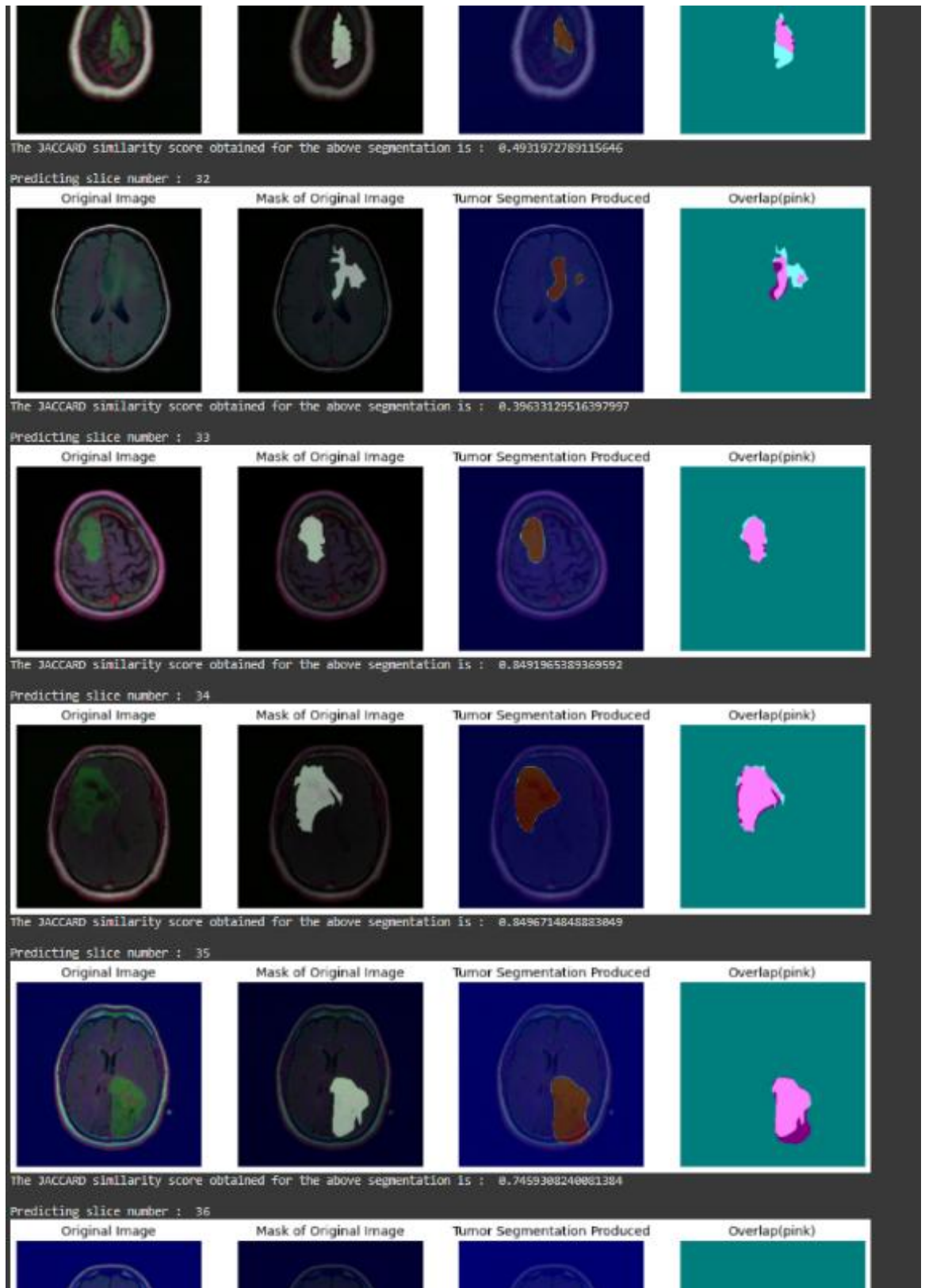


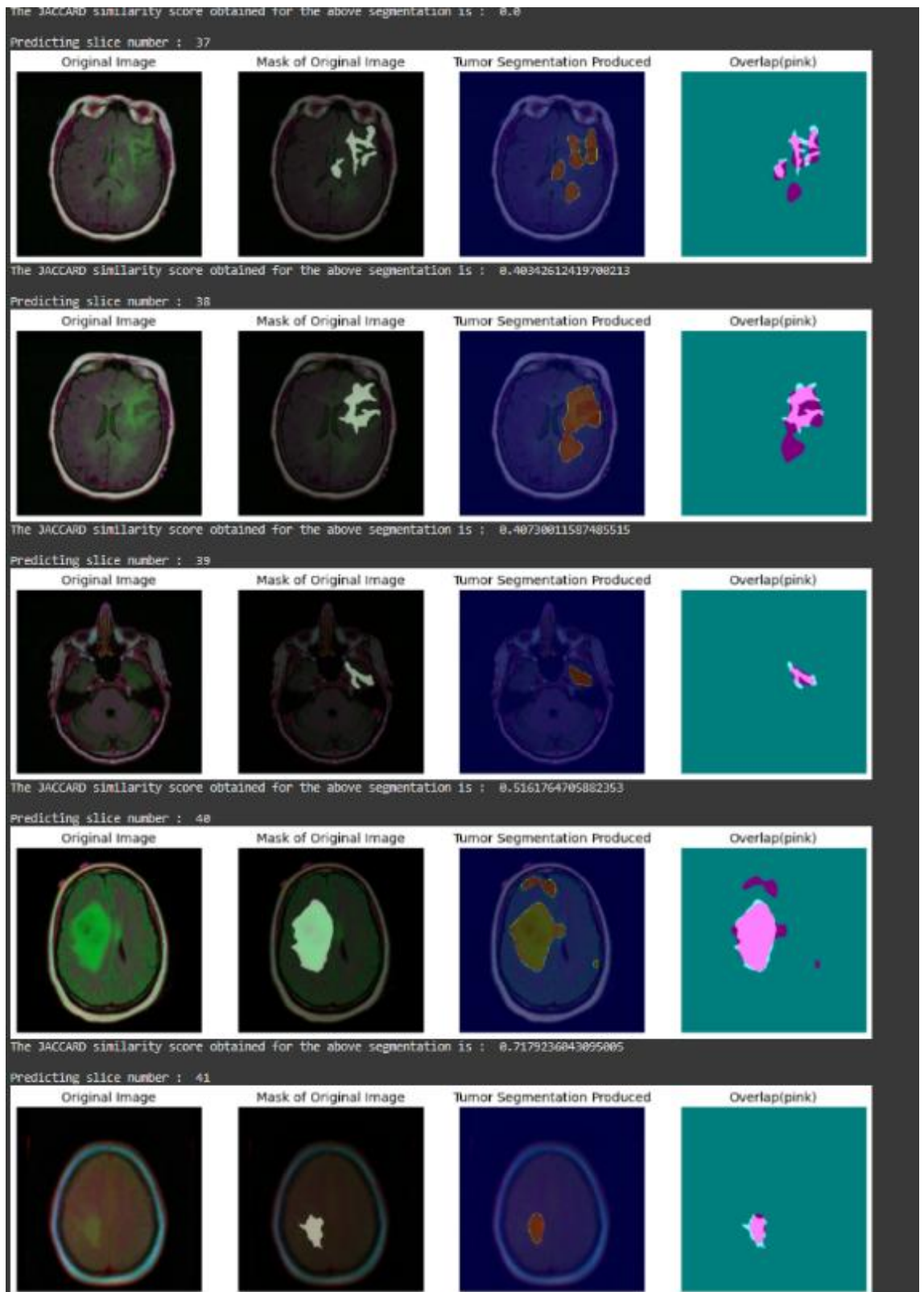


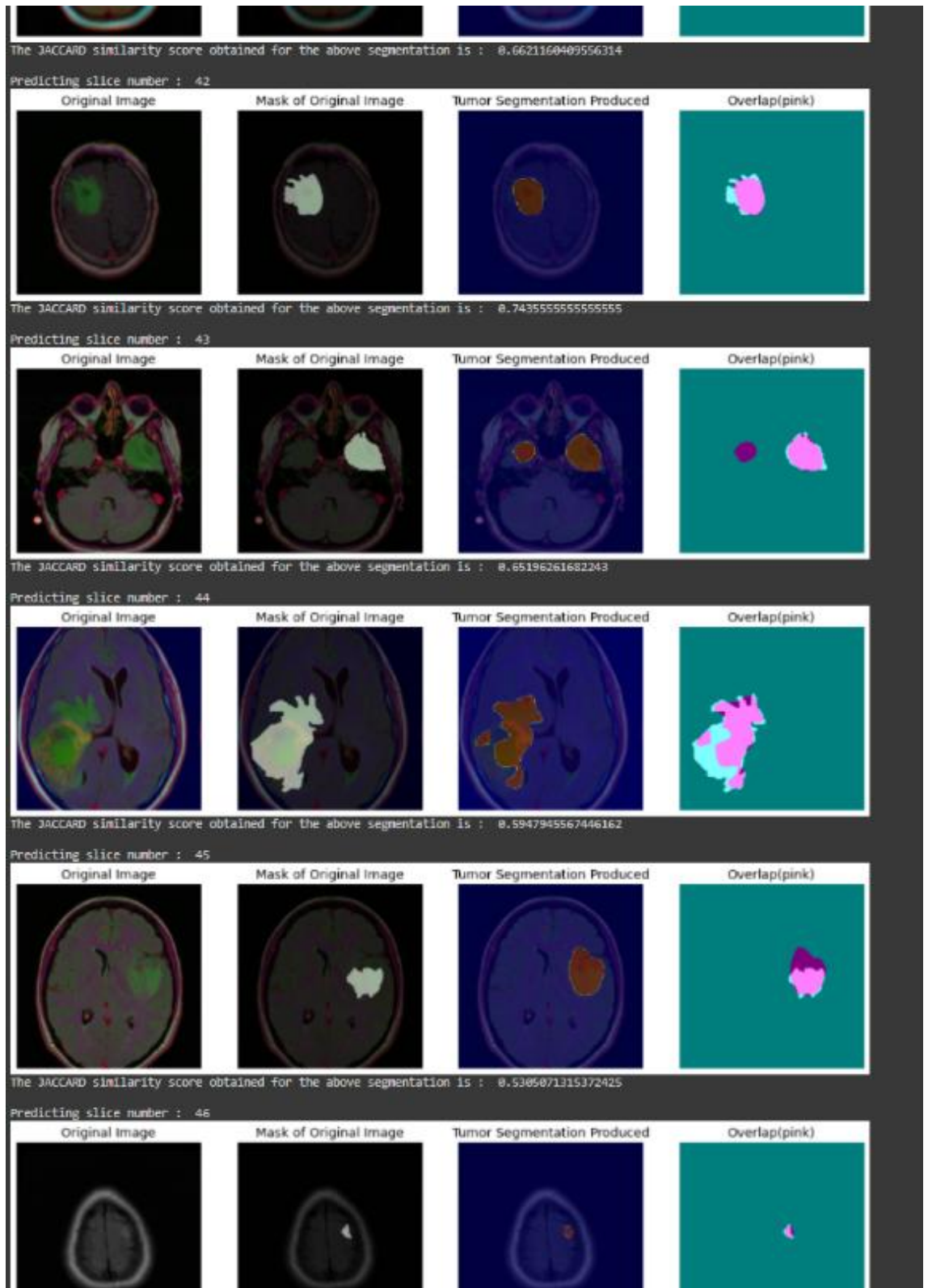


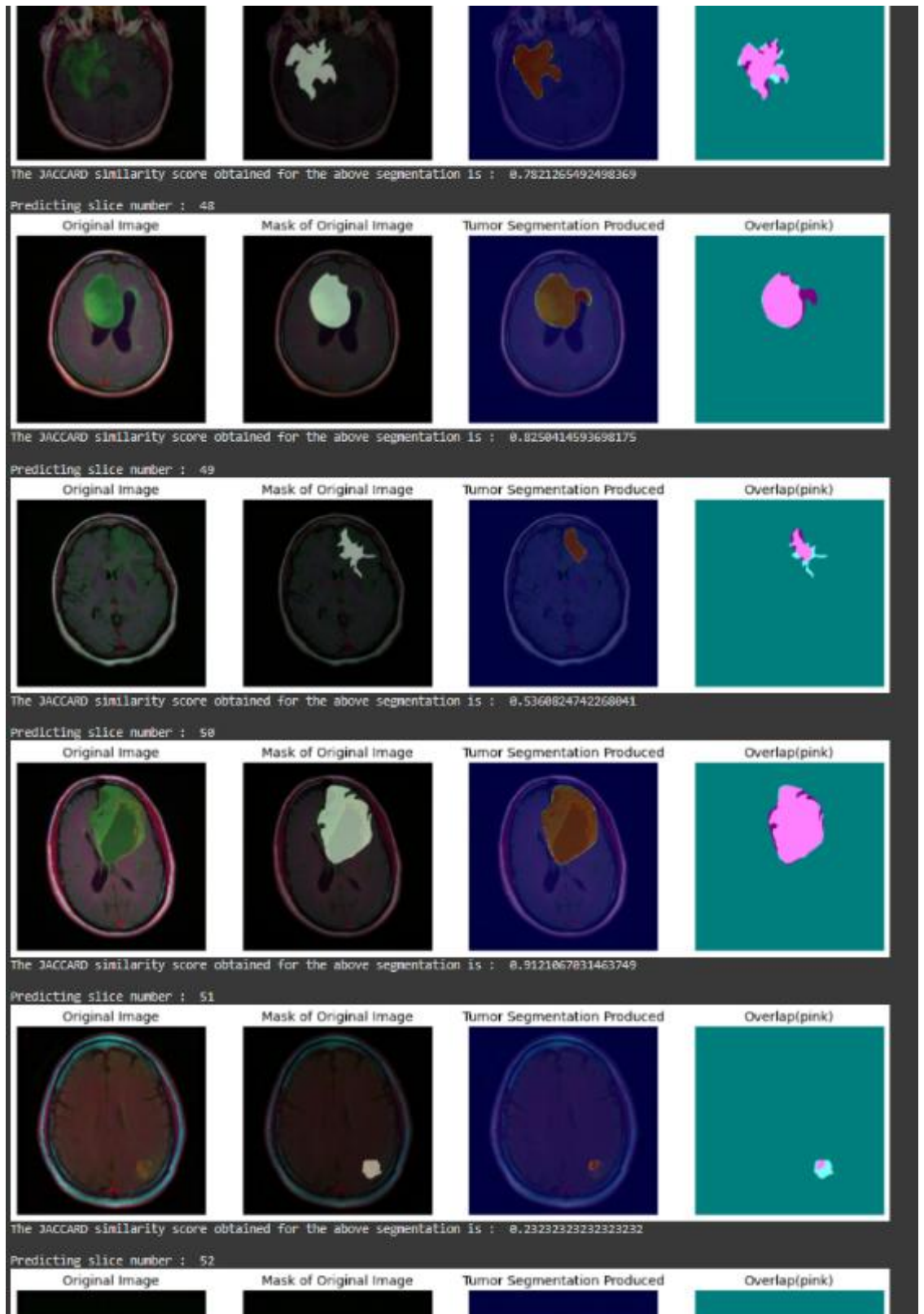


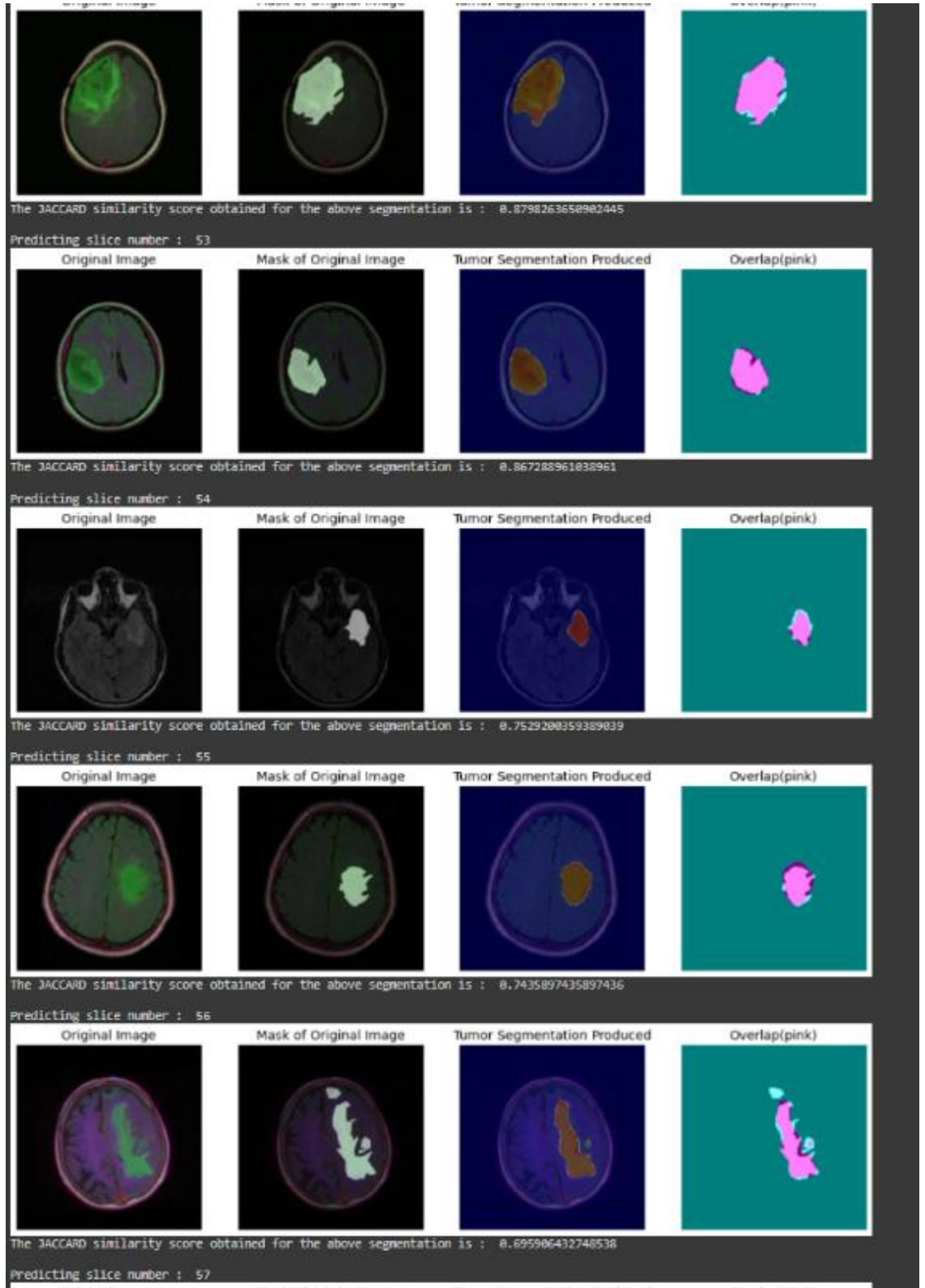


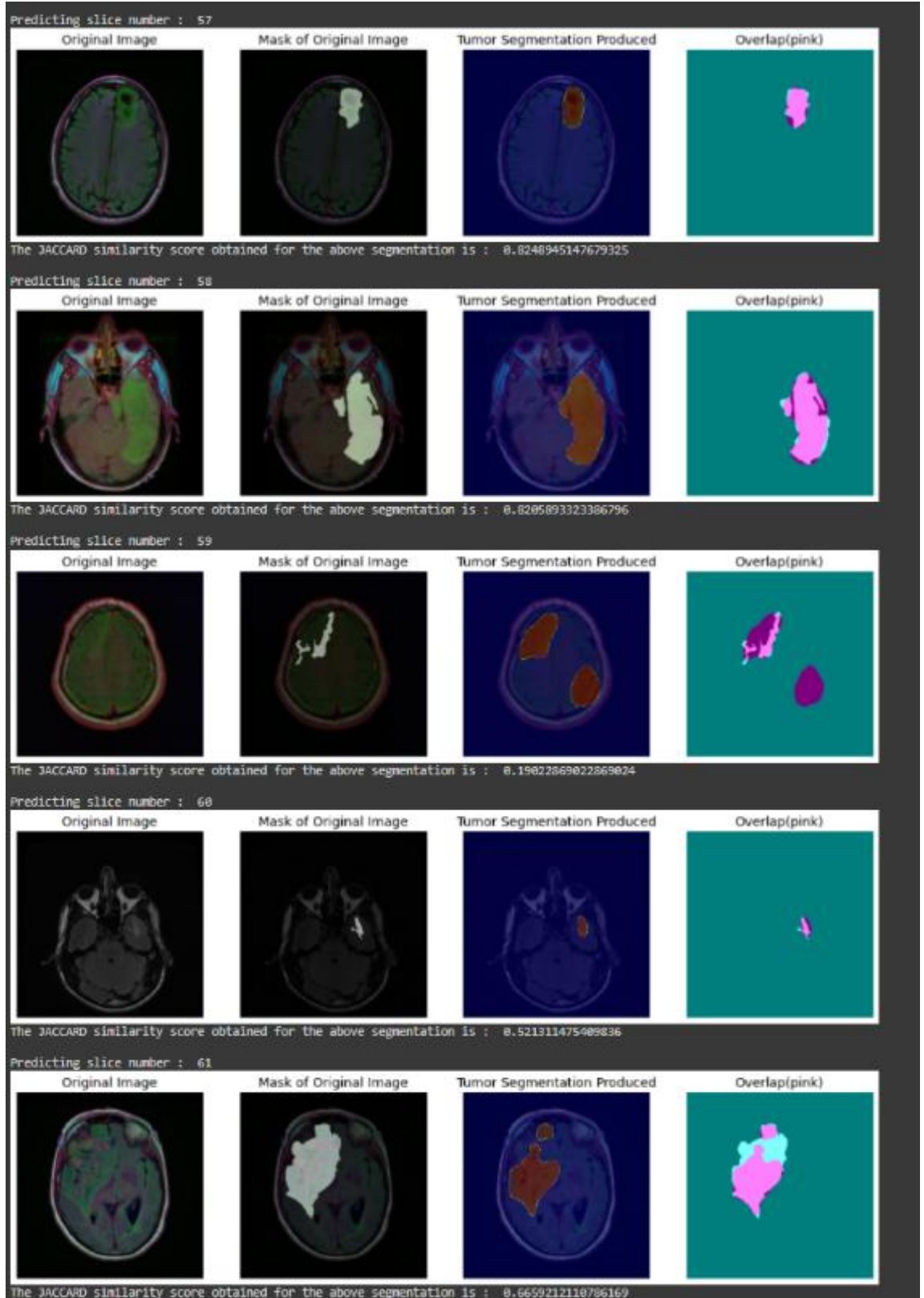


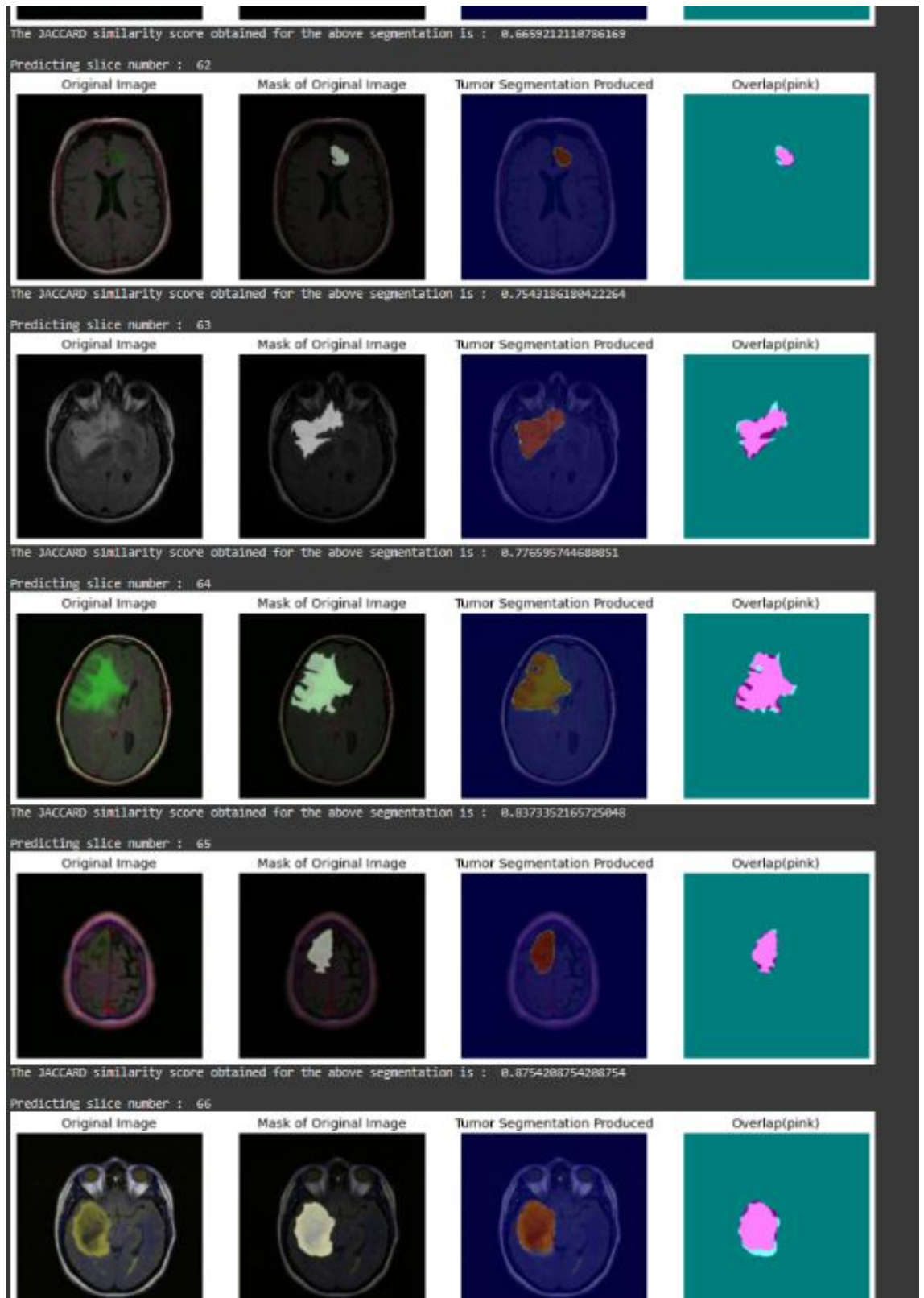


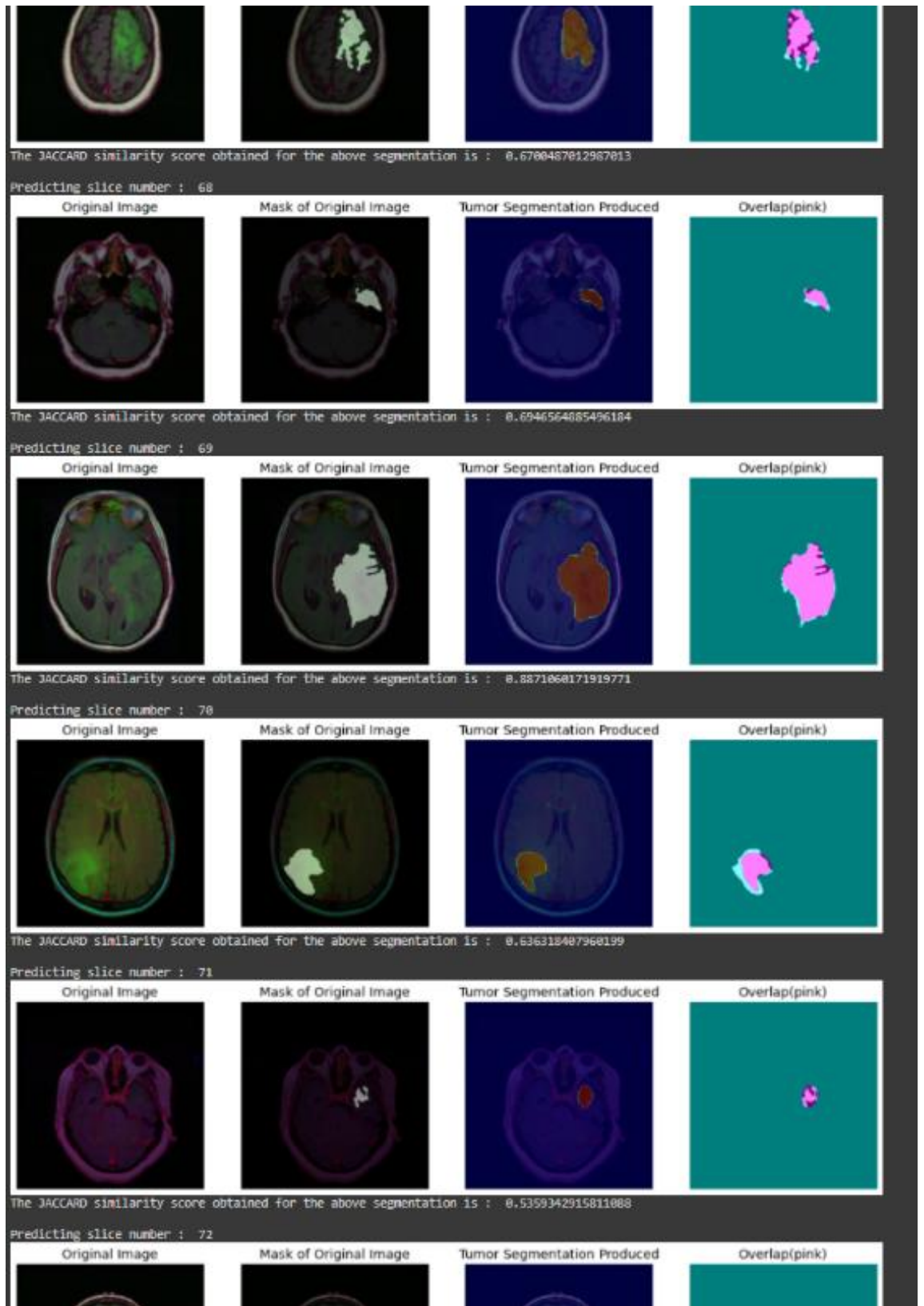


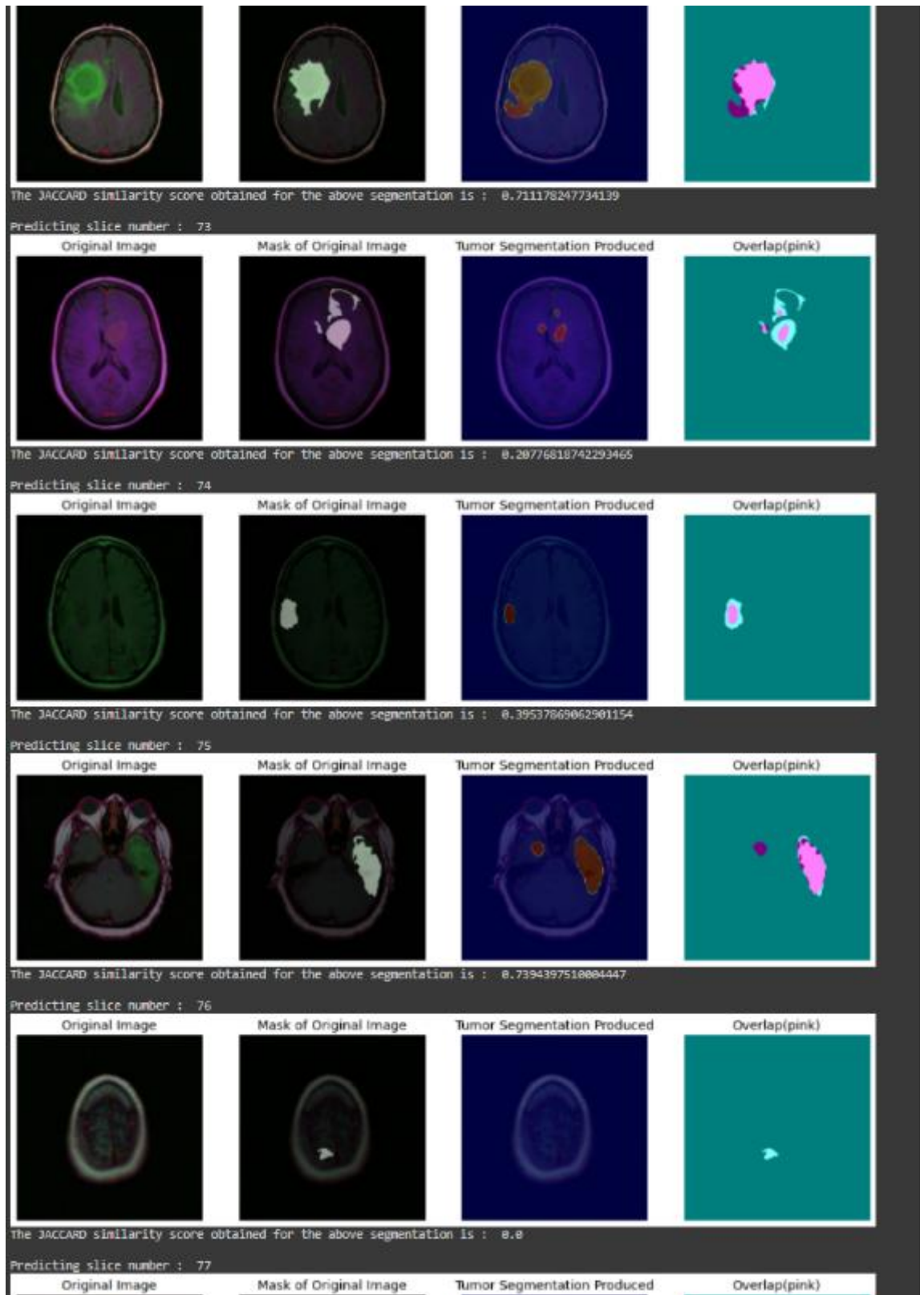


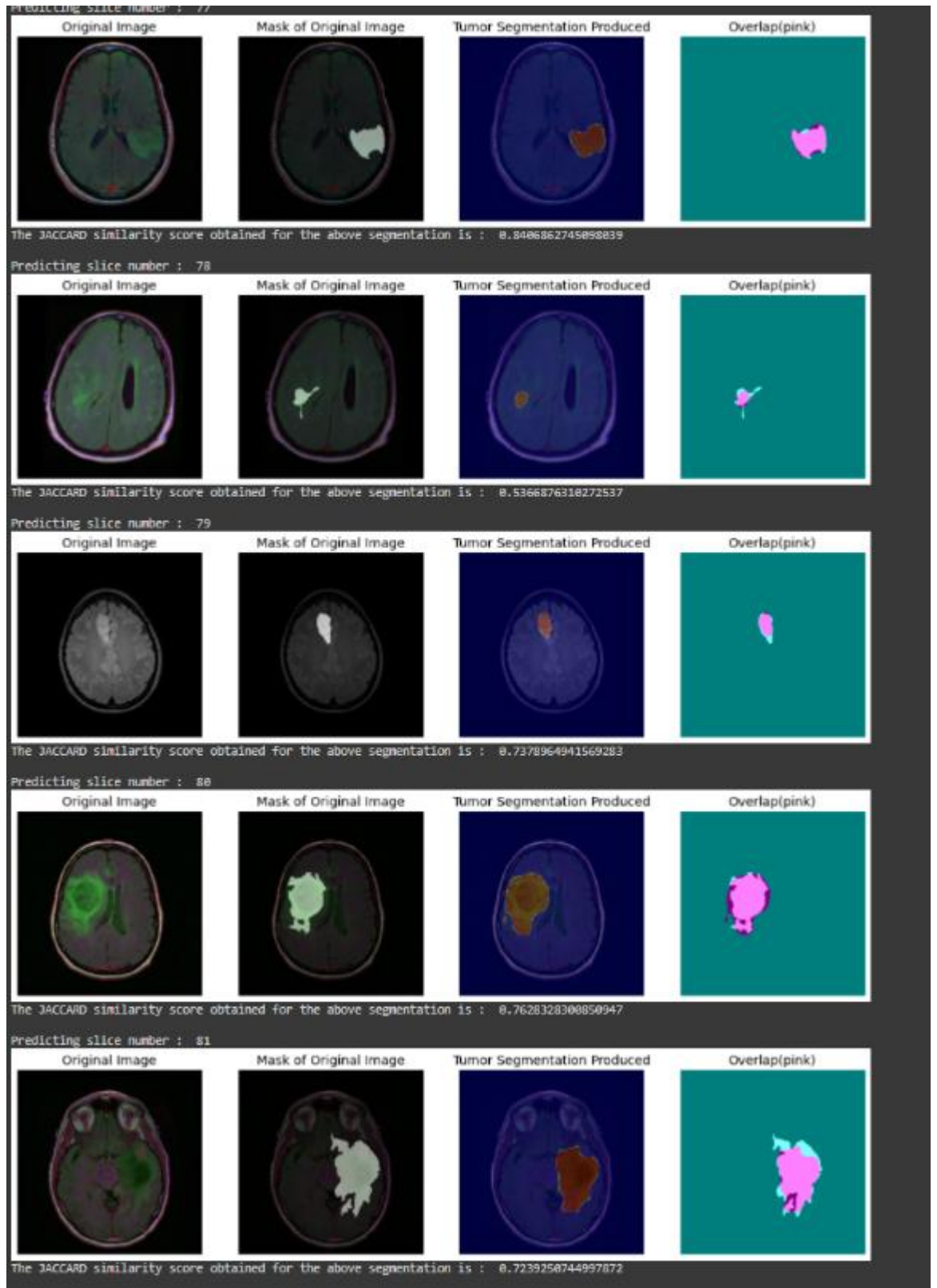


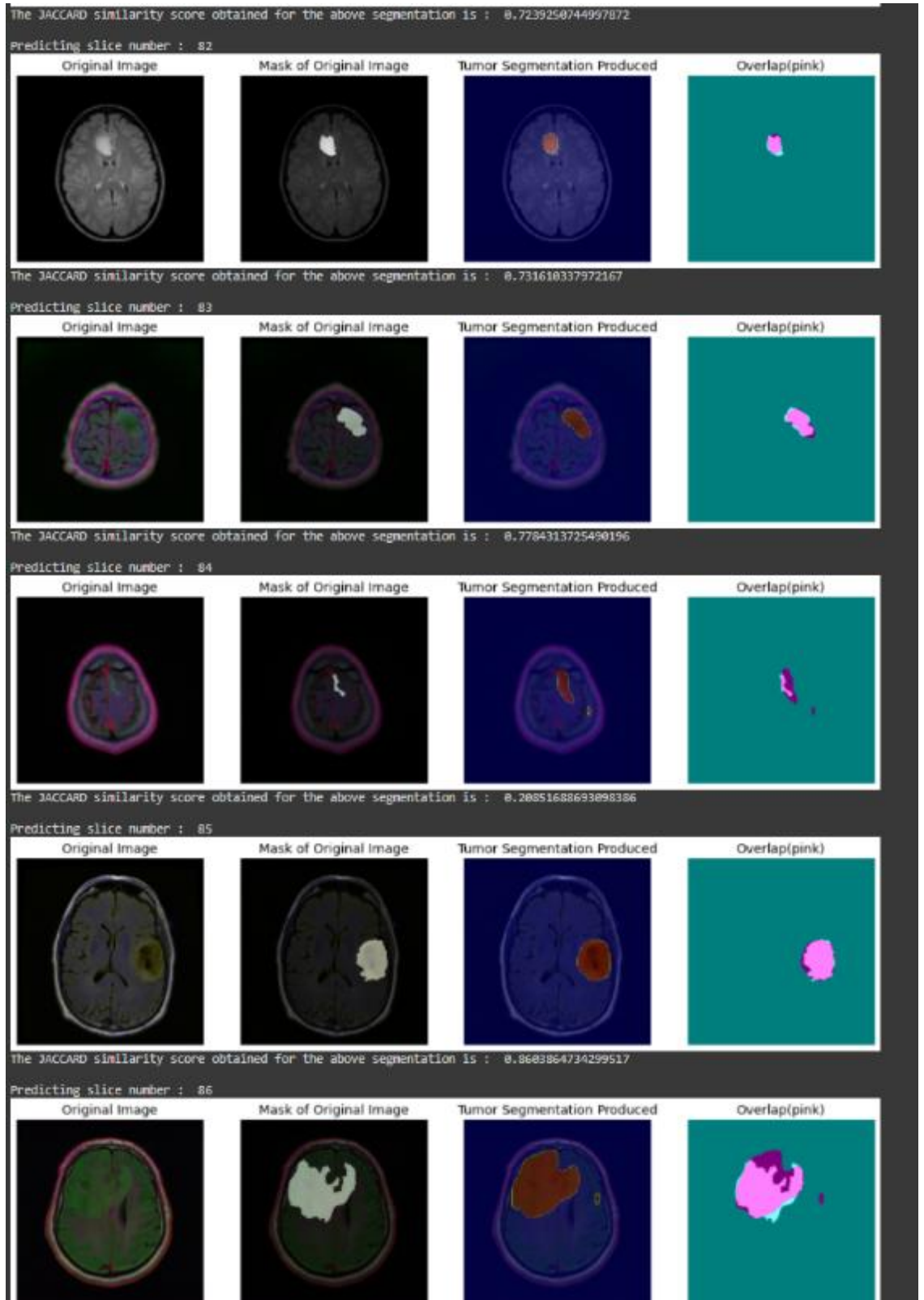


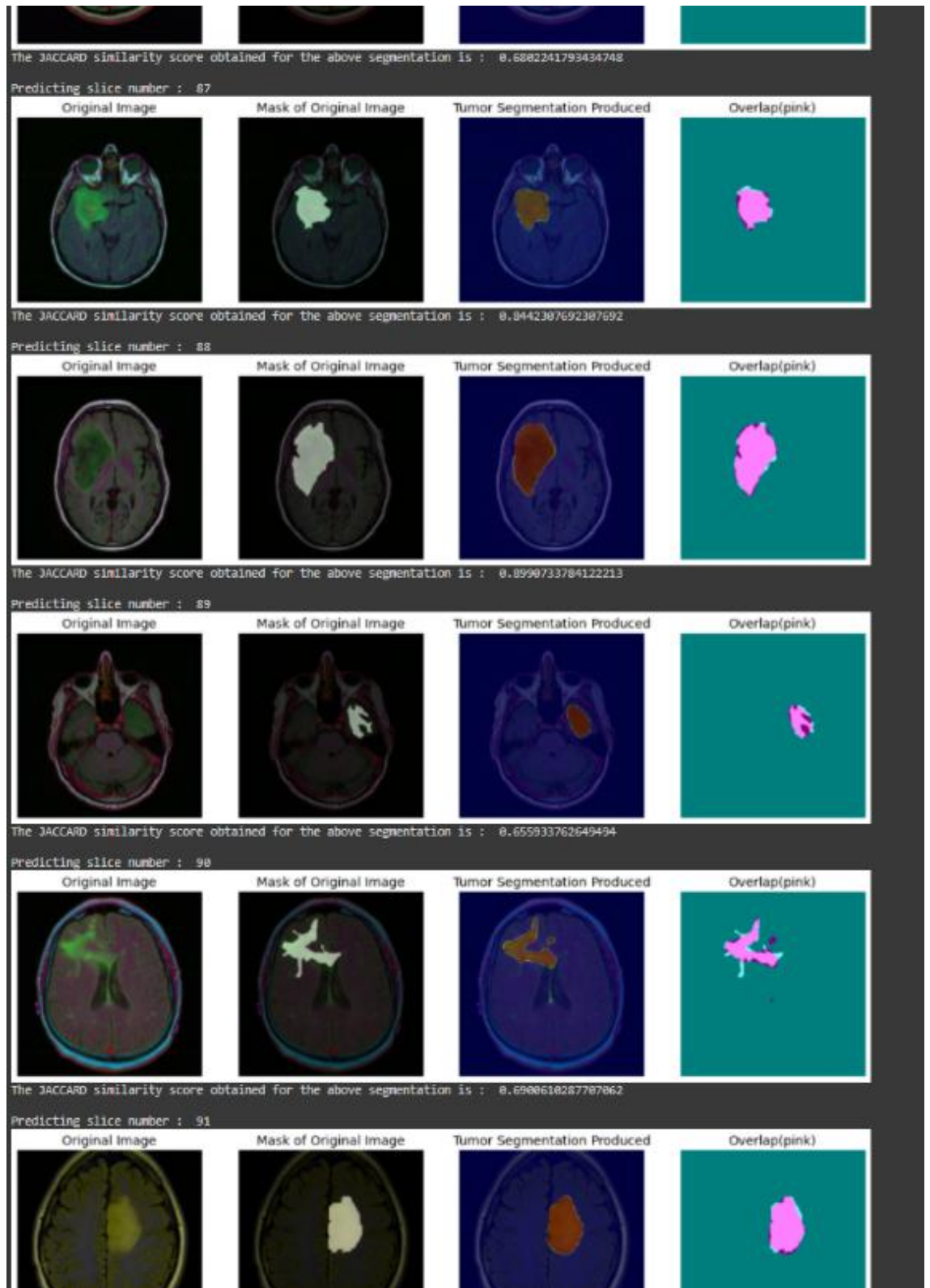


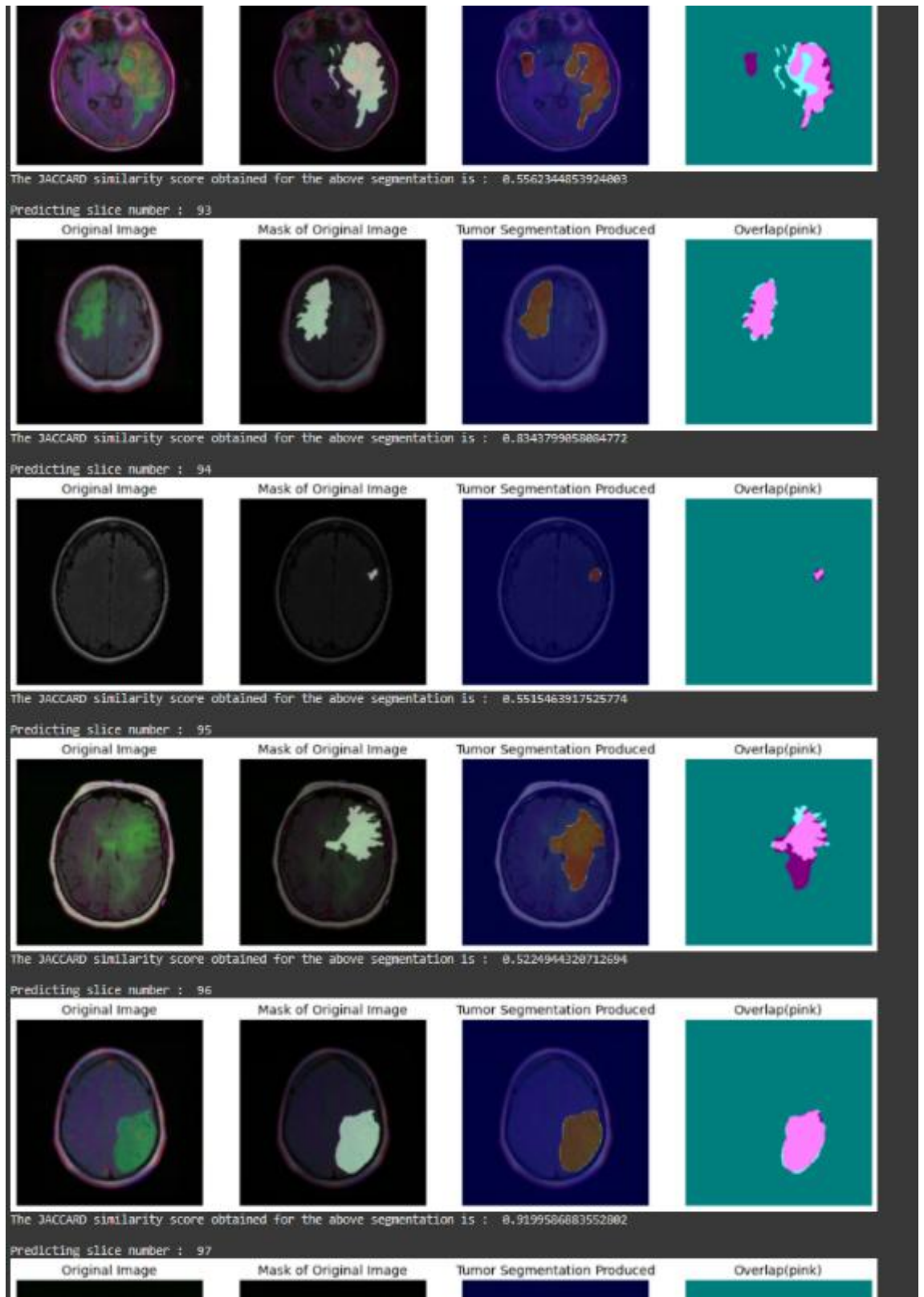


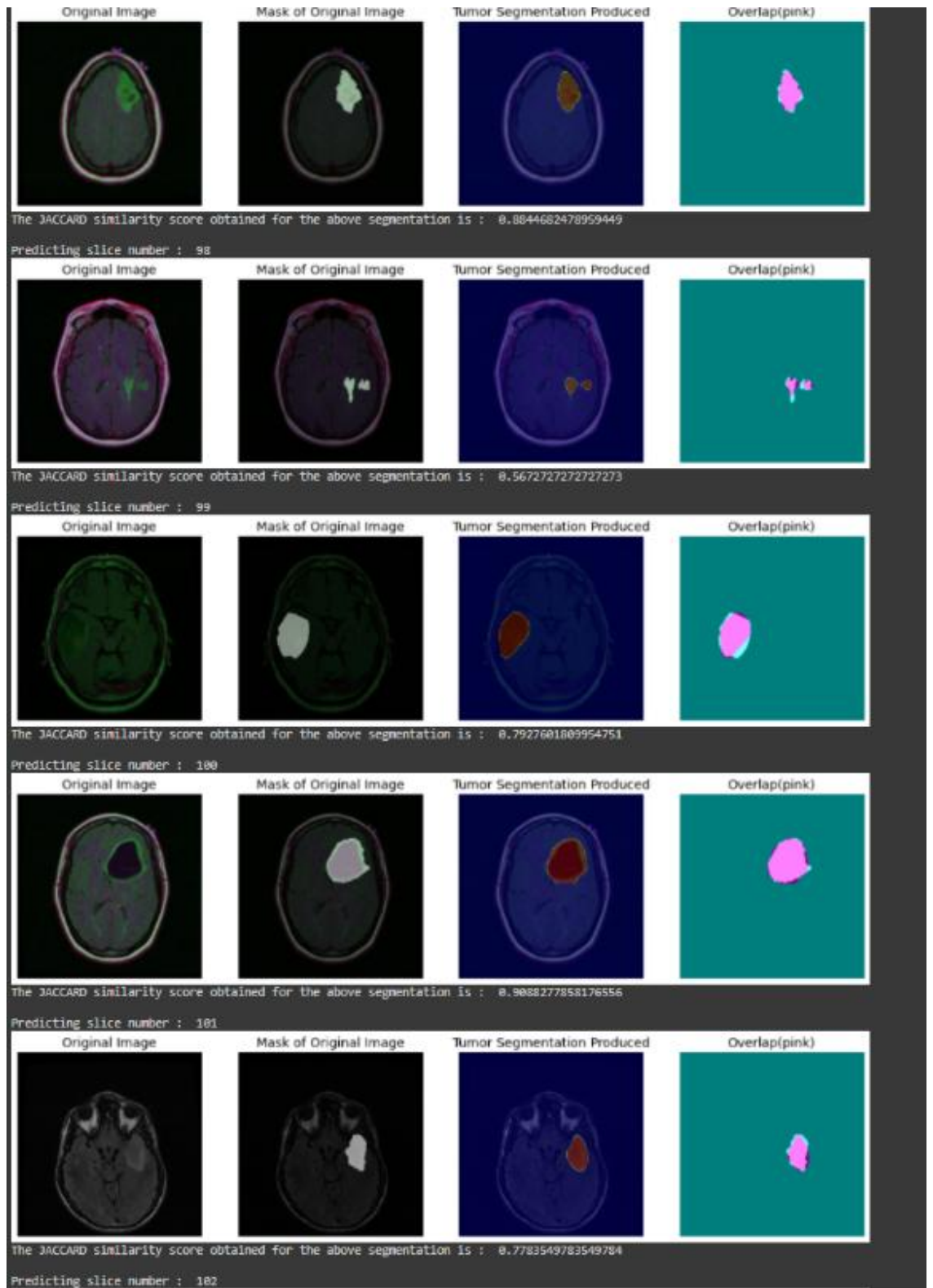


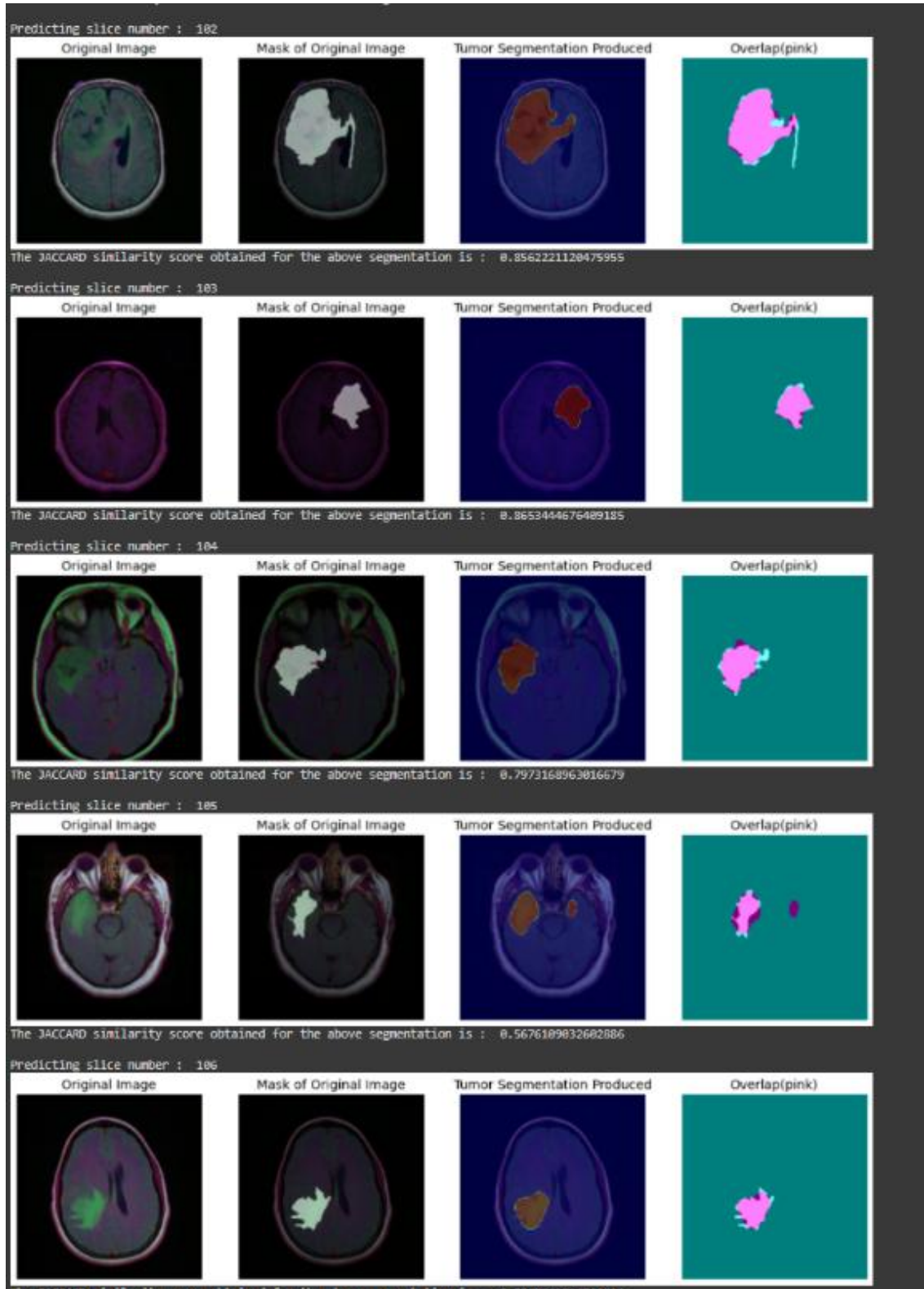


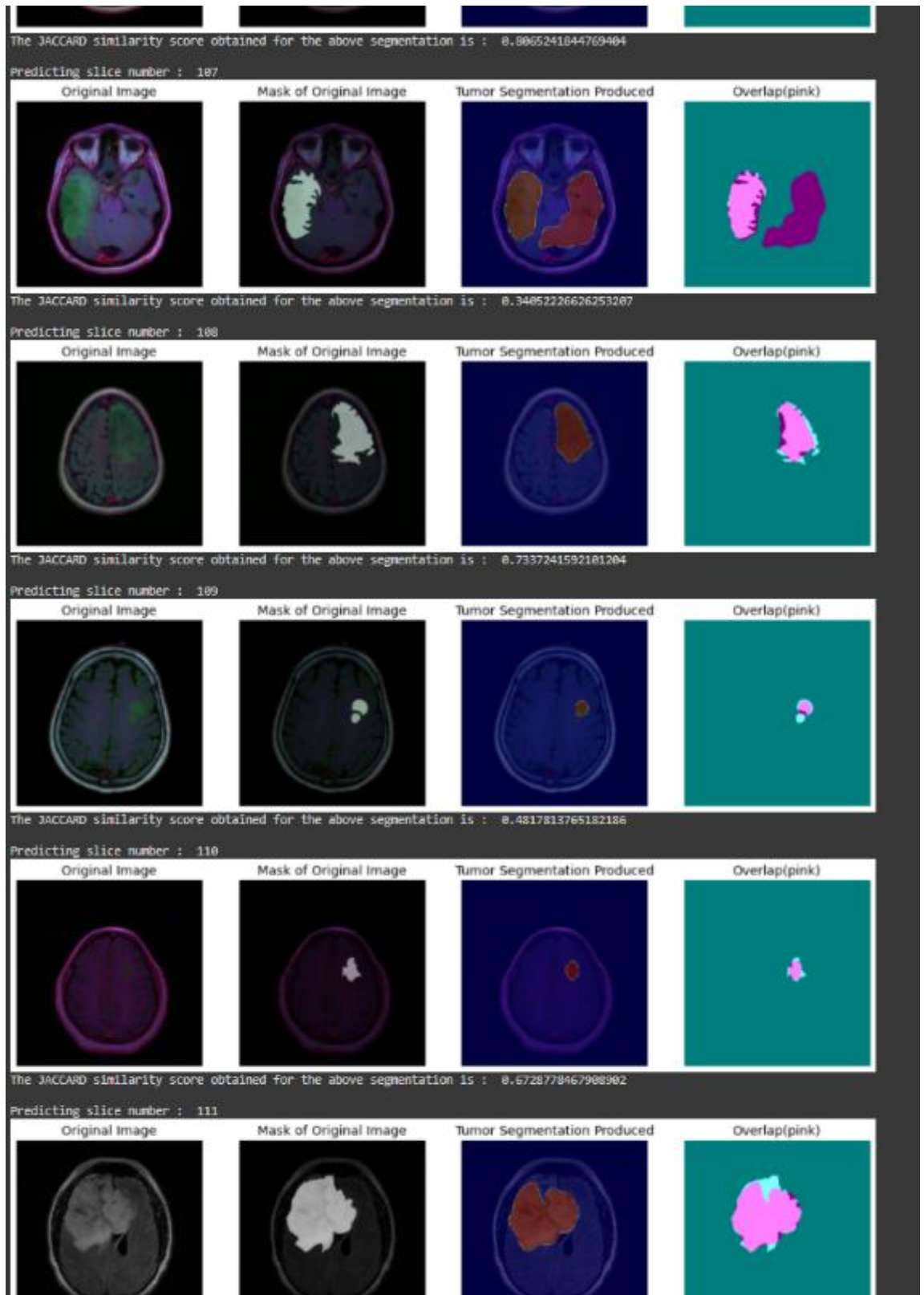


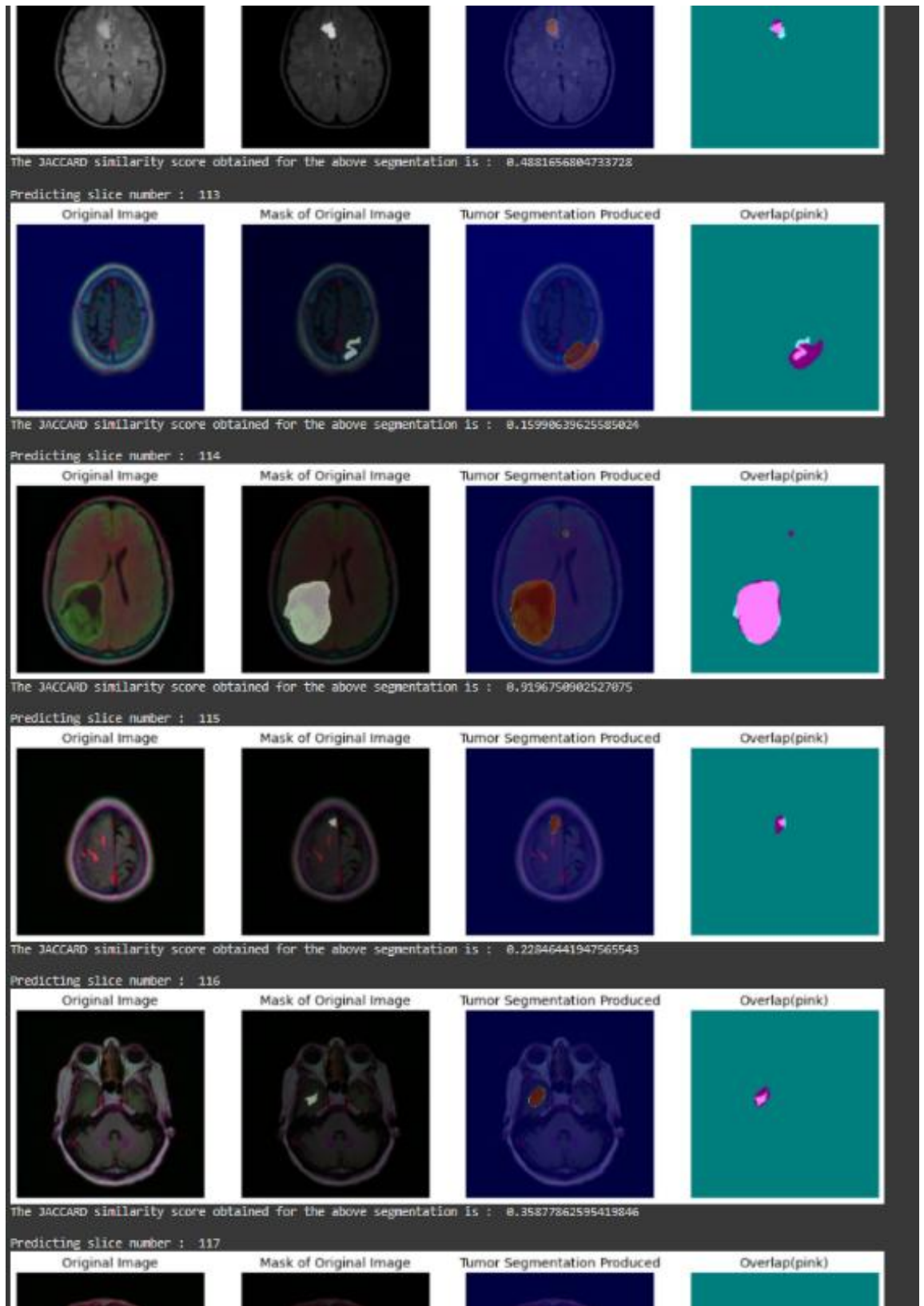


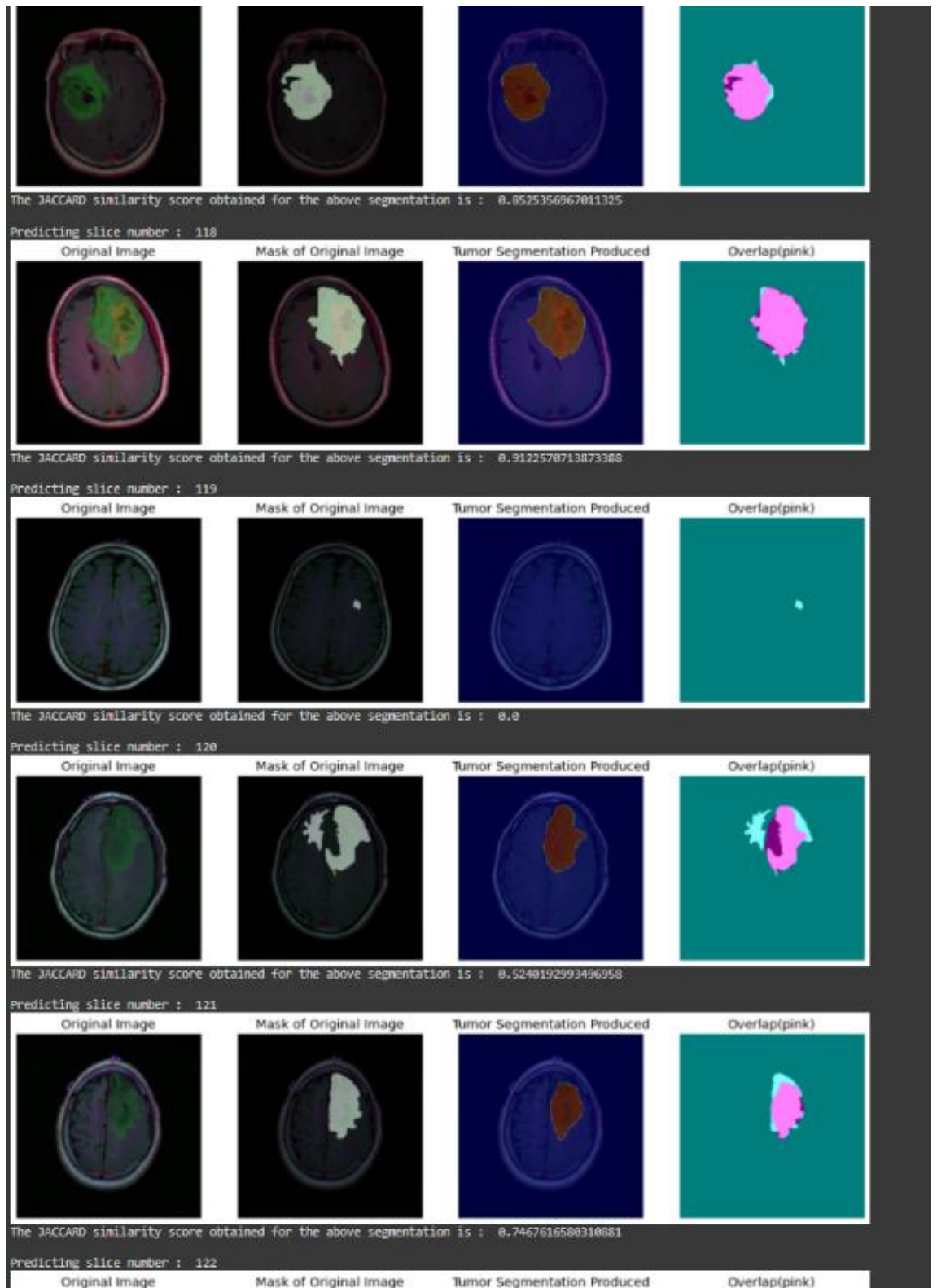


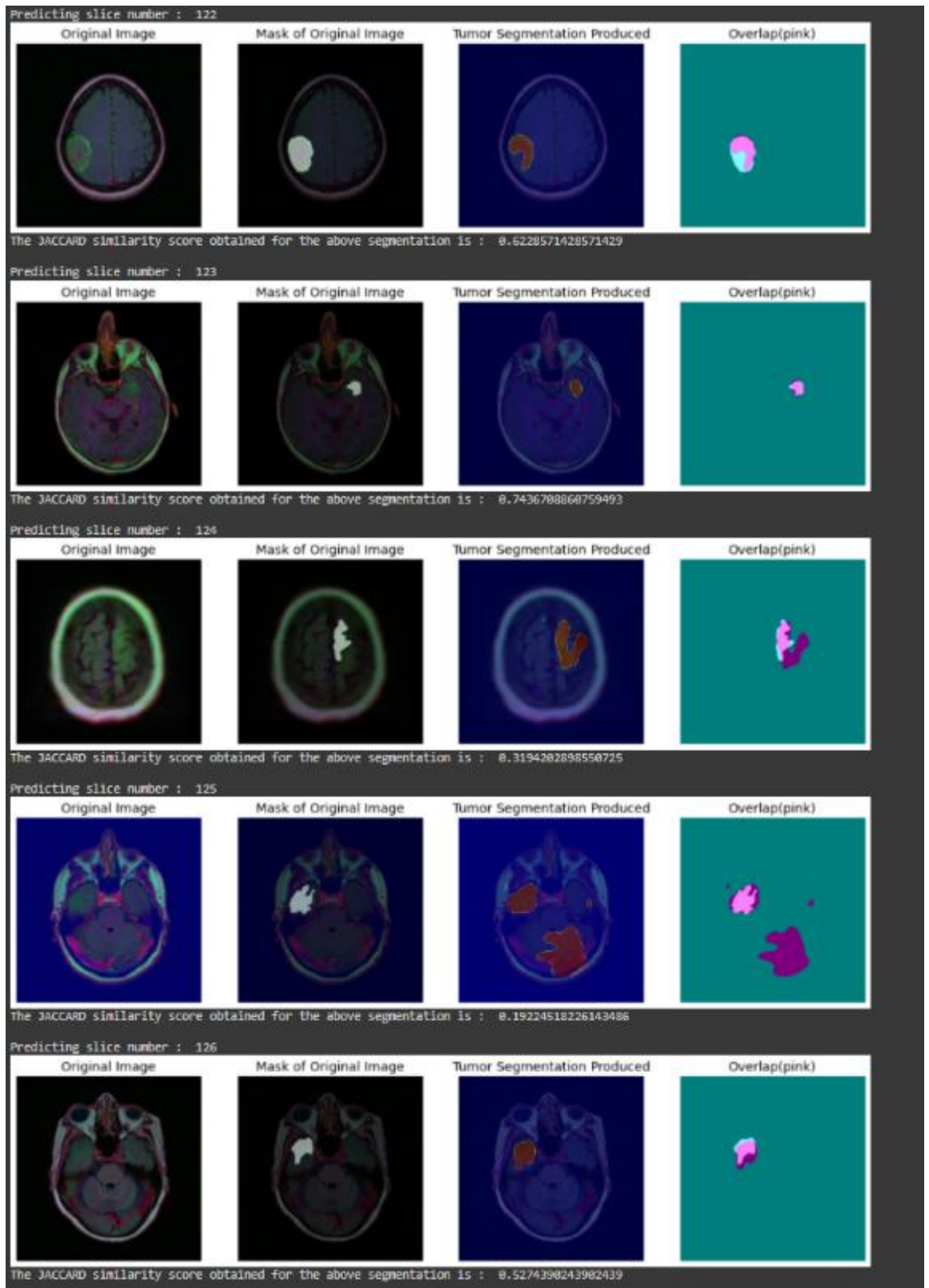


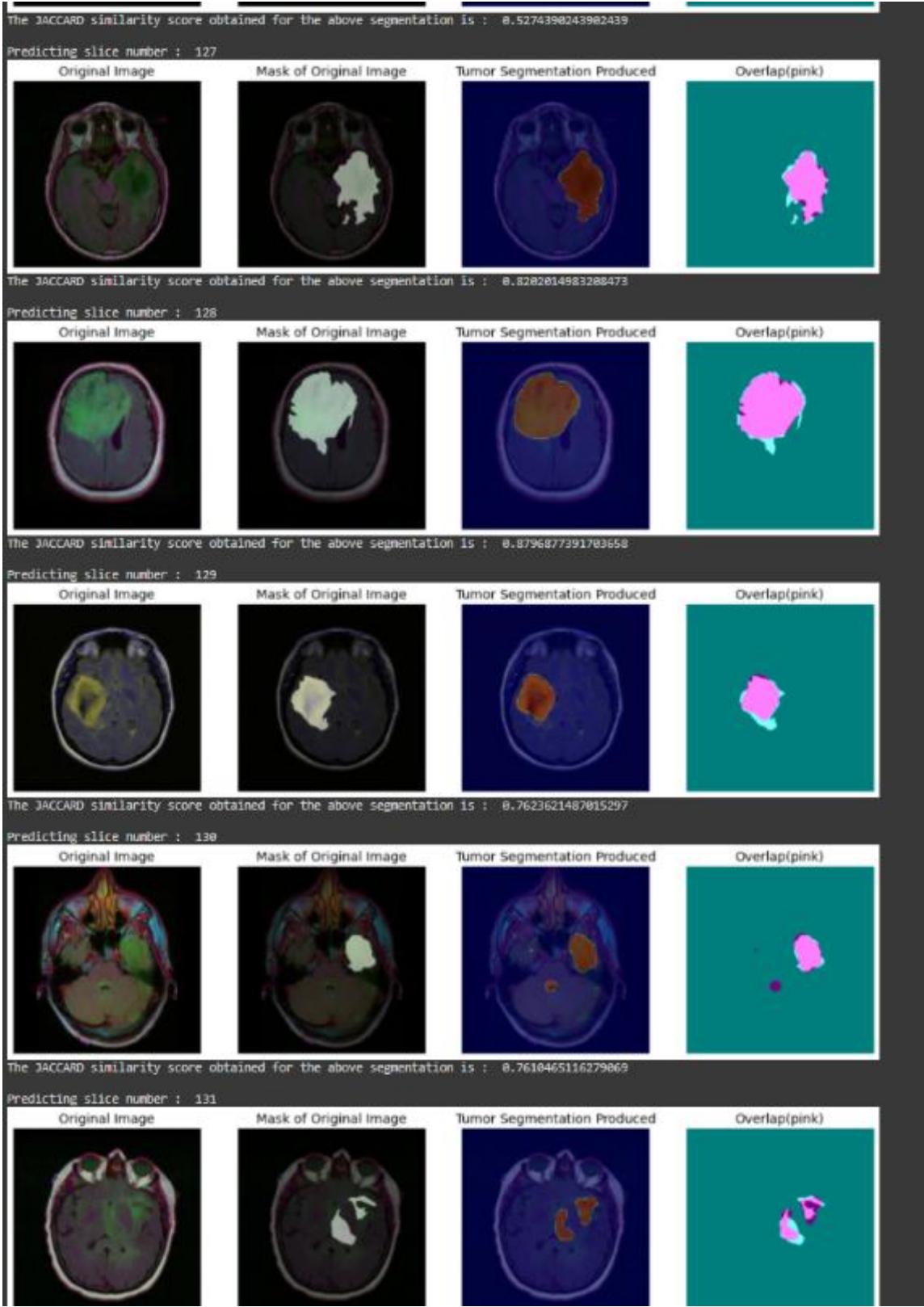


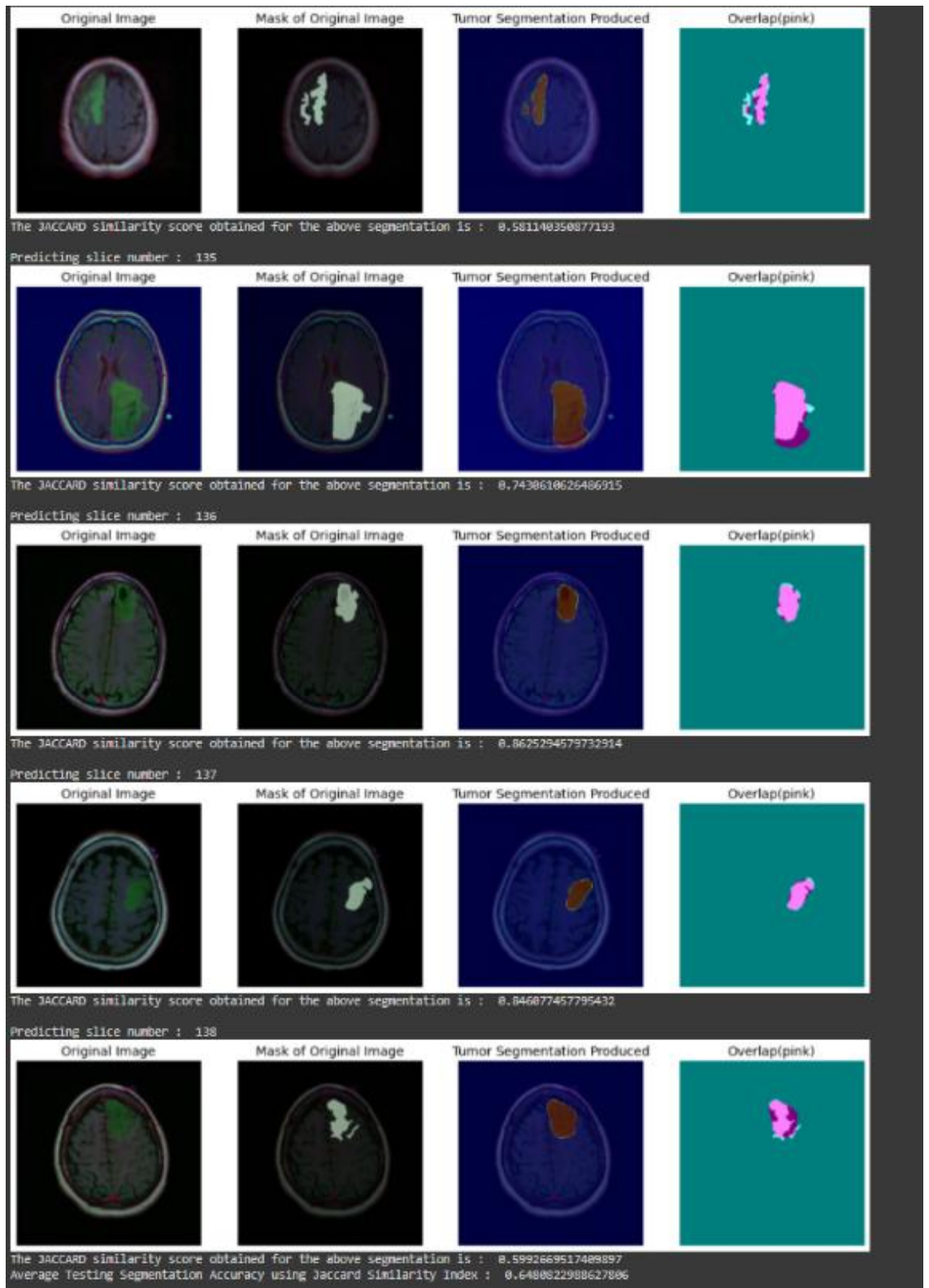




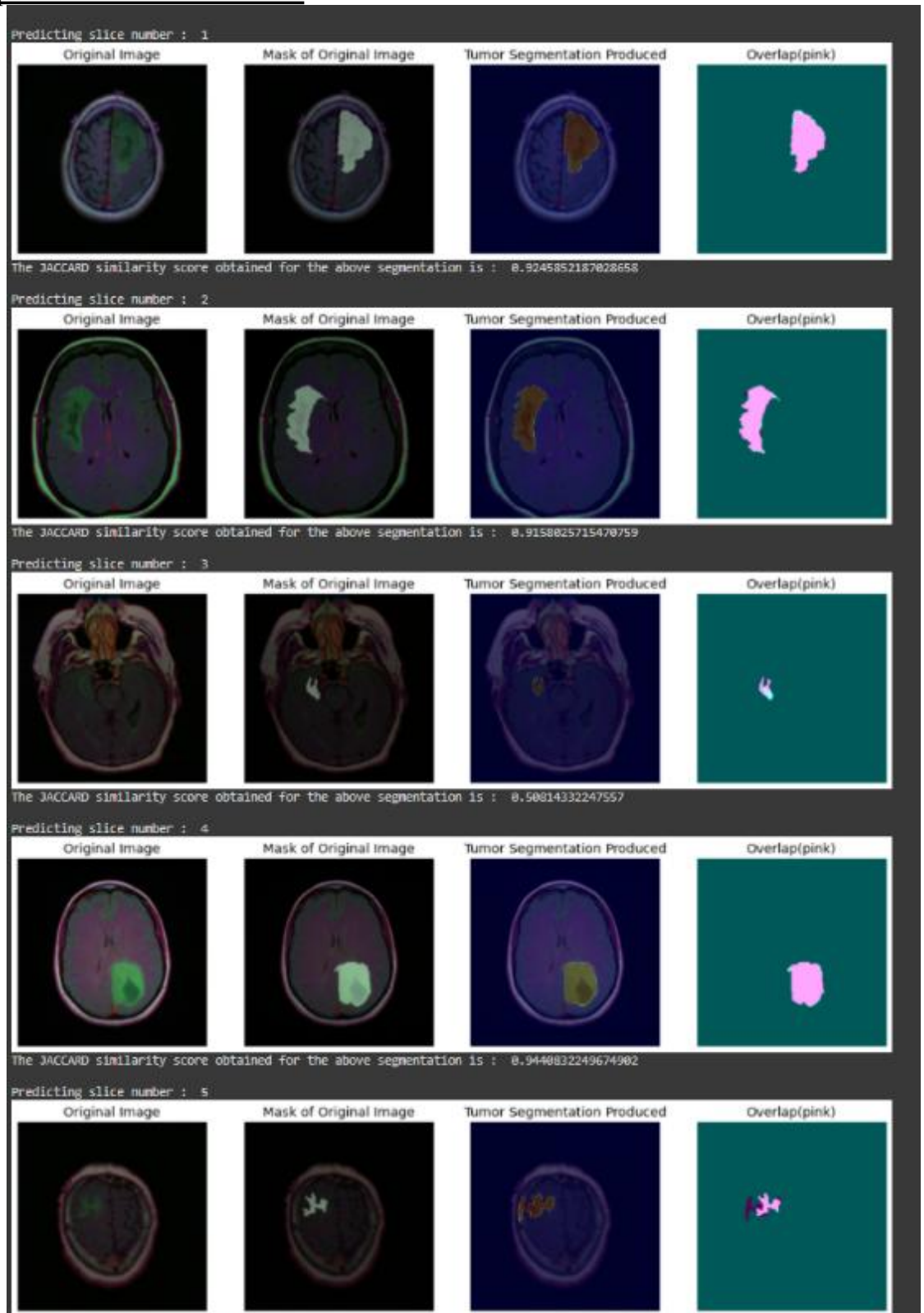


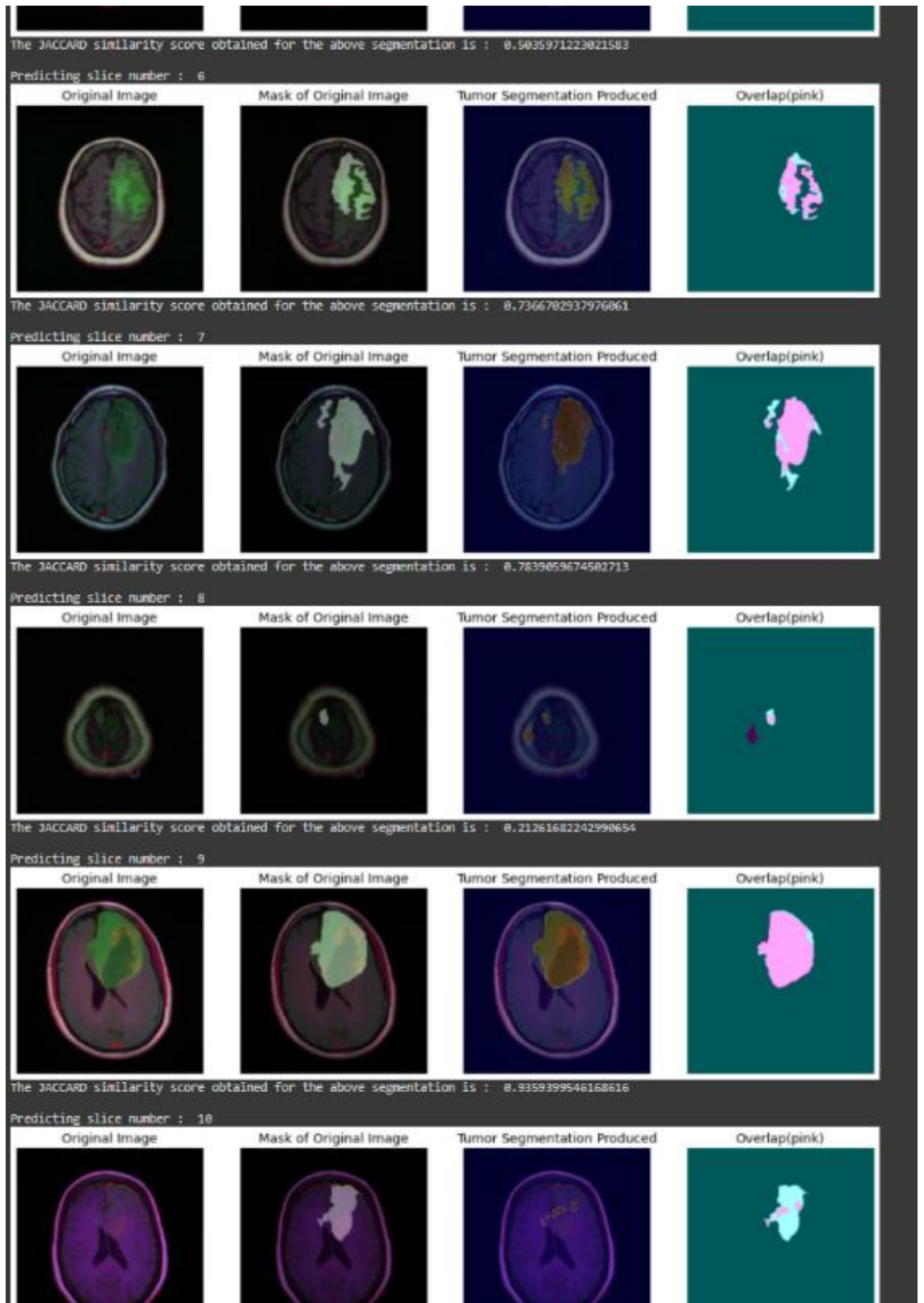


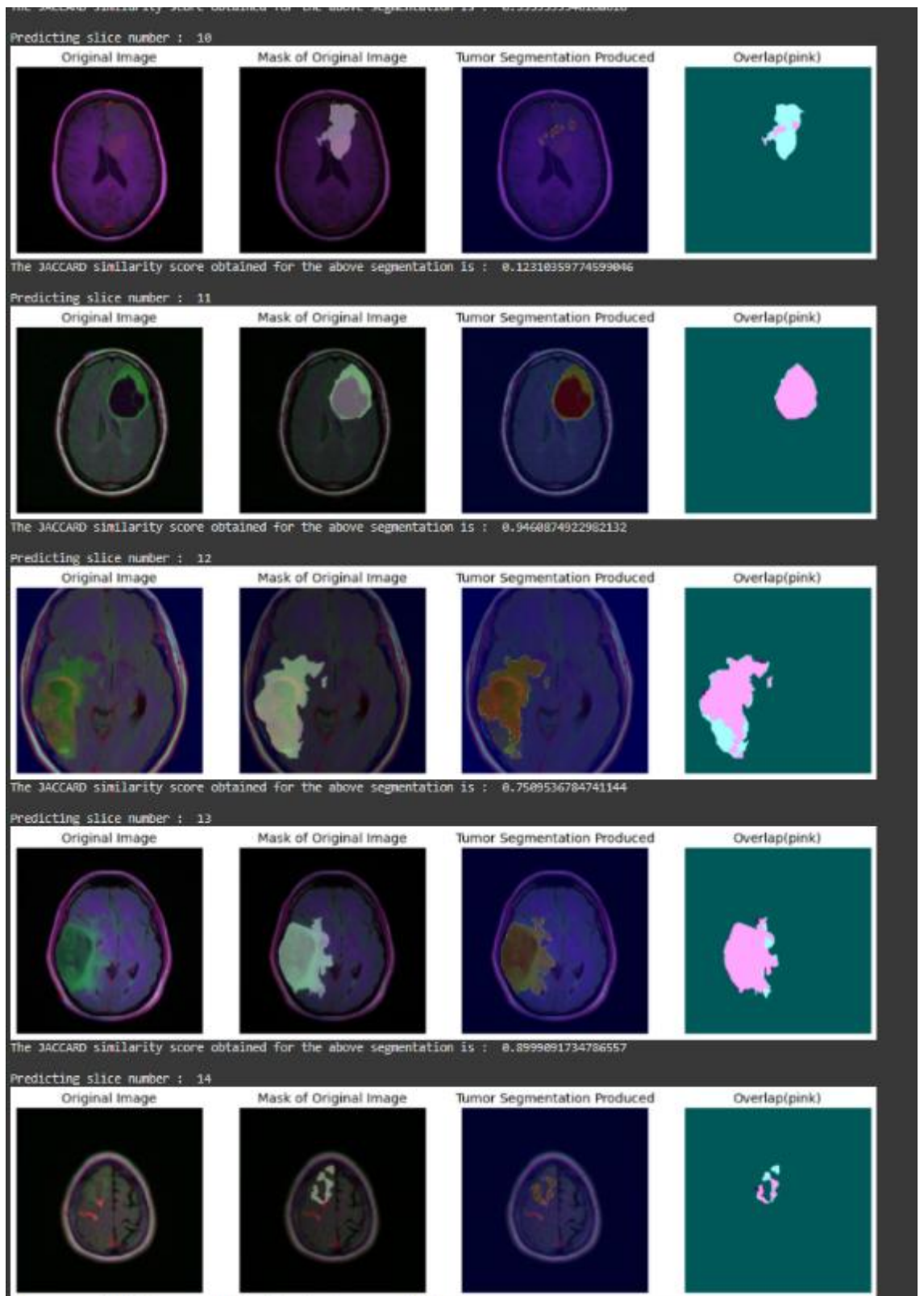


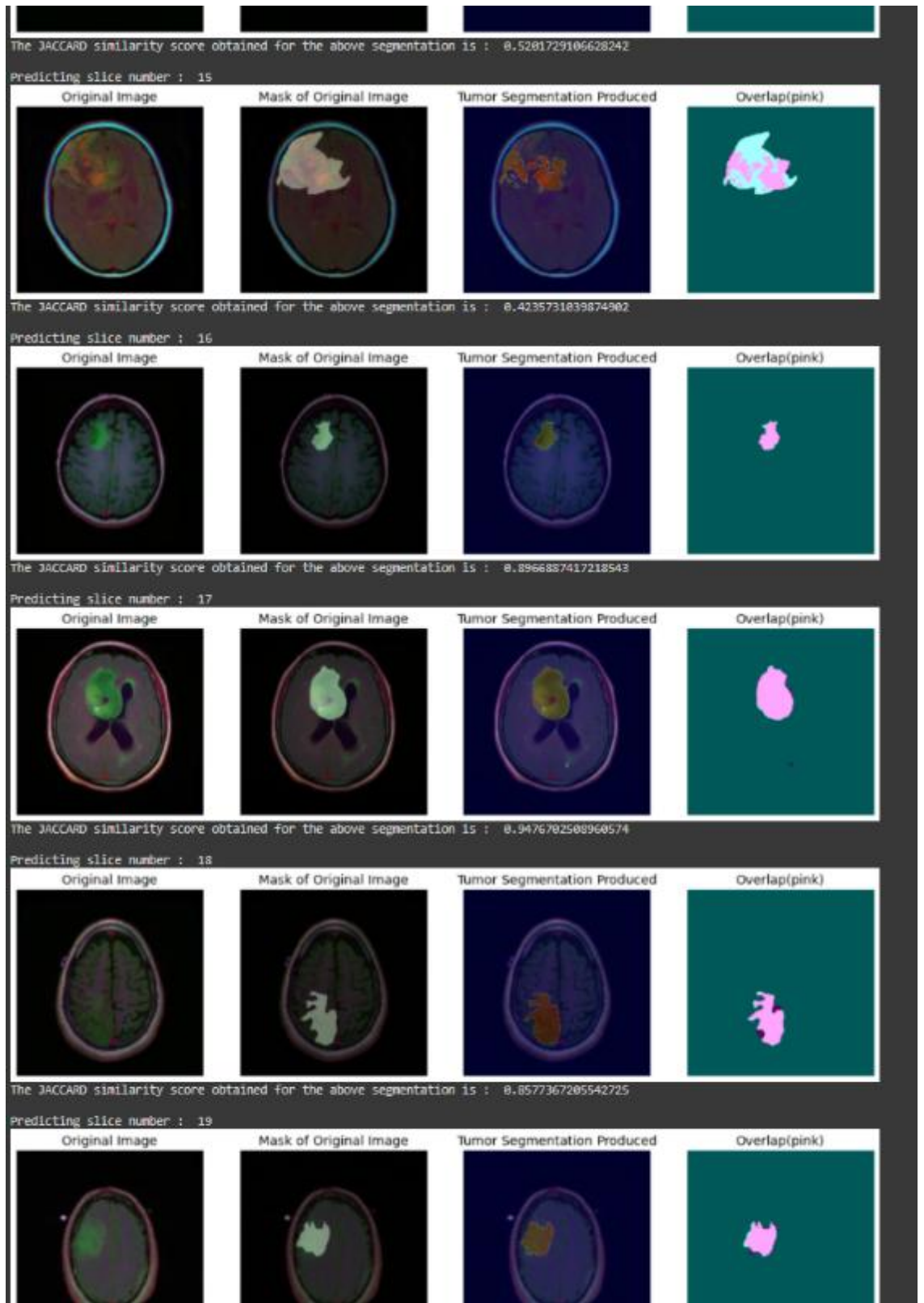


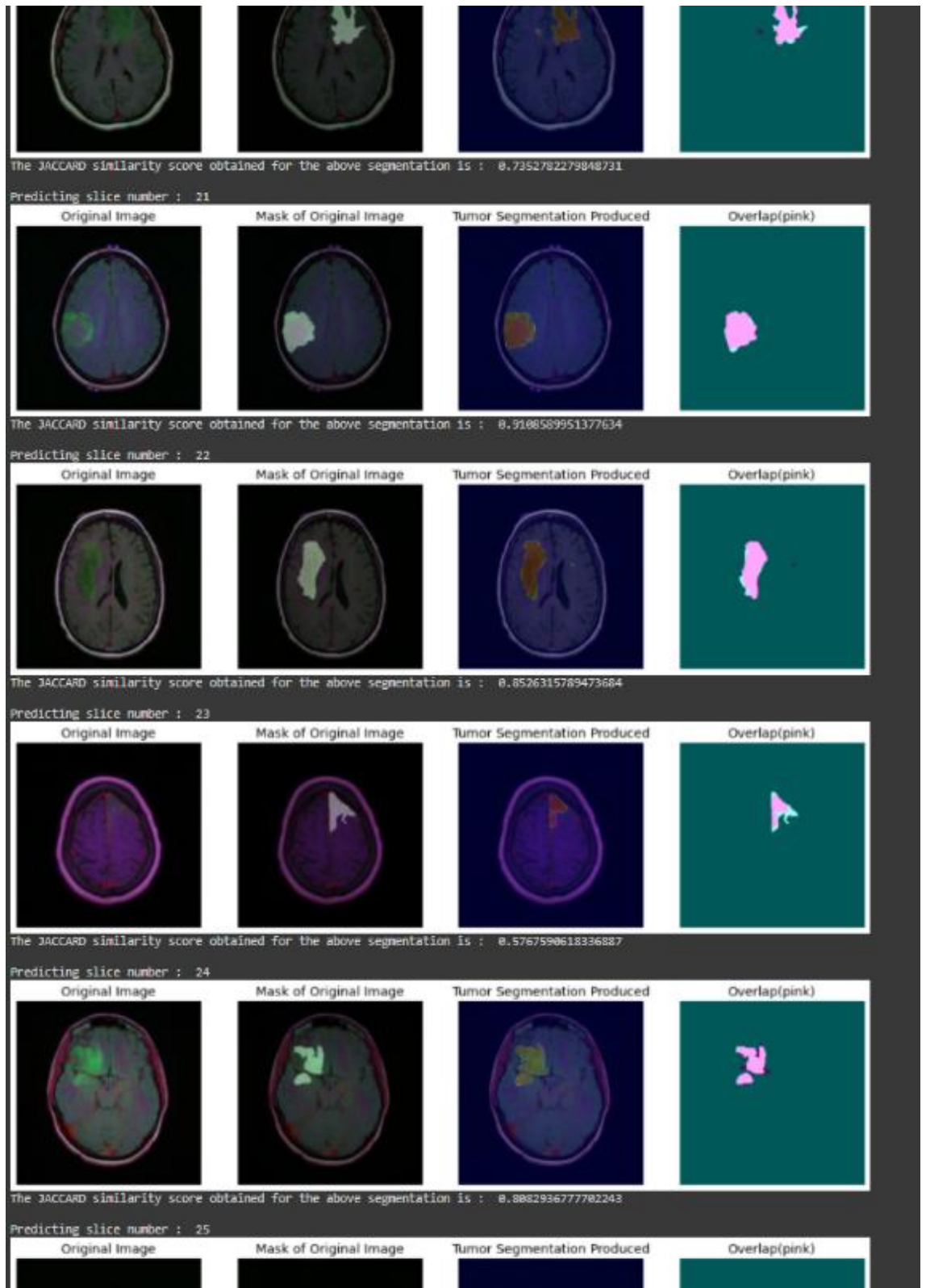
2. Experimental Model No. 6

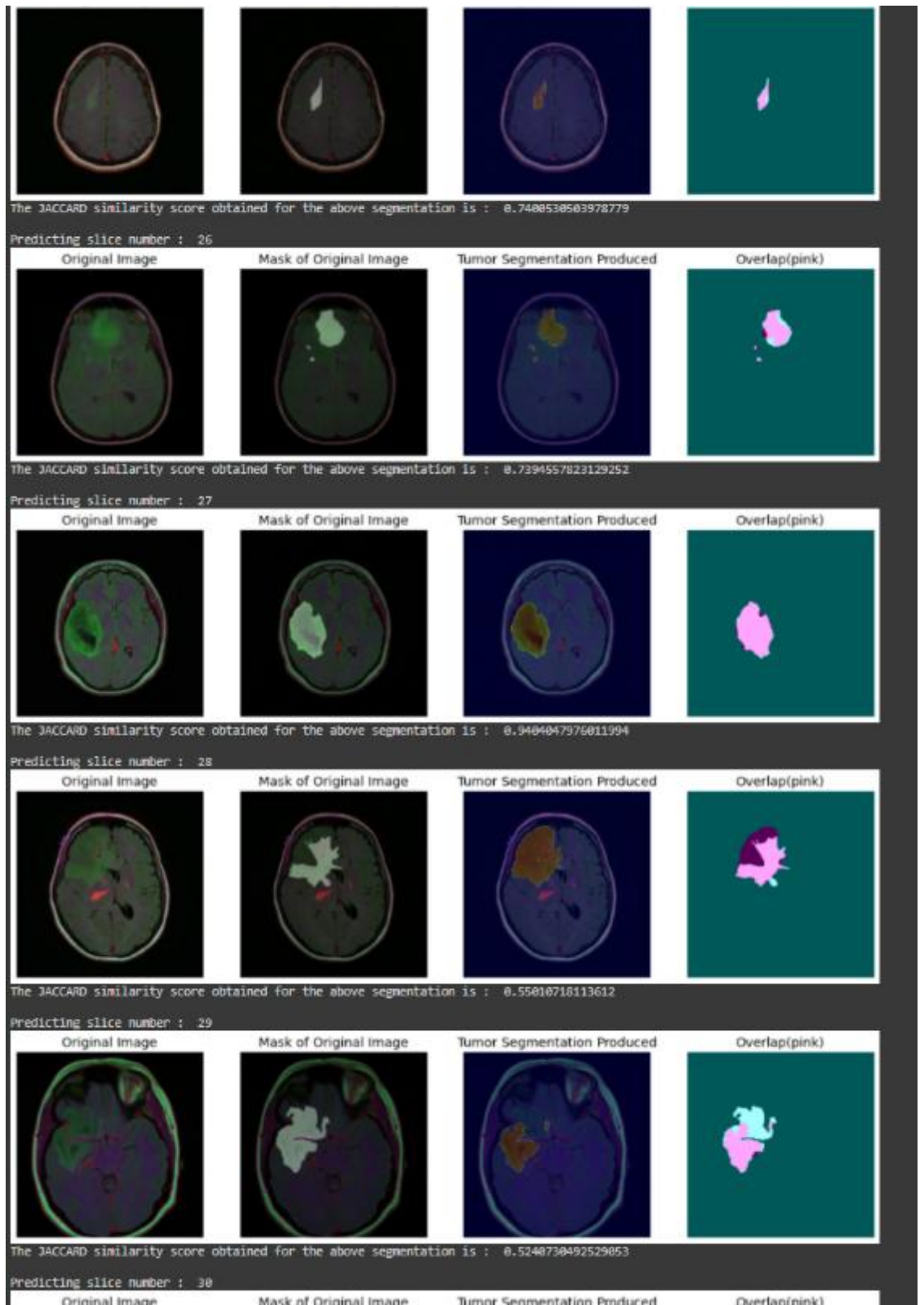


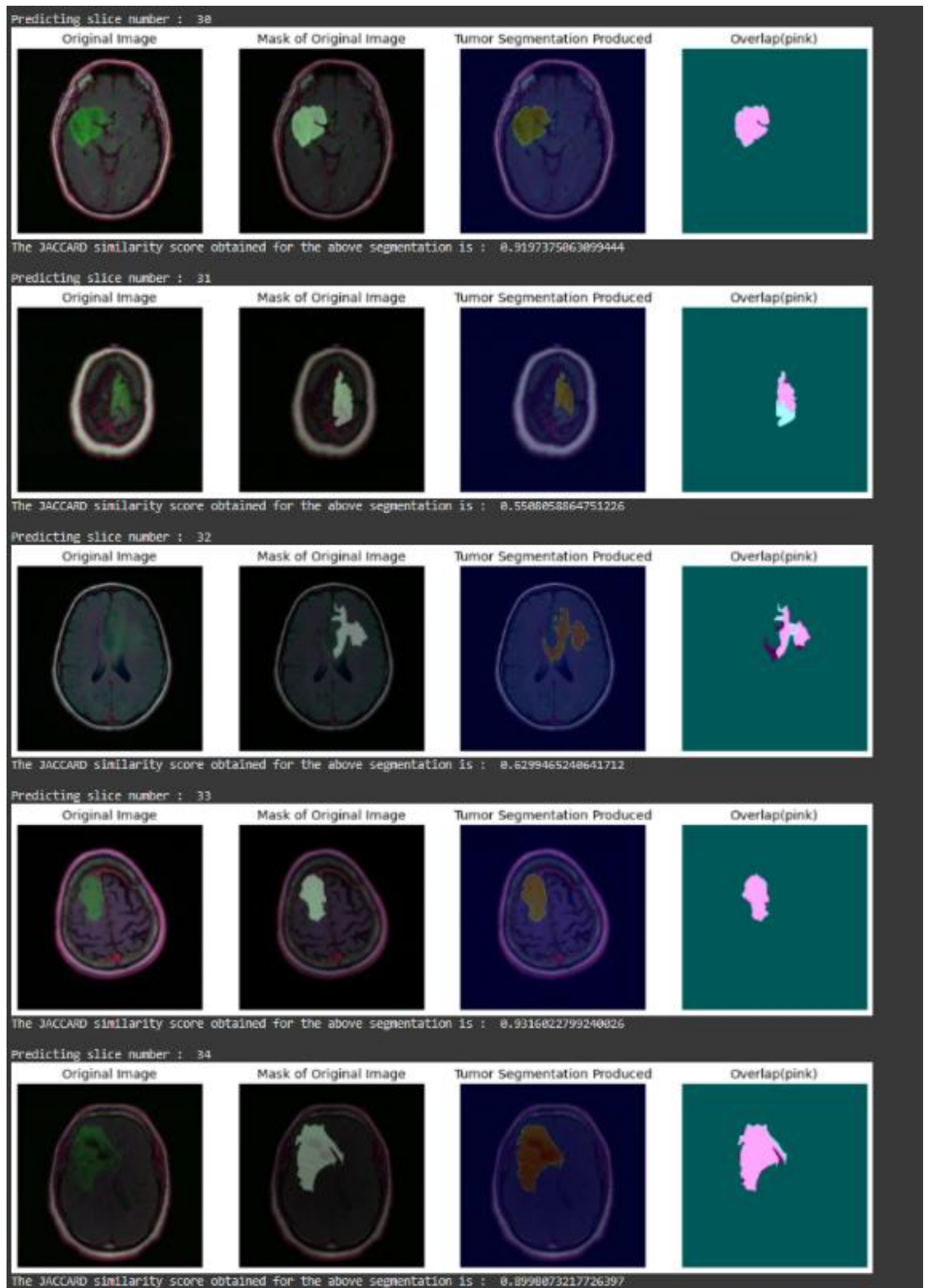


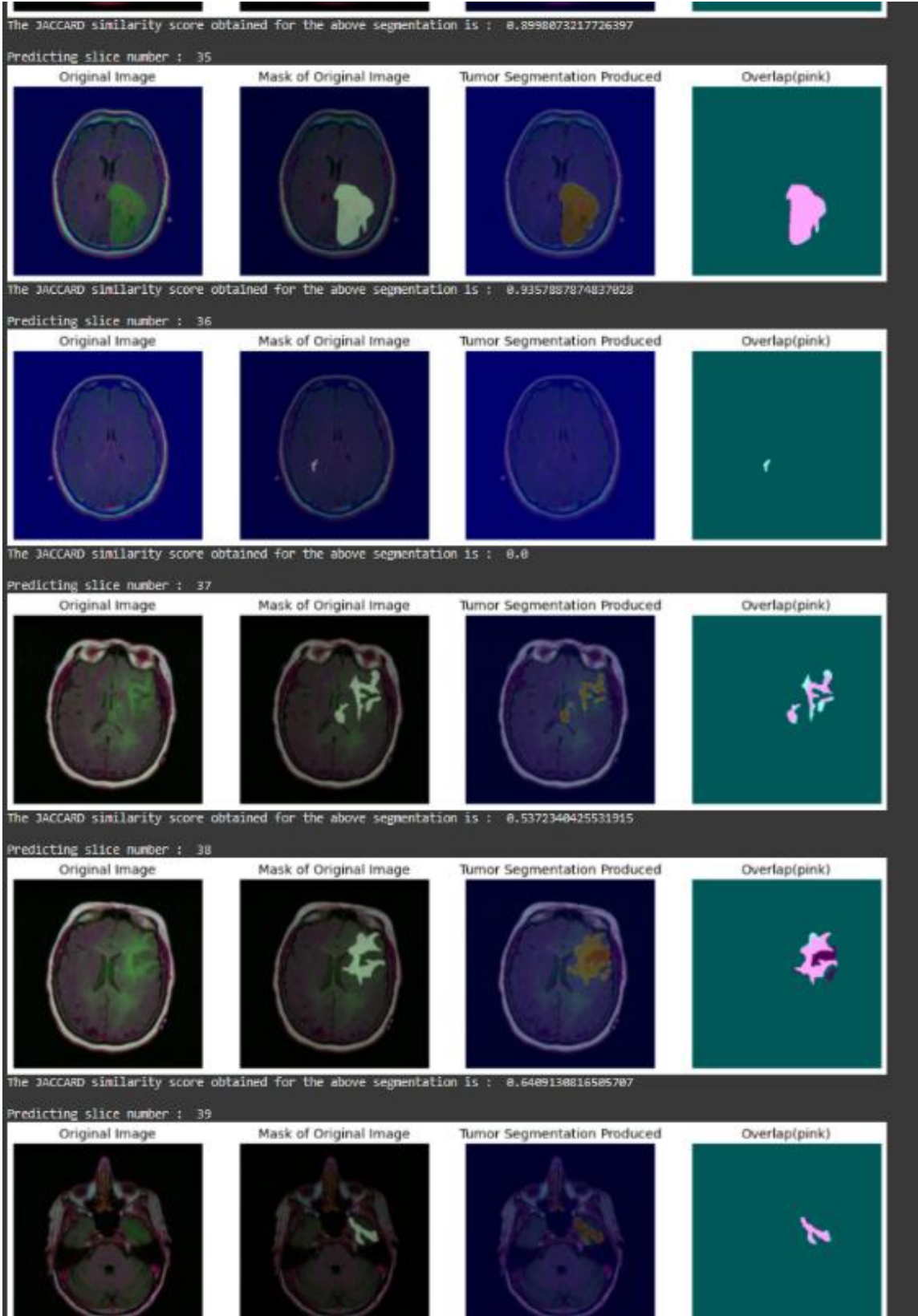


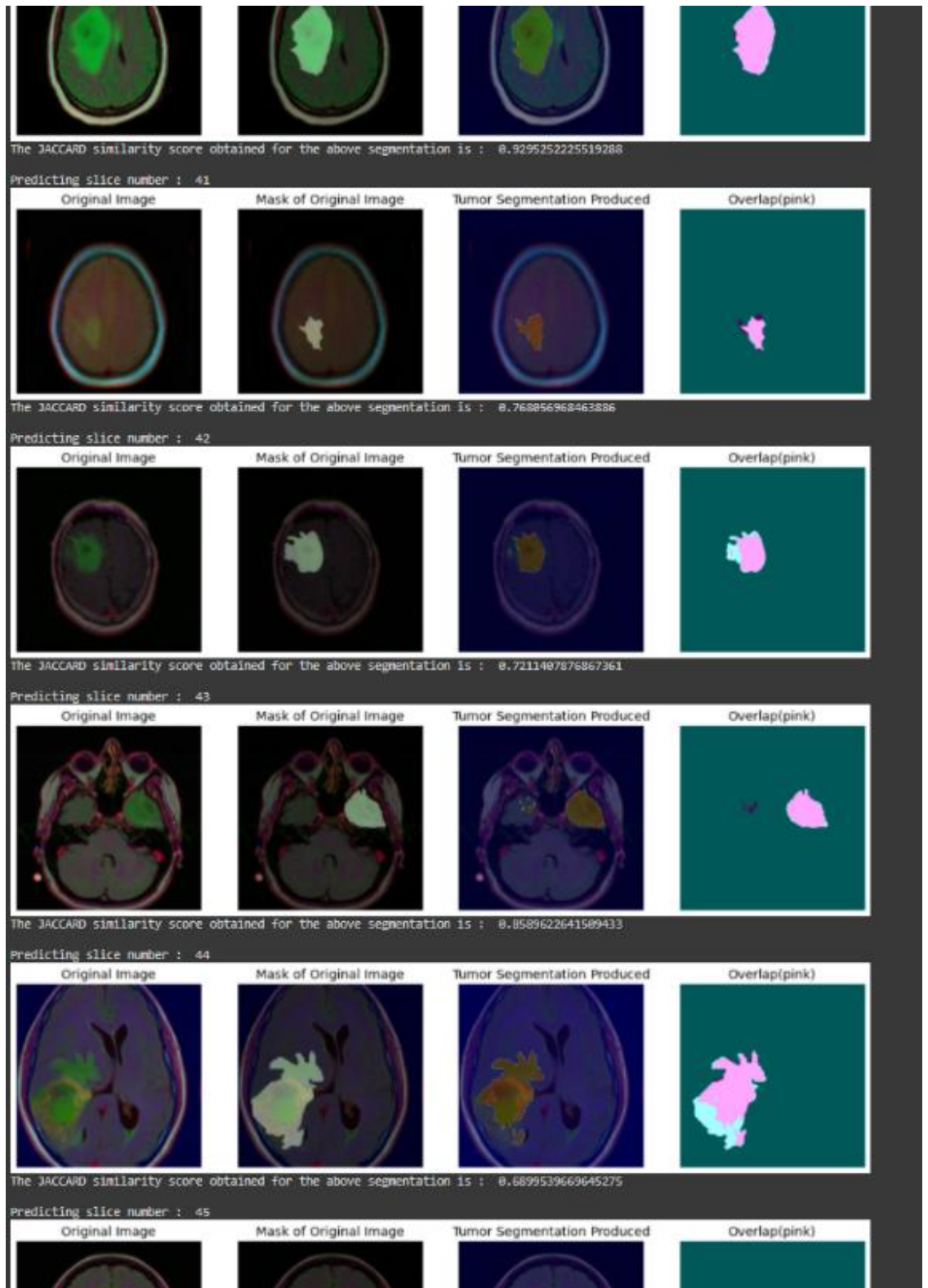


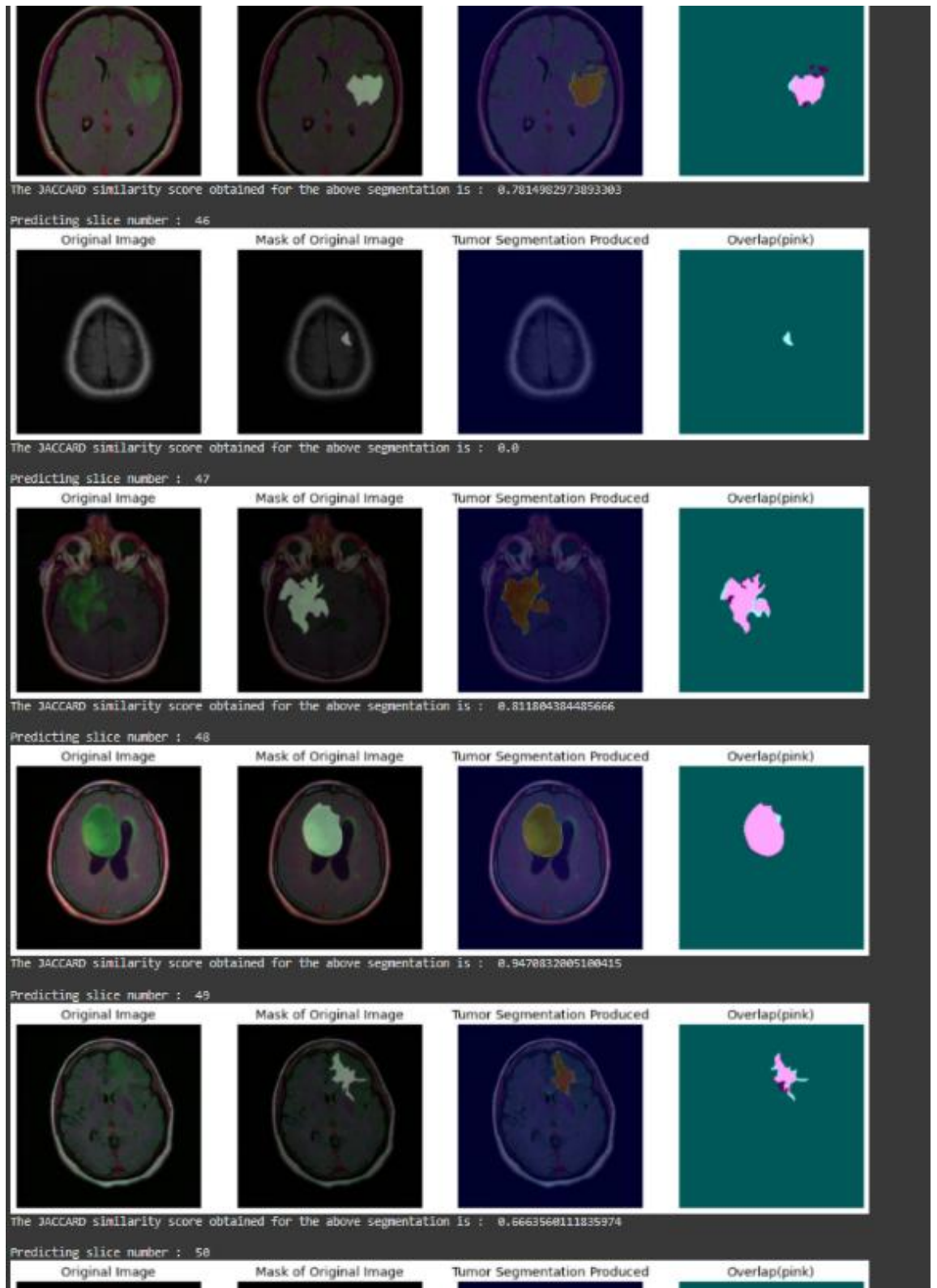


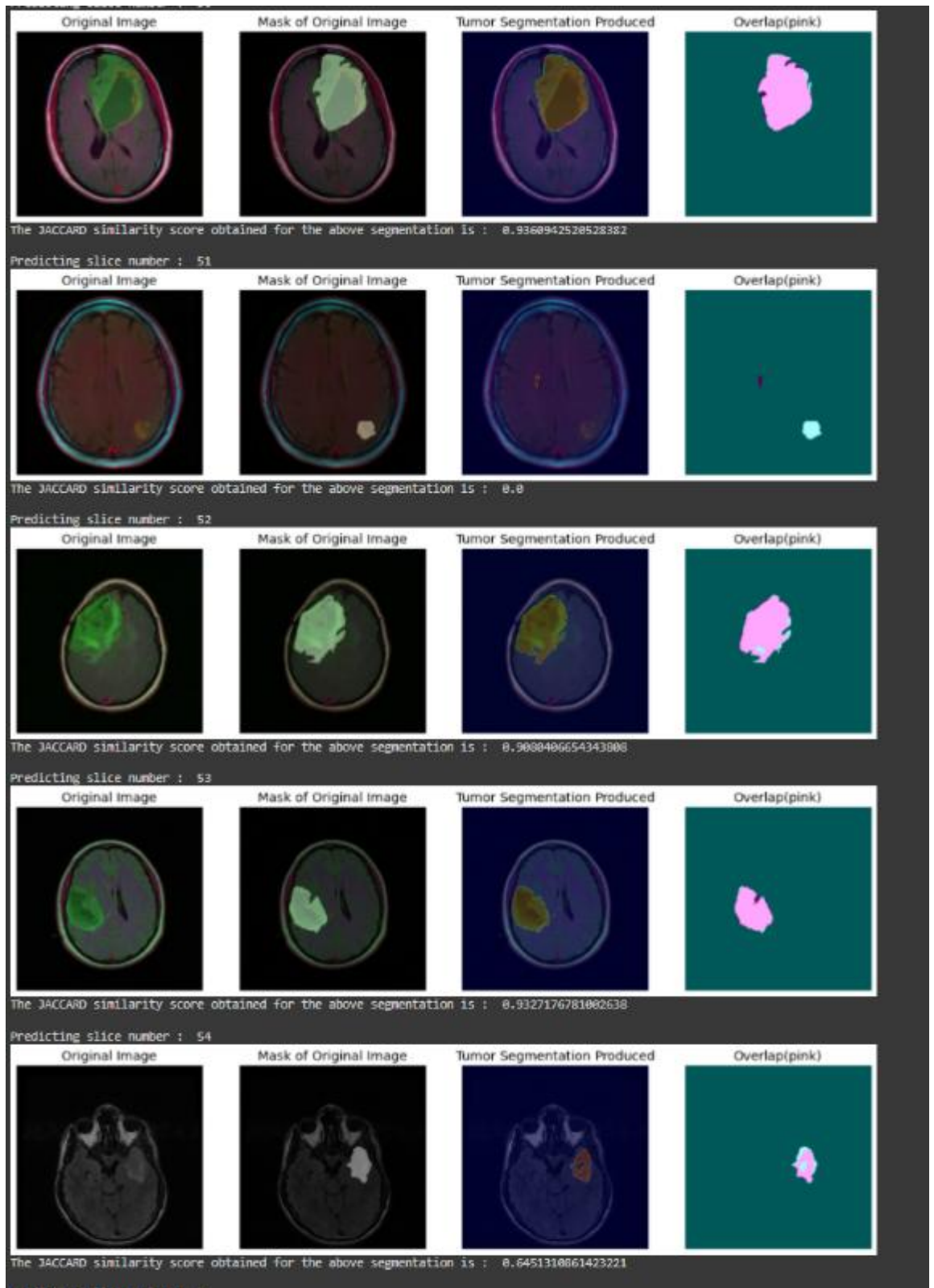


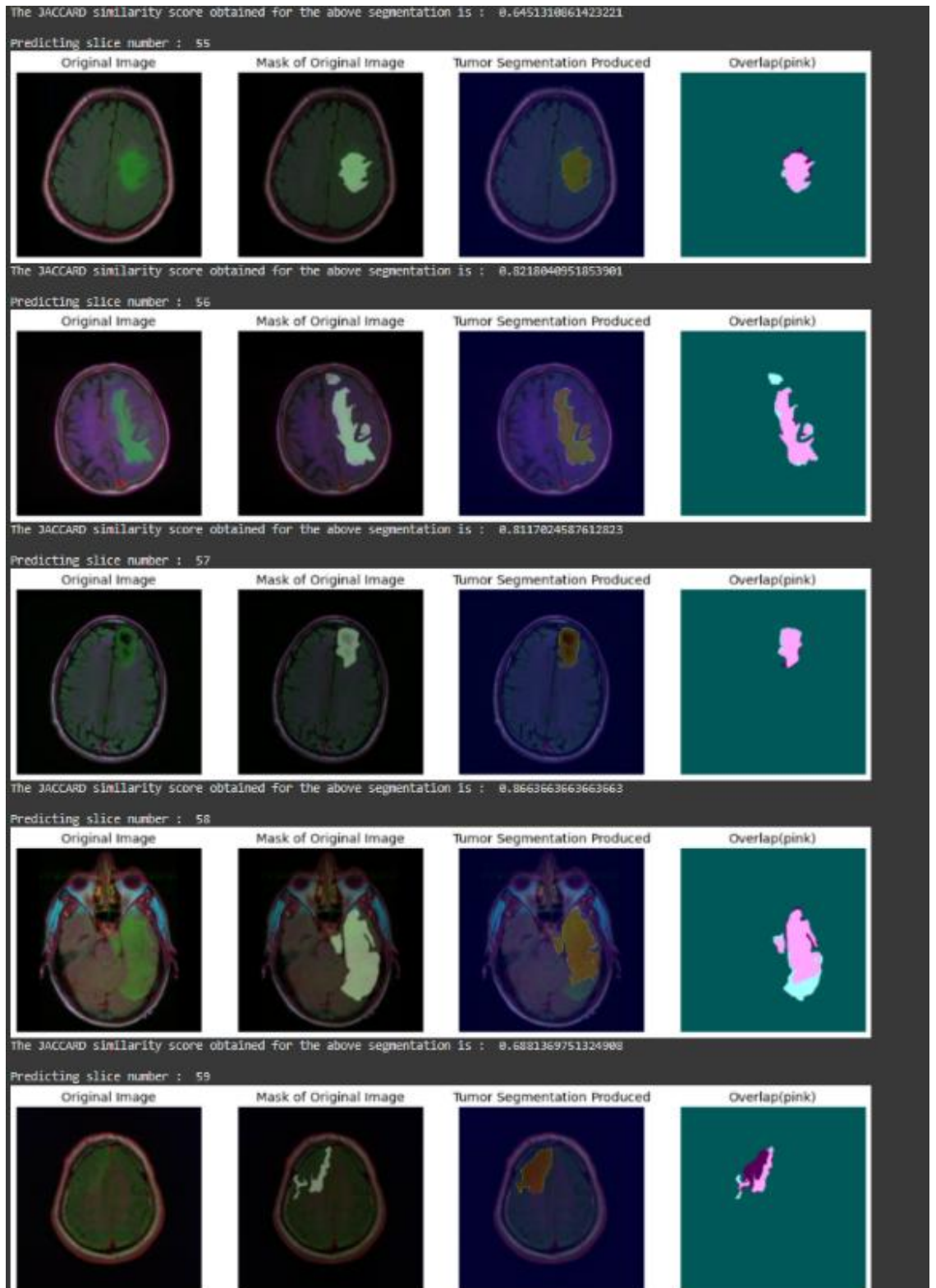


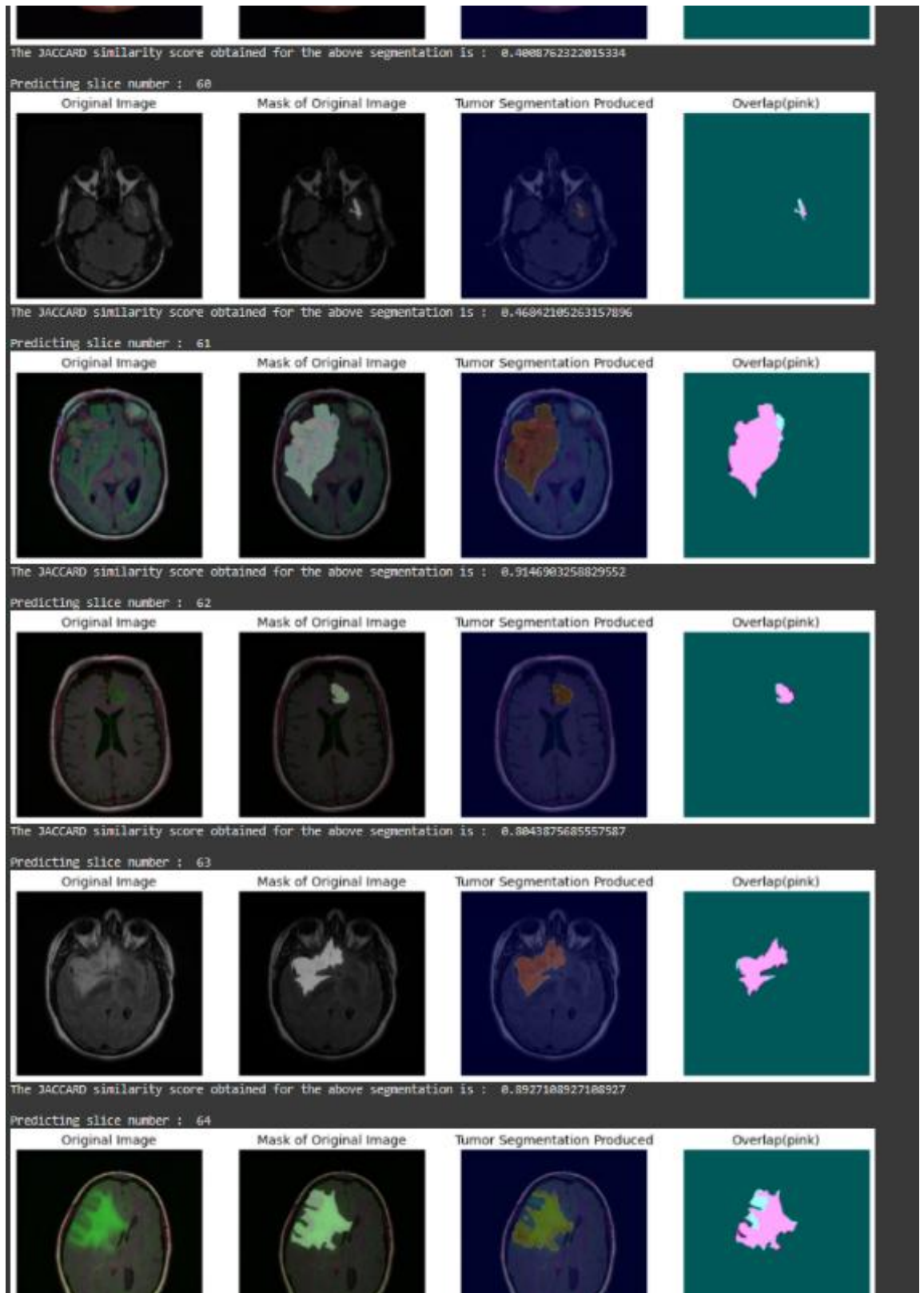


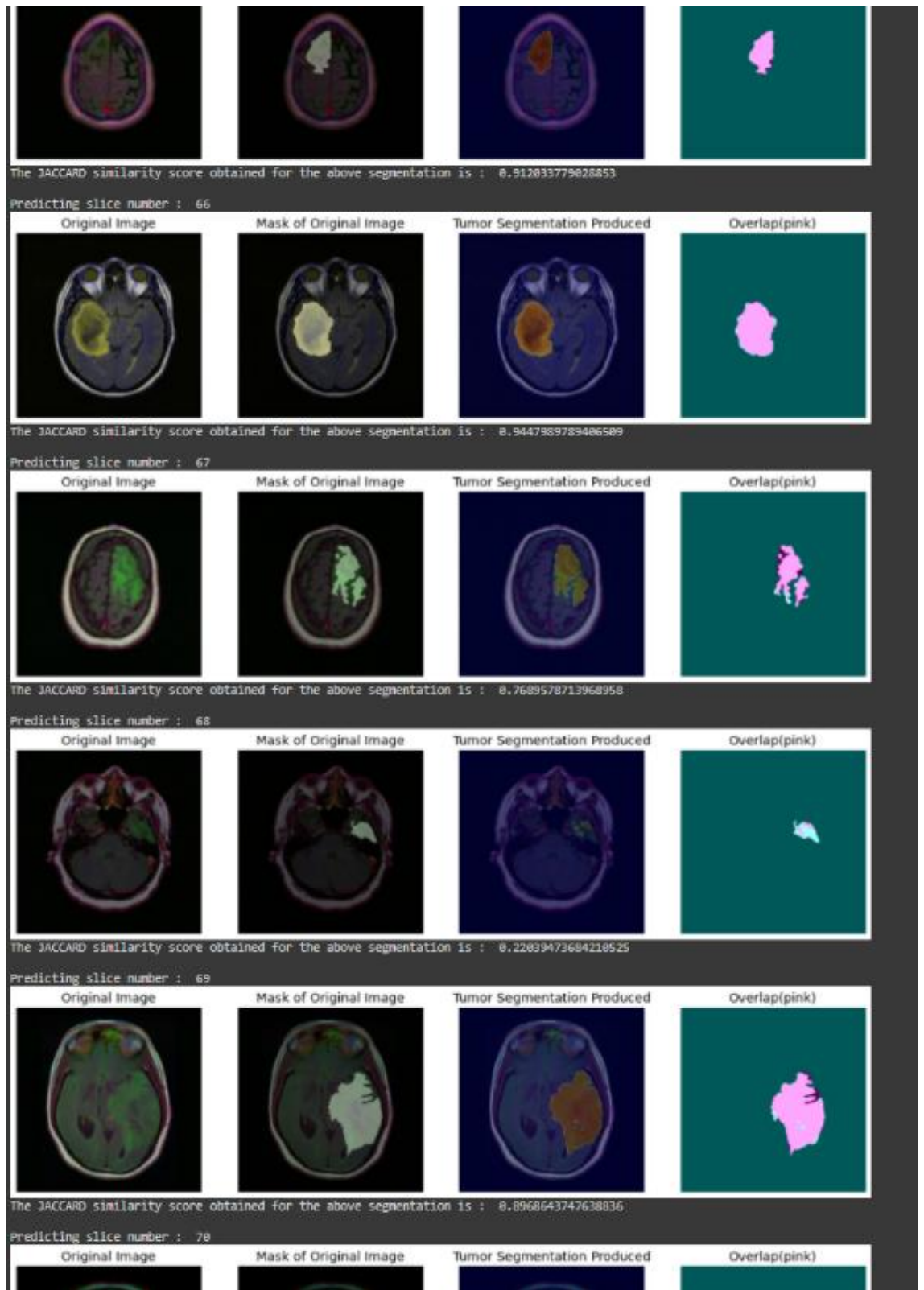


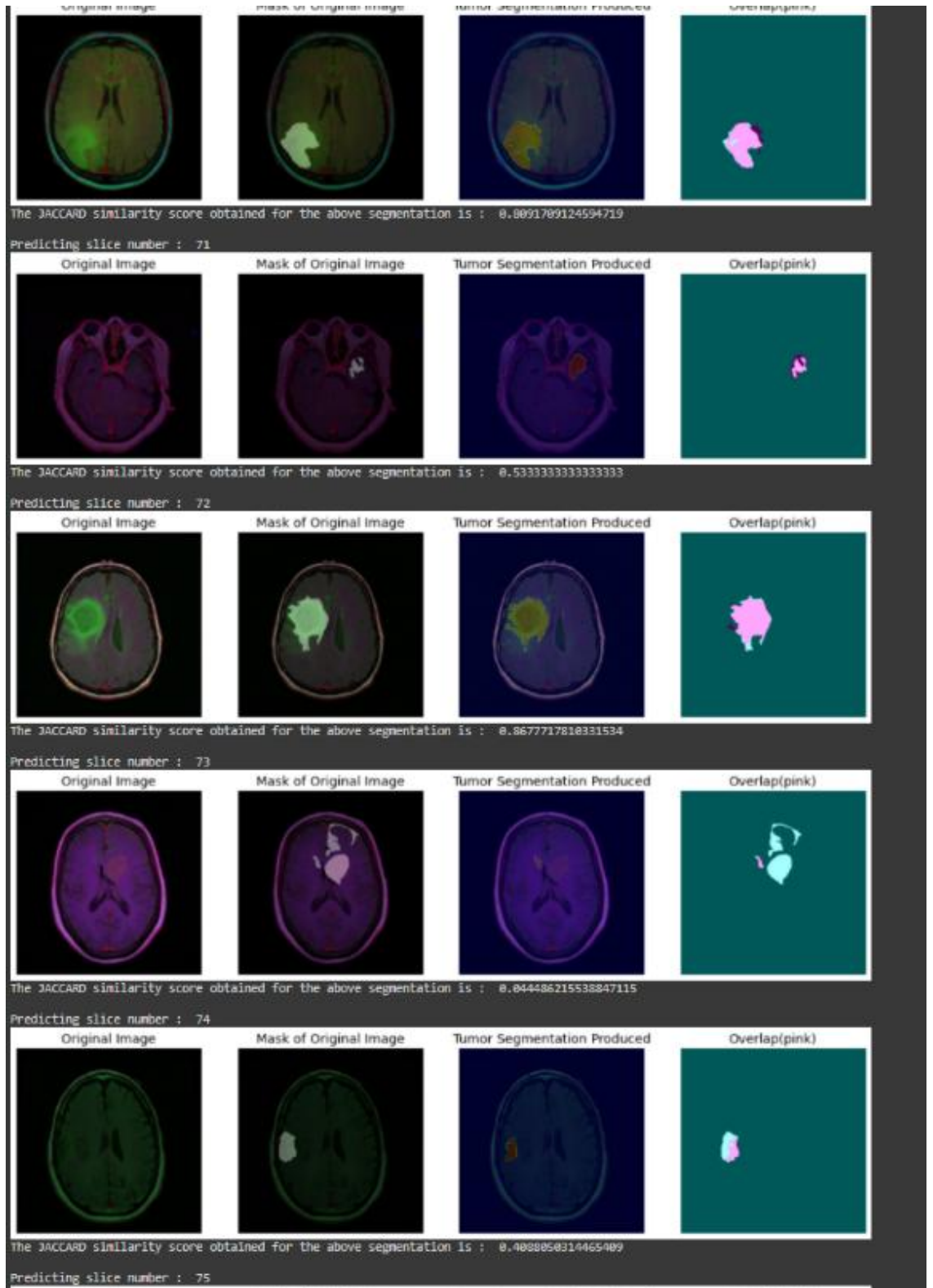


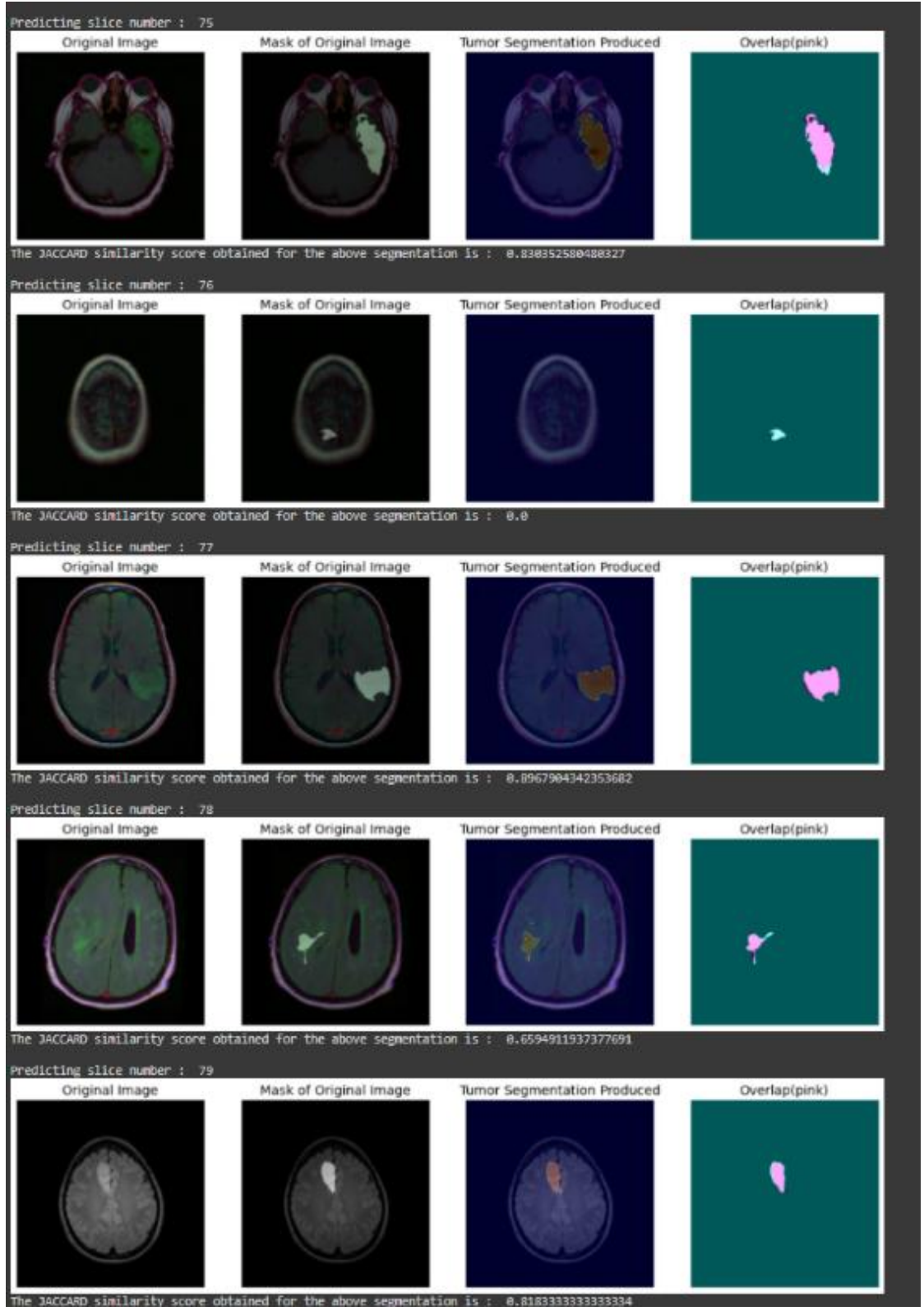


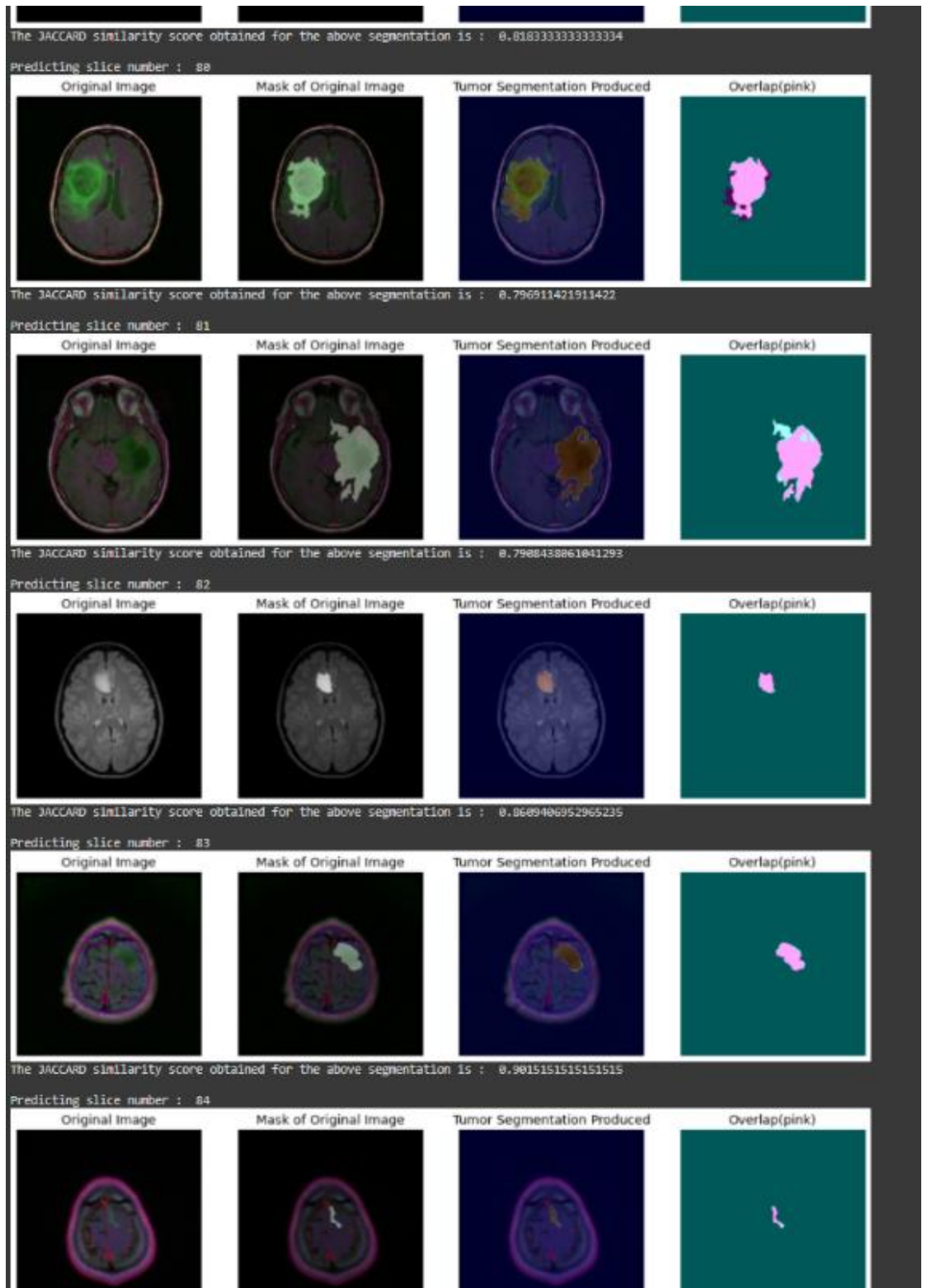


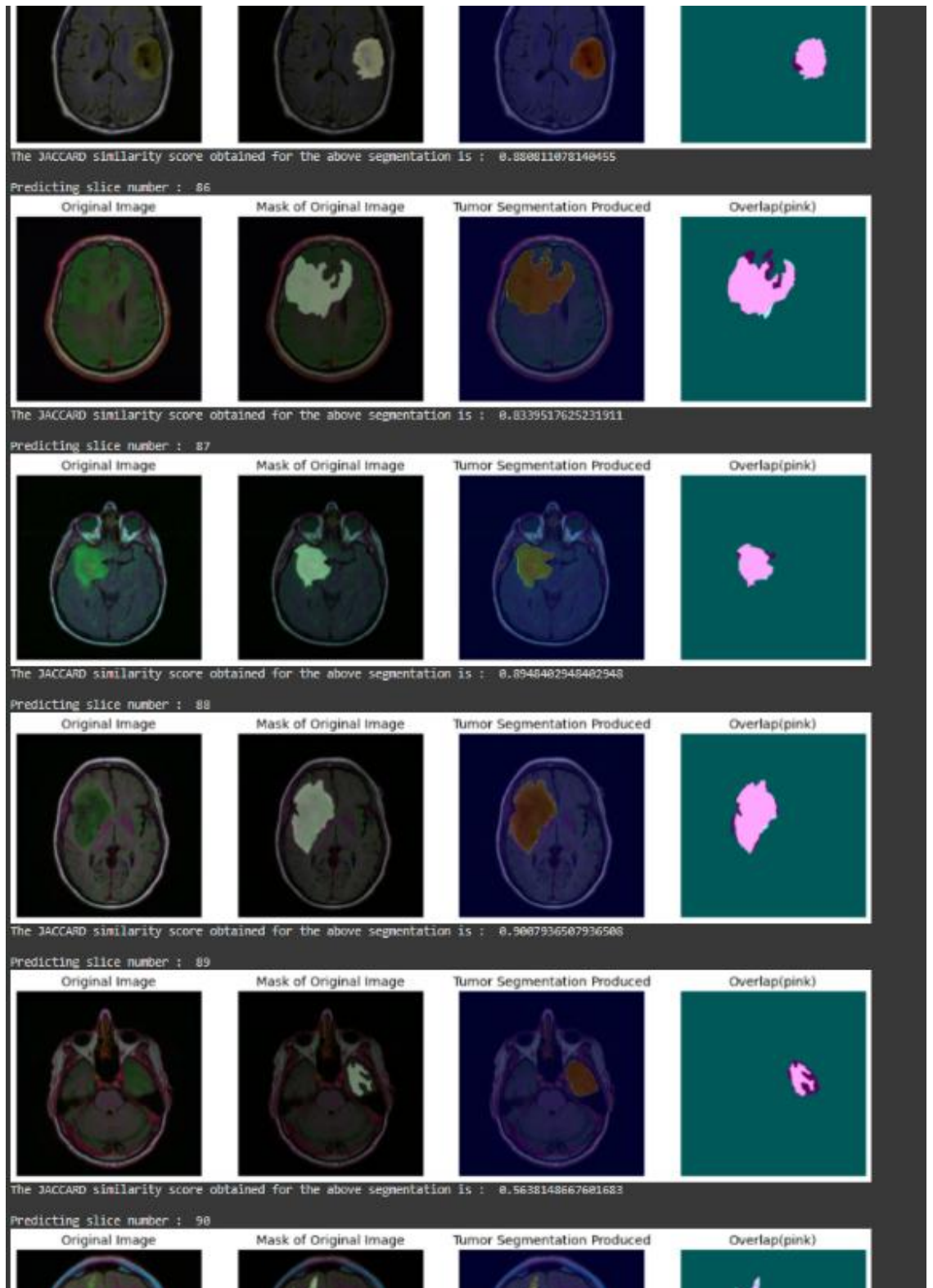


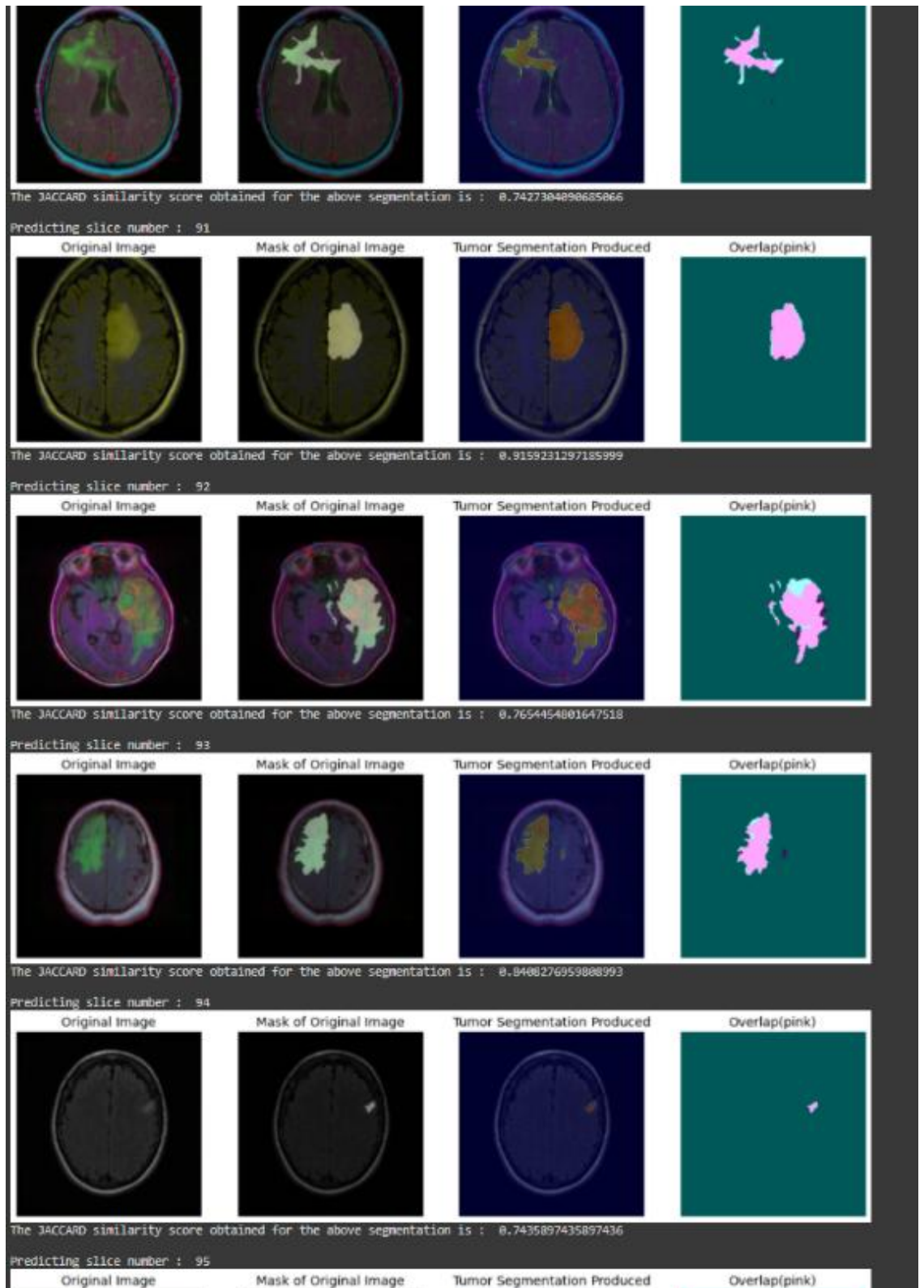


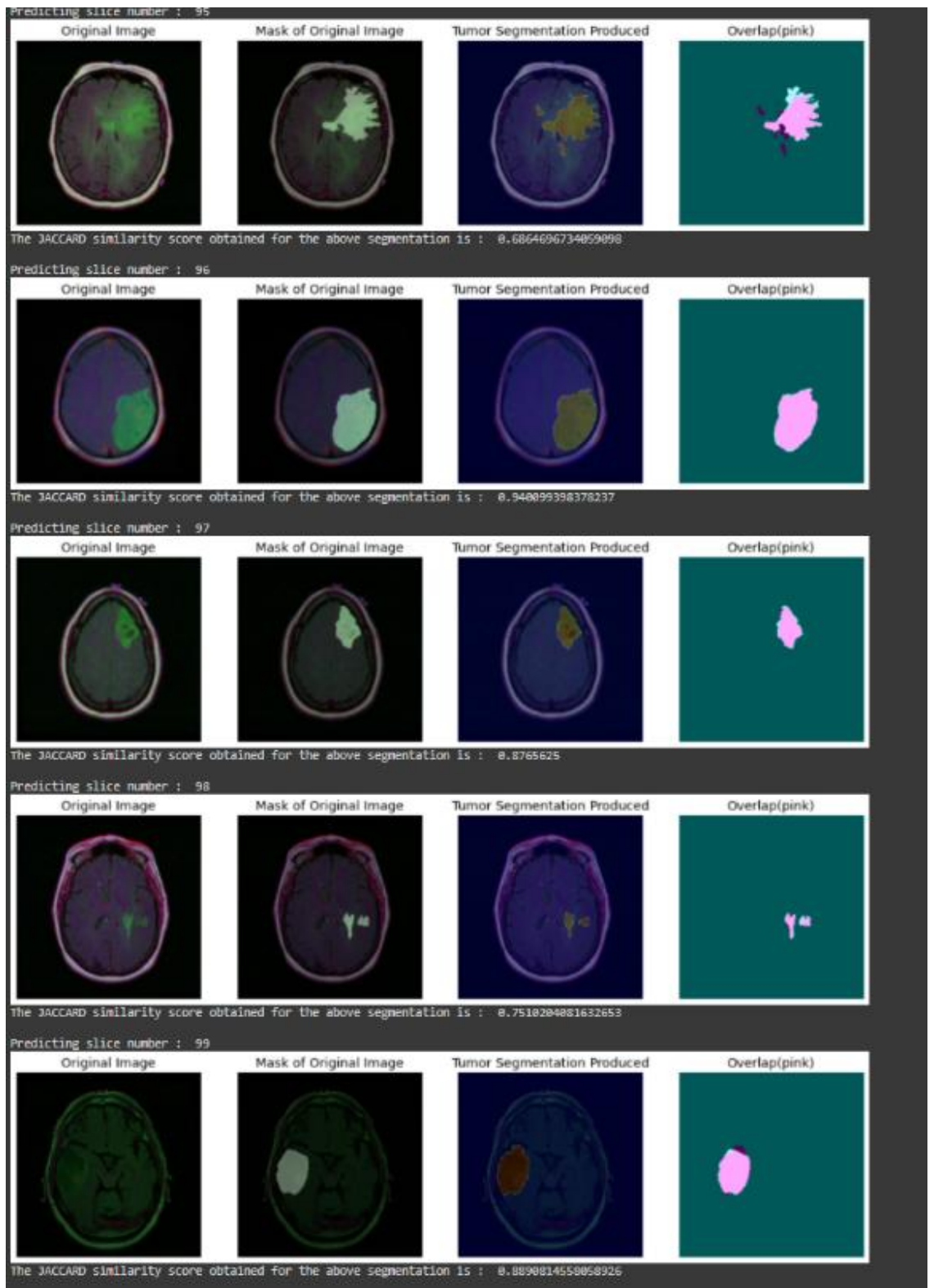


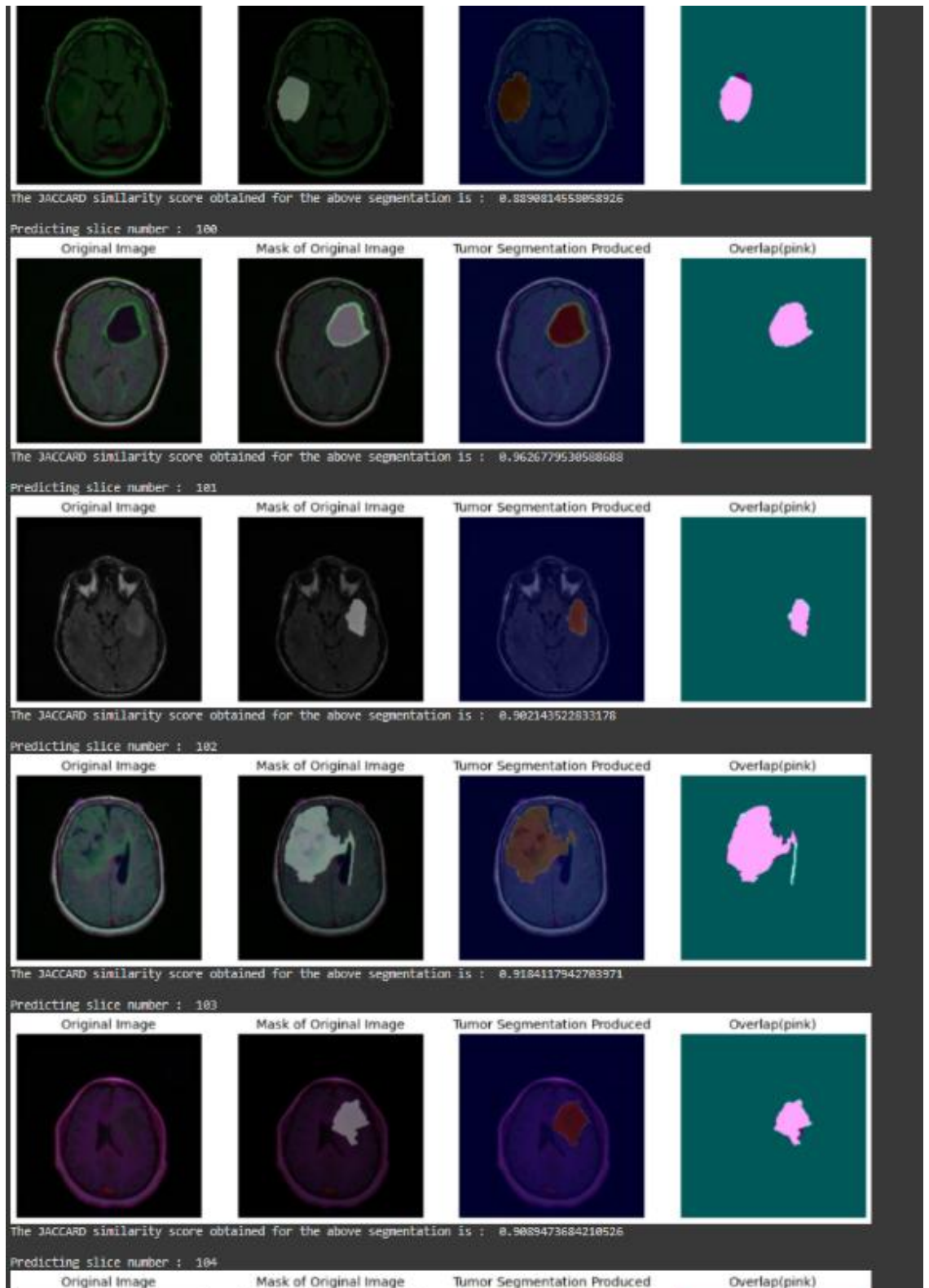


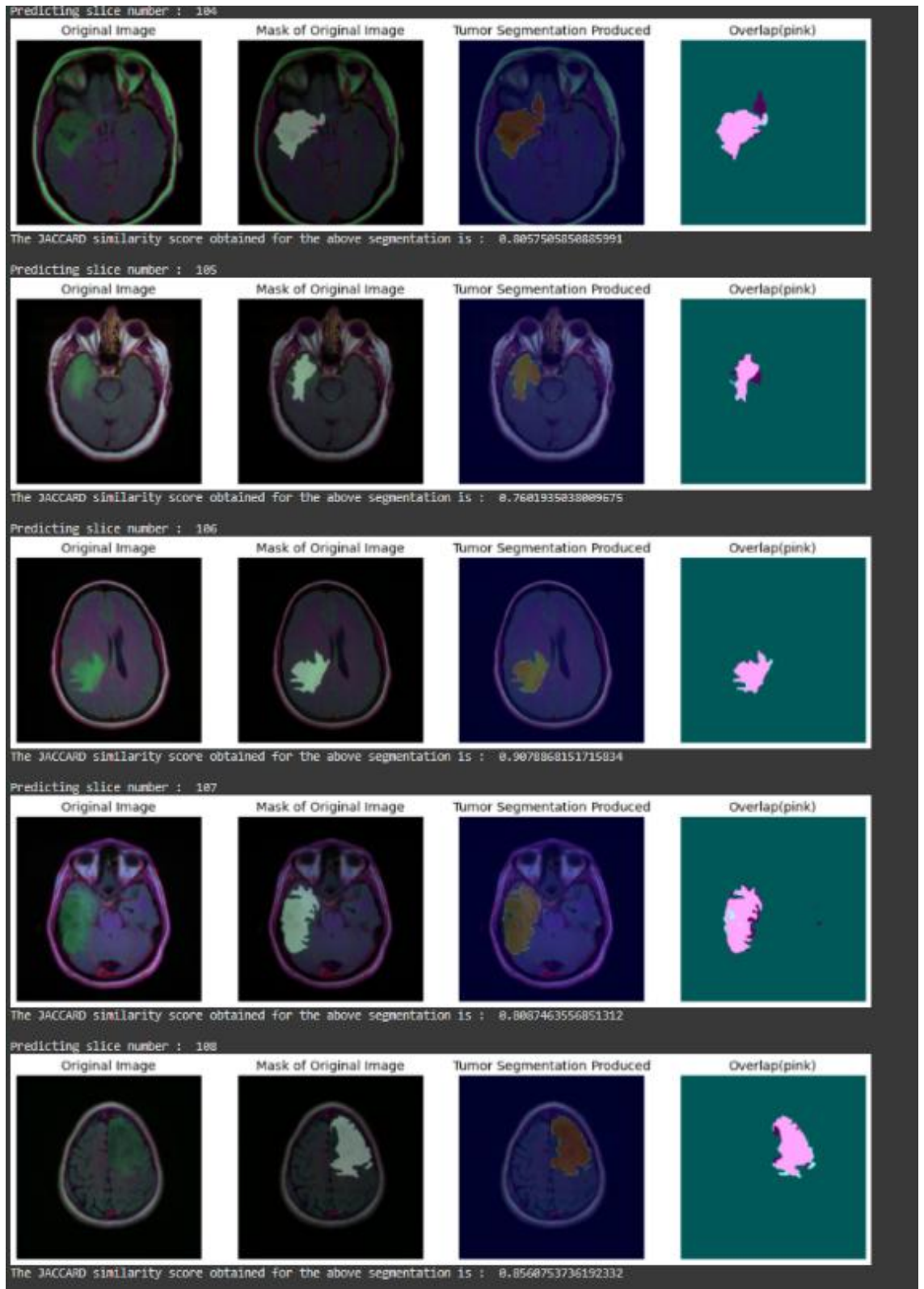


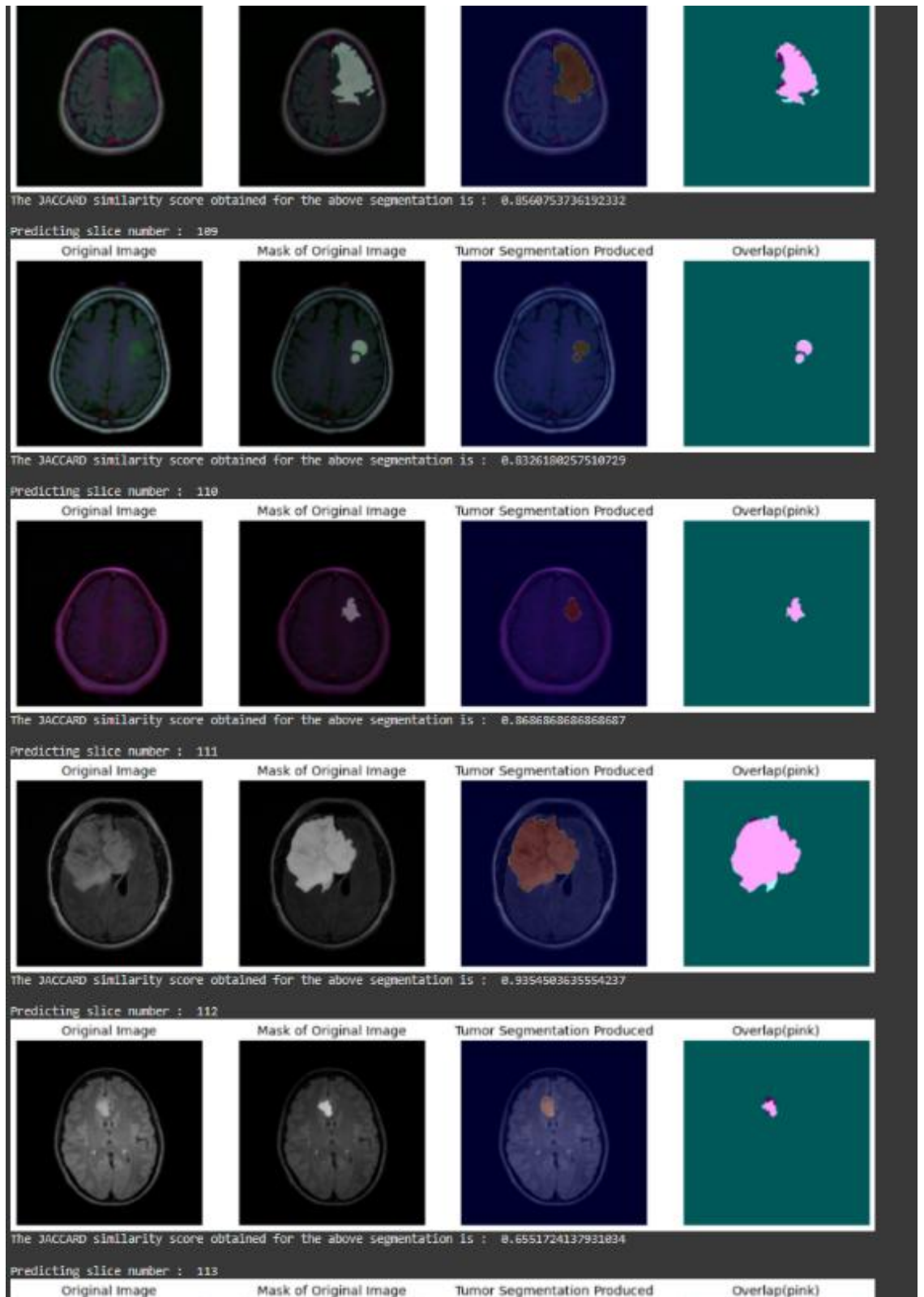


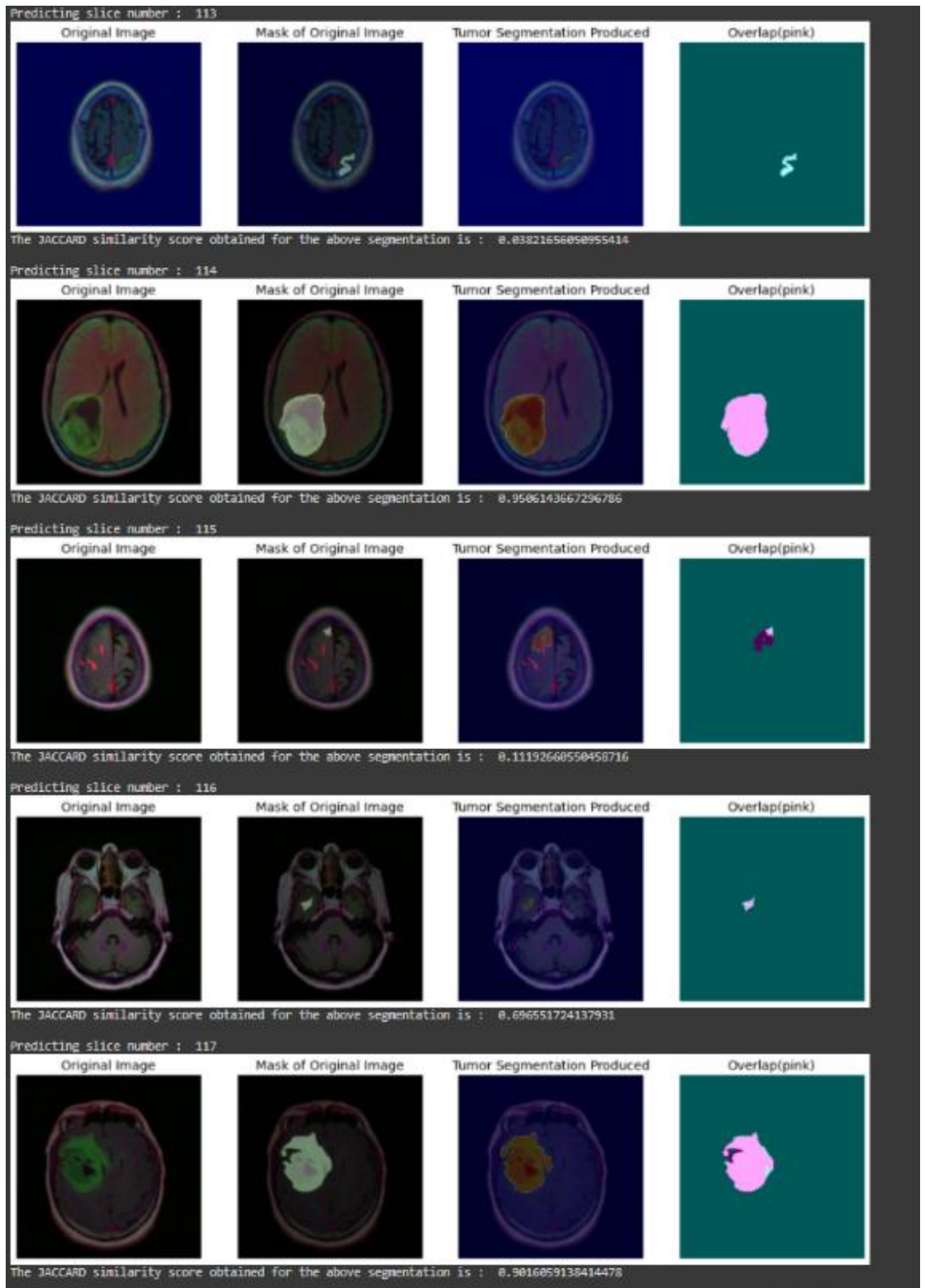


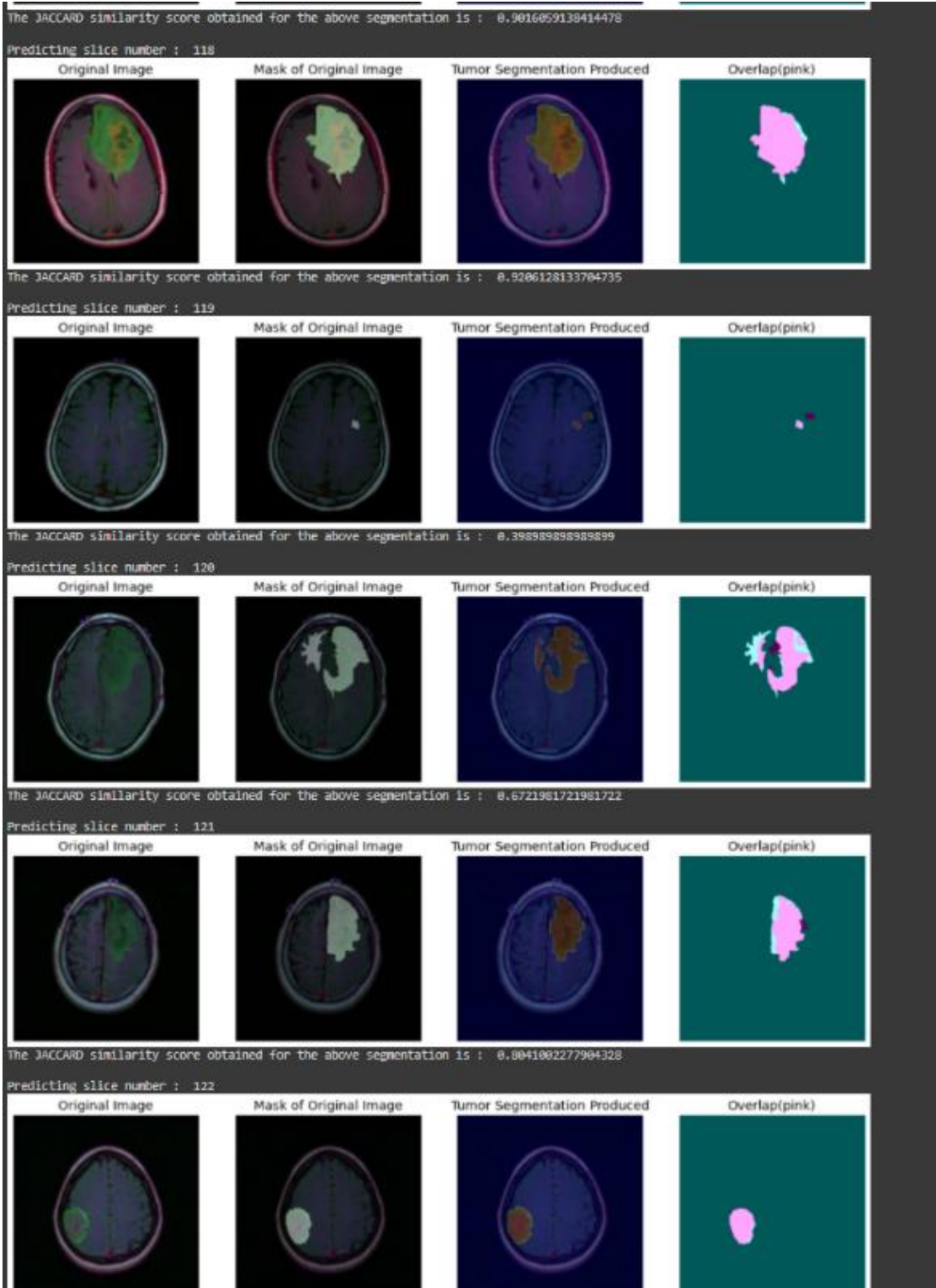


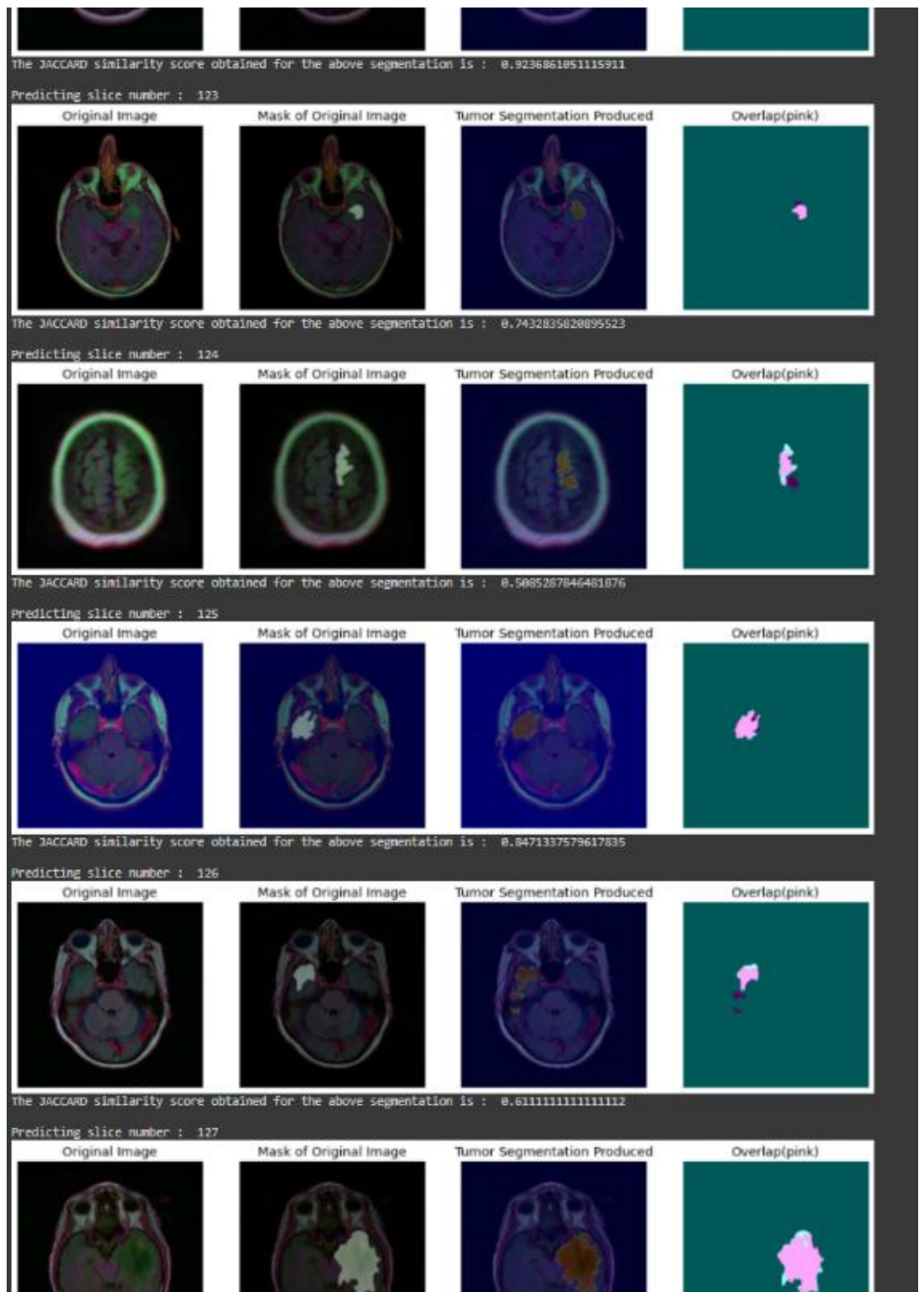


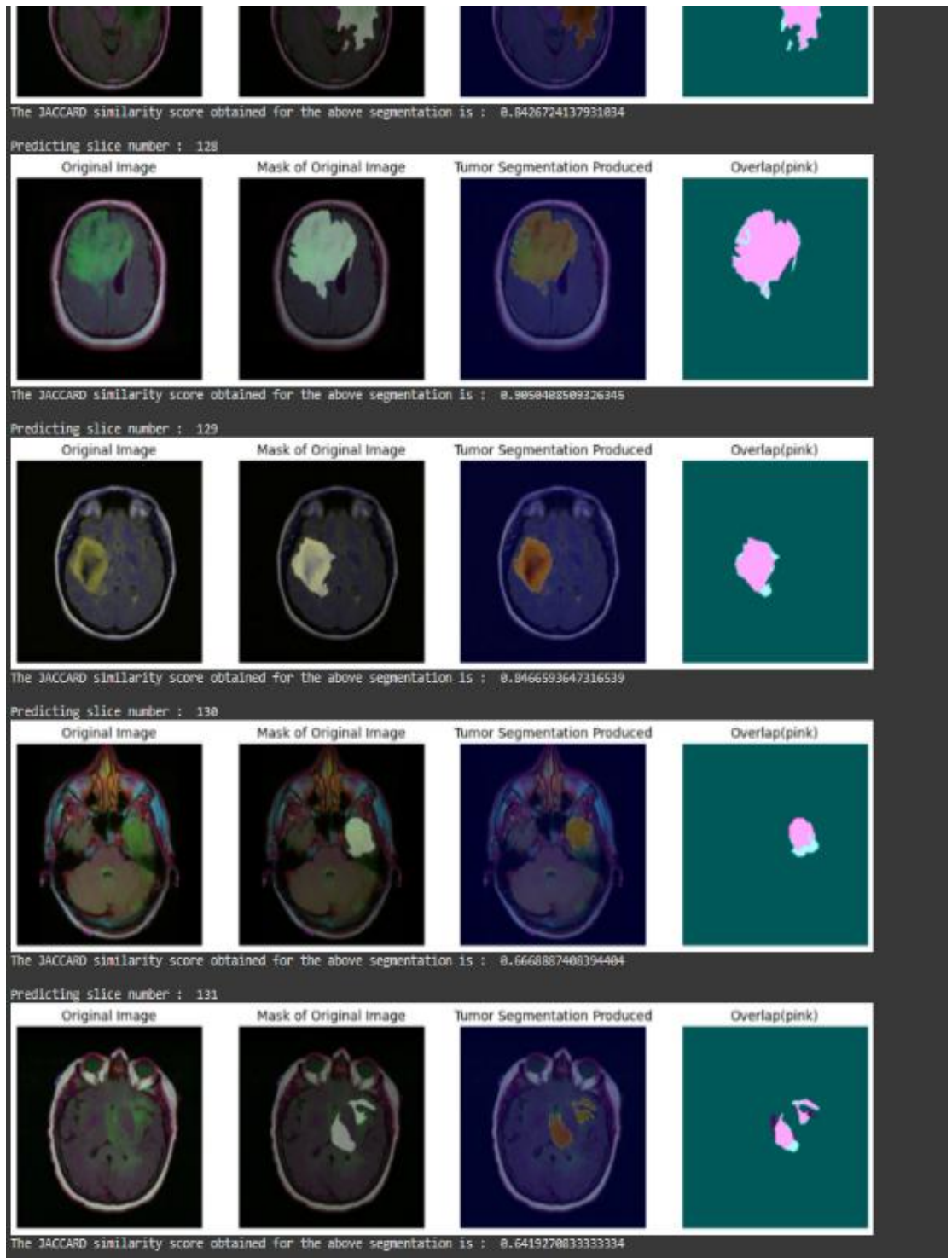


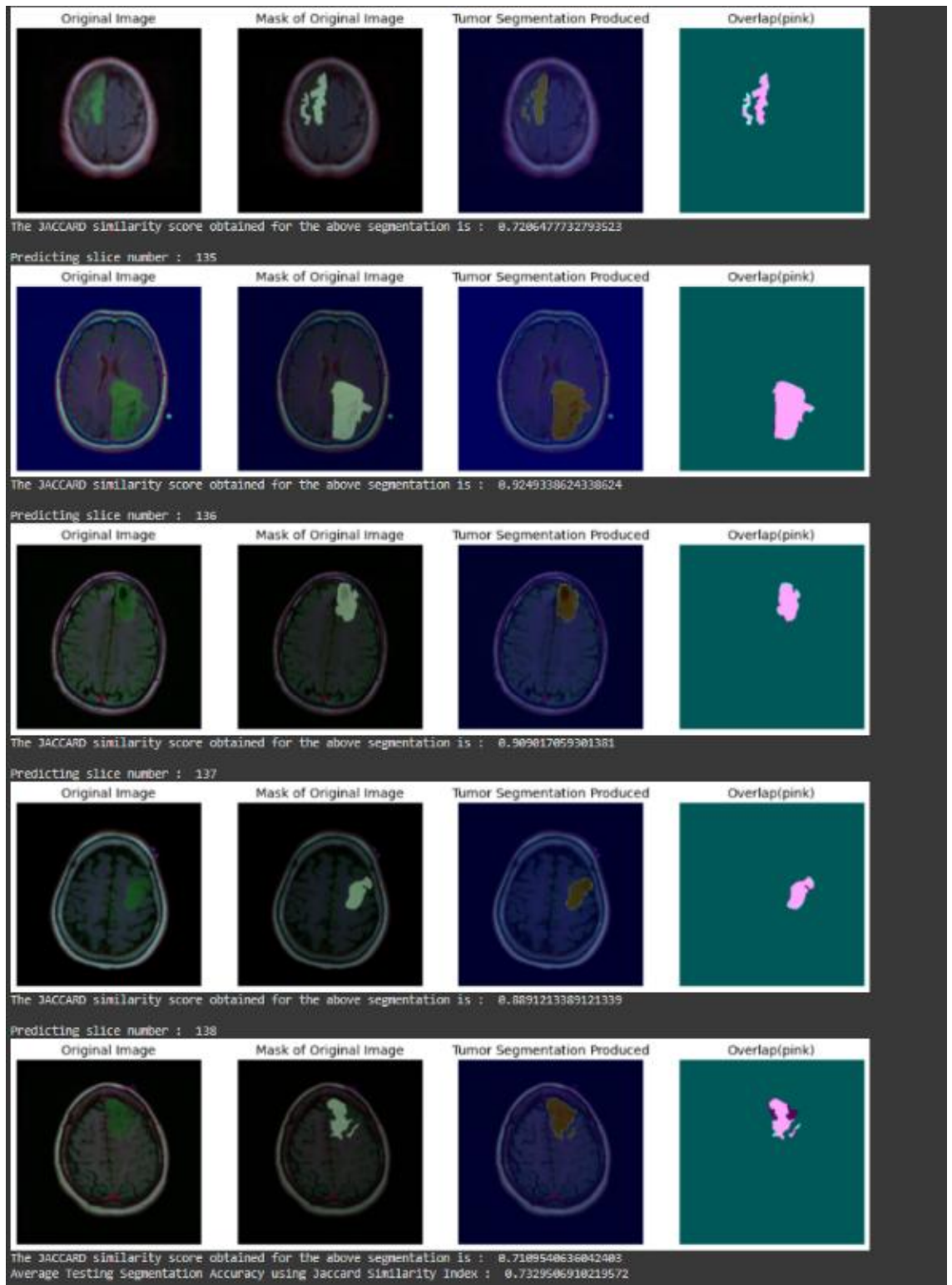




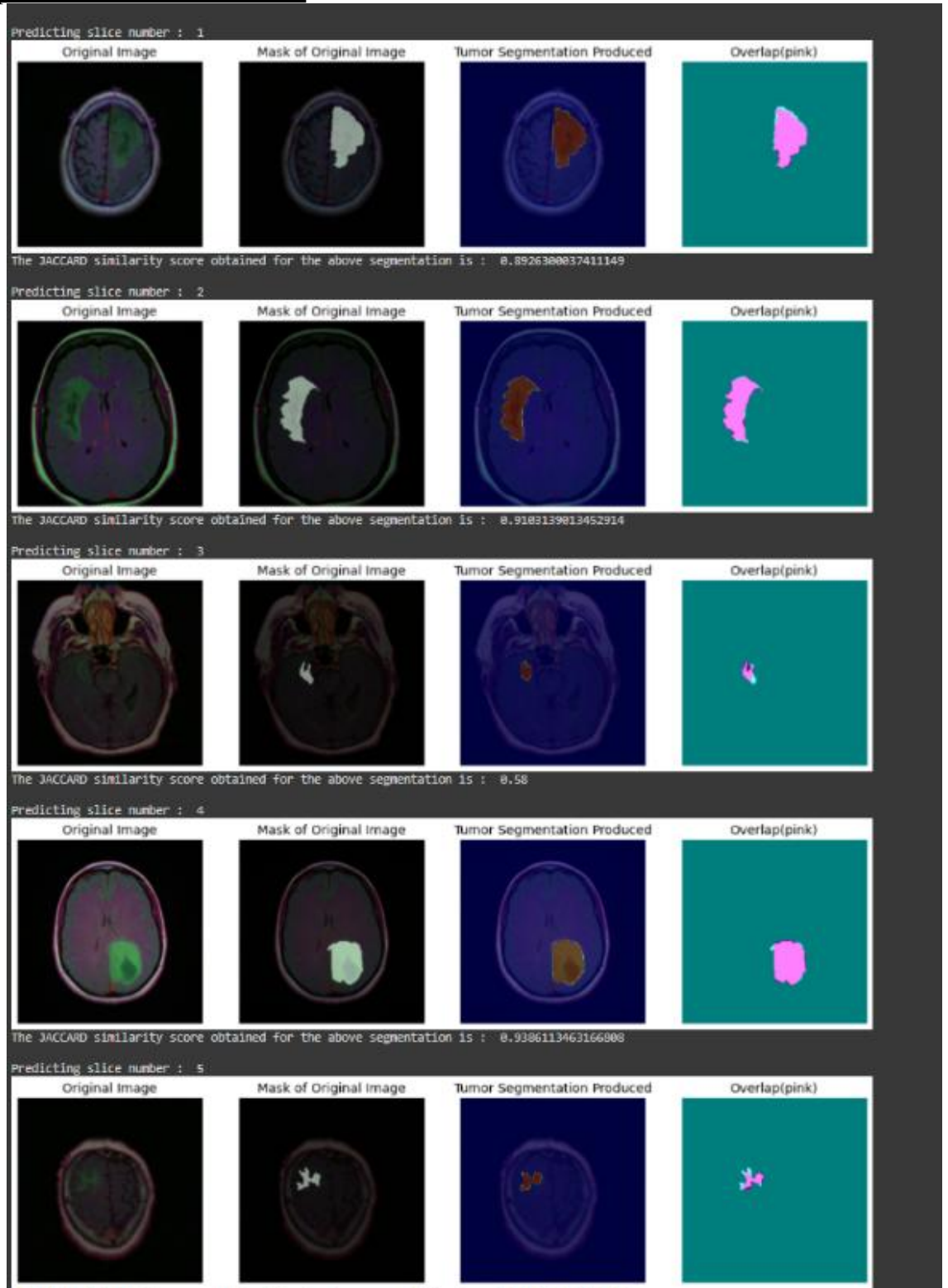


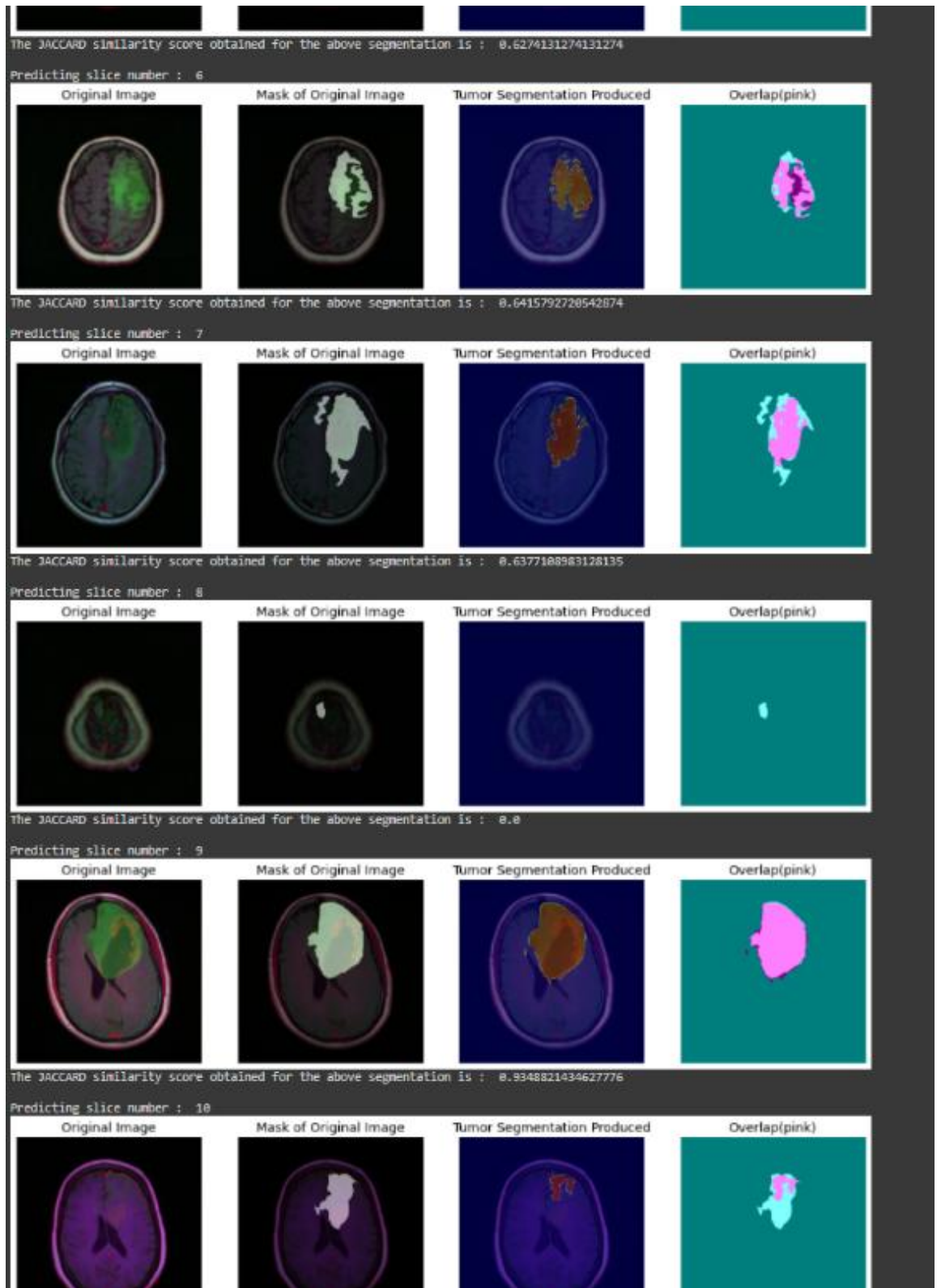


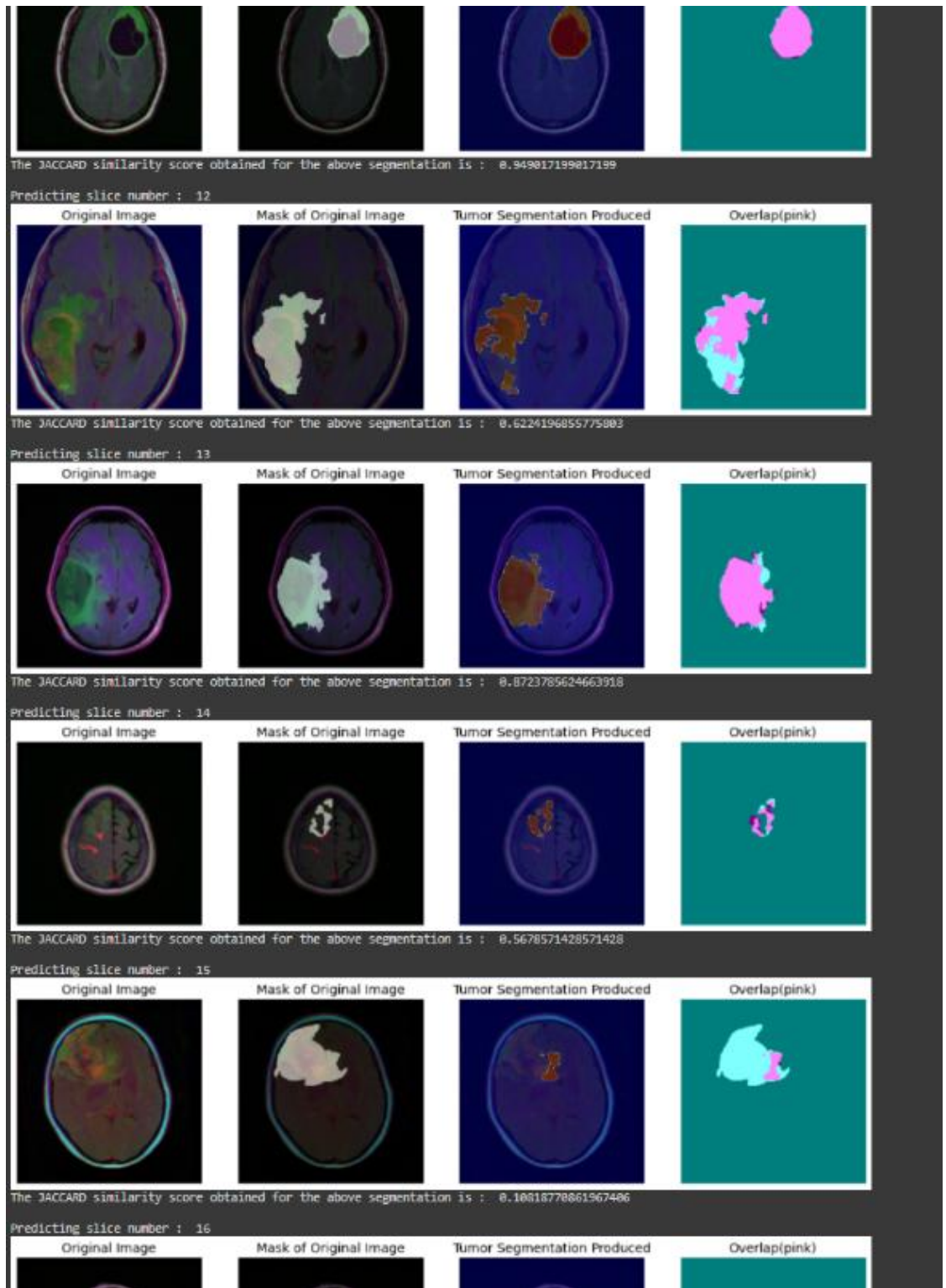


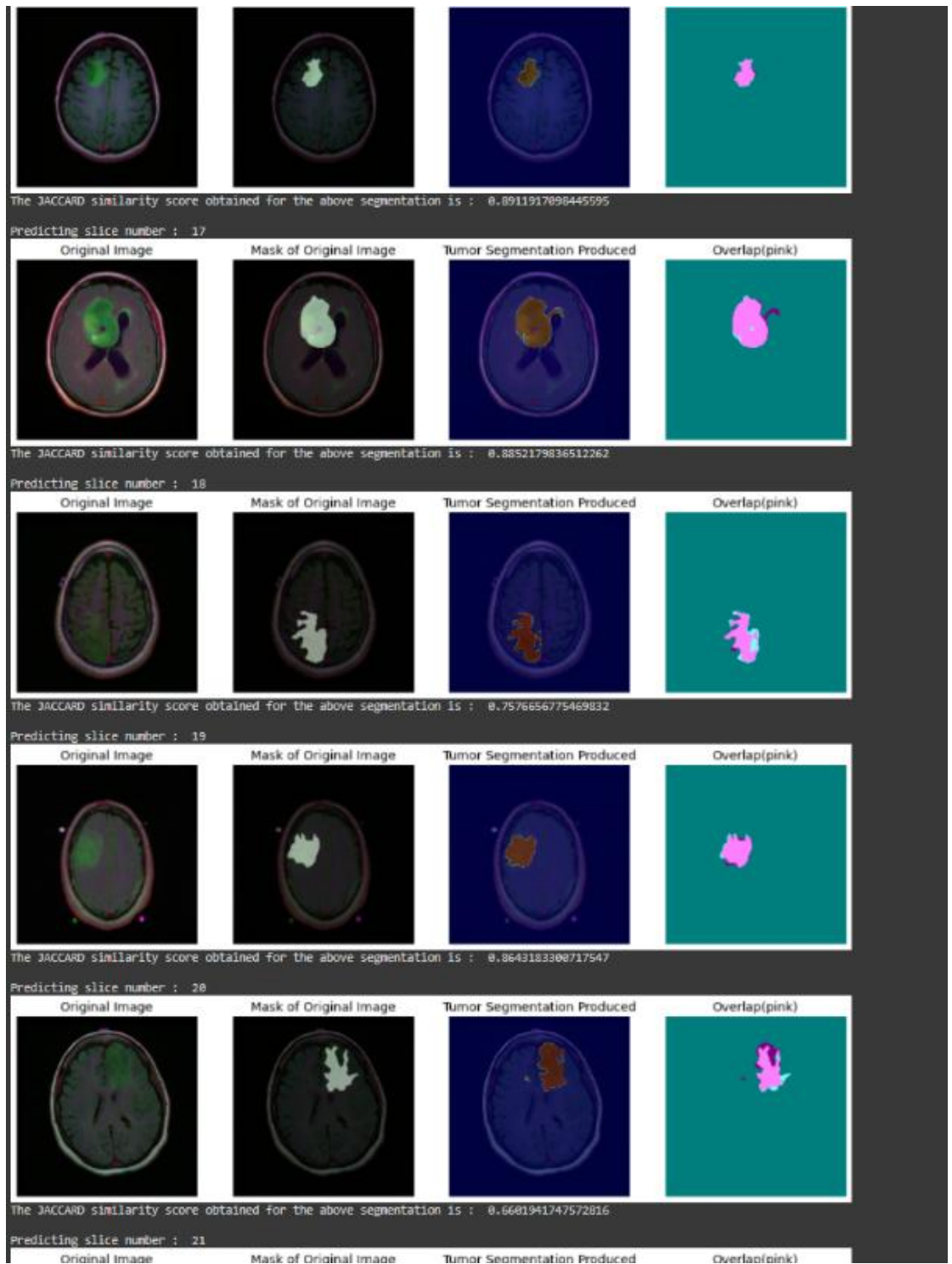


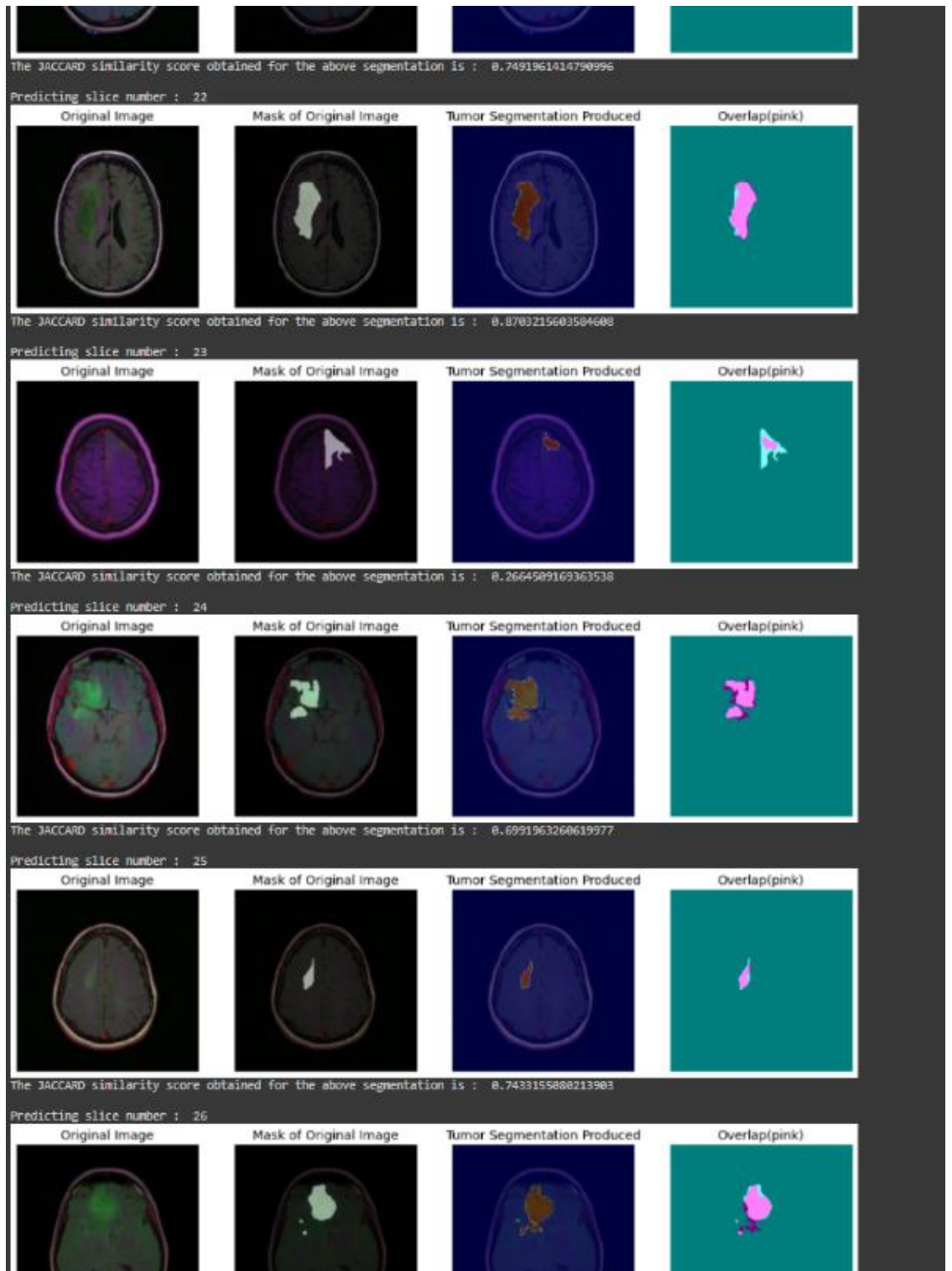
3. Experimental Model No. 8

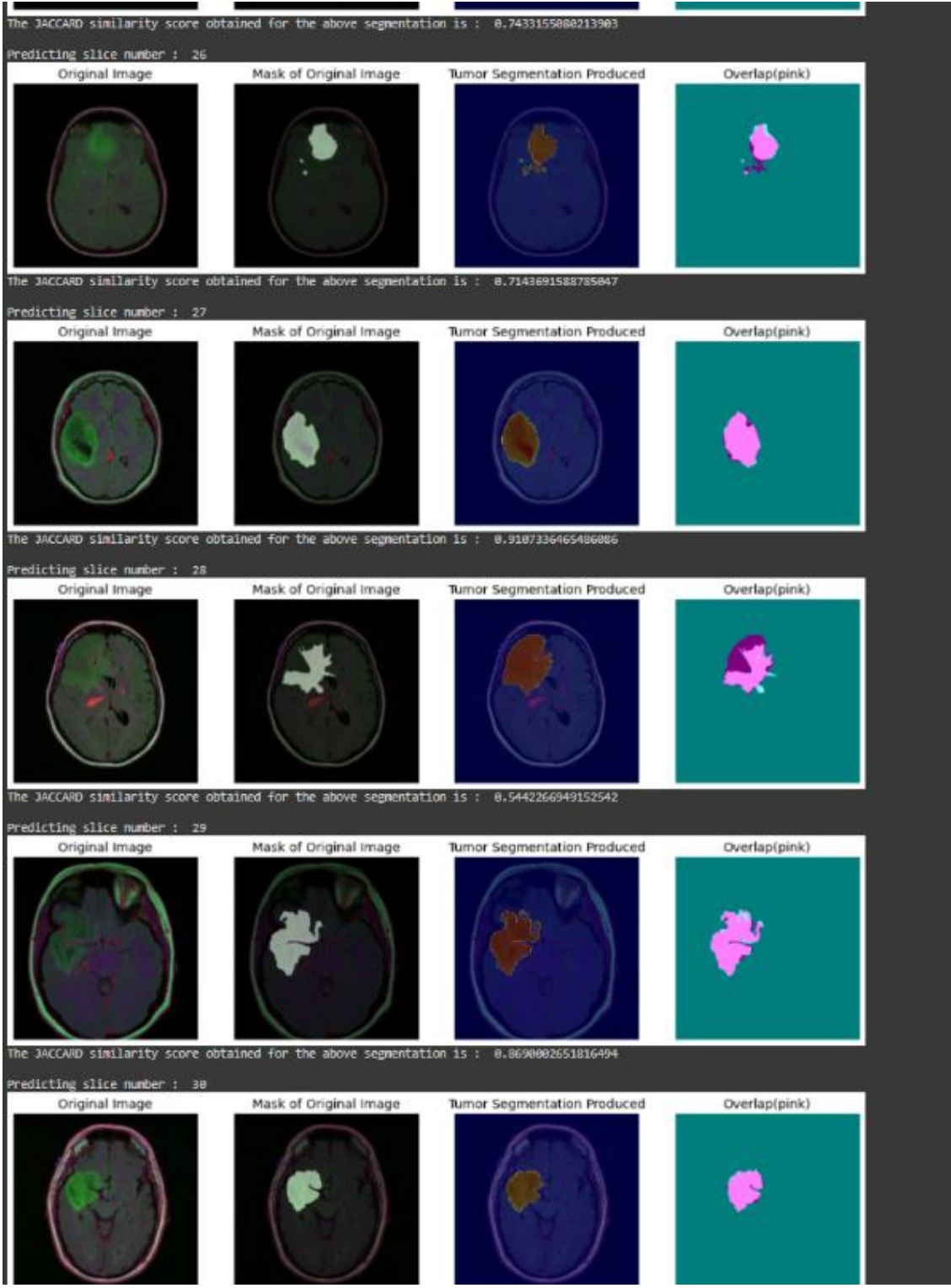


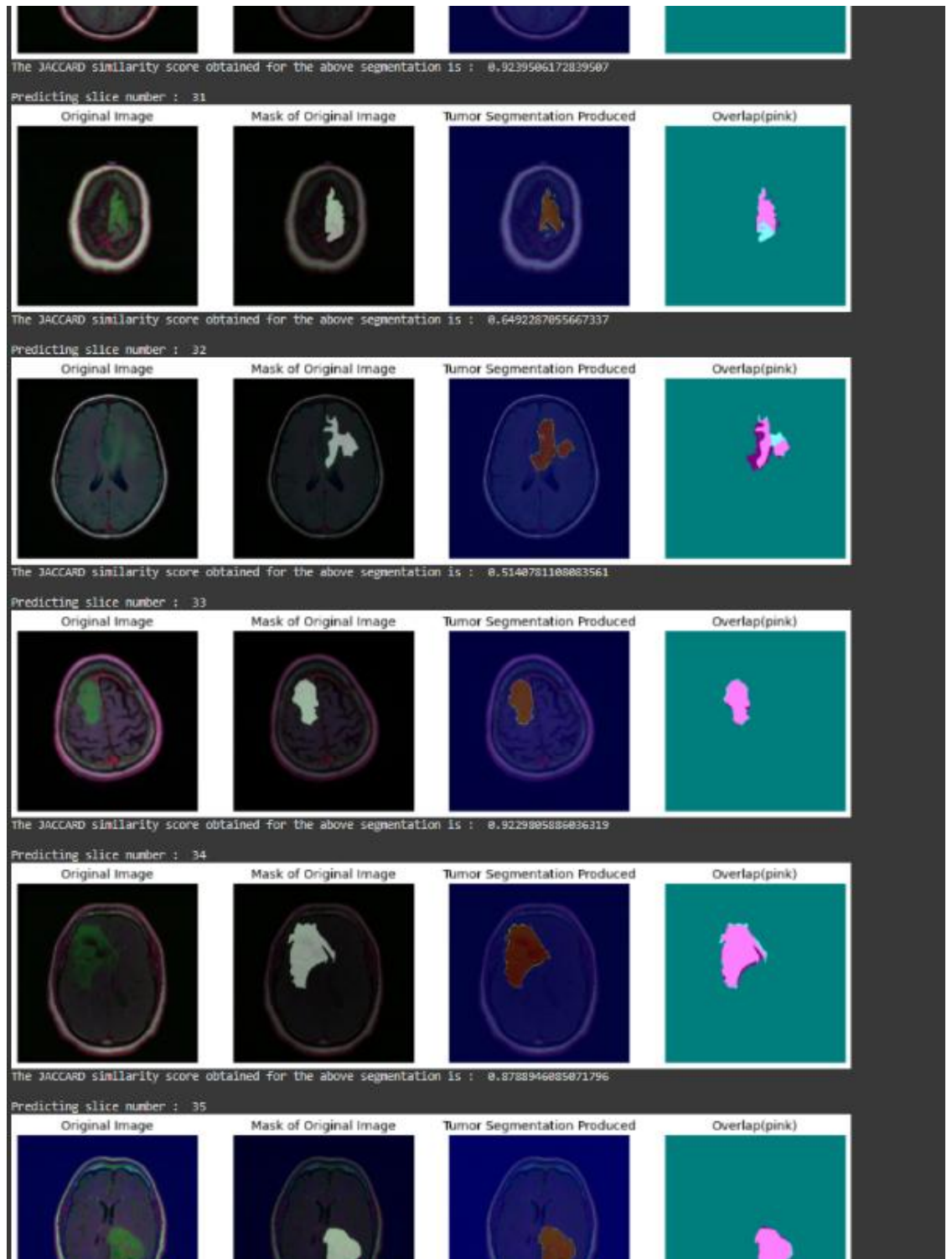


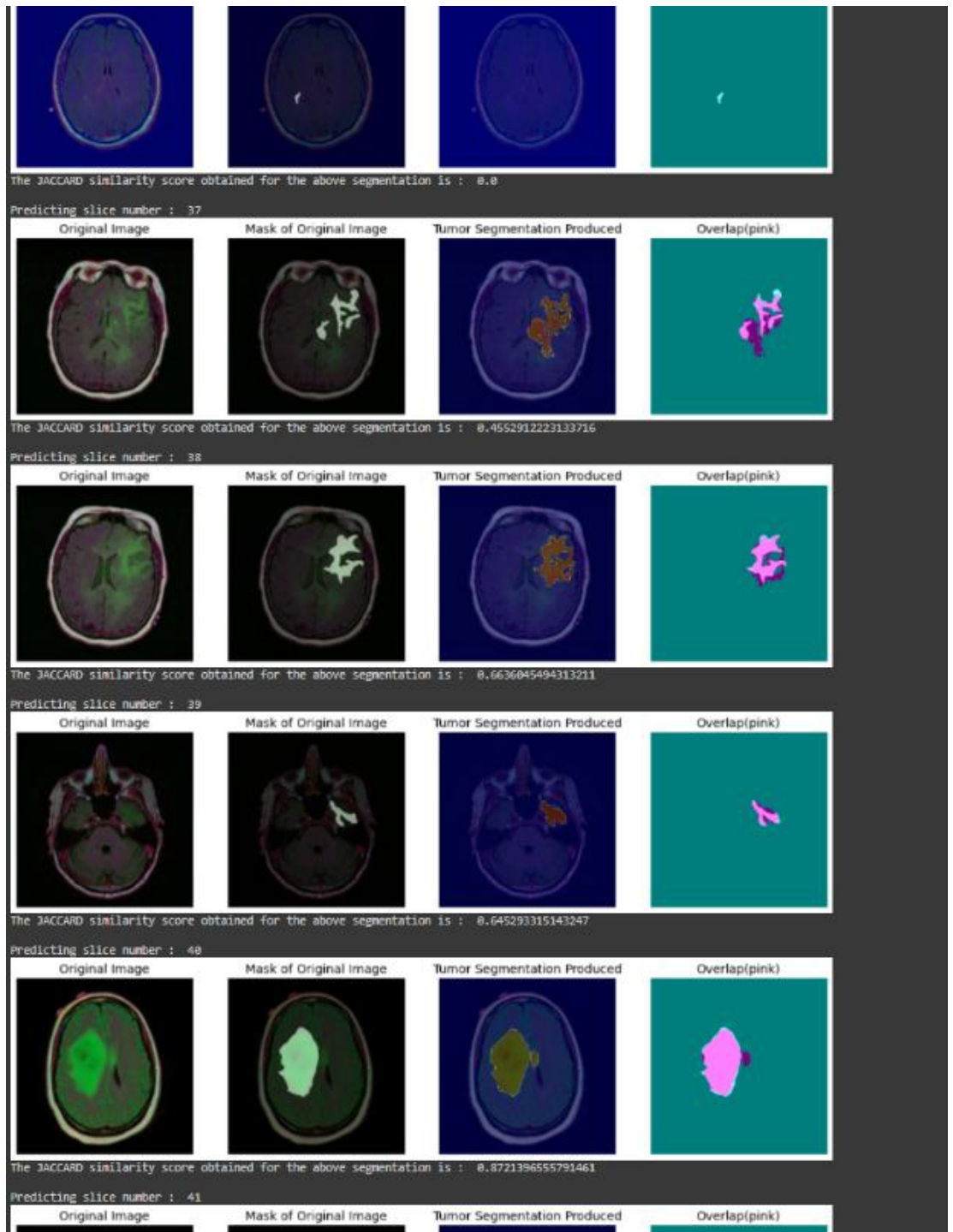


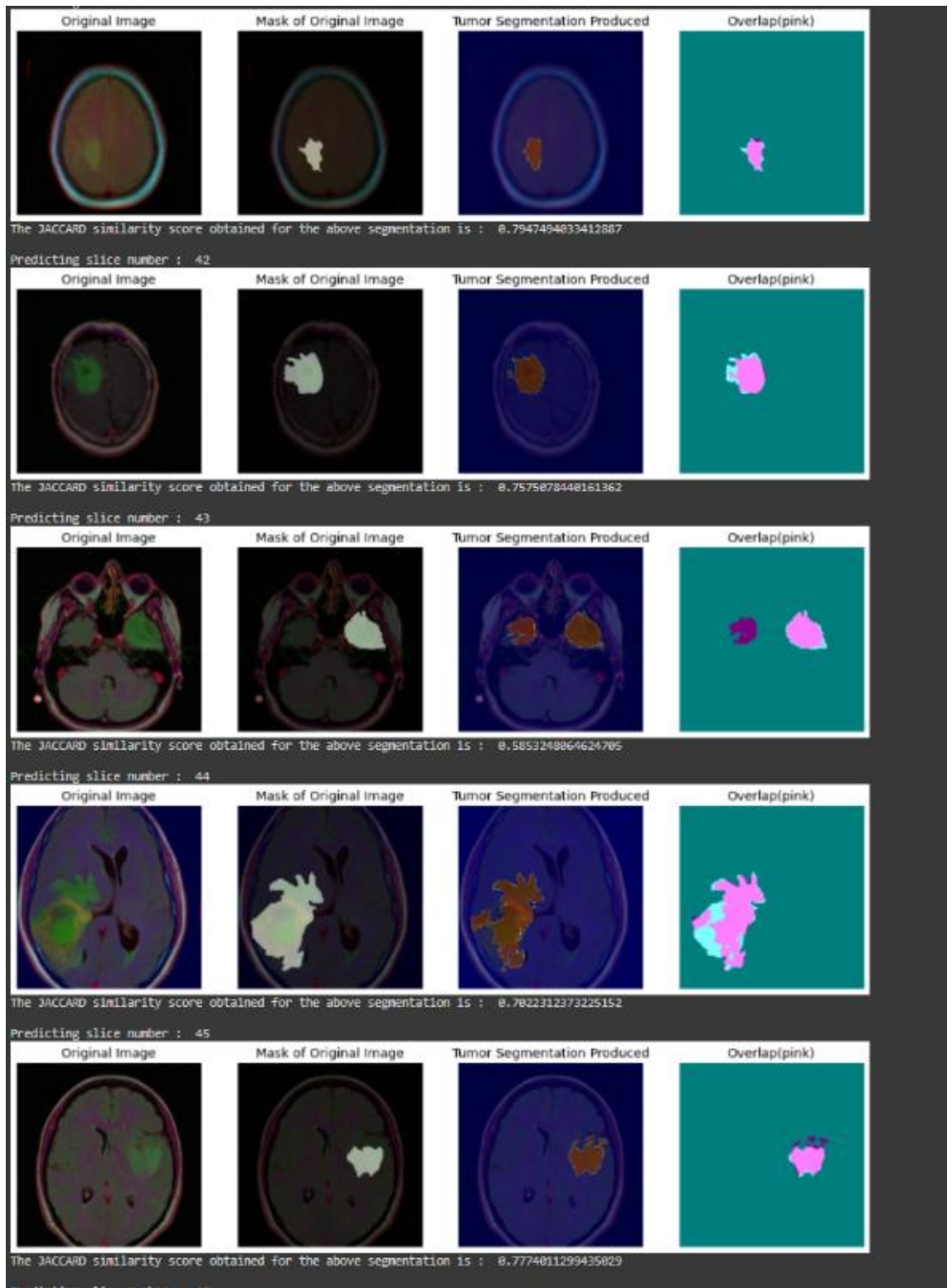


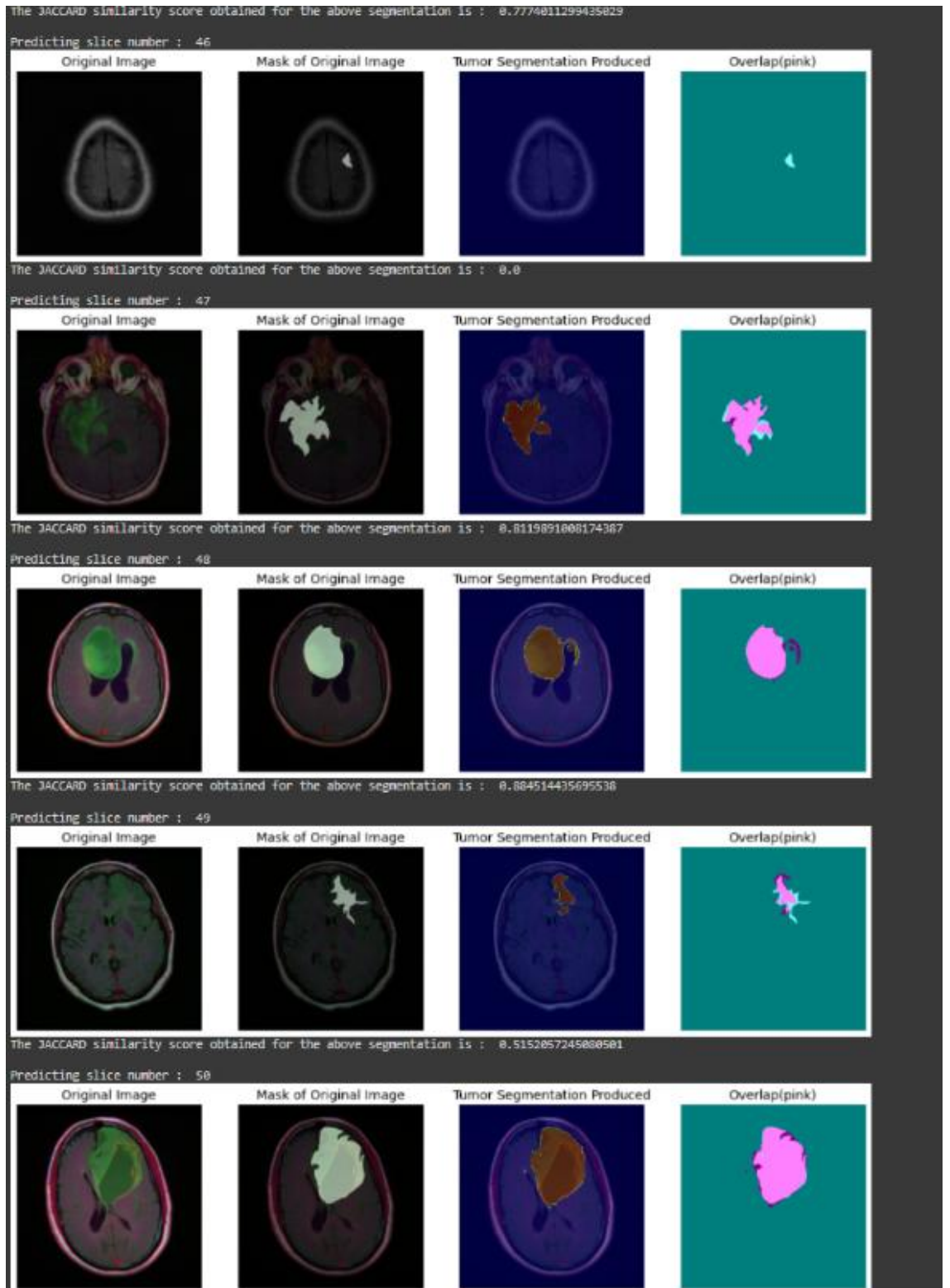


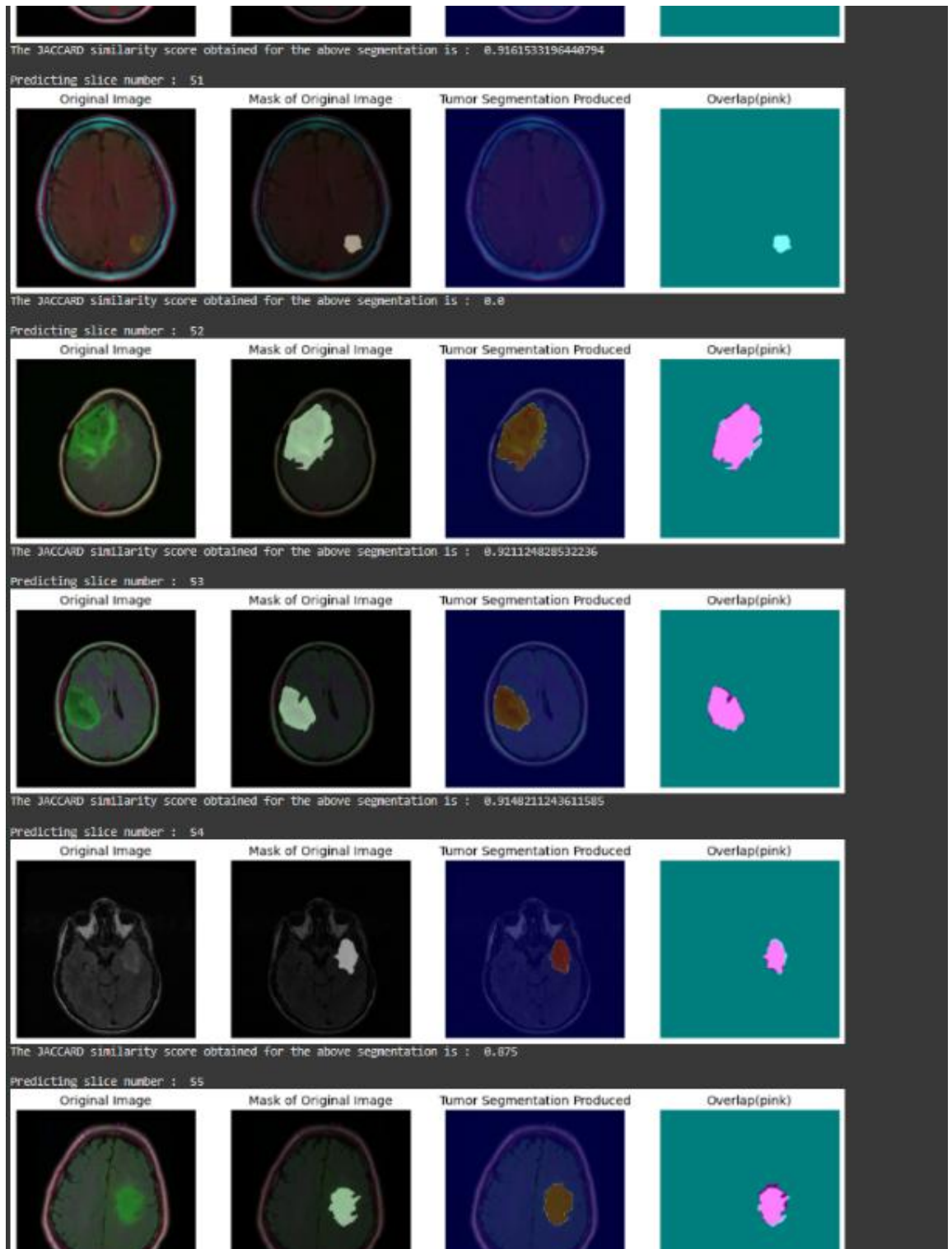


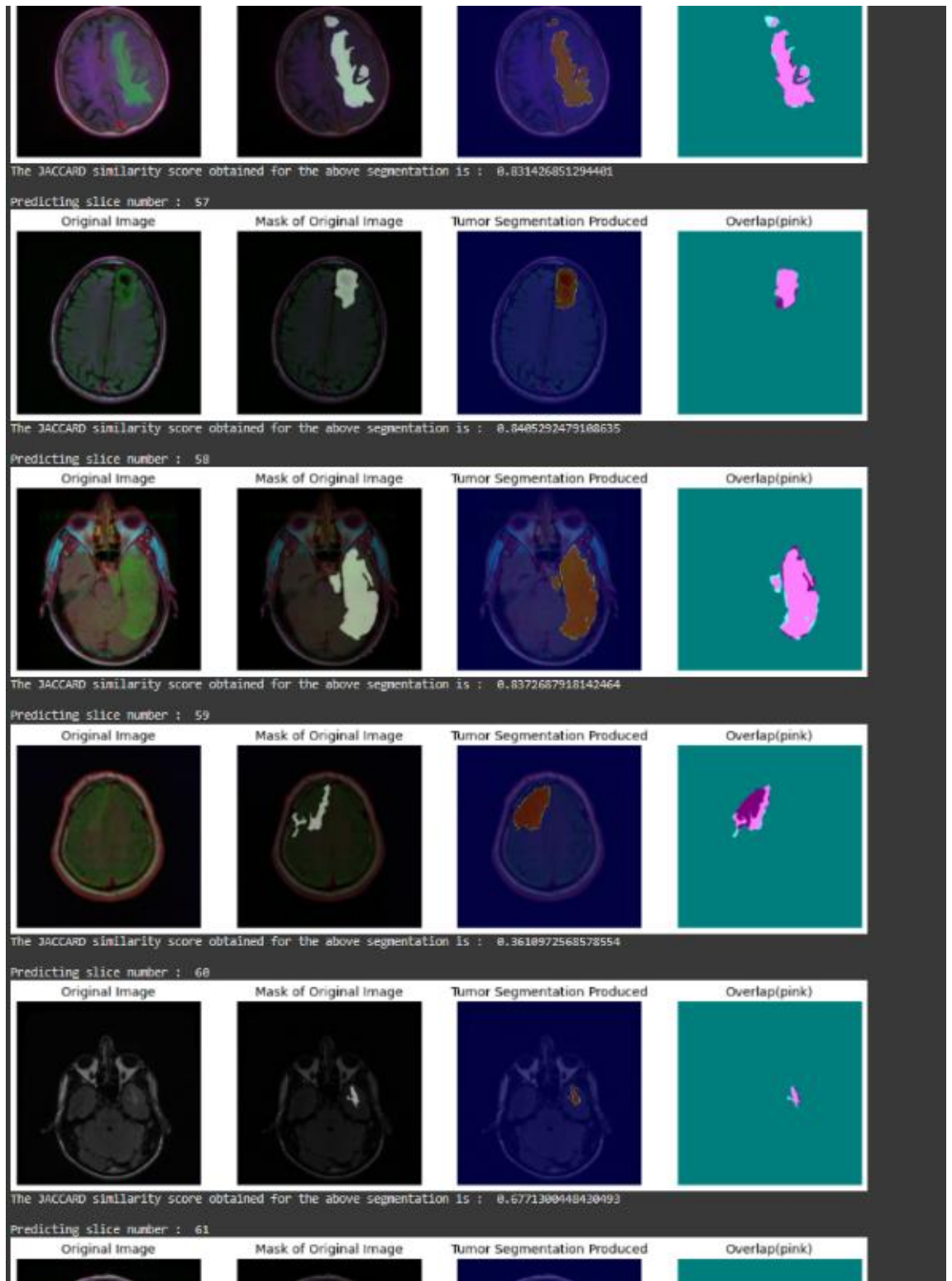


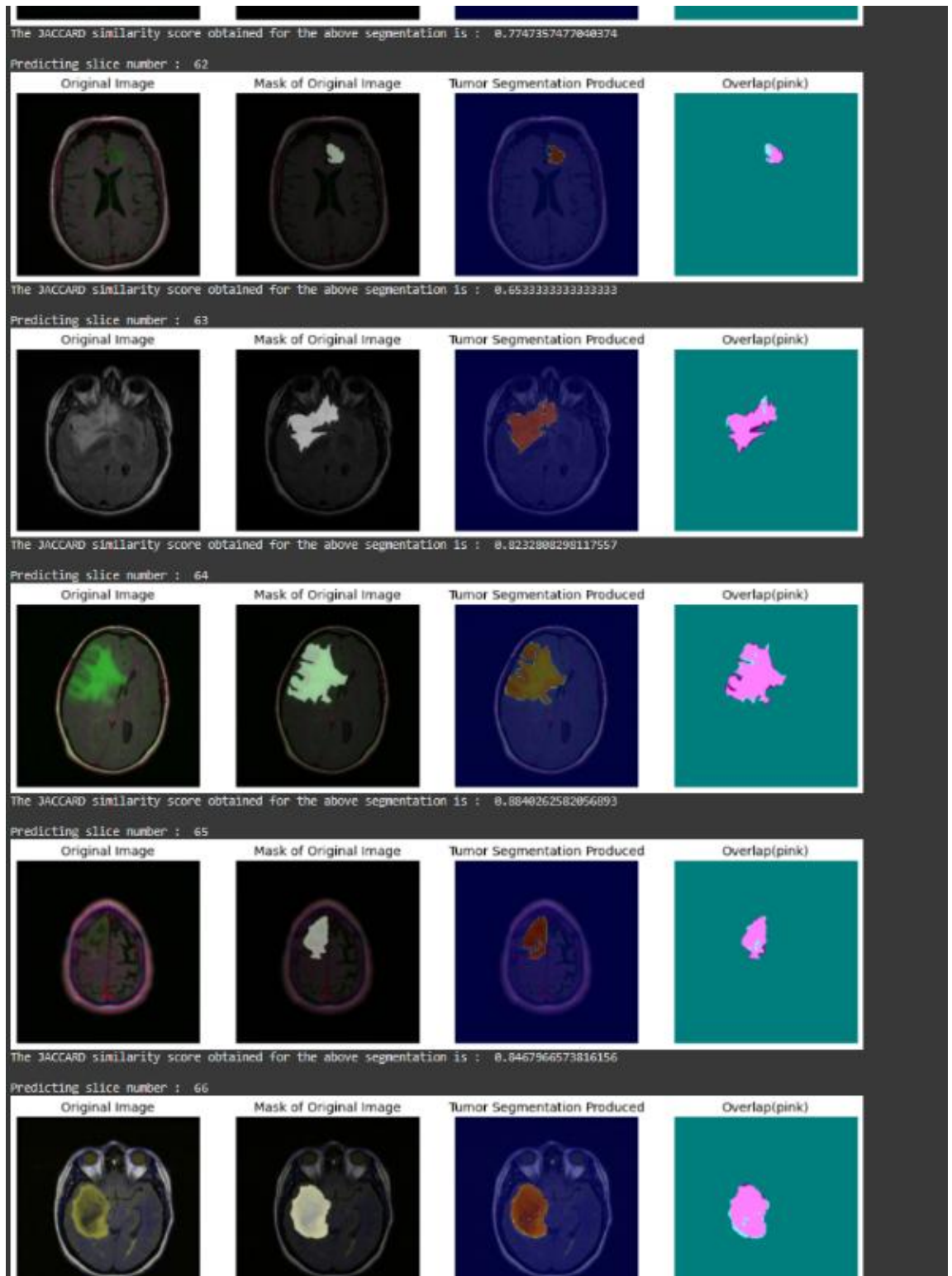


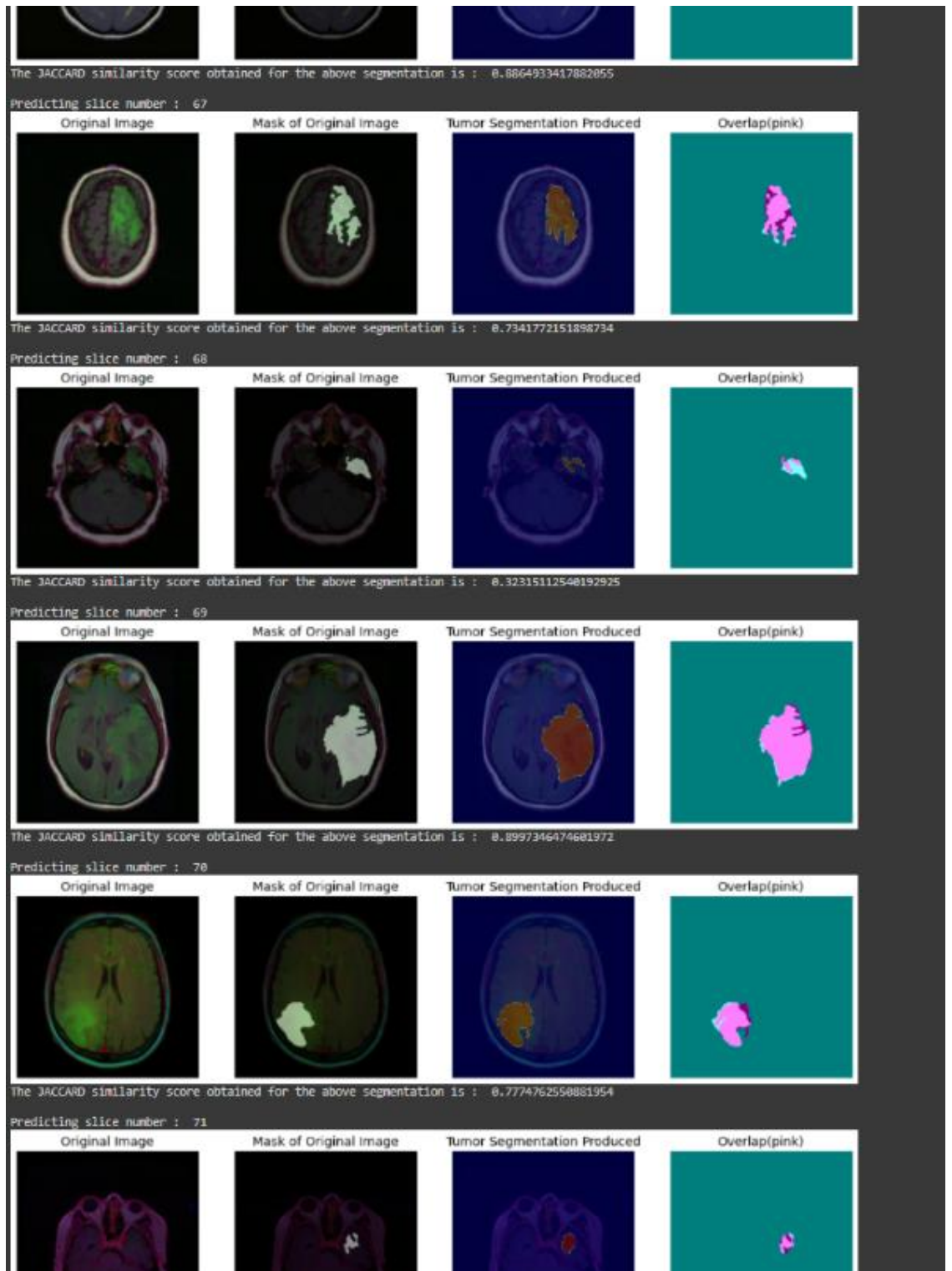


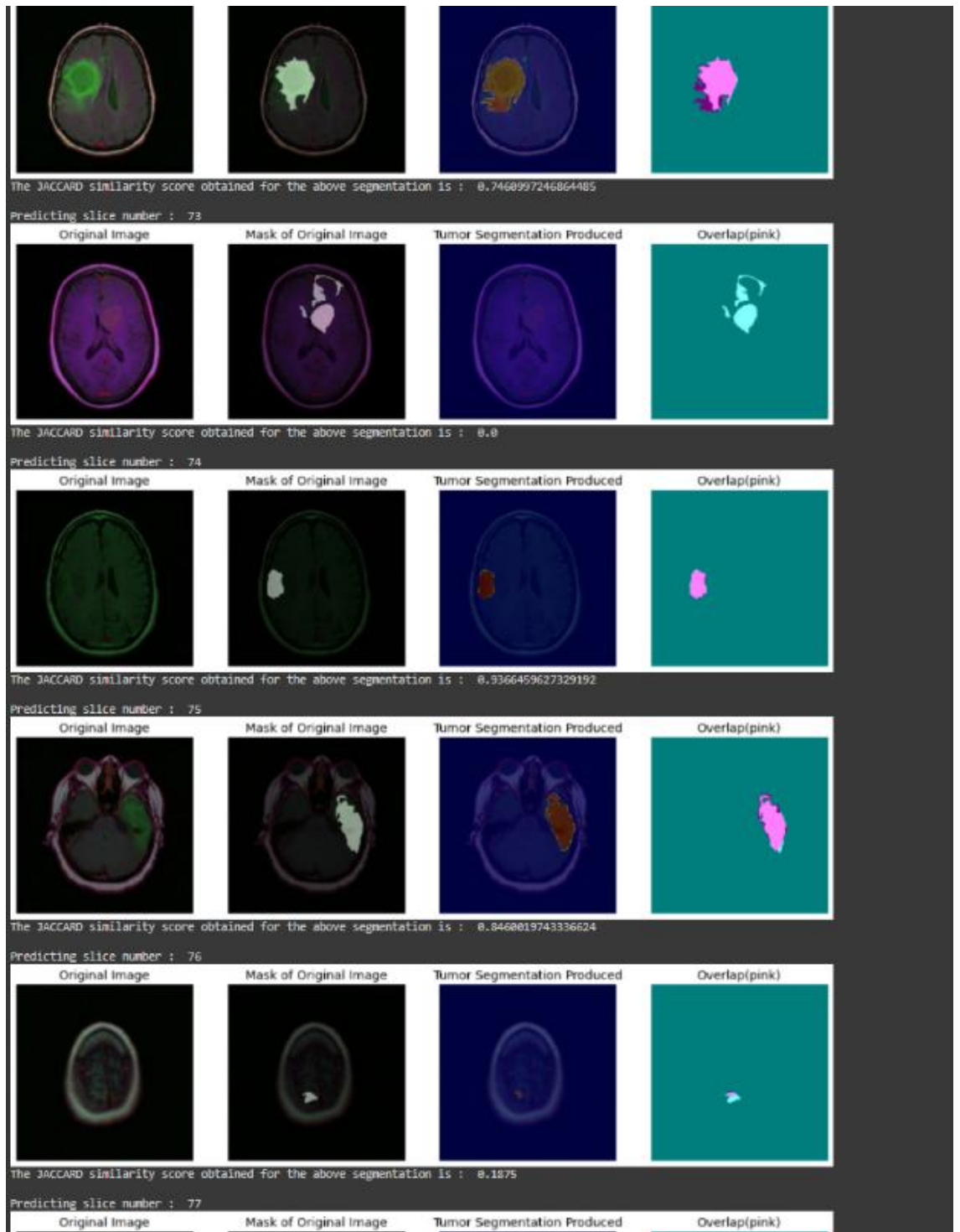


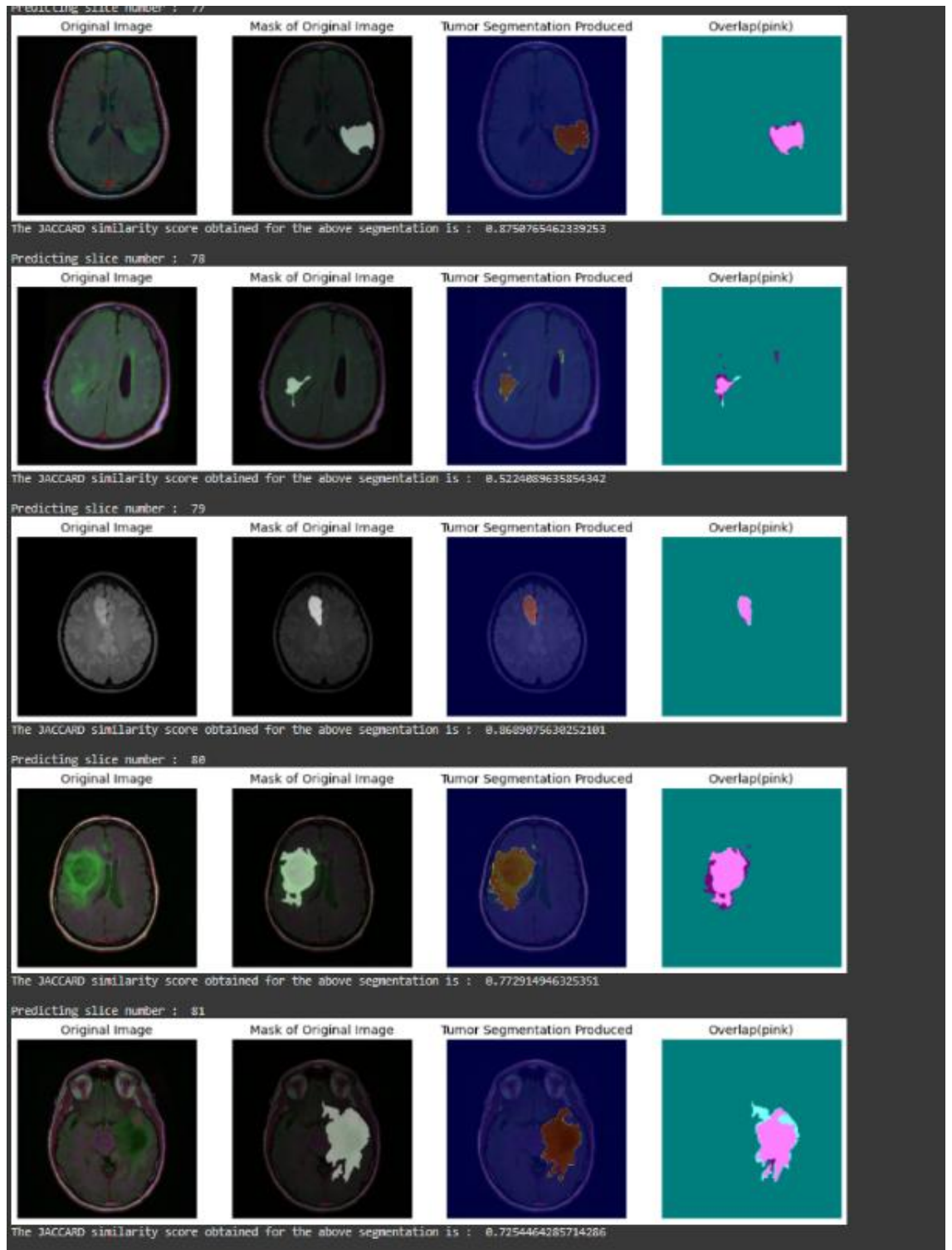


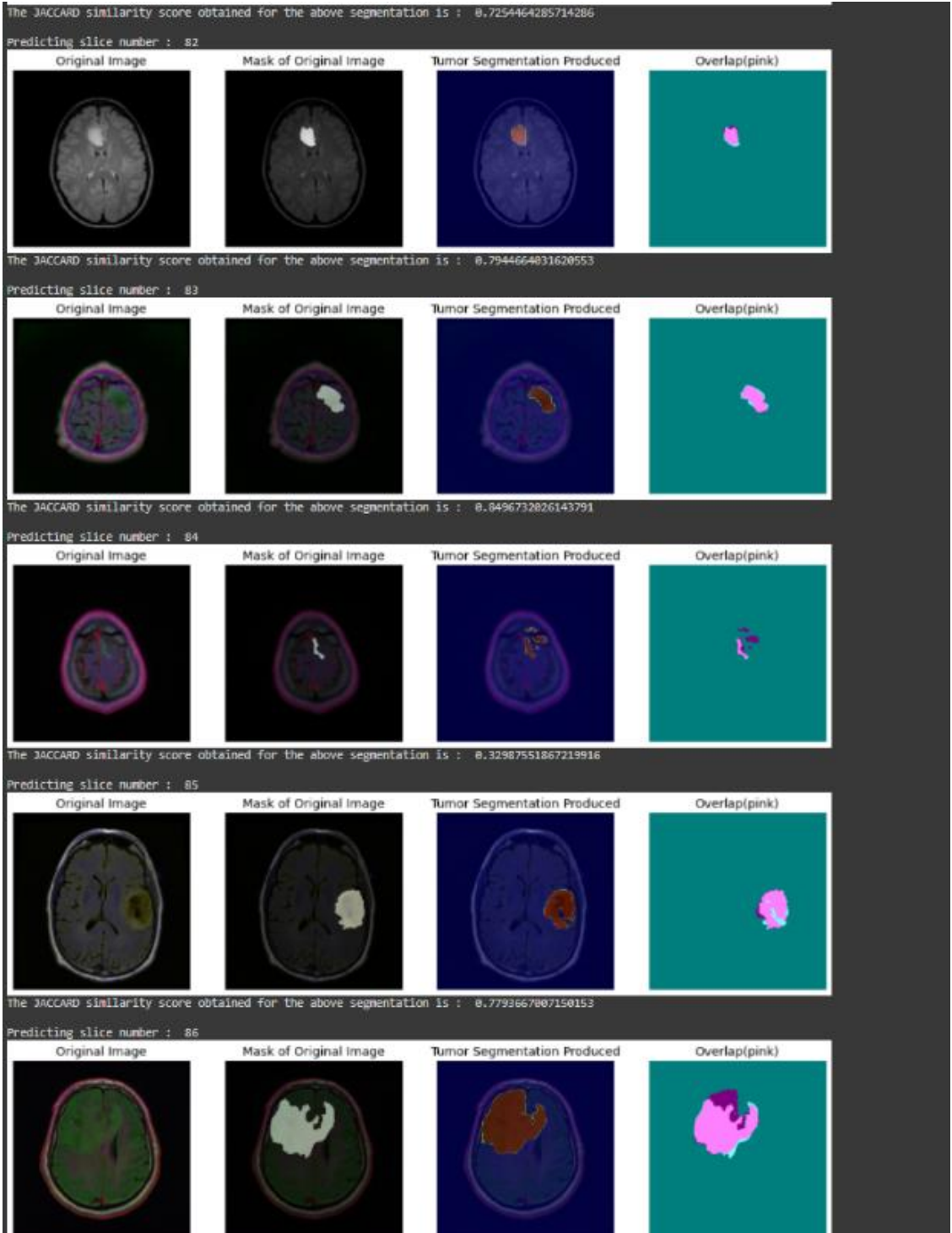


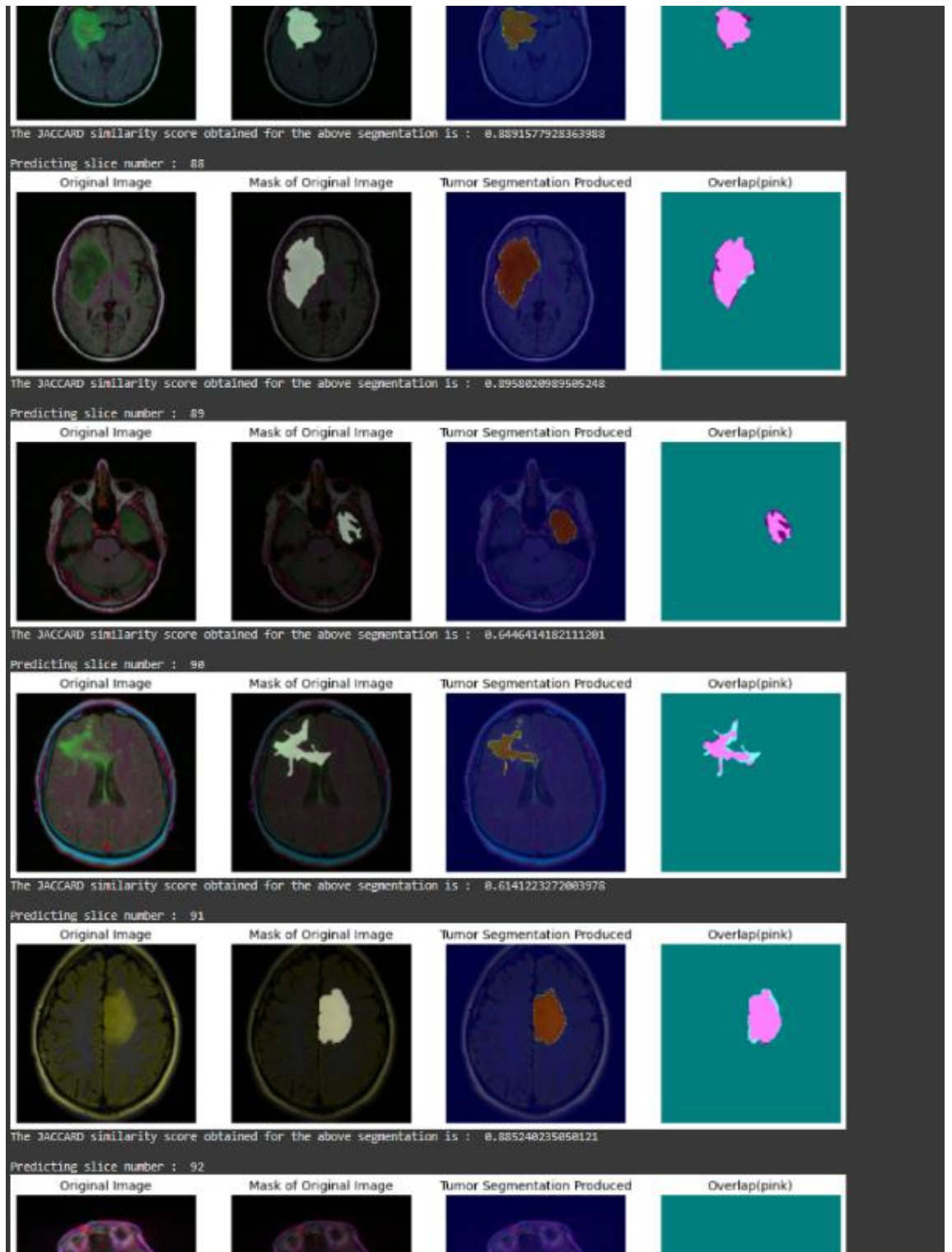


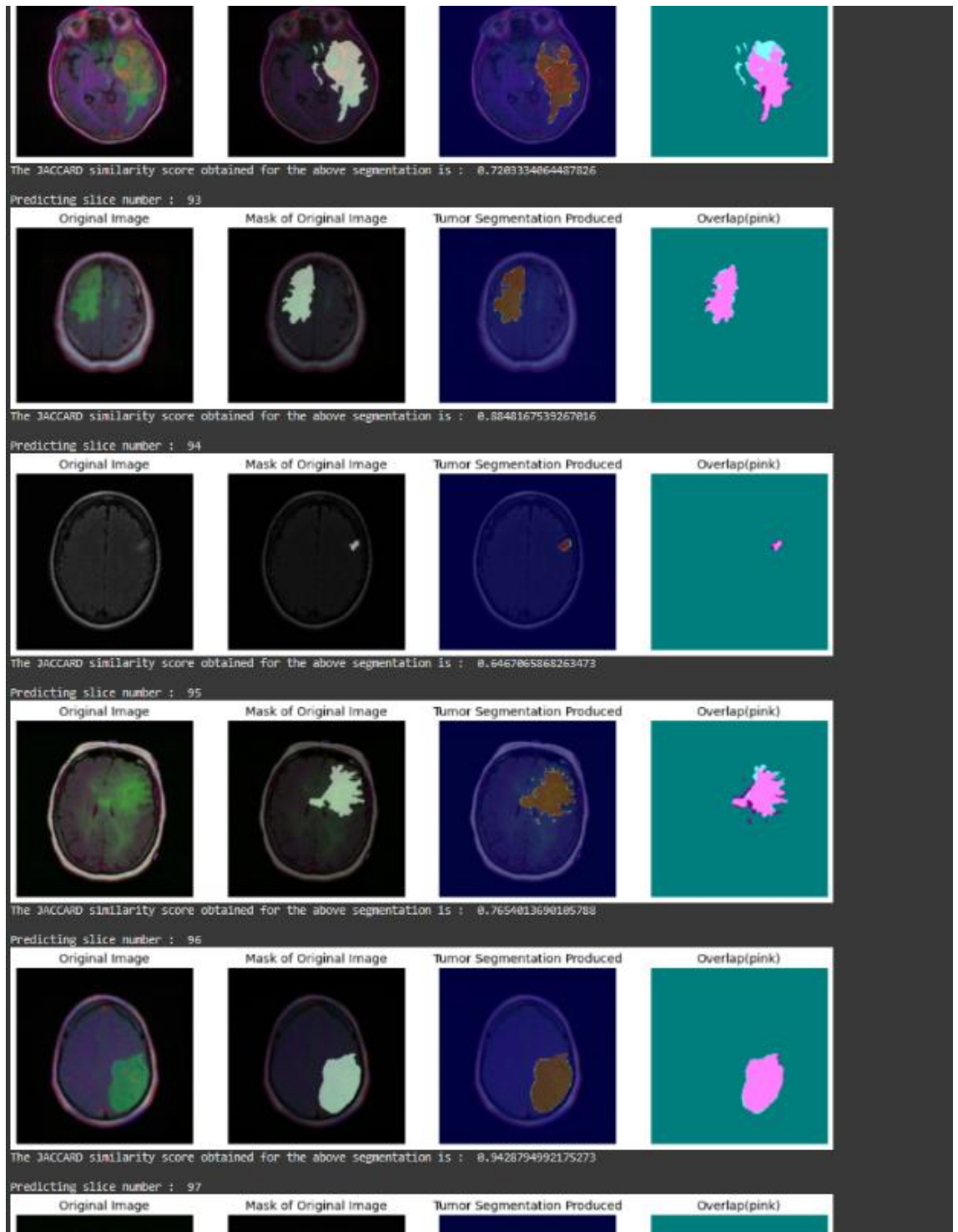


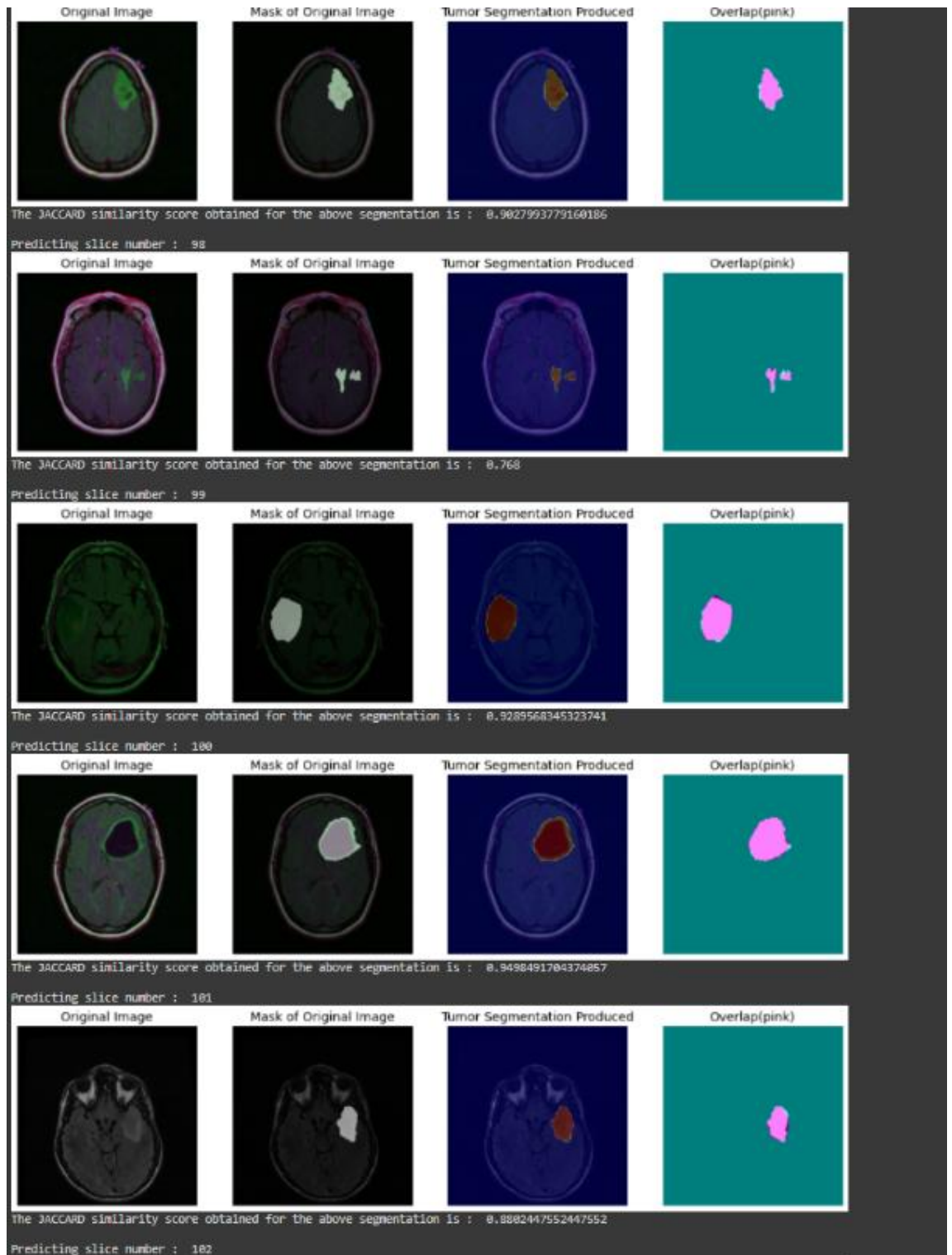


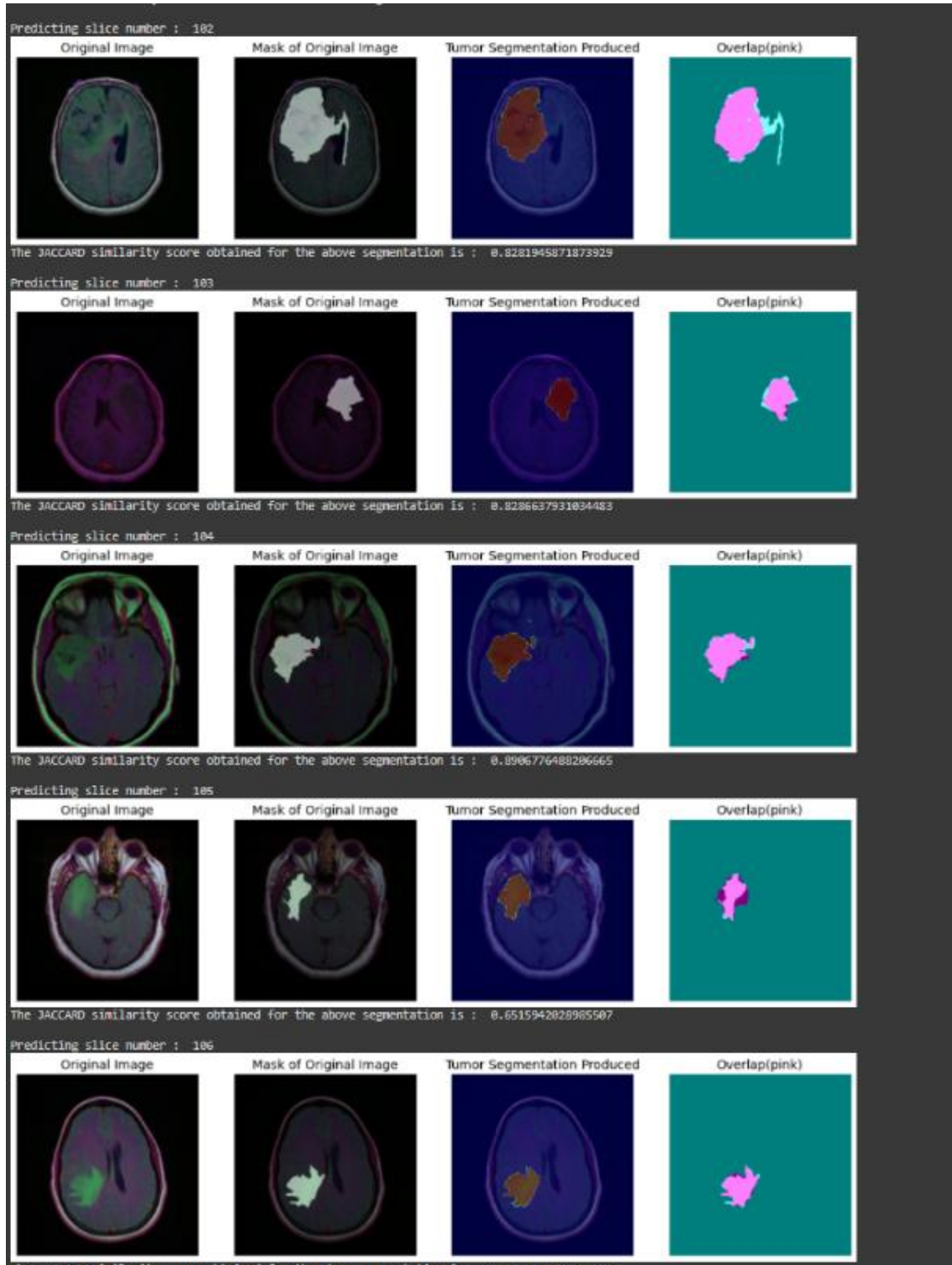


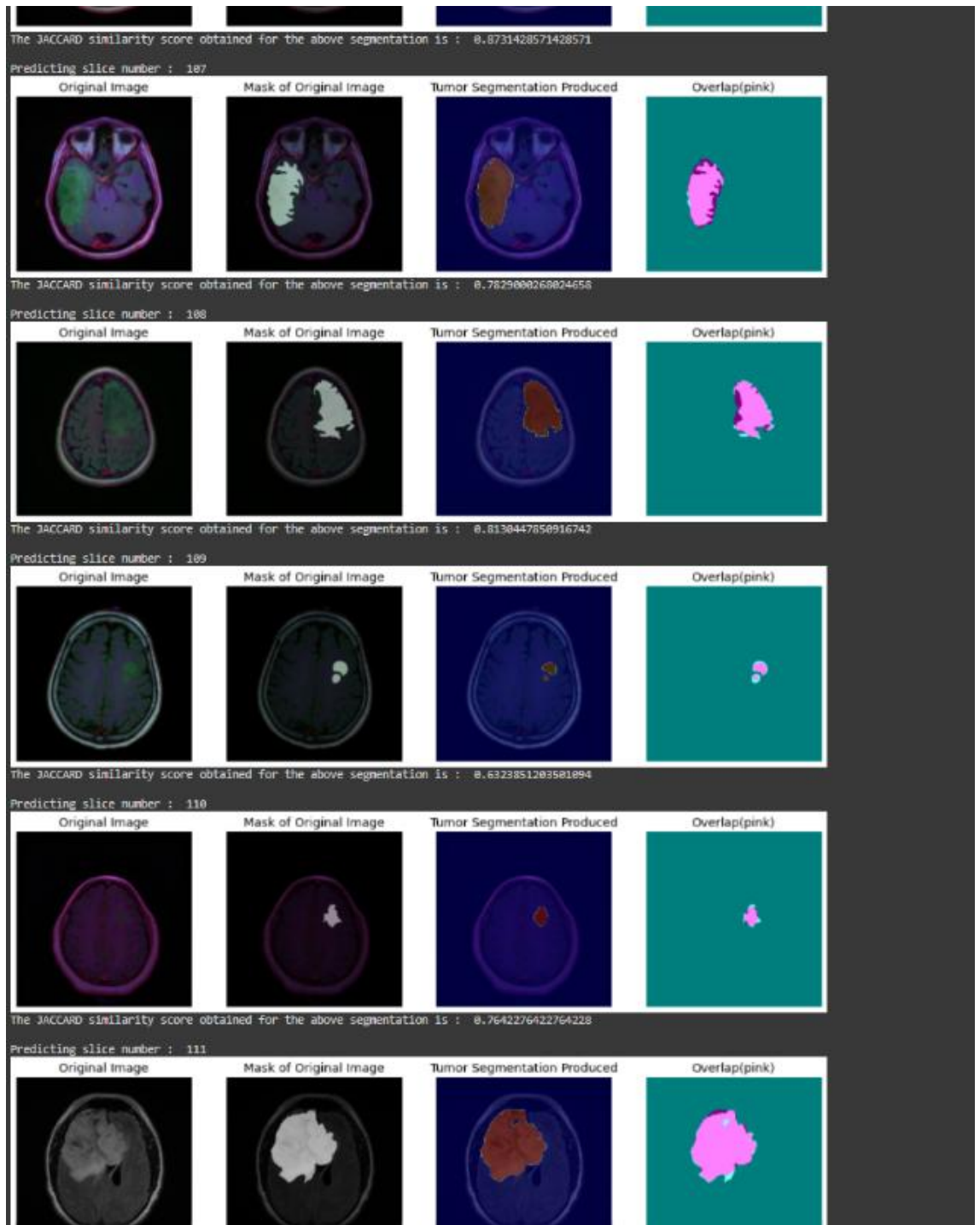


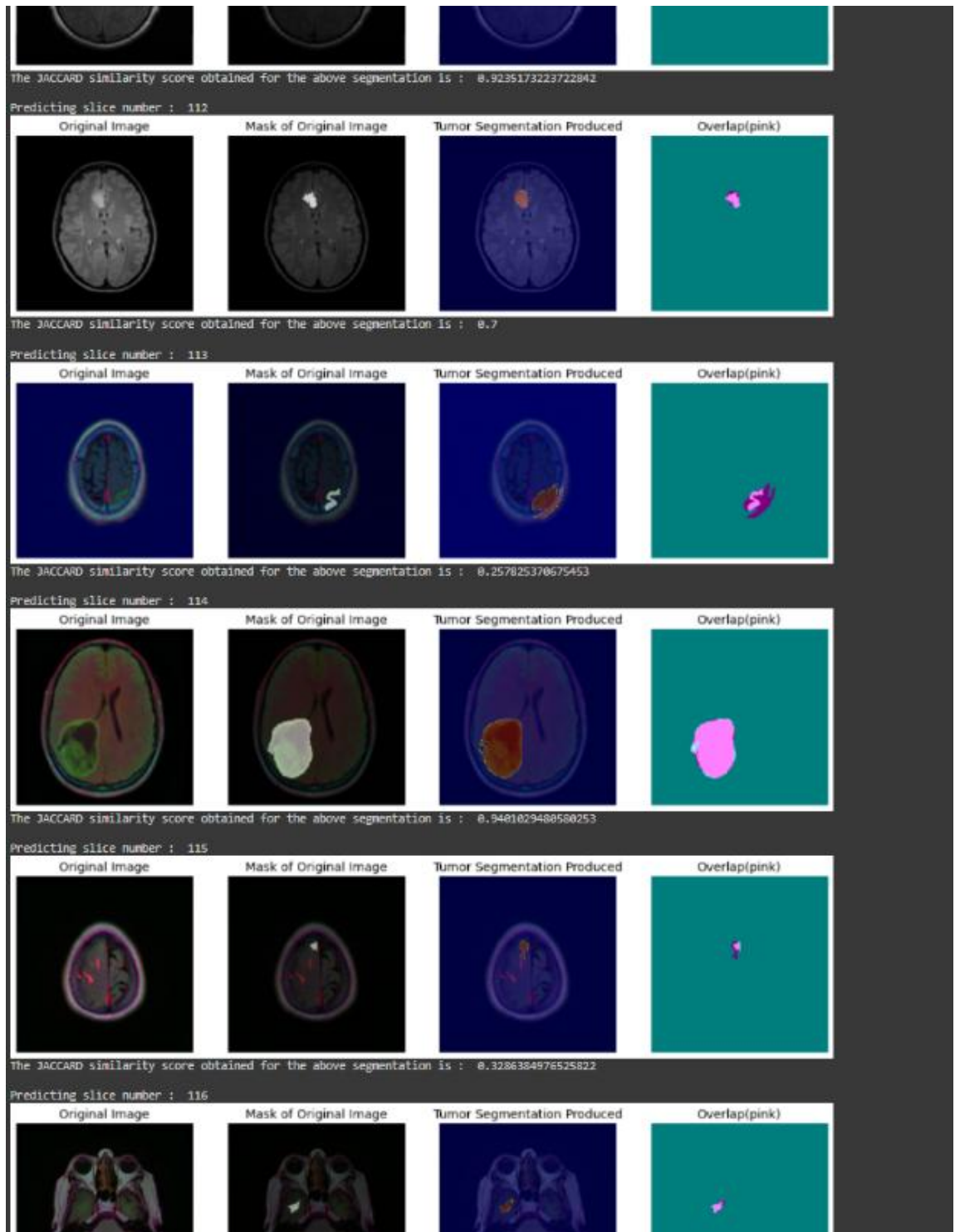


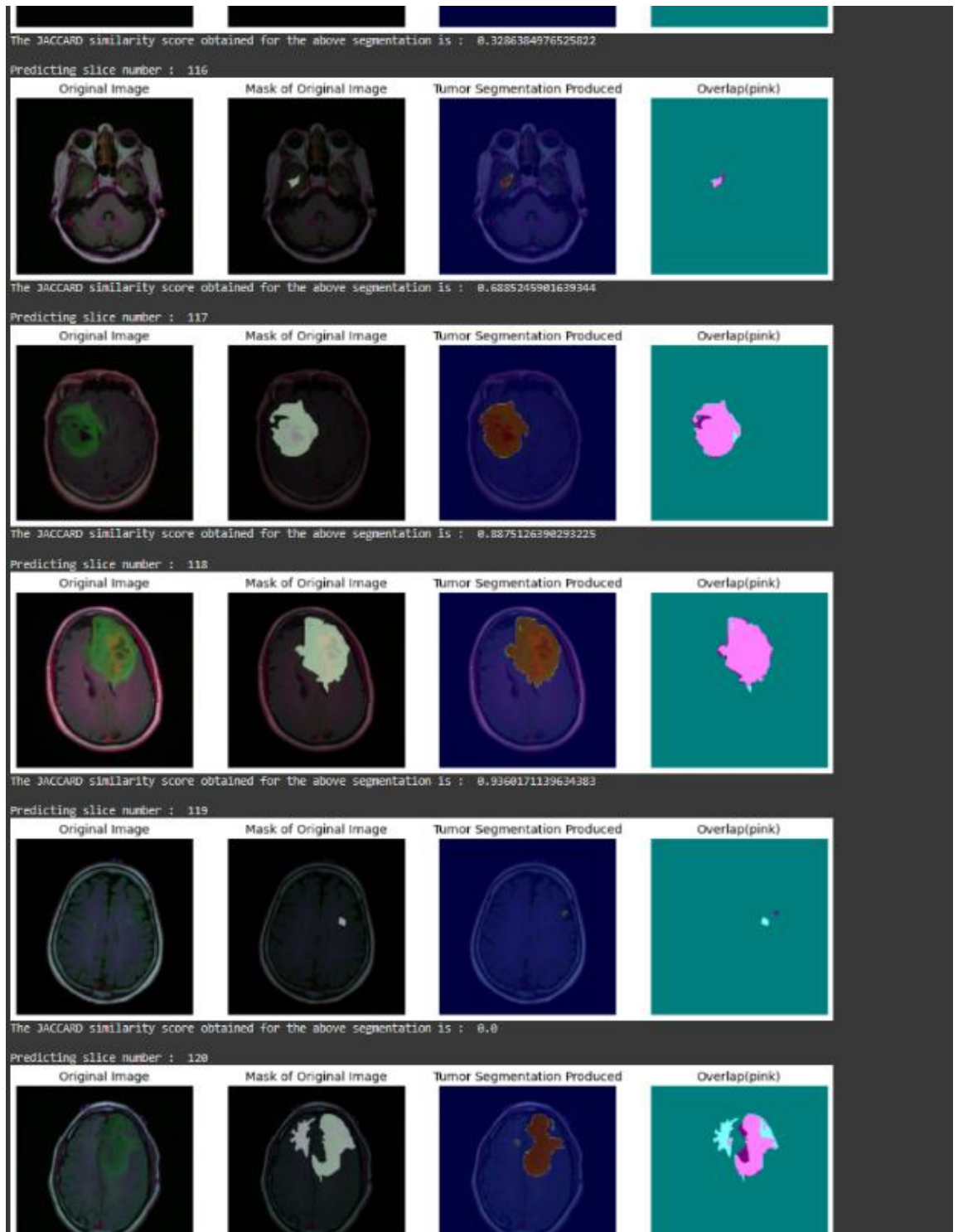


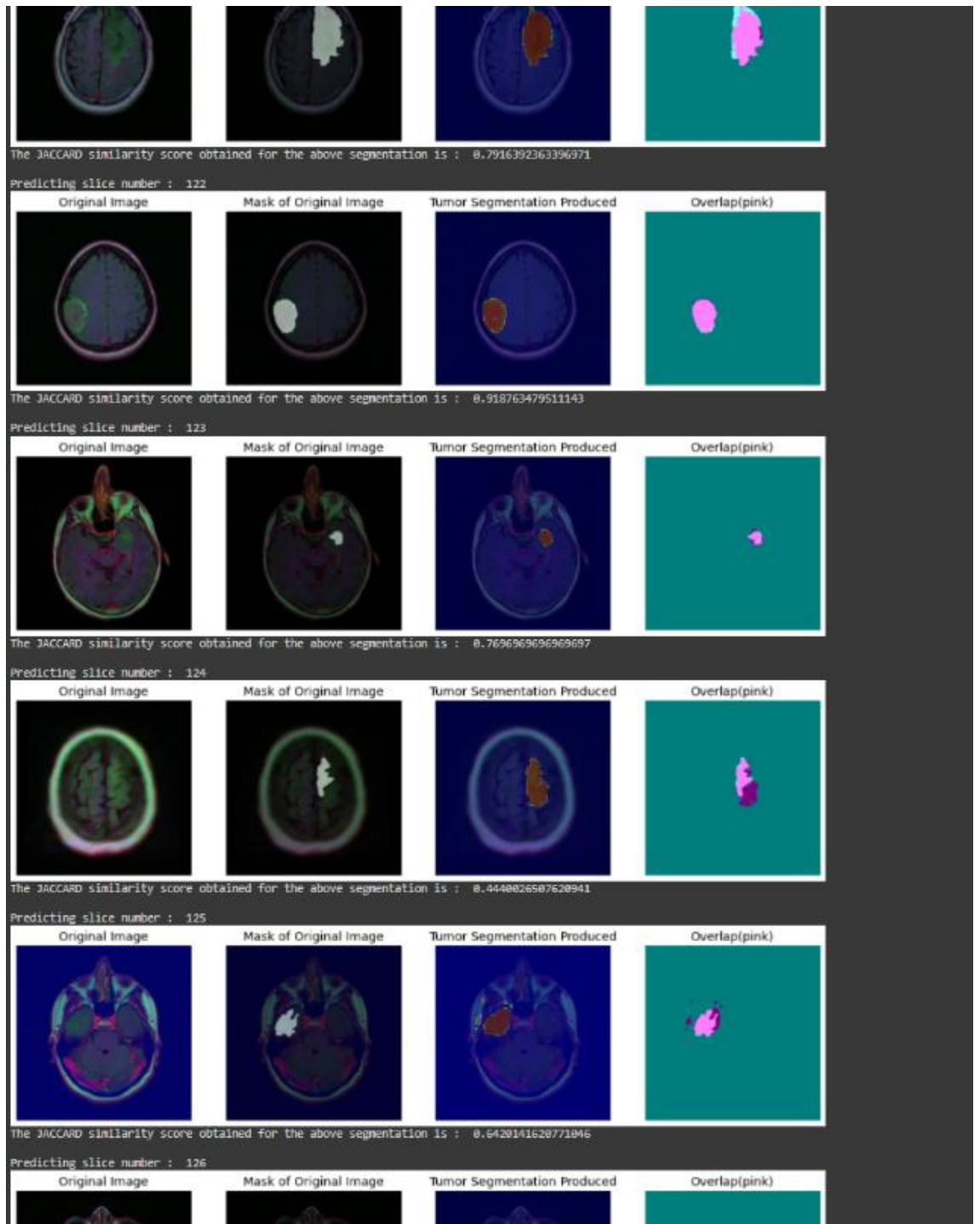


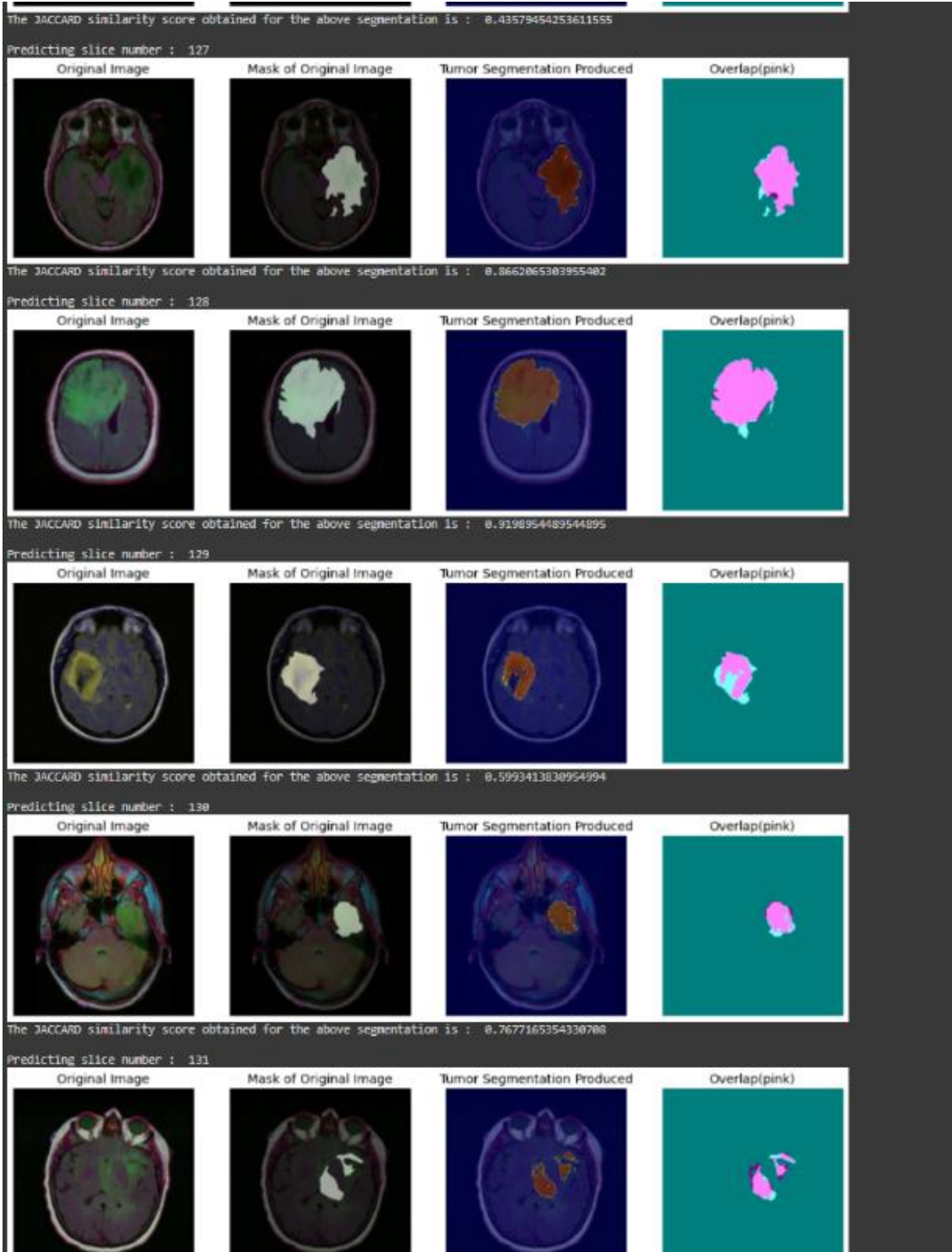


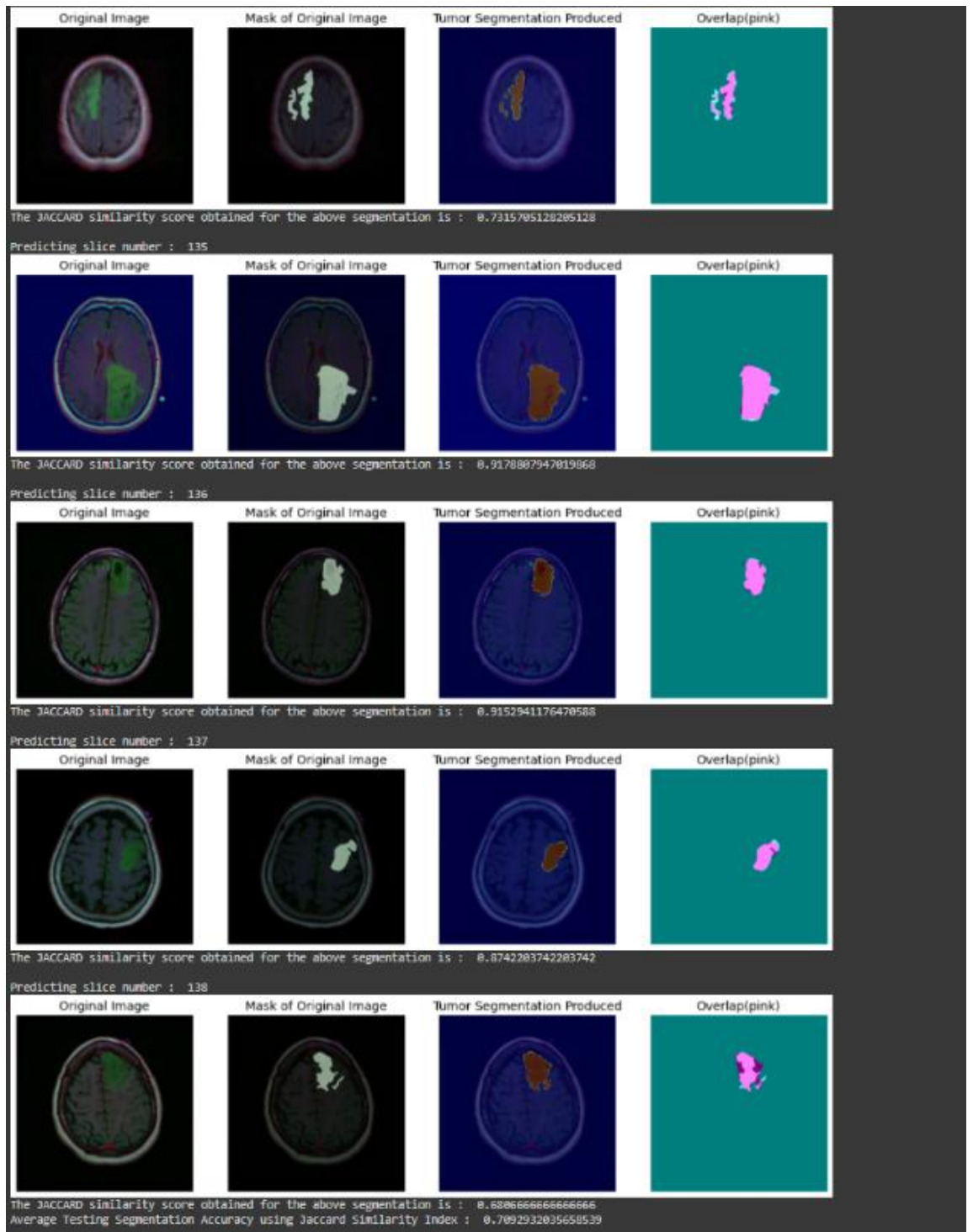




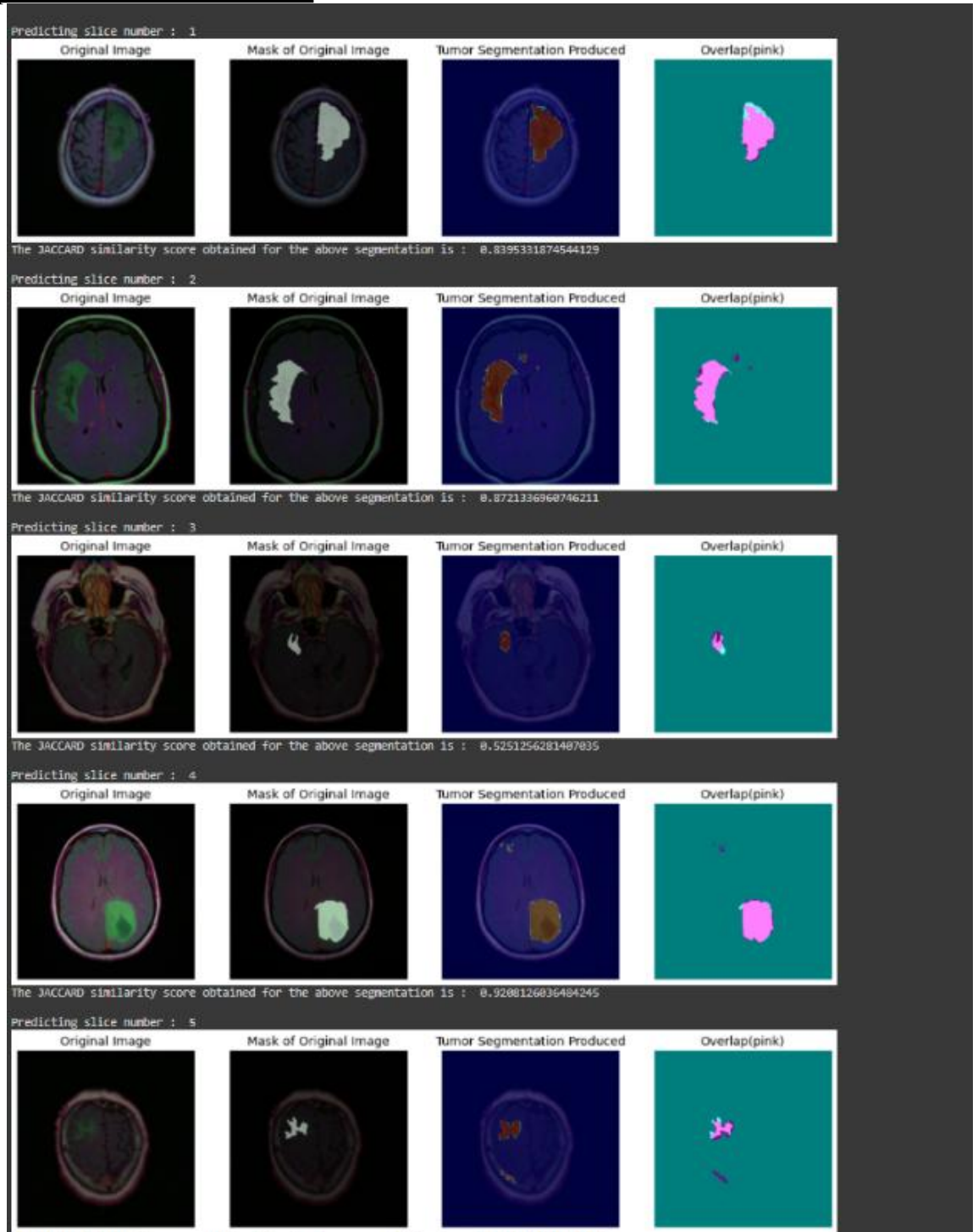


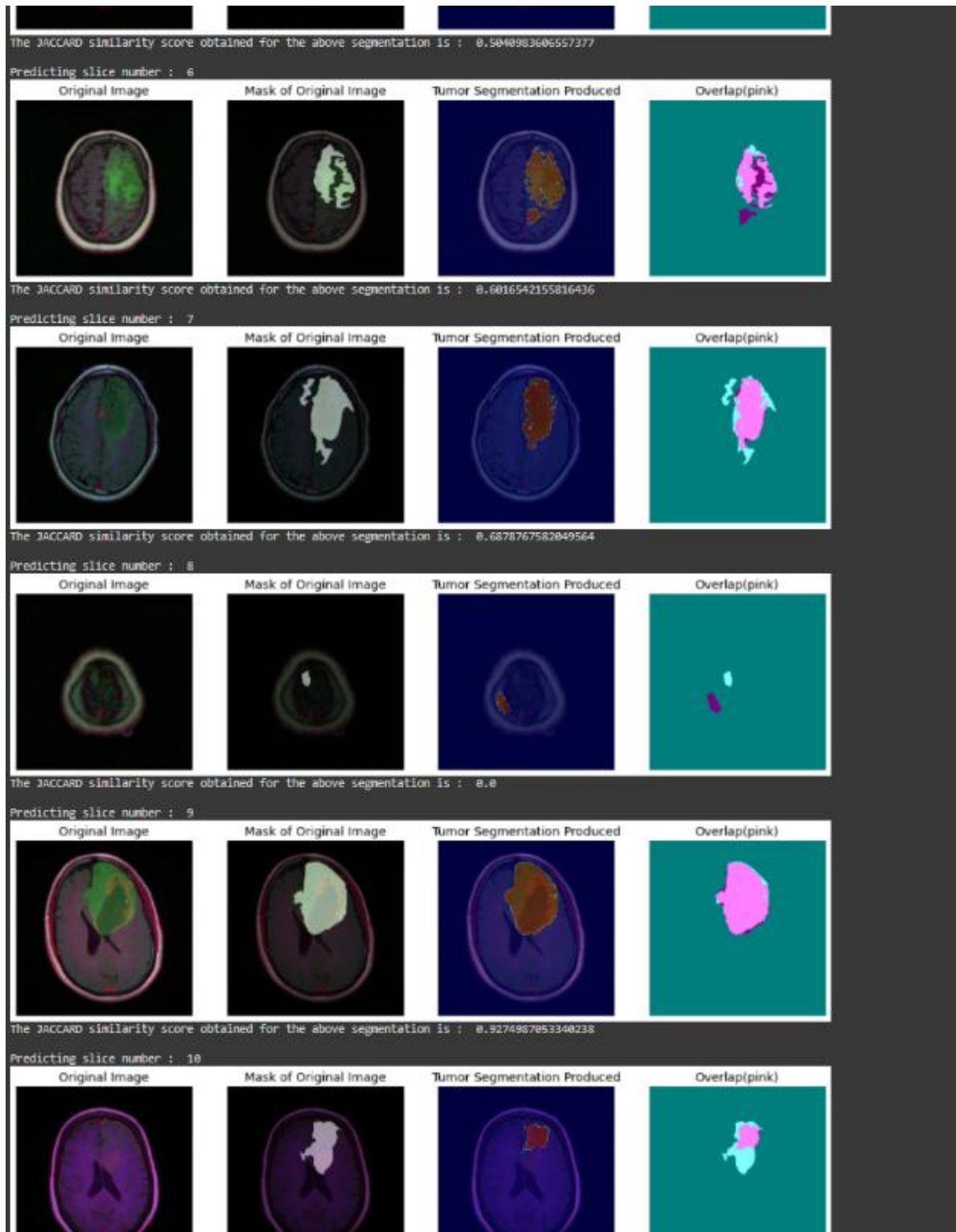


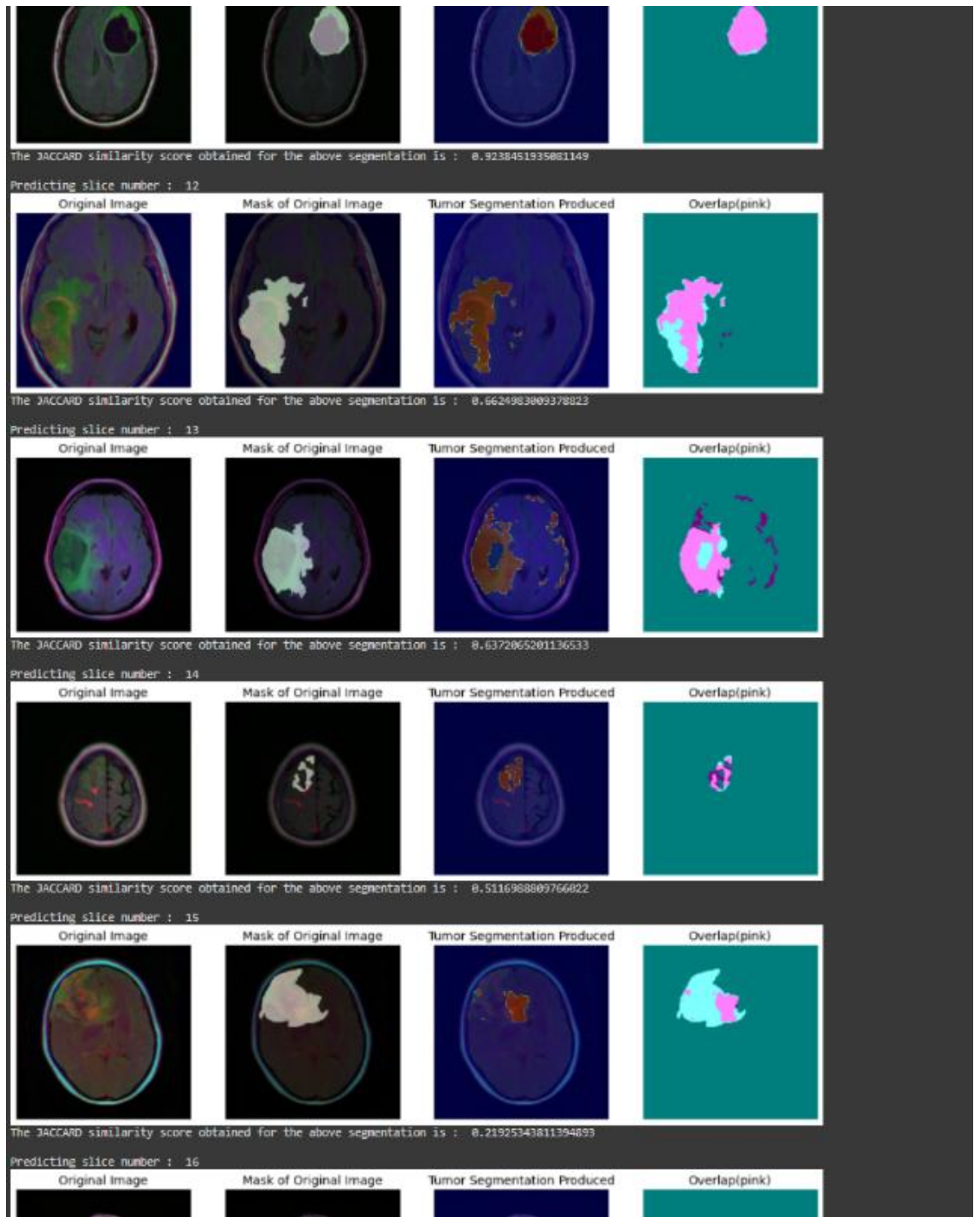


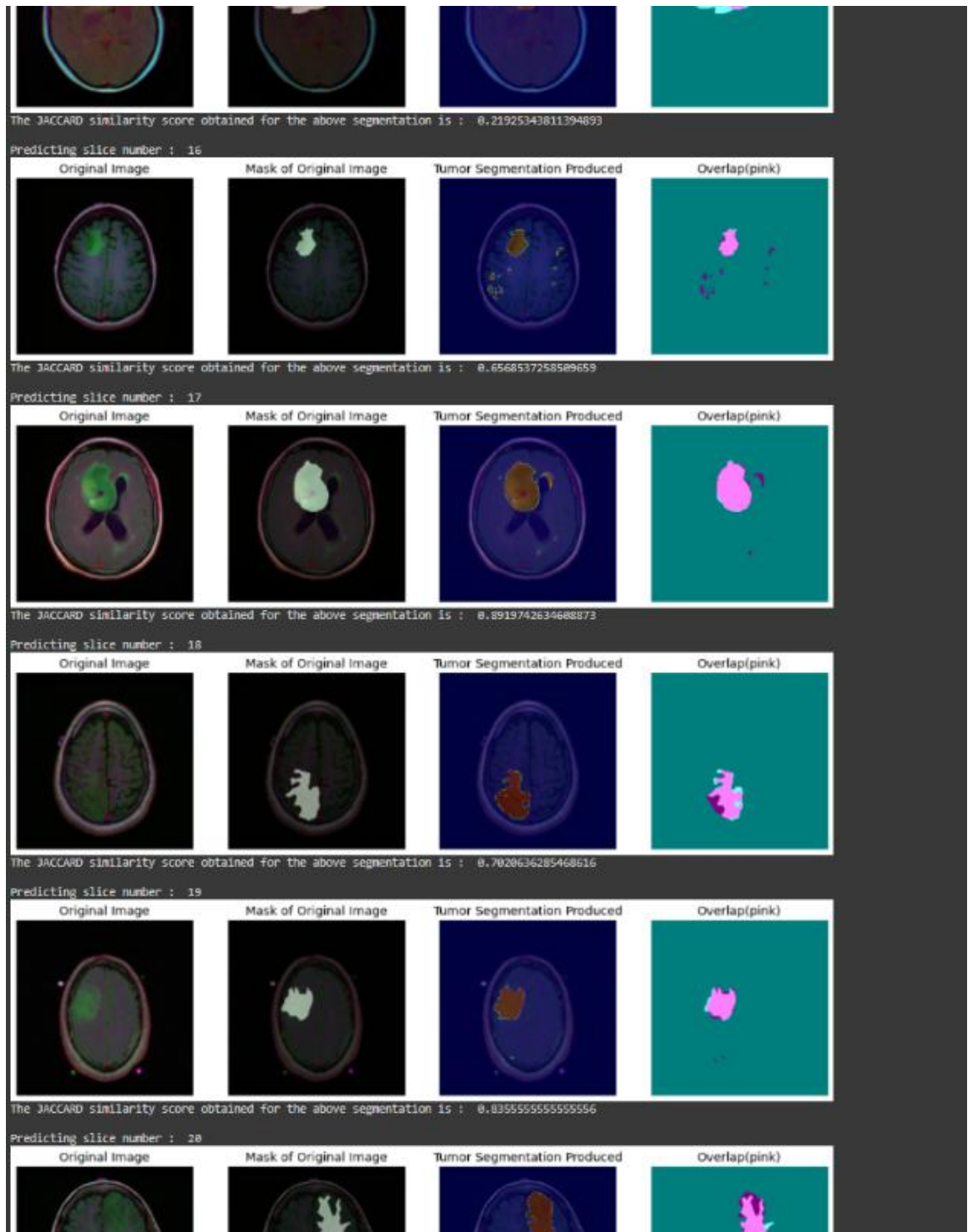


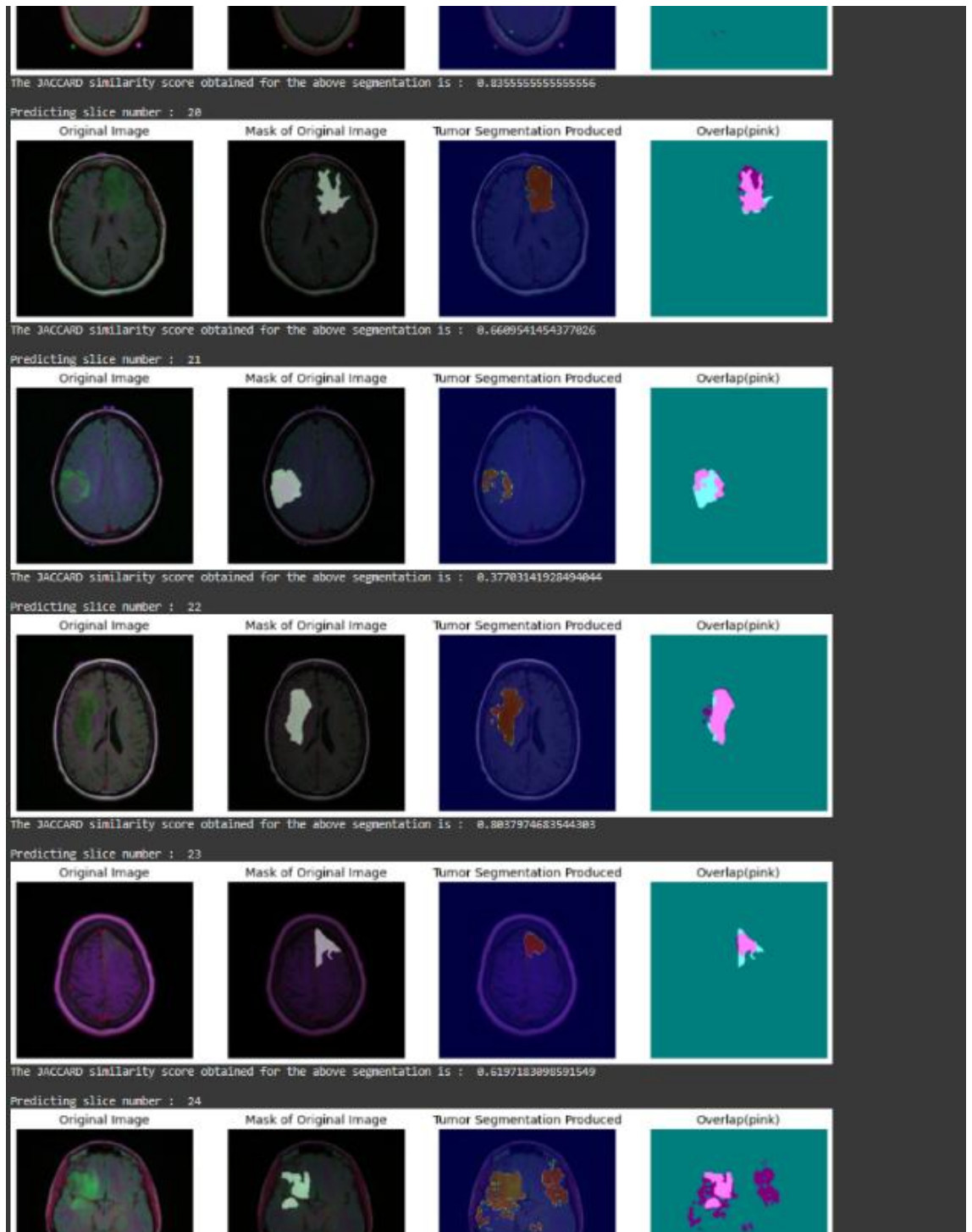
4. Experimental Model No. 9

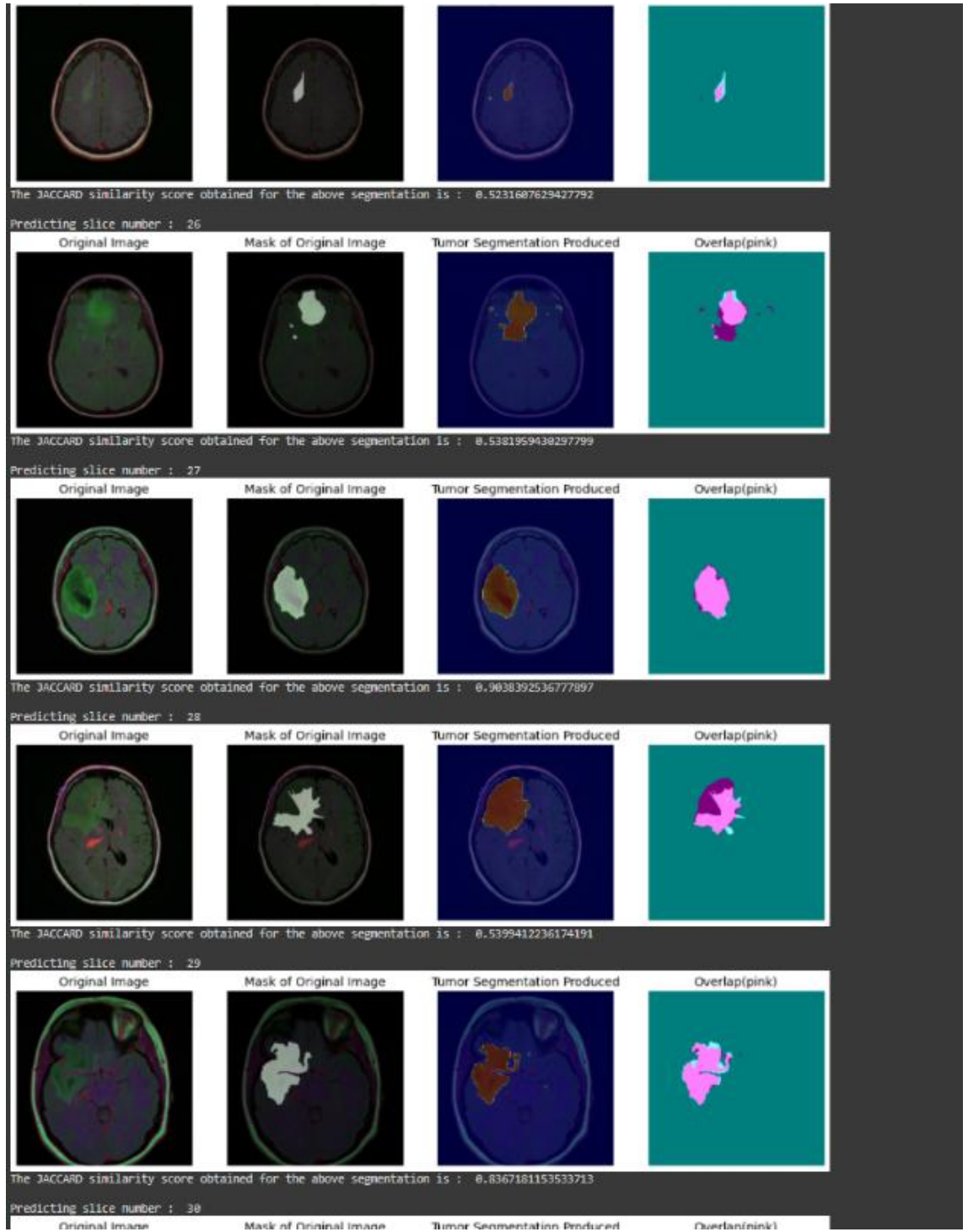


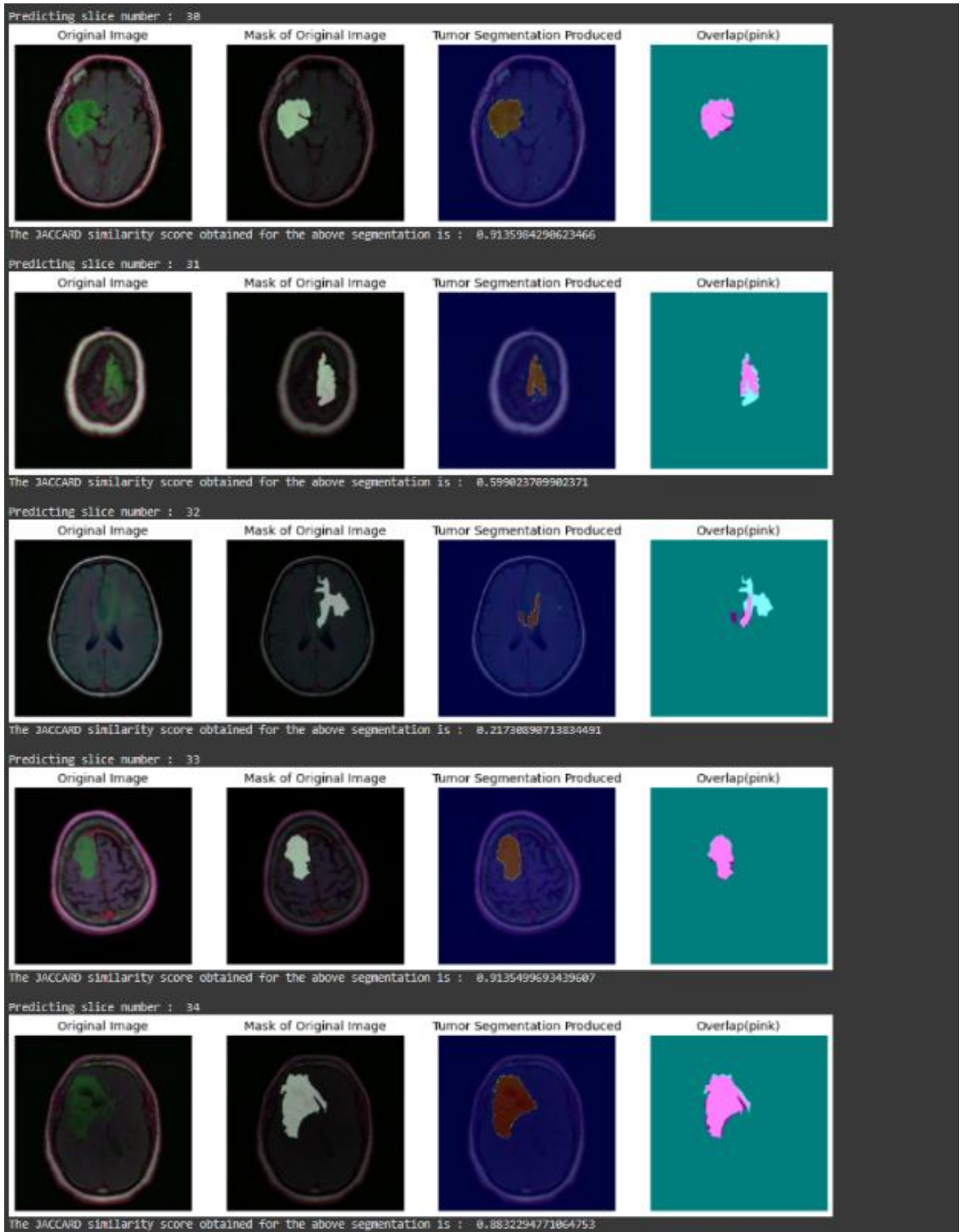


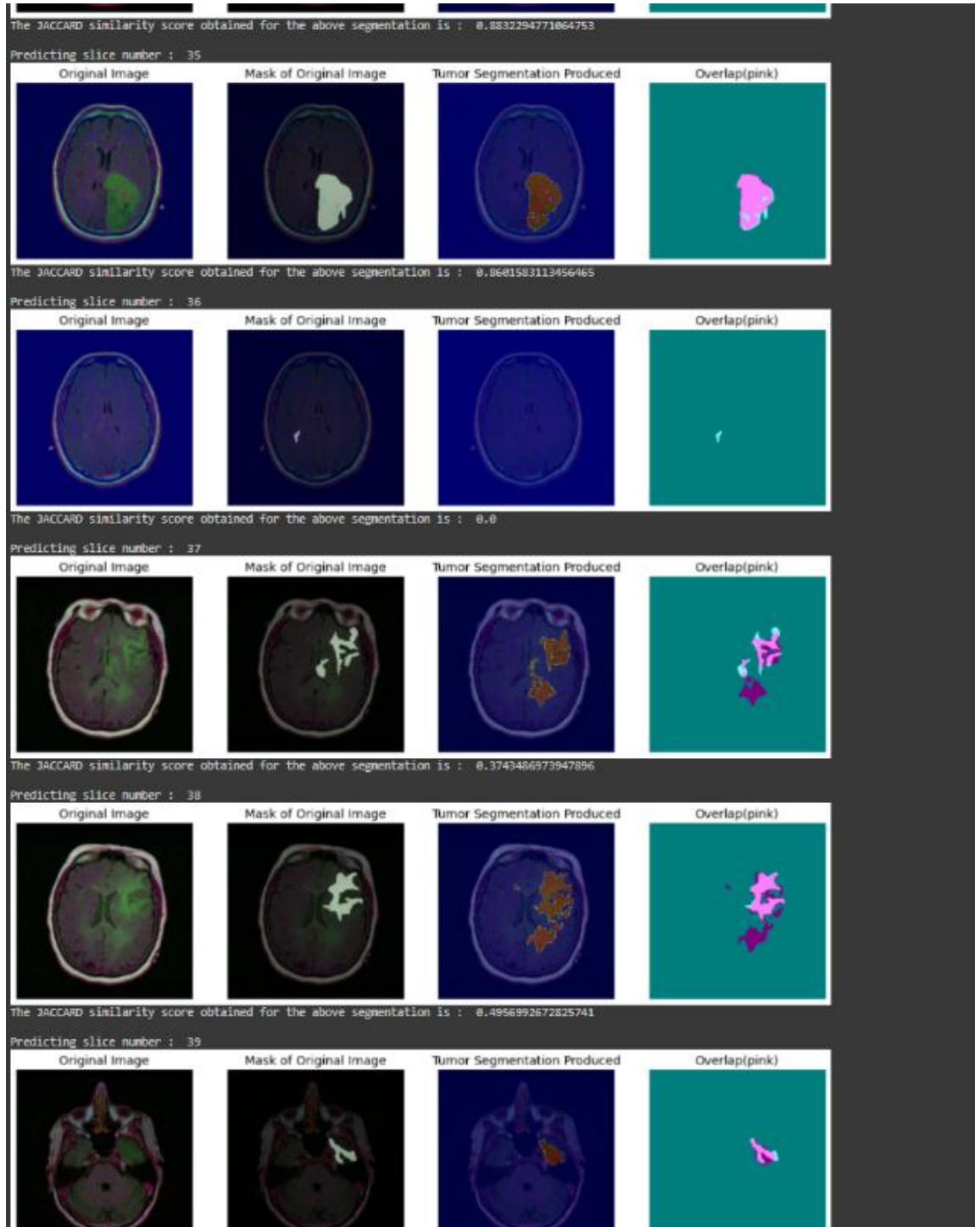


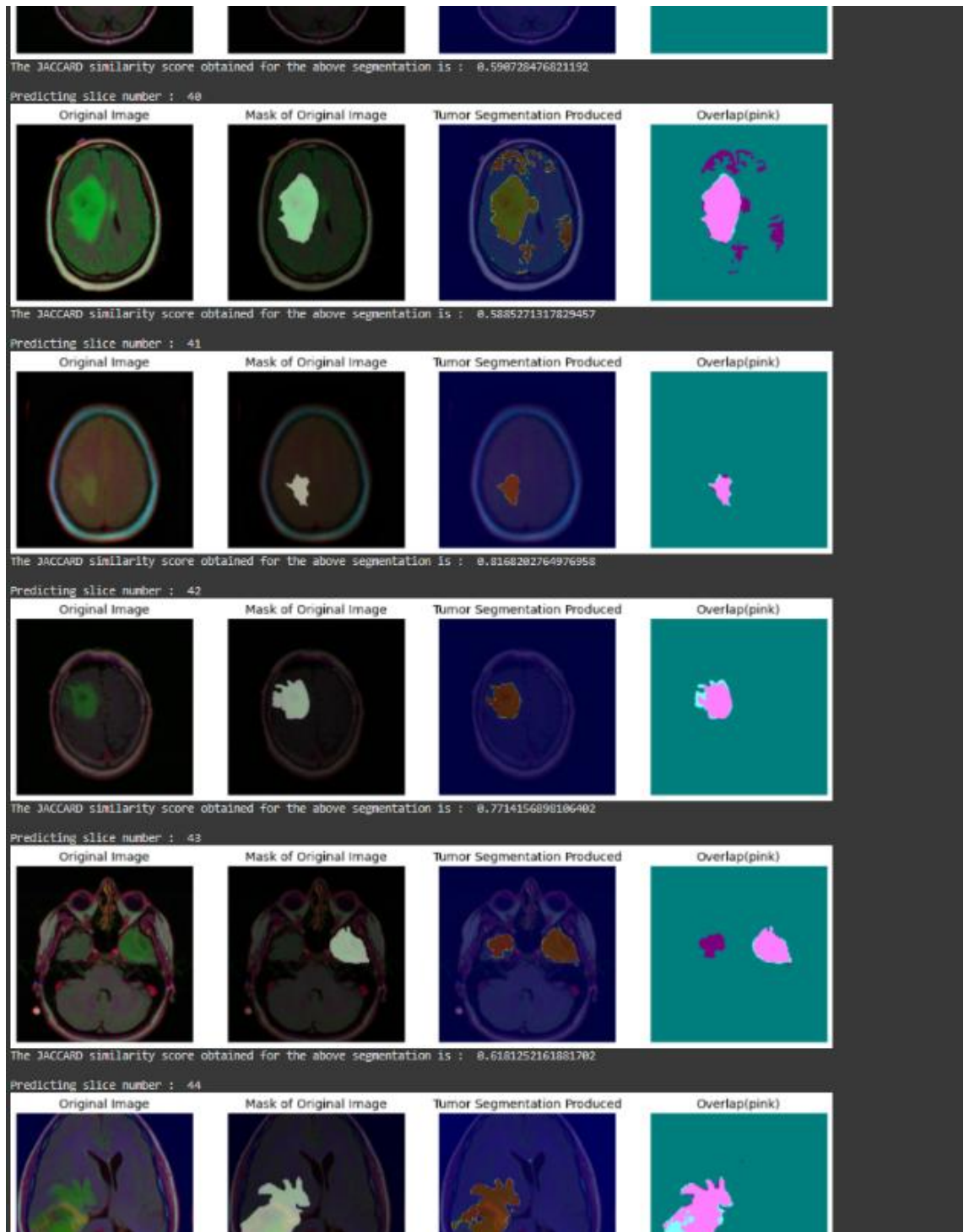


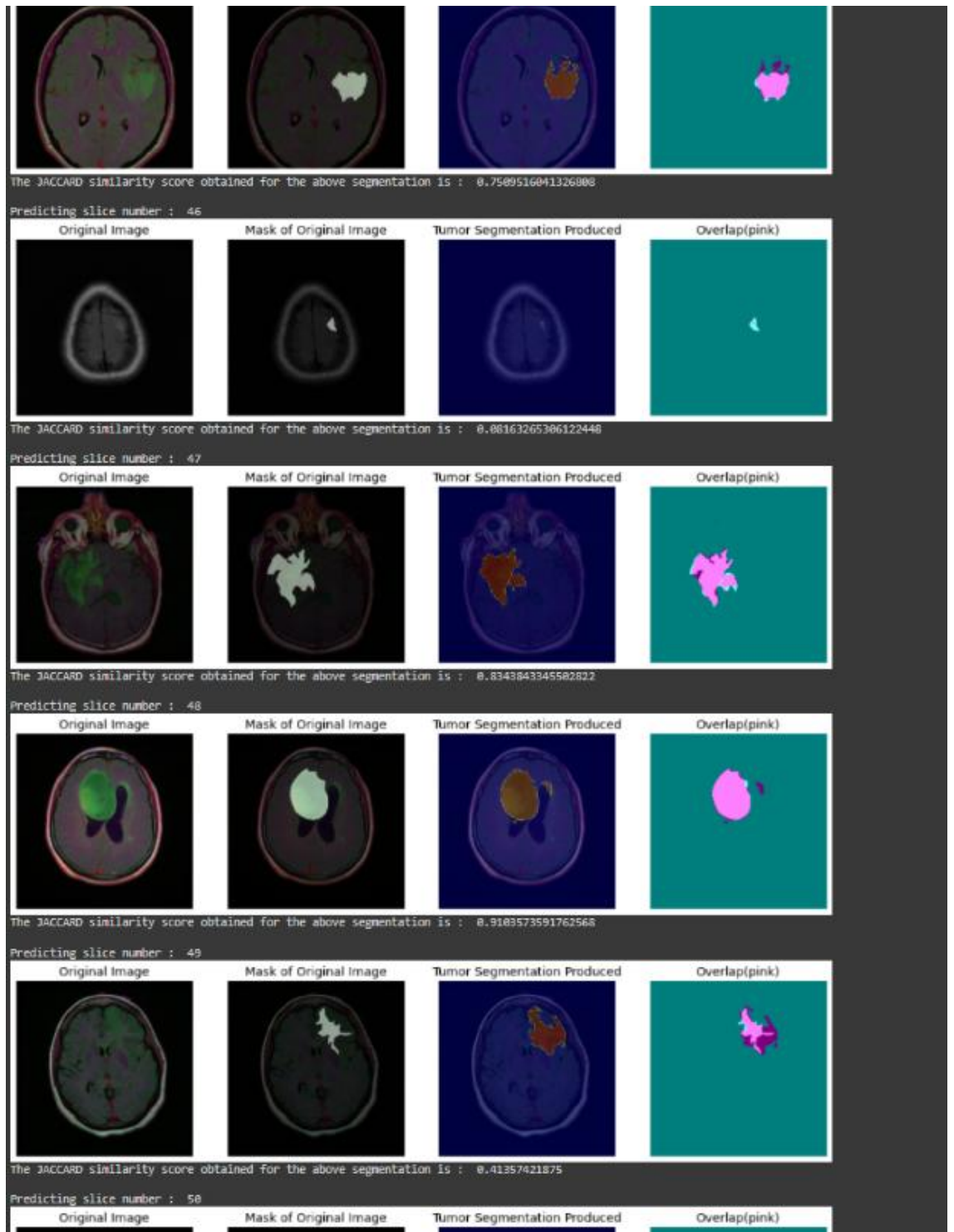


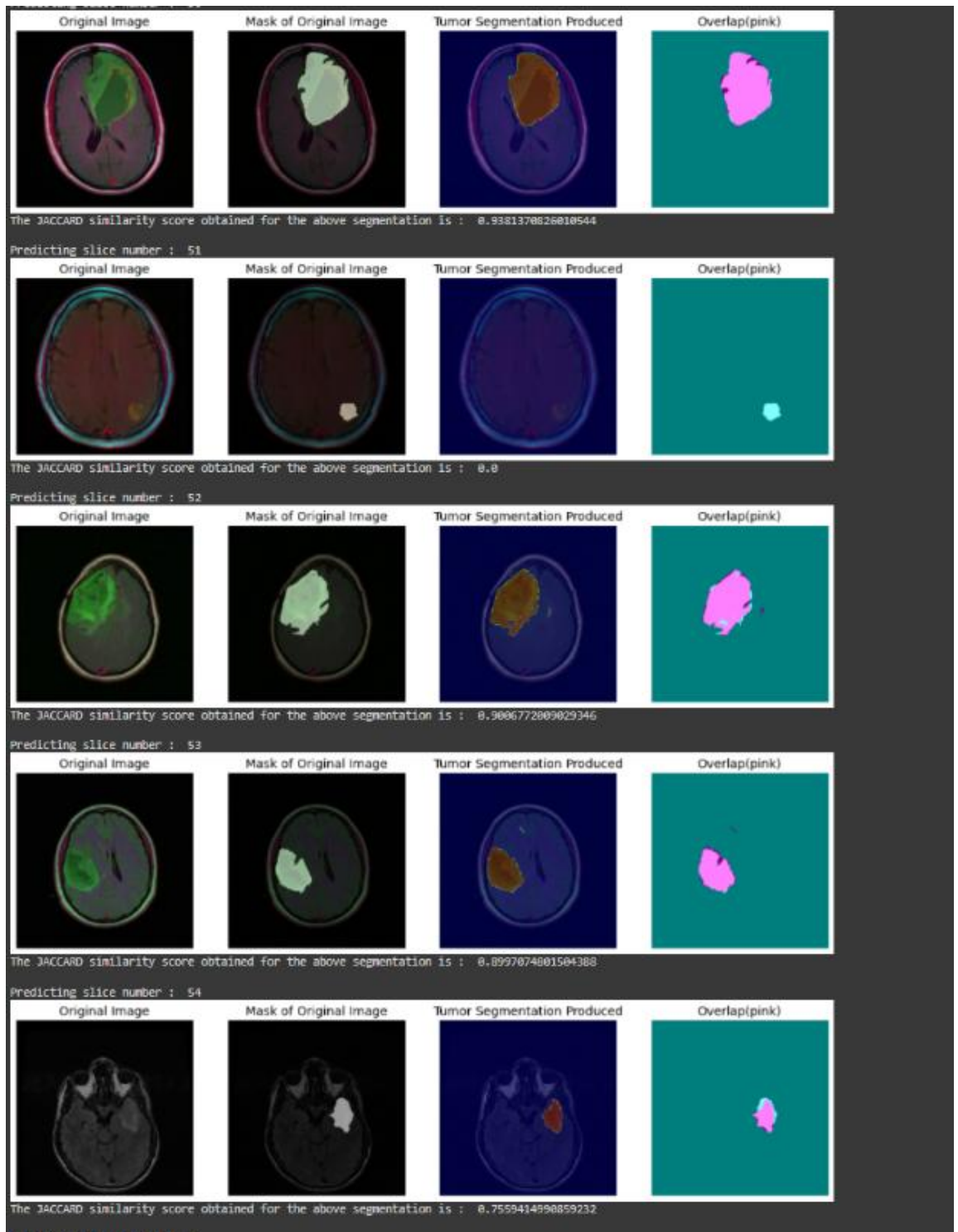


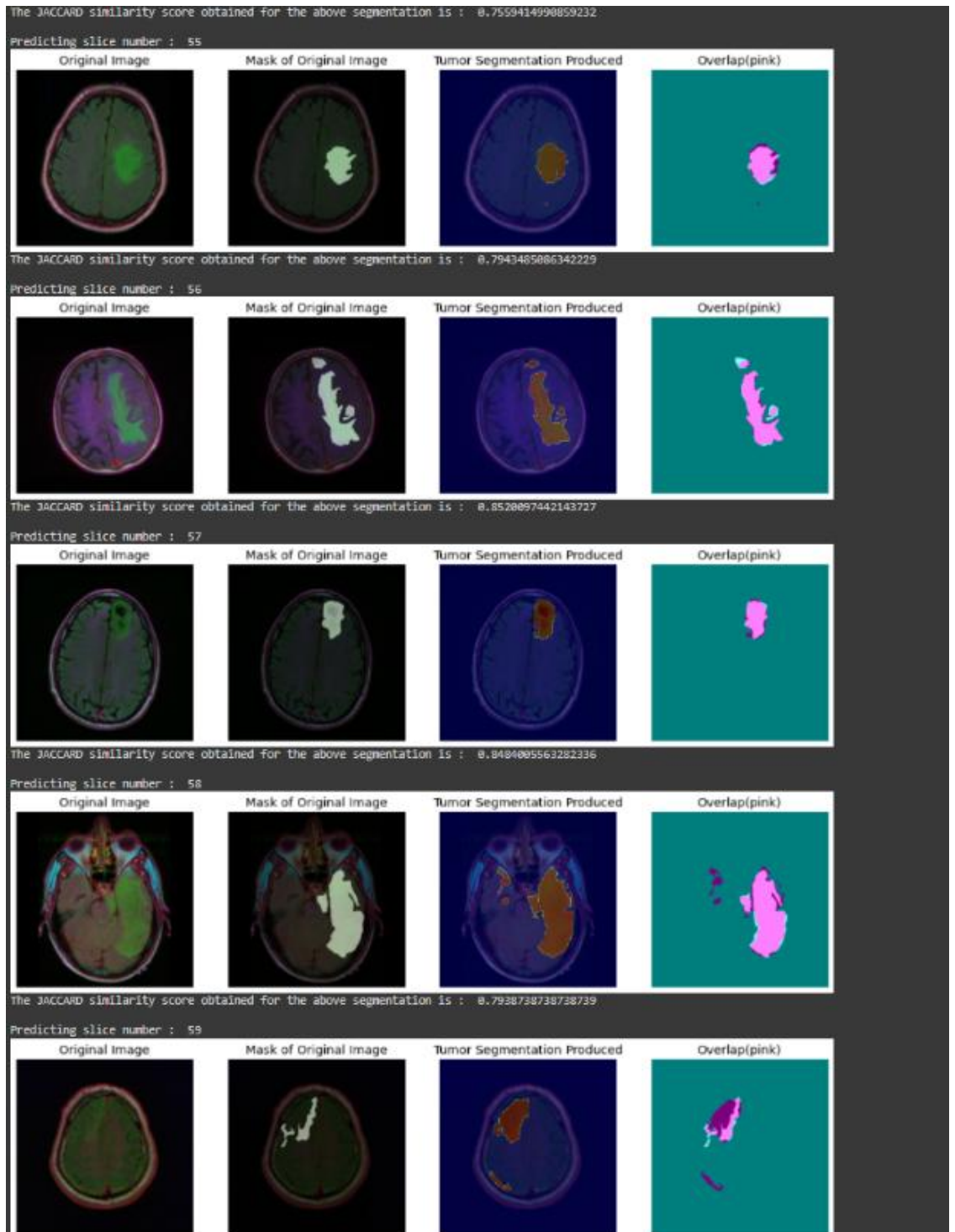


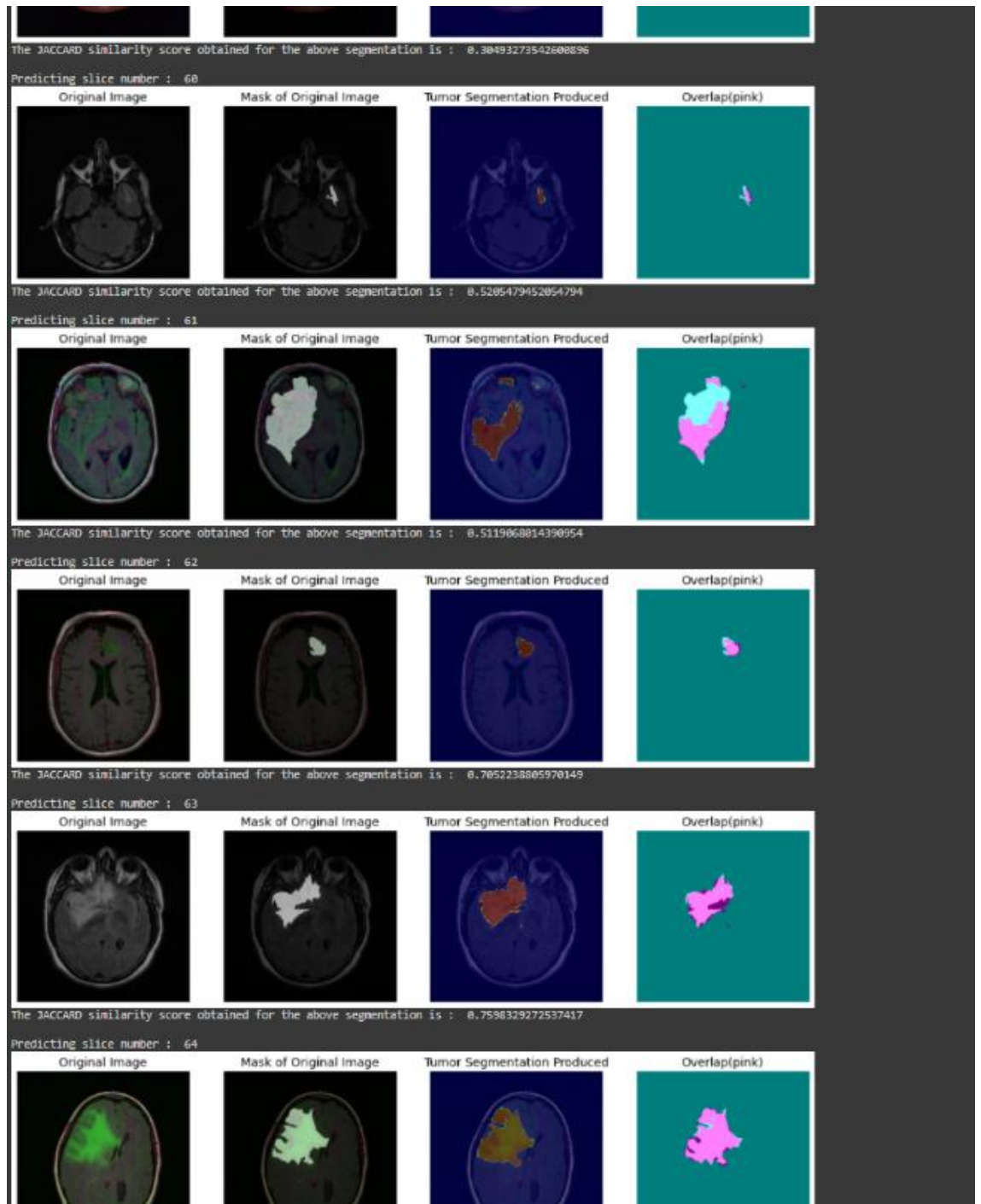


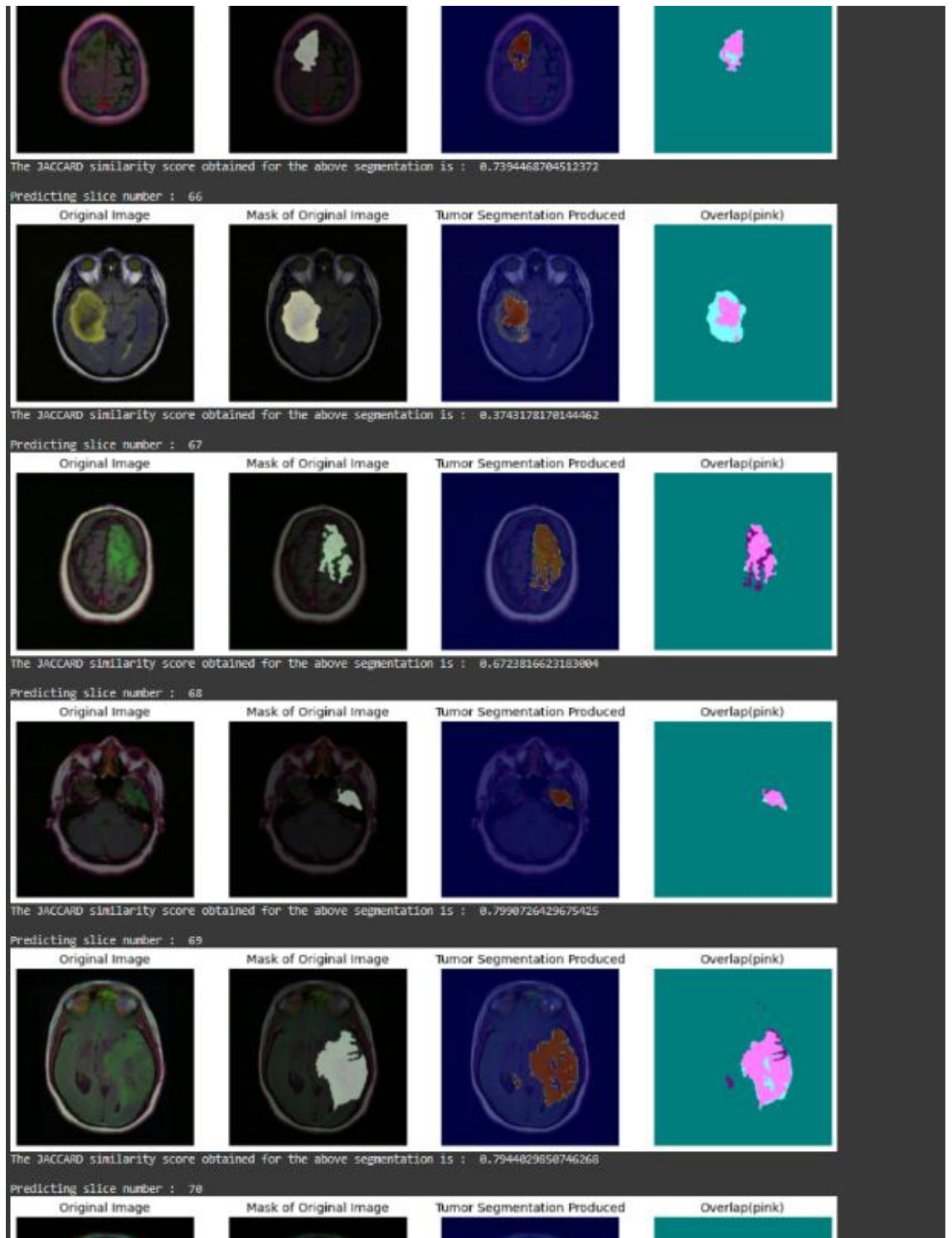


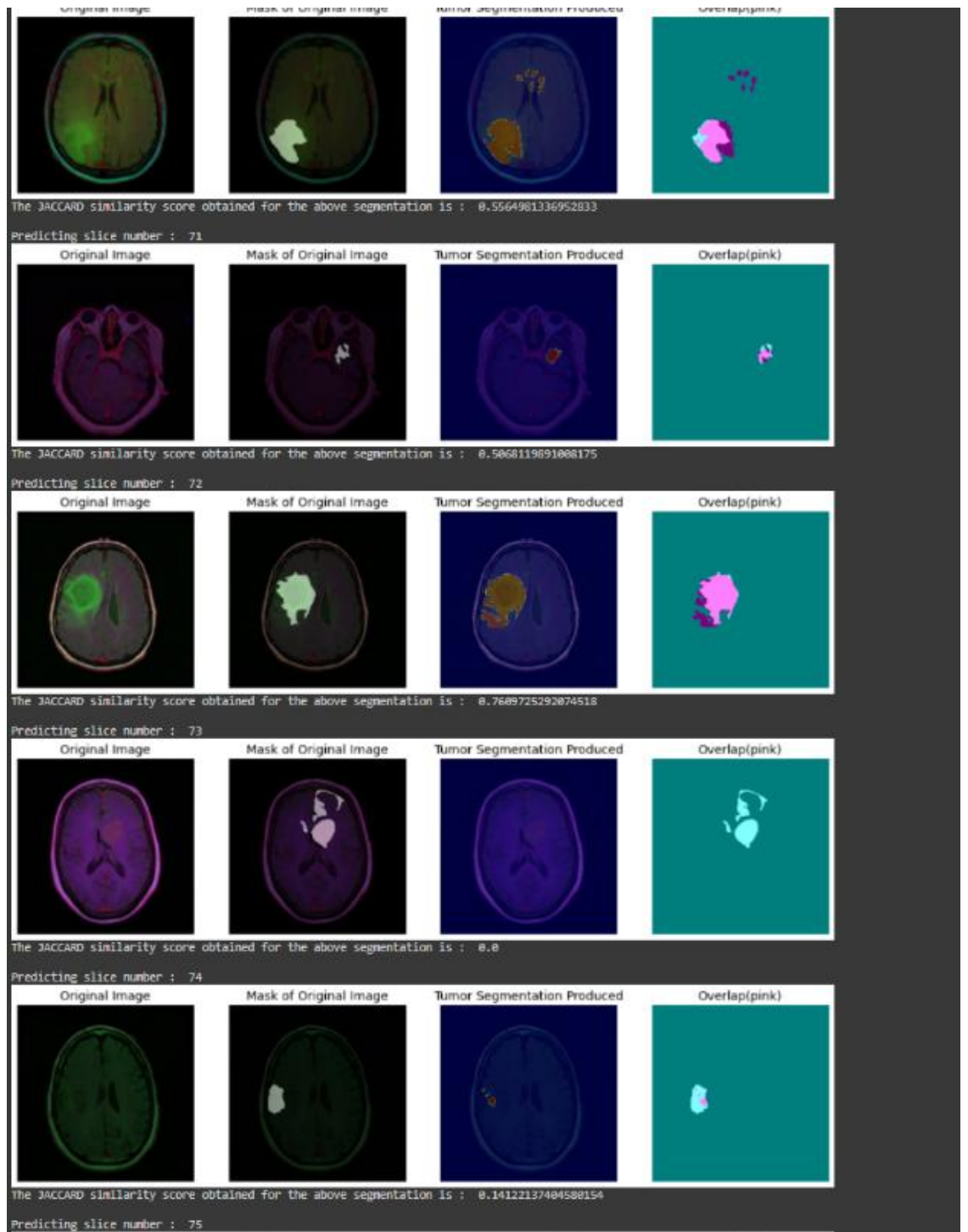


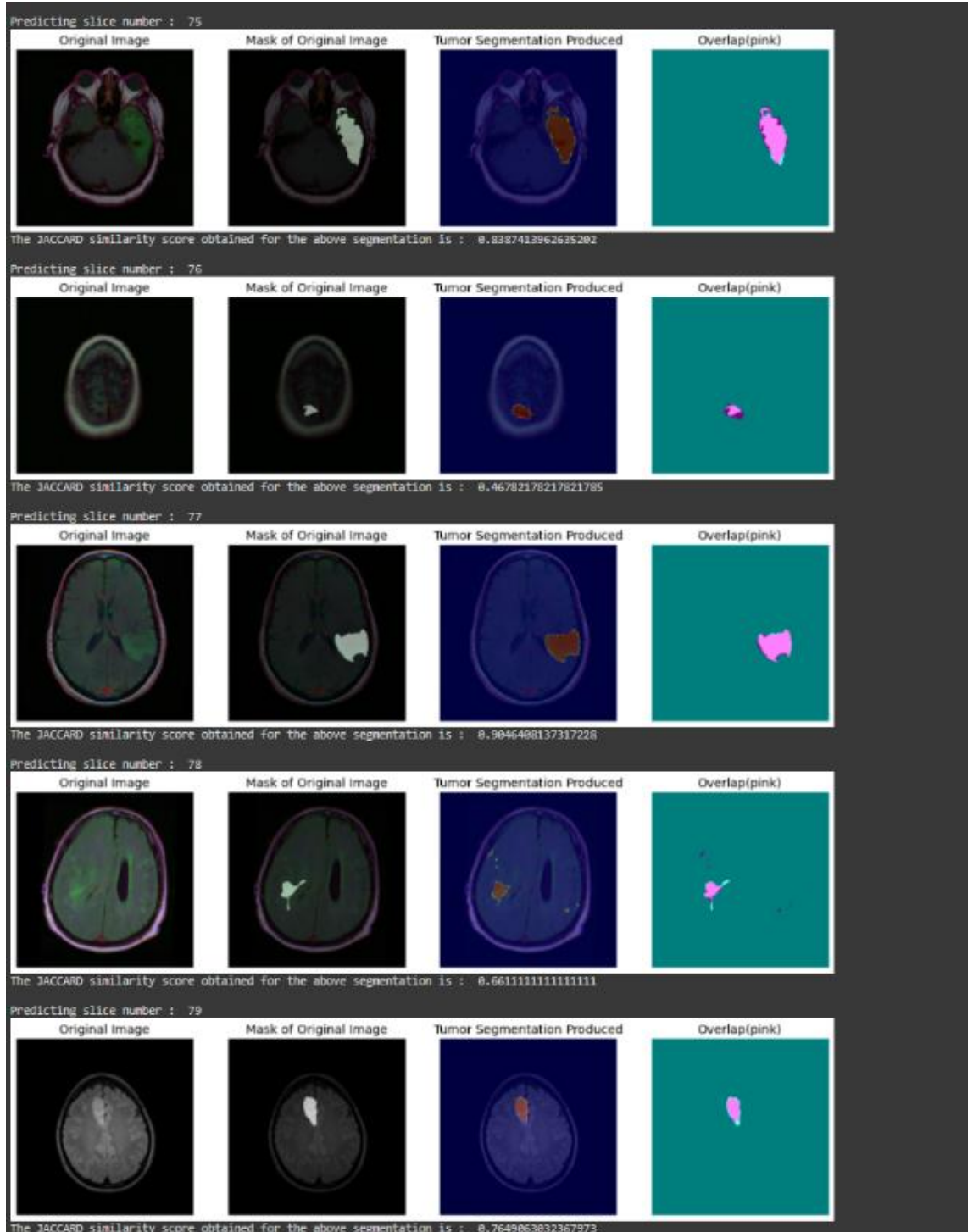


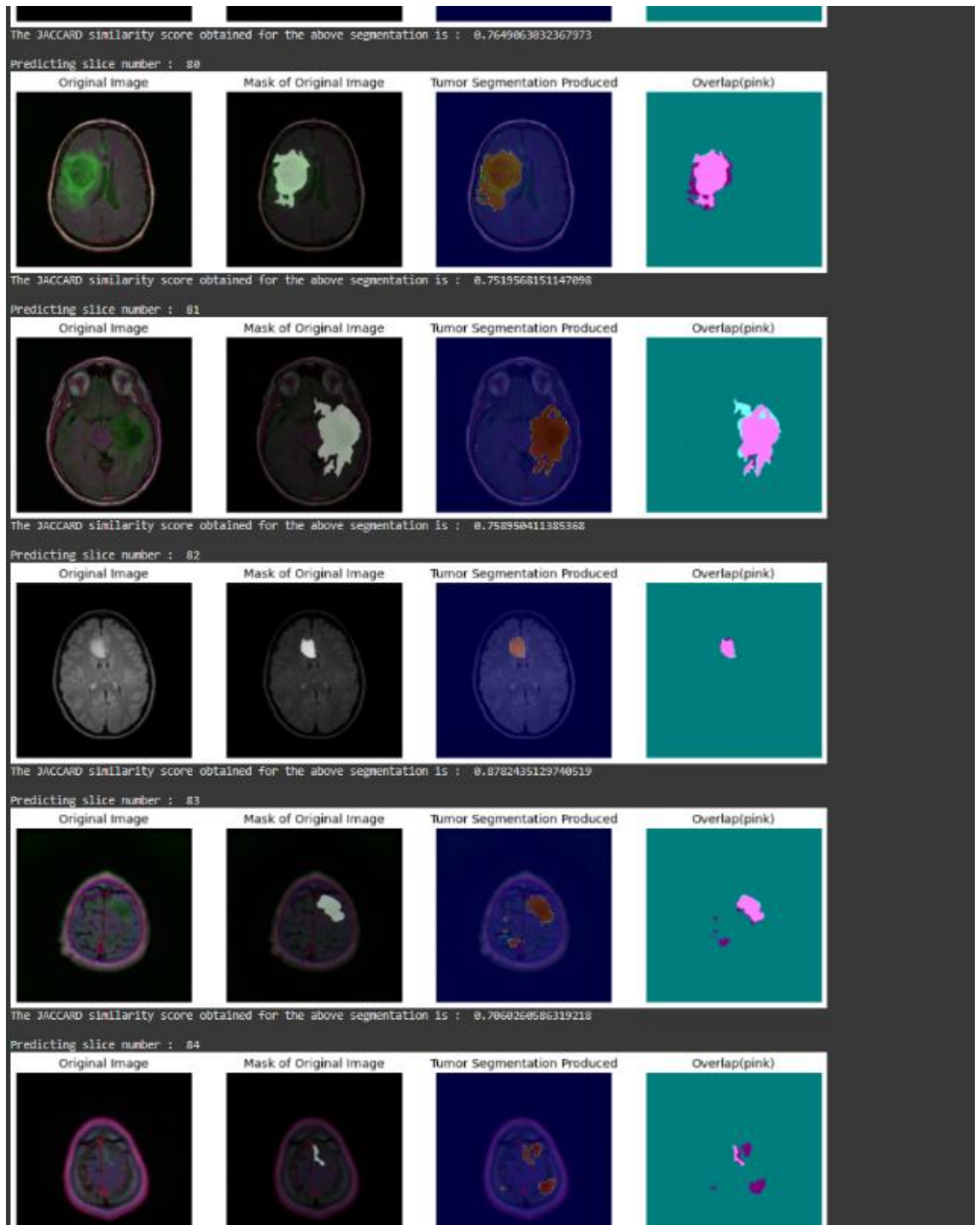


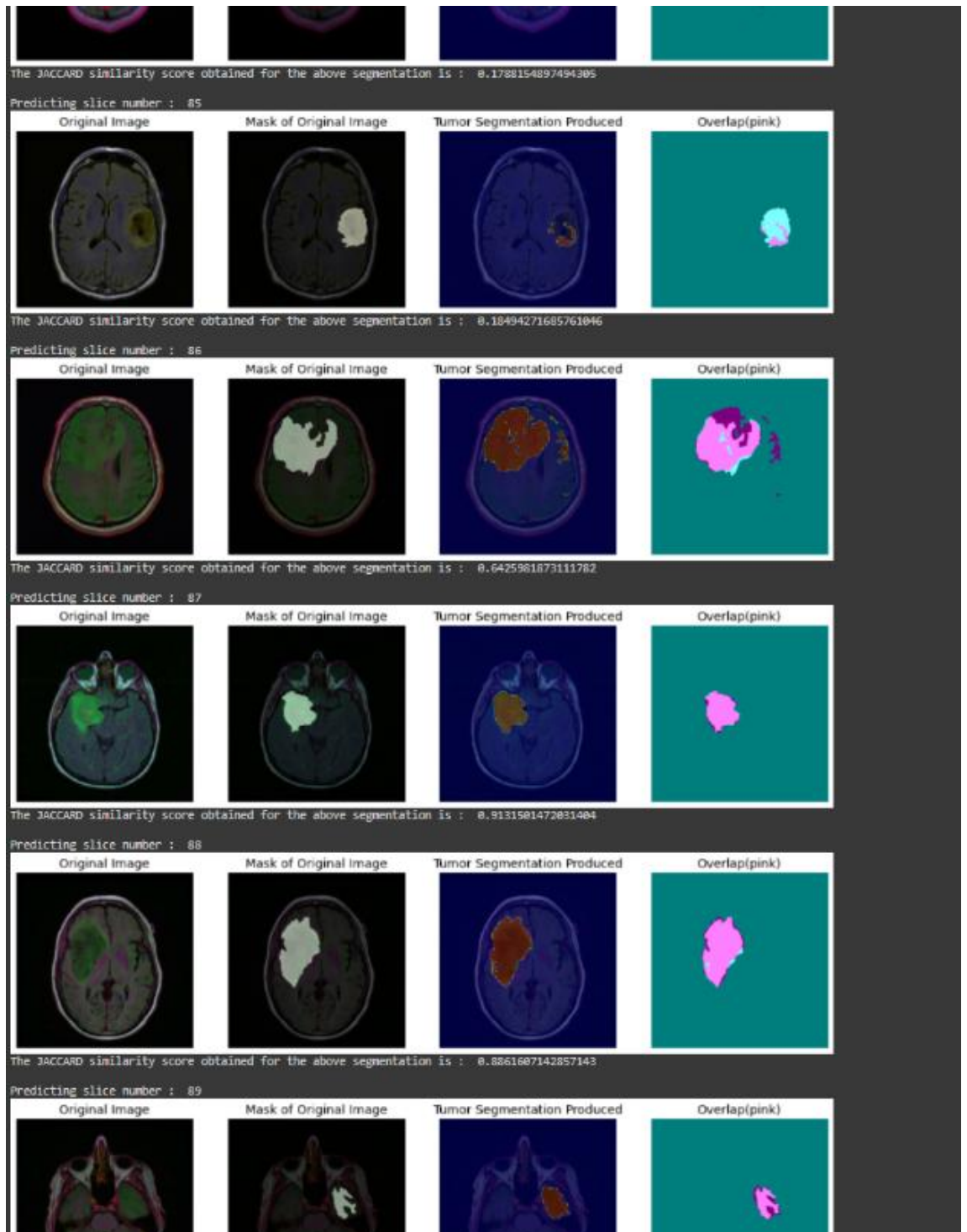


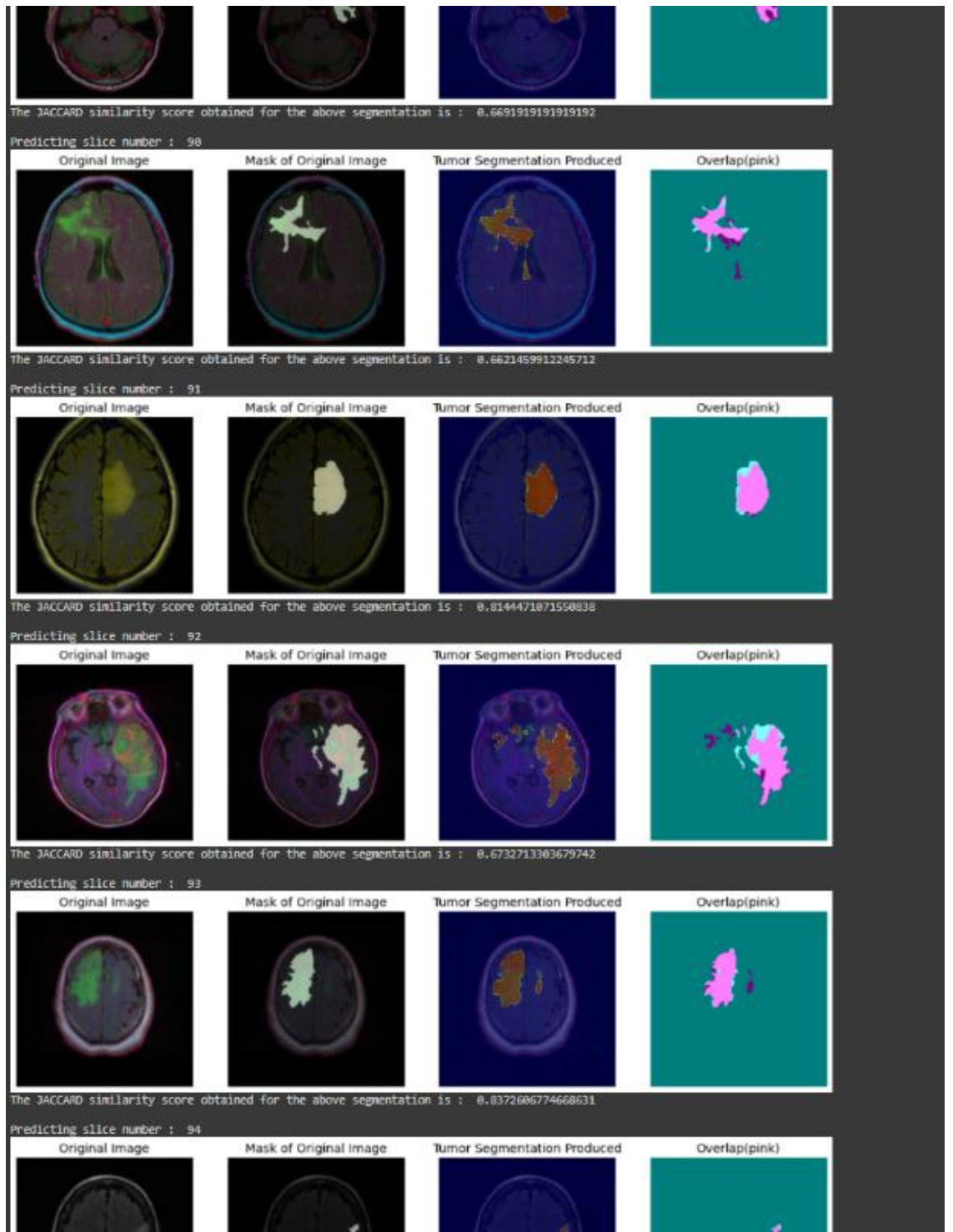


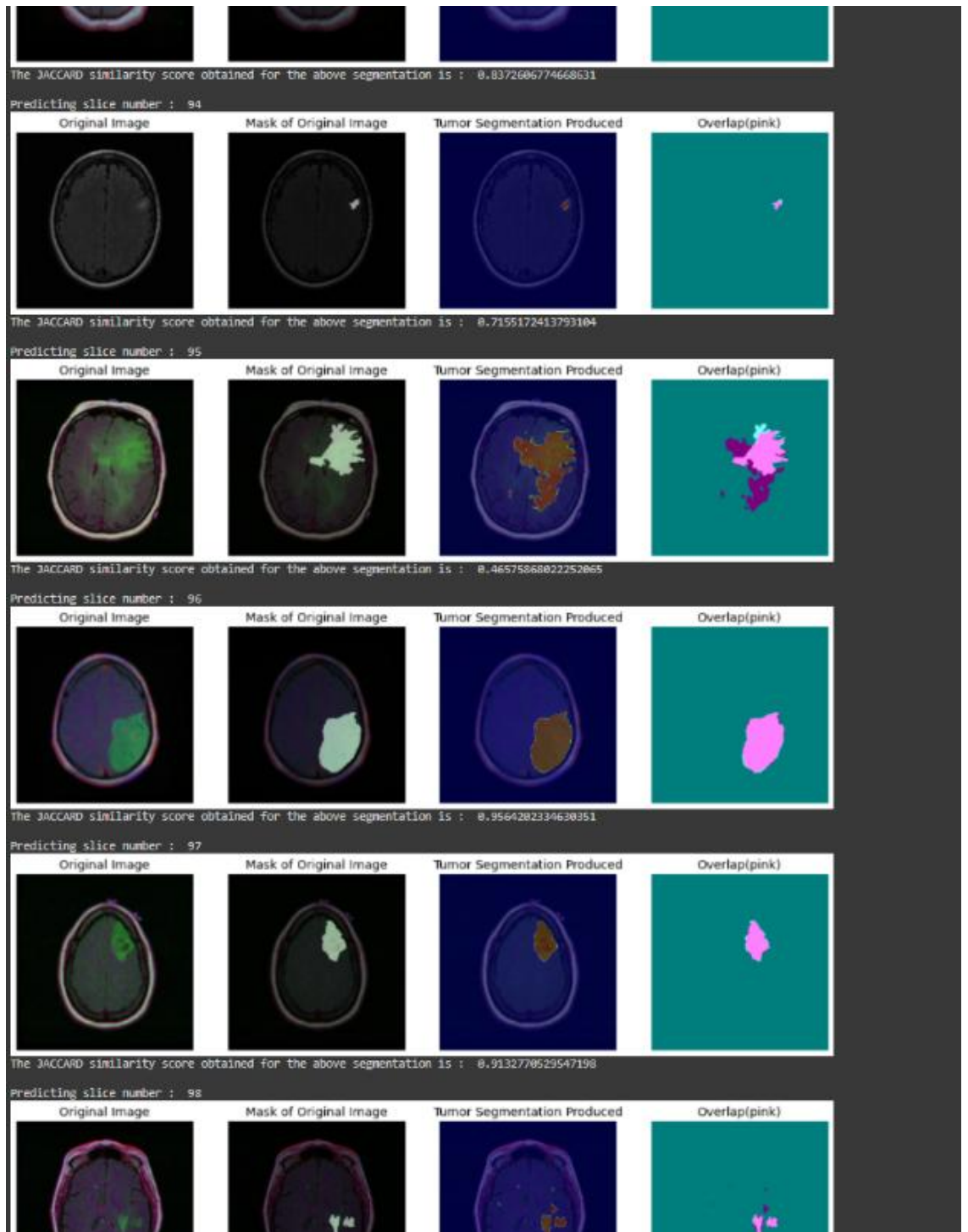


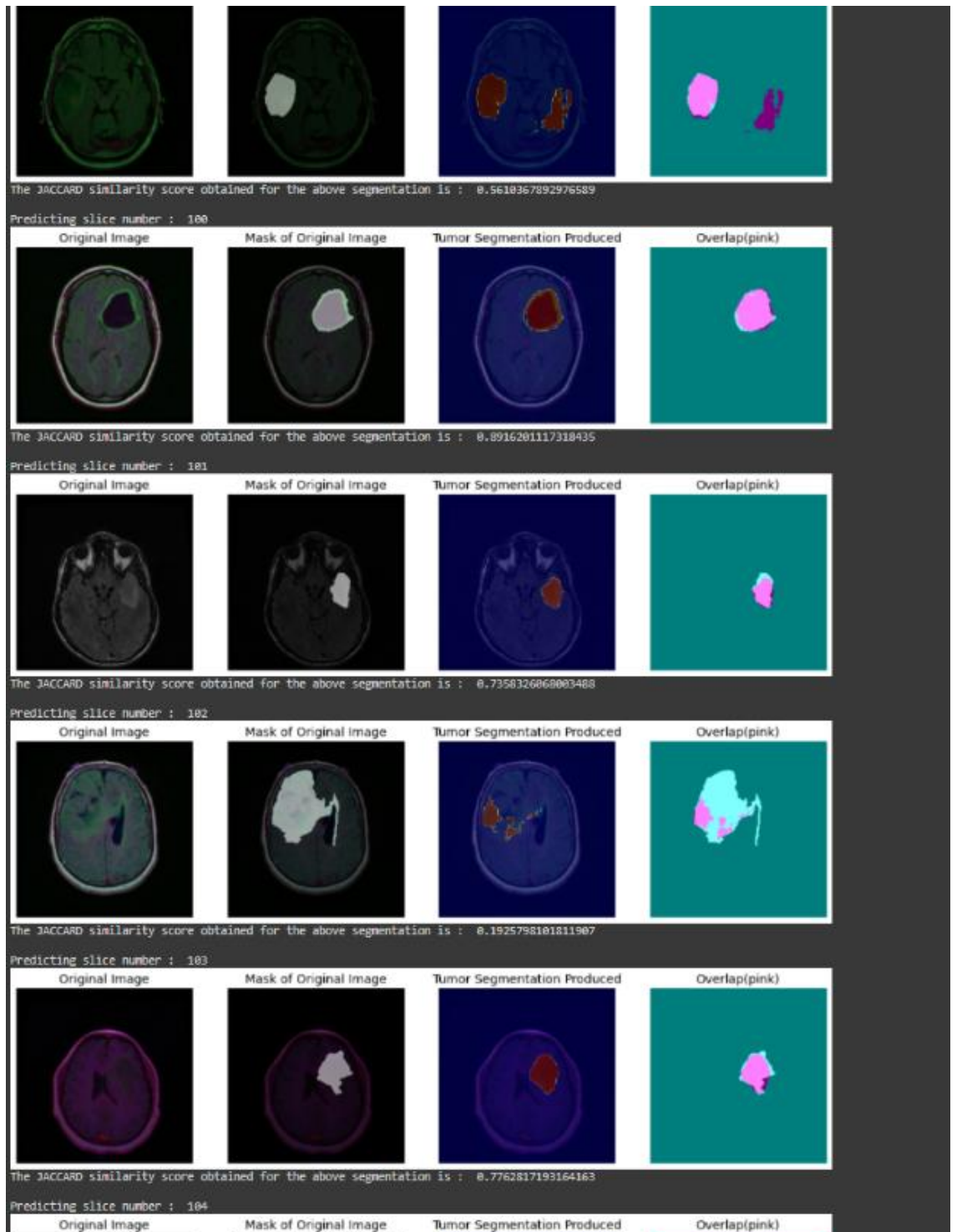


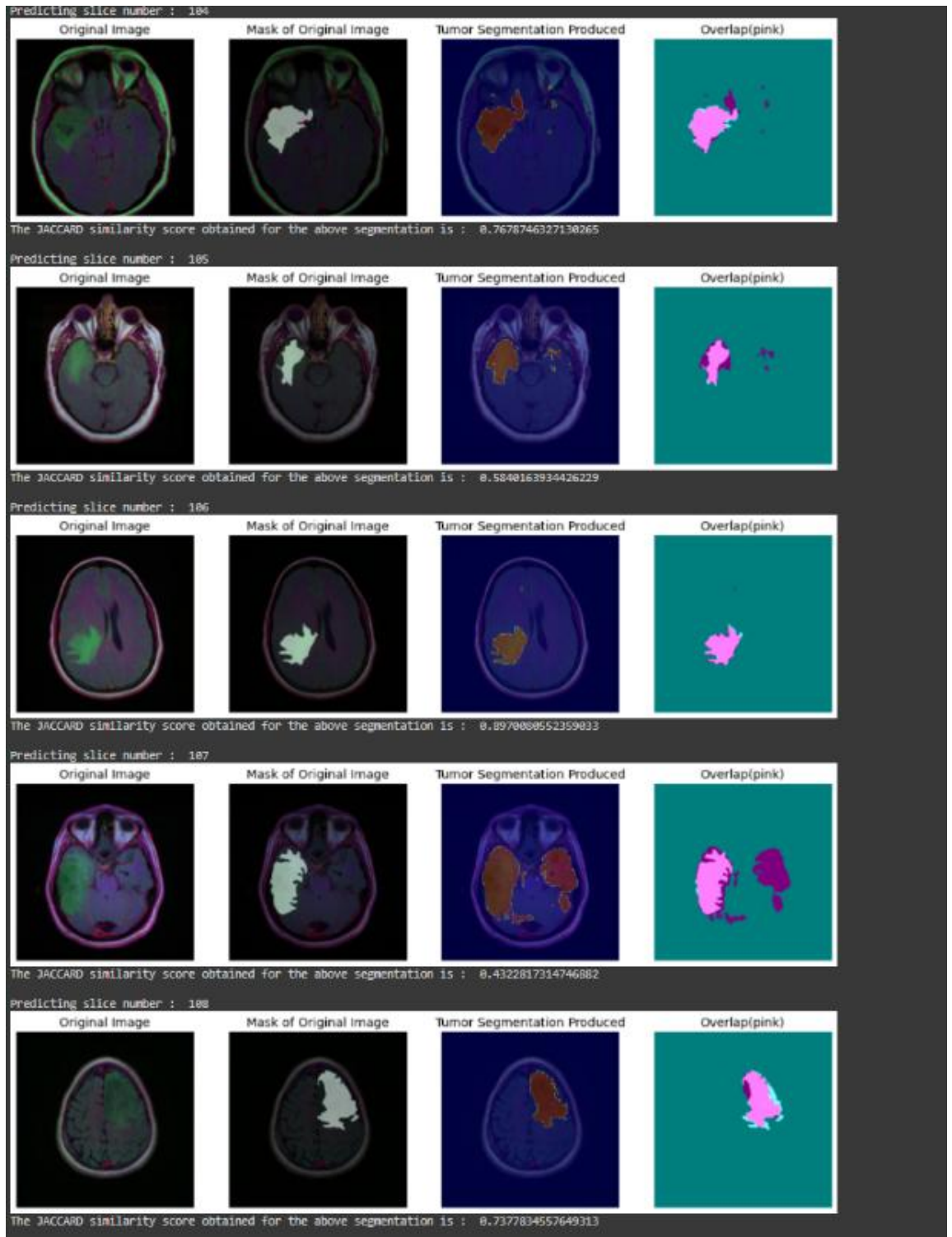


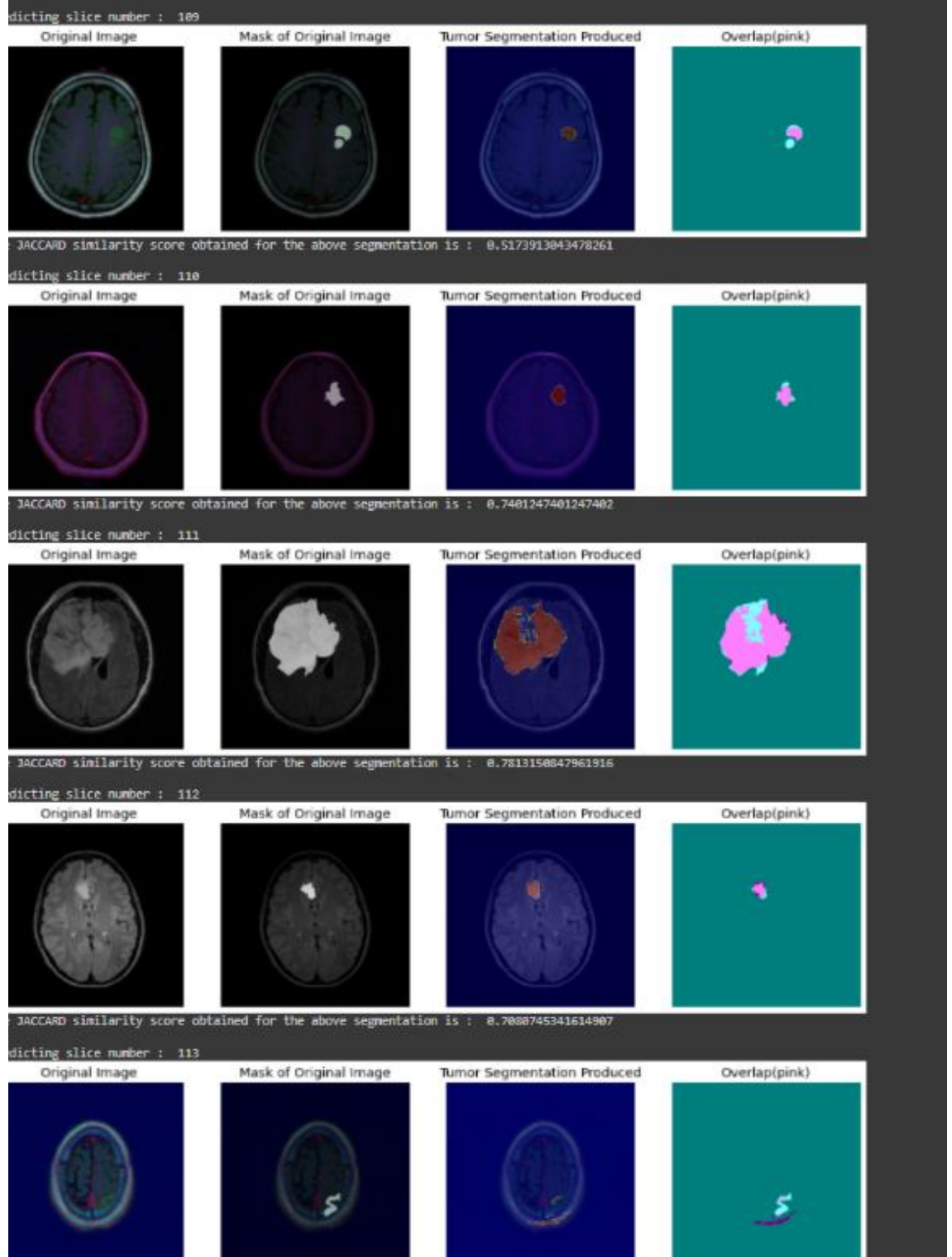


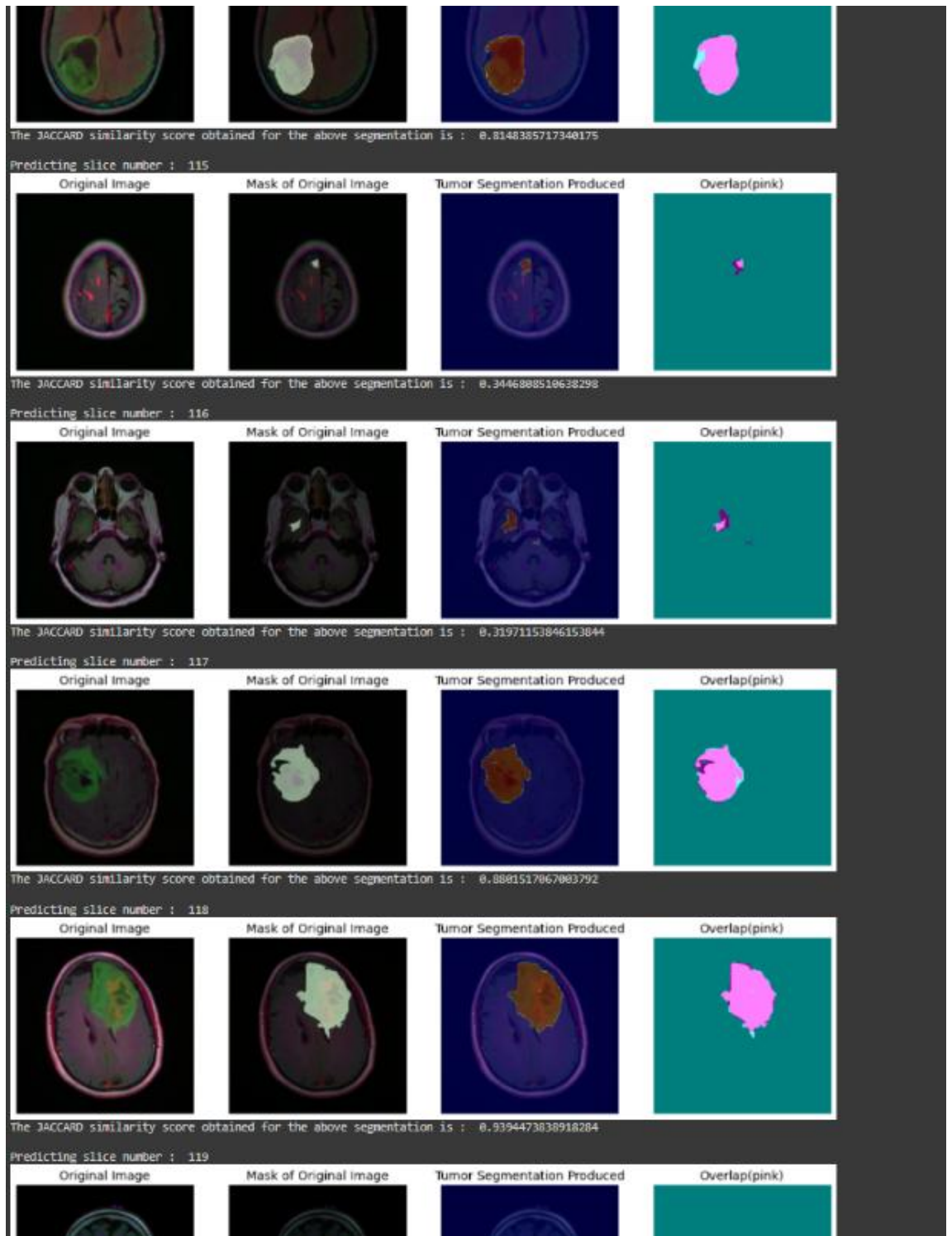


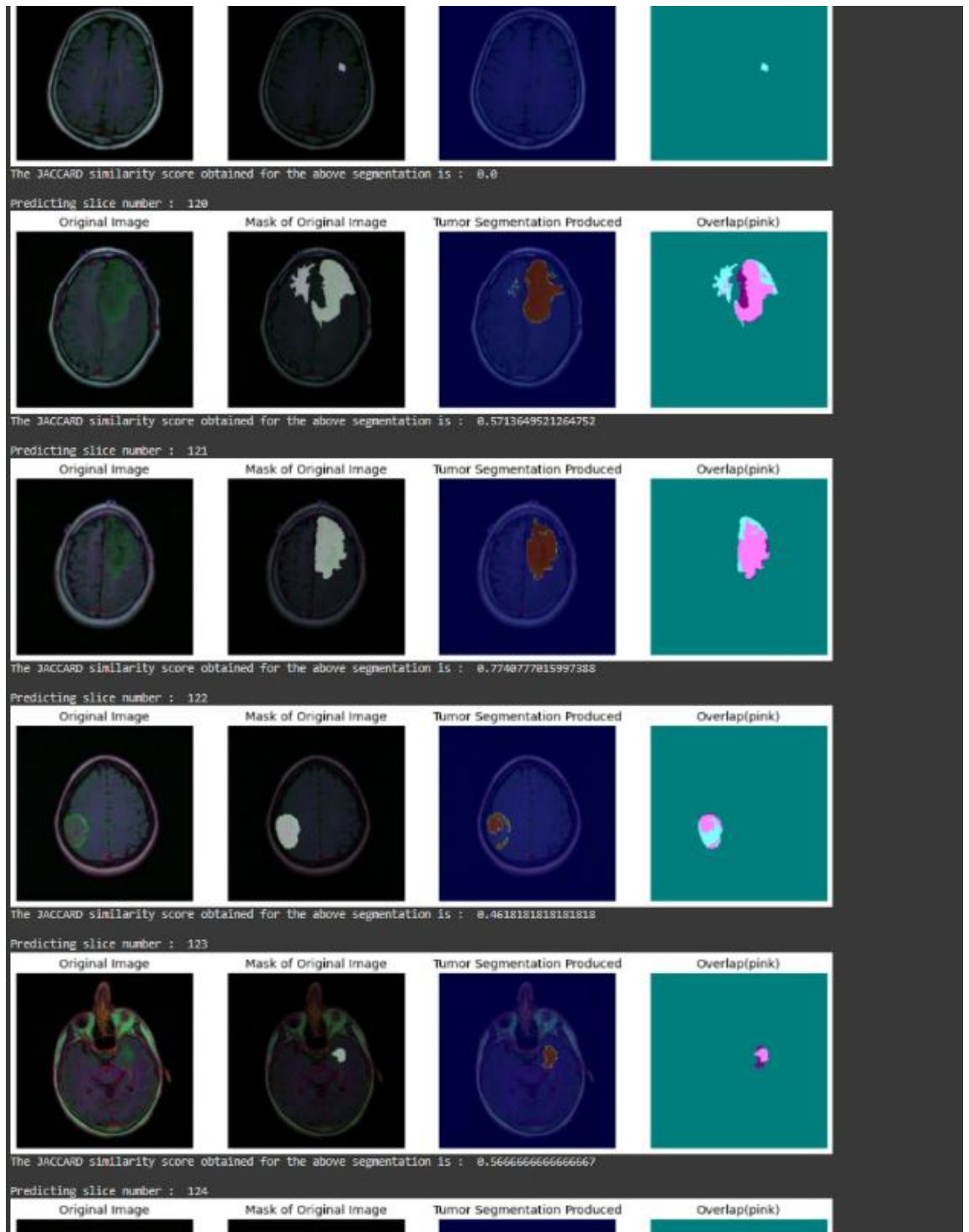


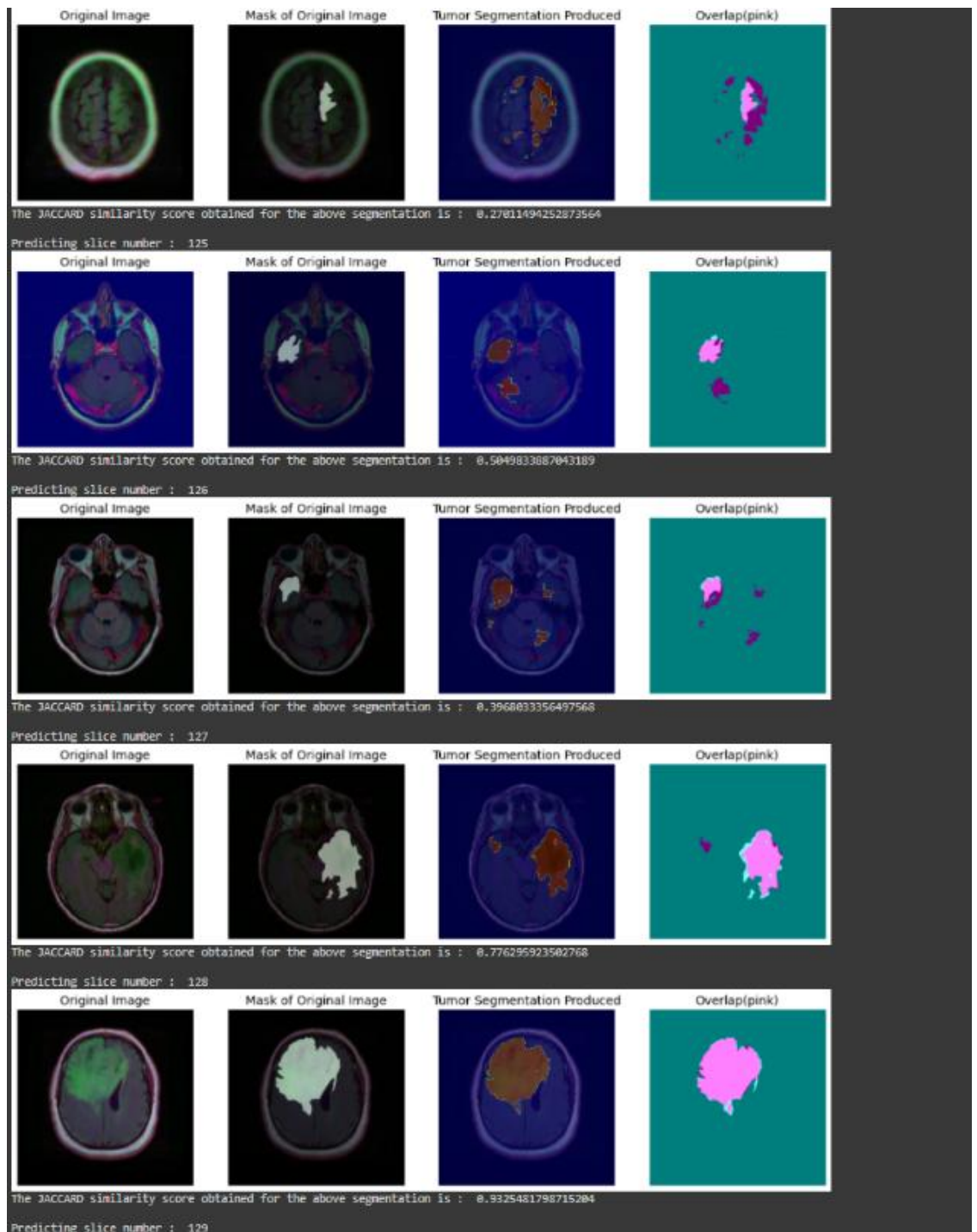


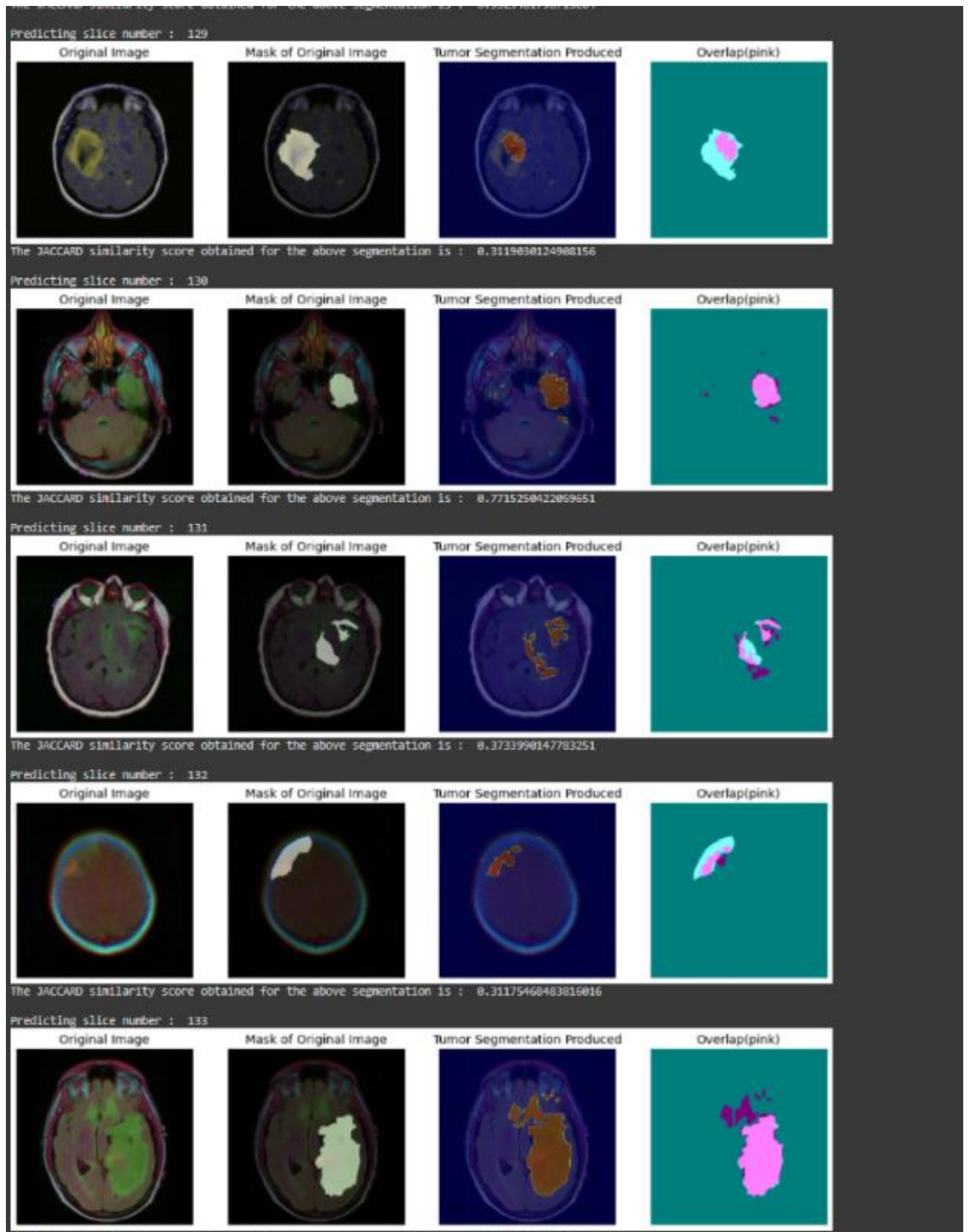


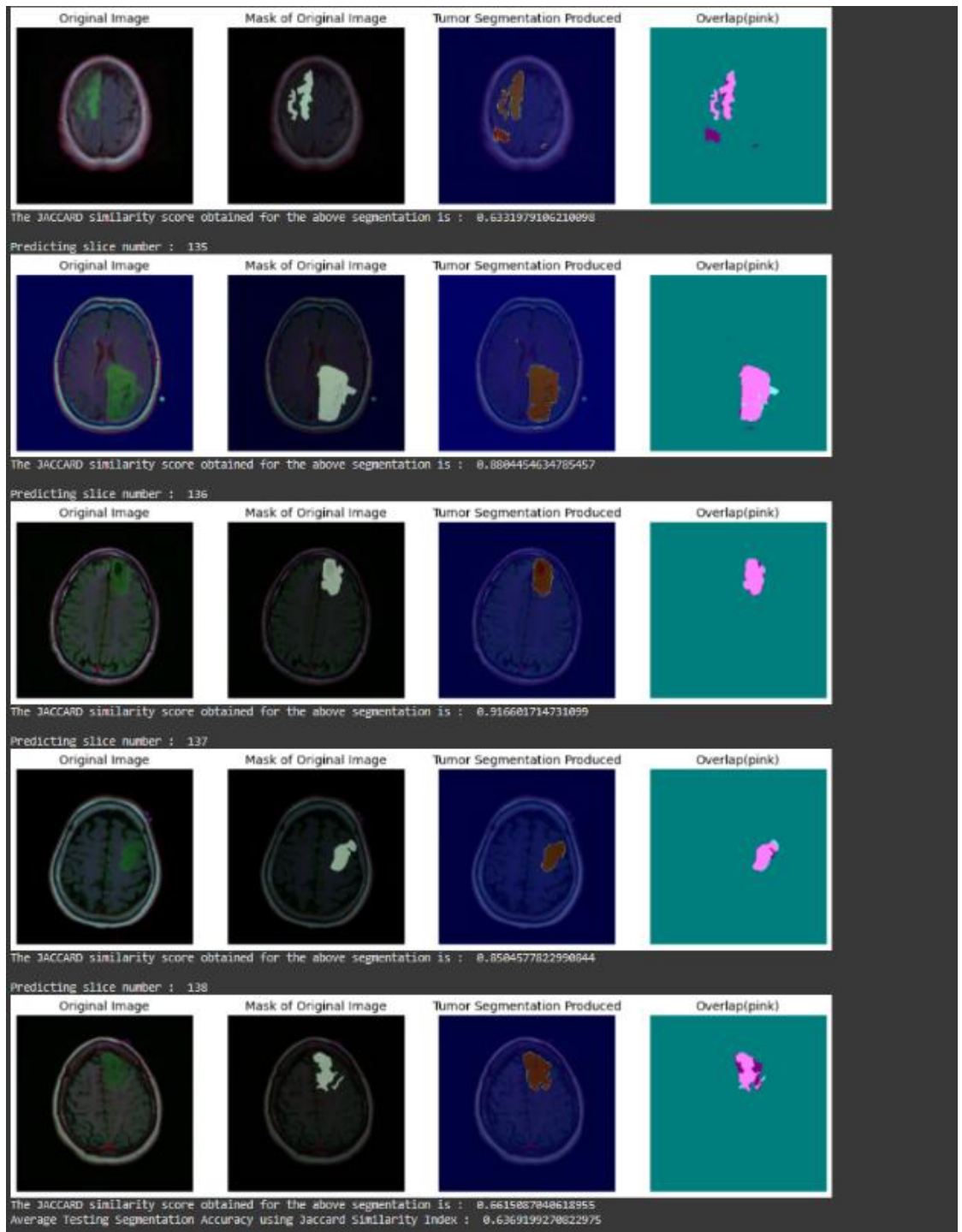




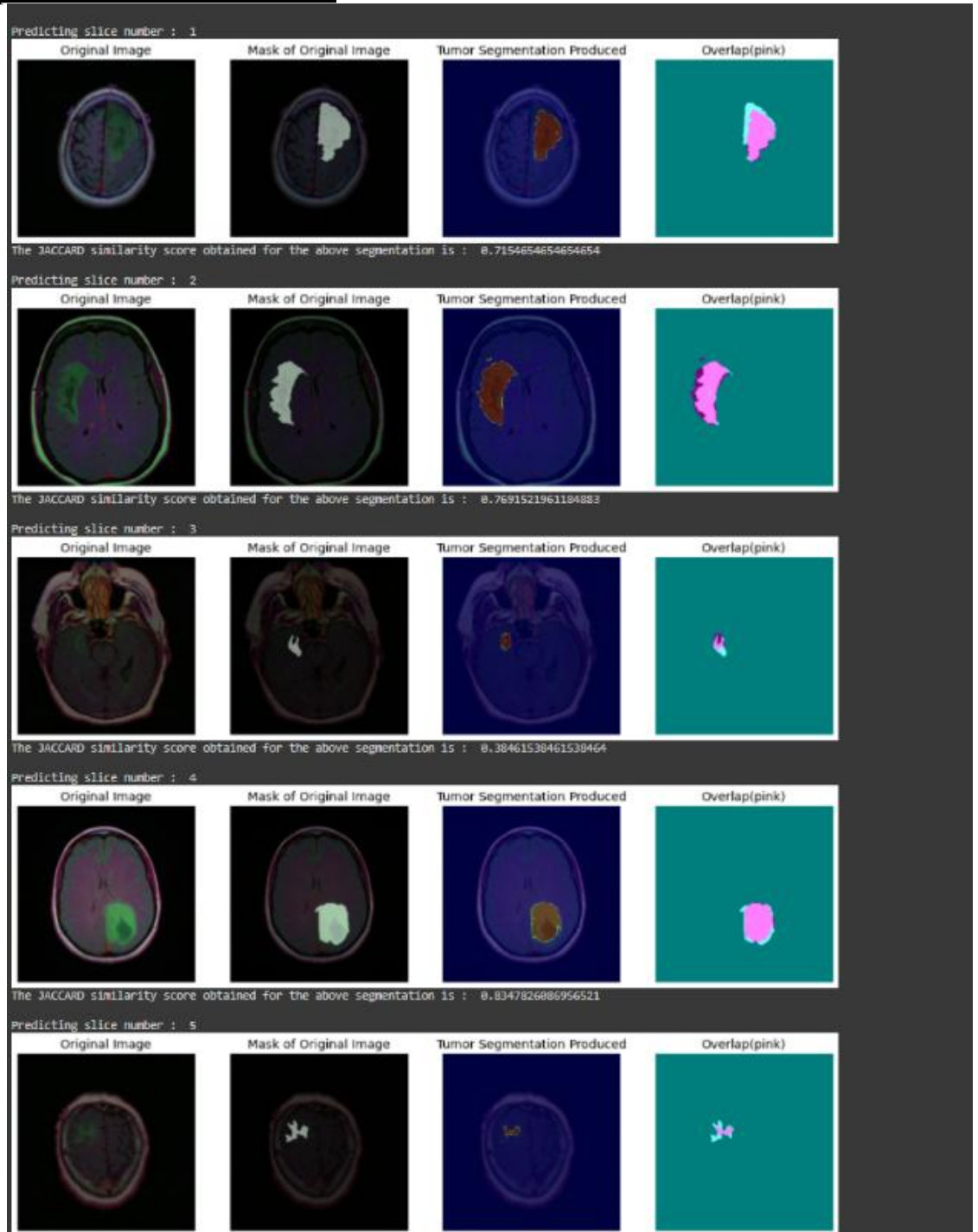


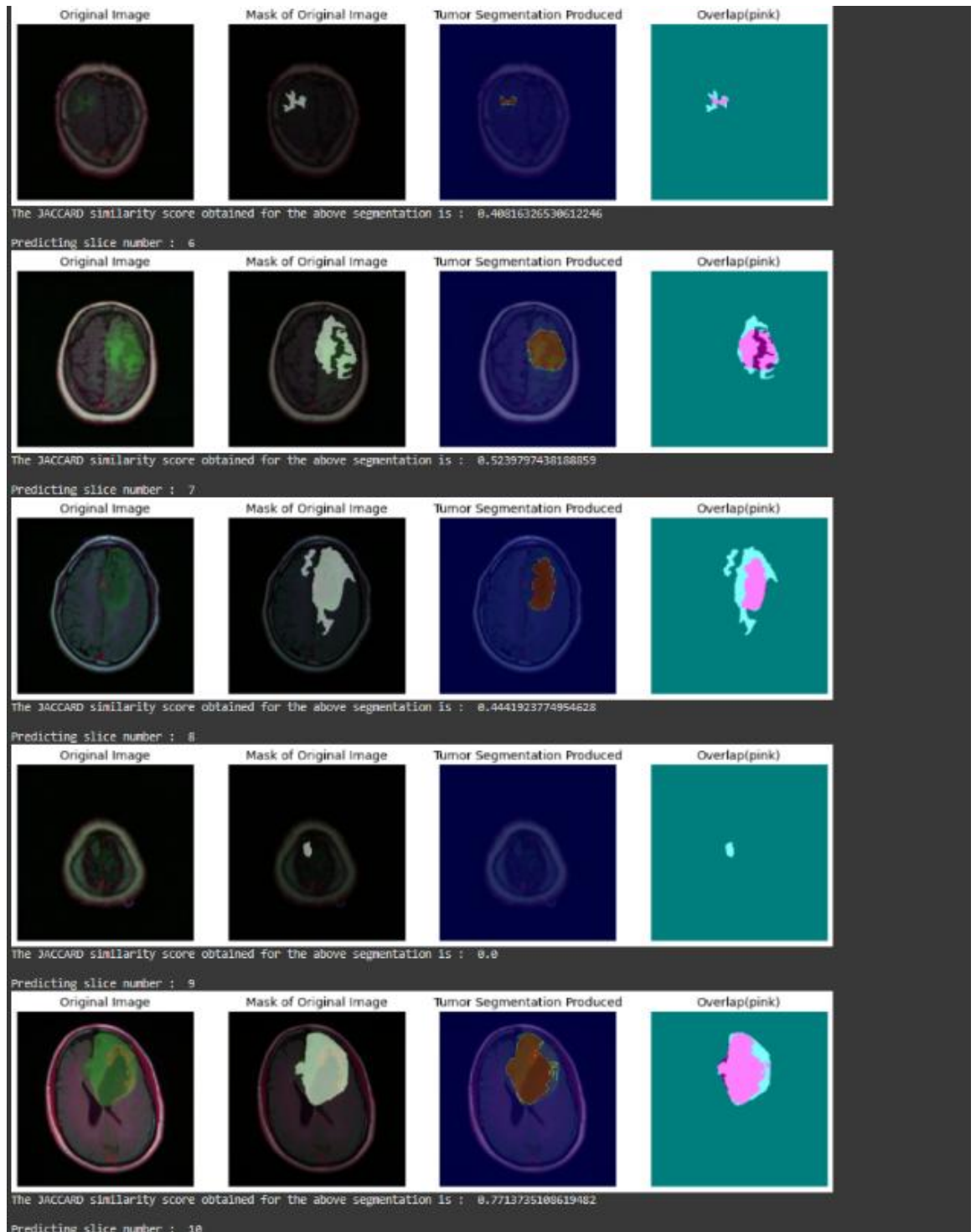


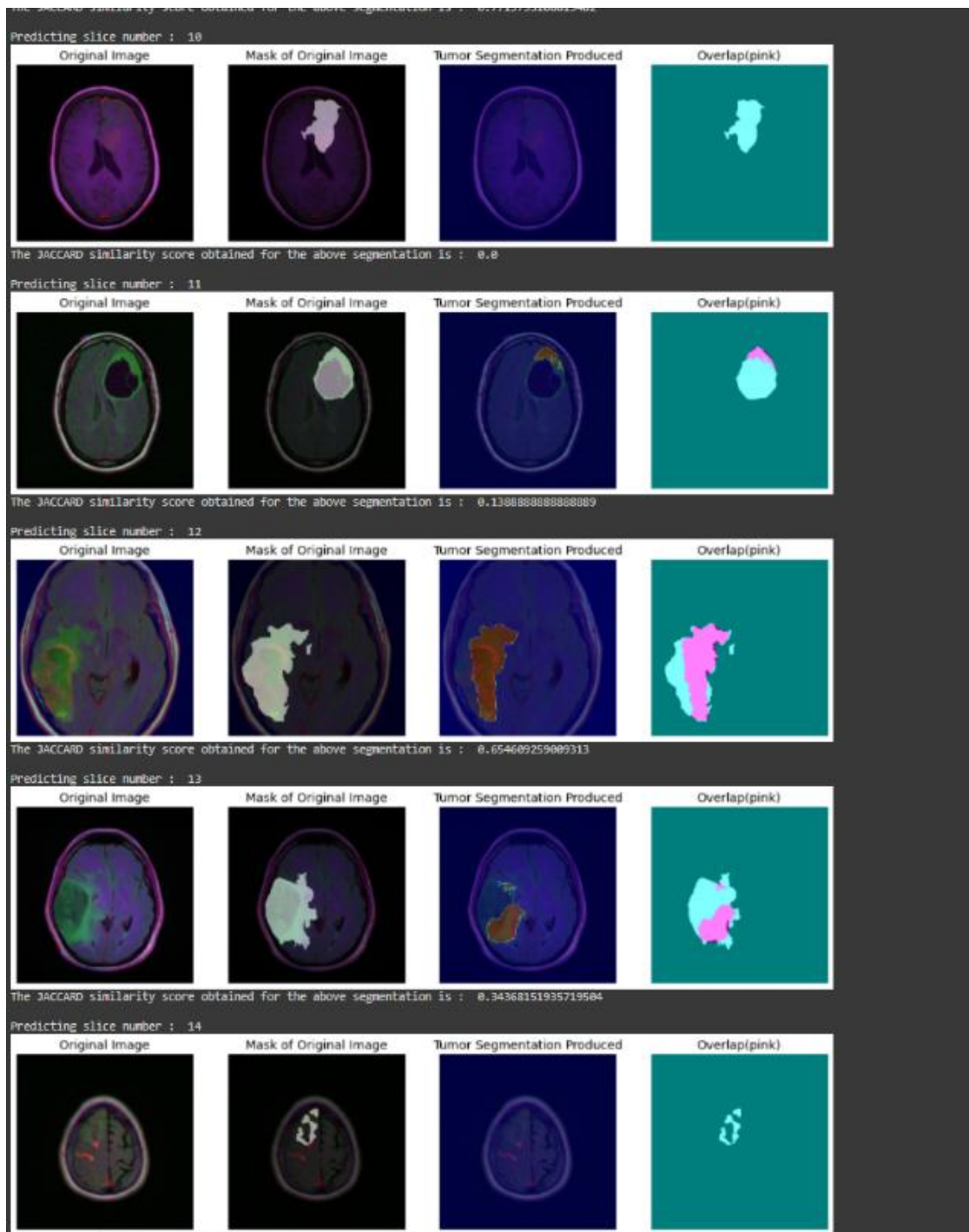


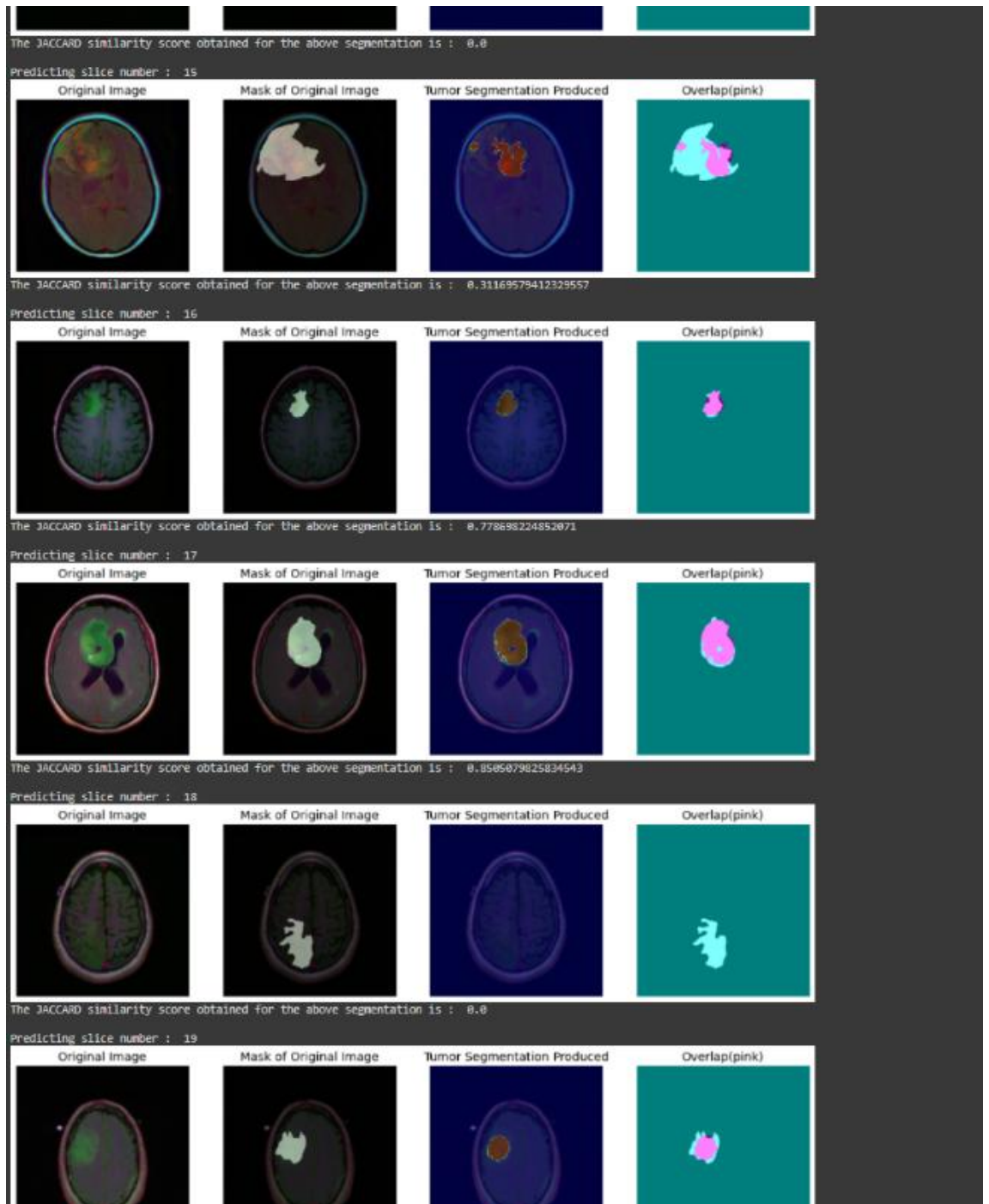


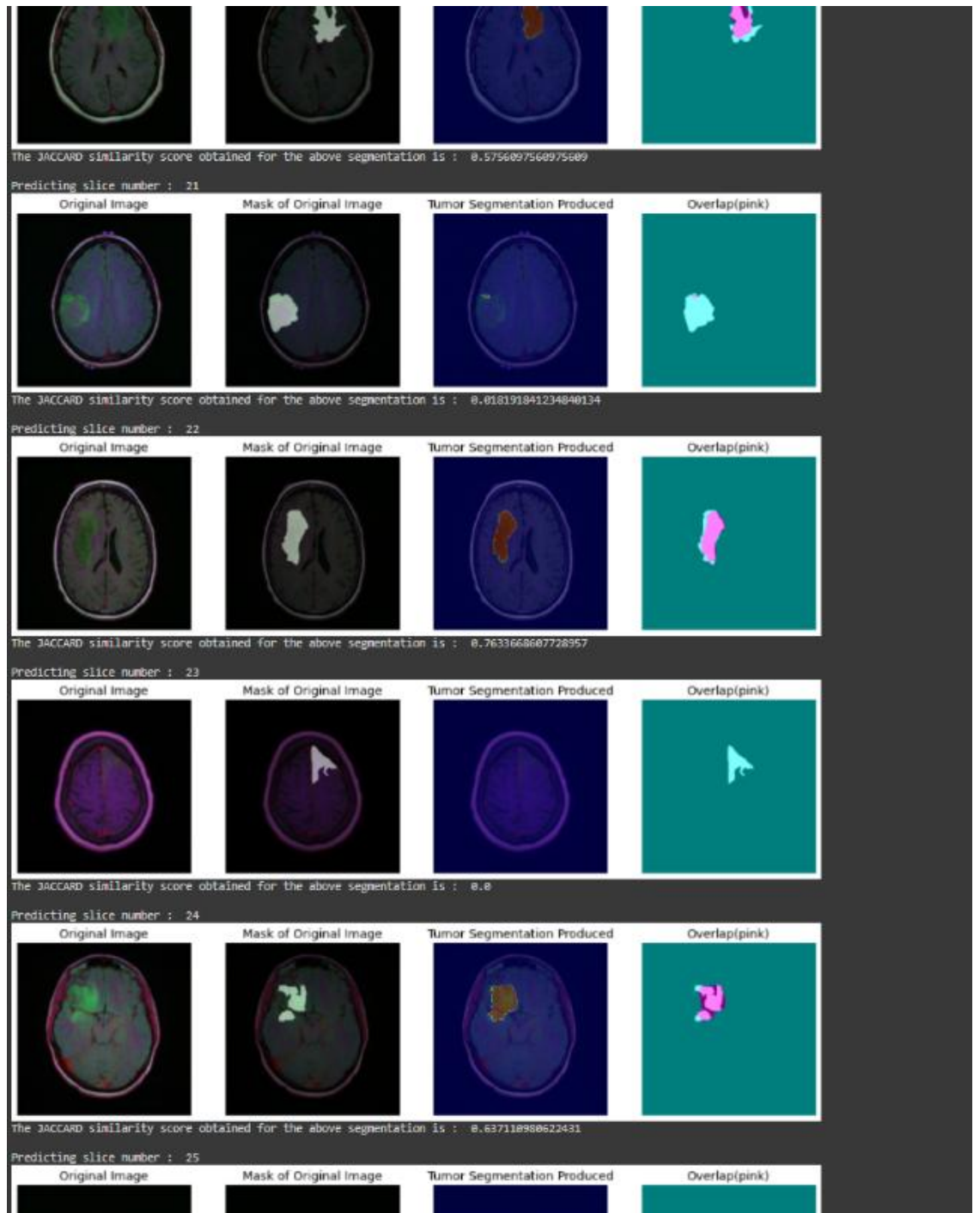
5. Experimental Model No. 11

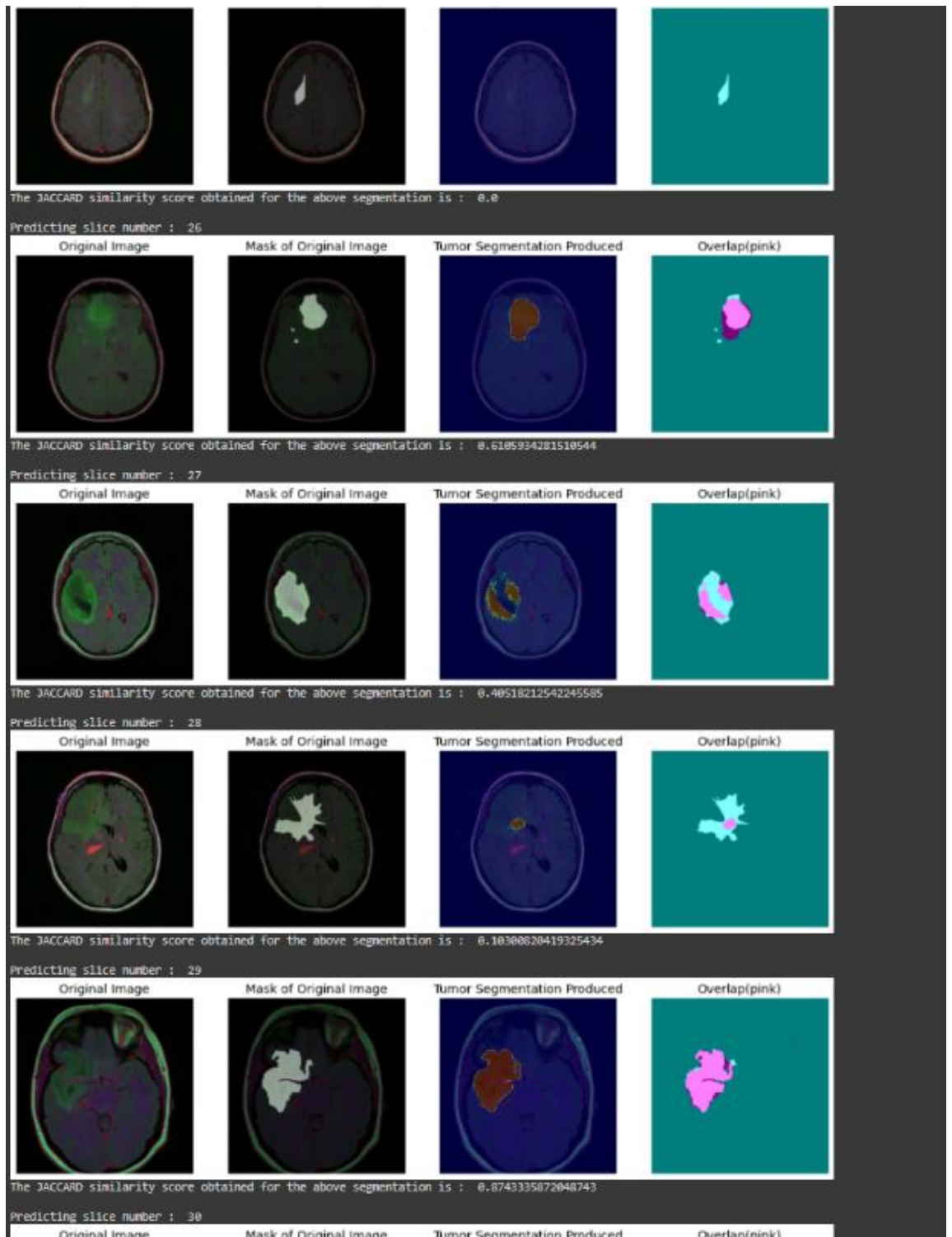


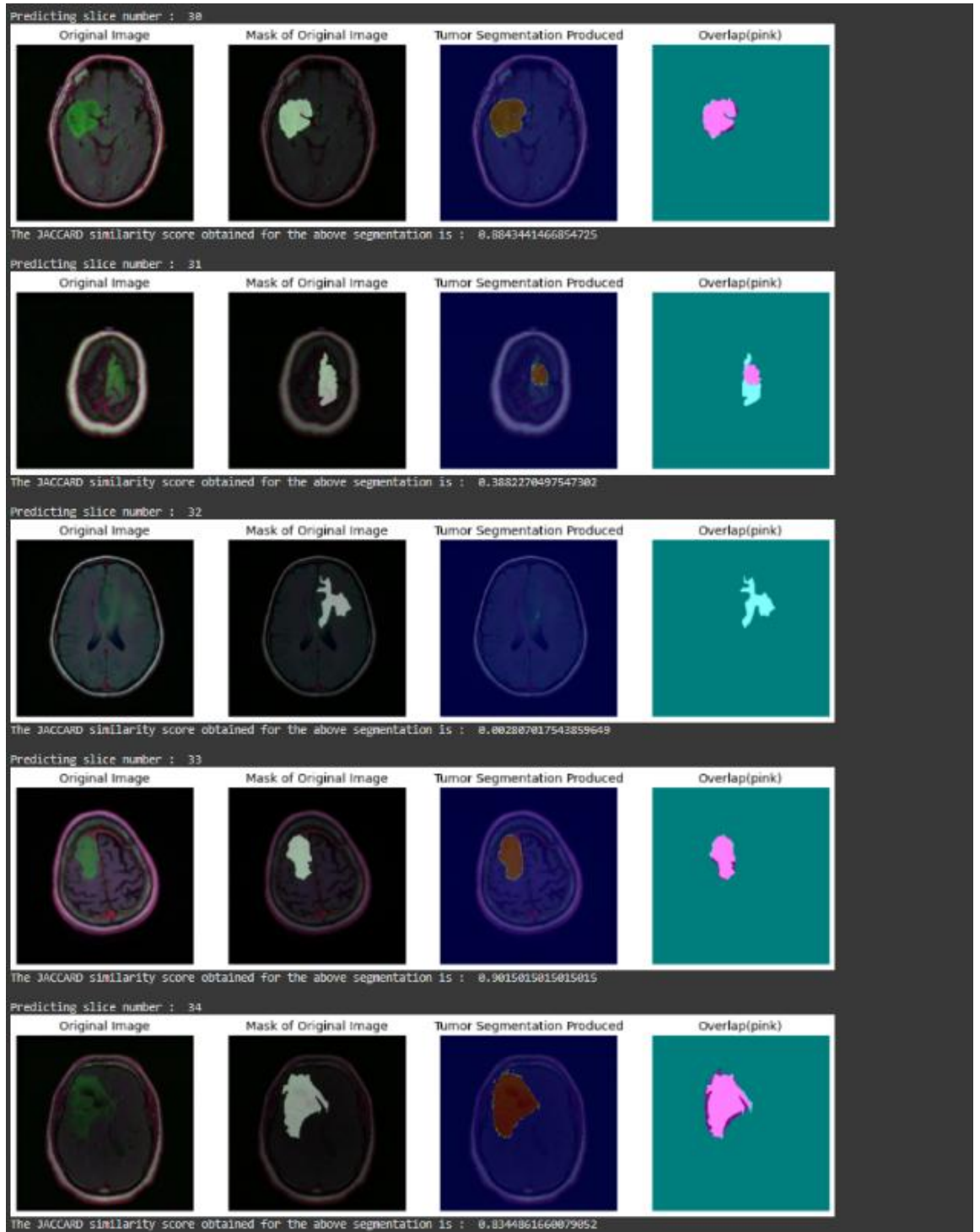


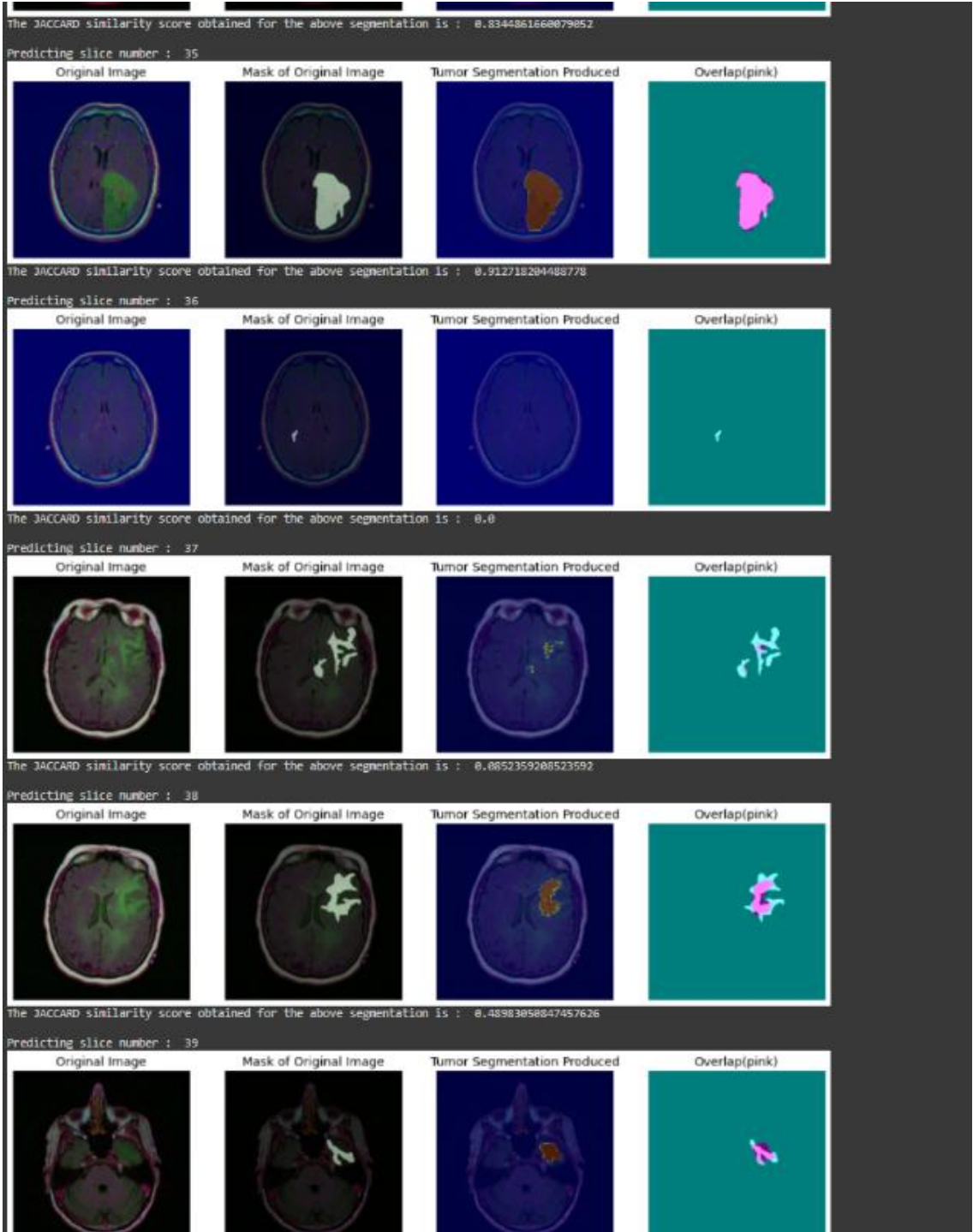


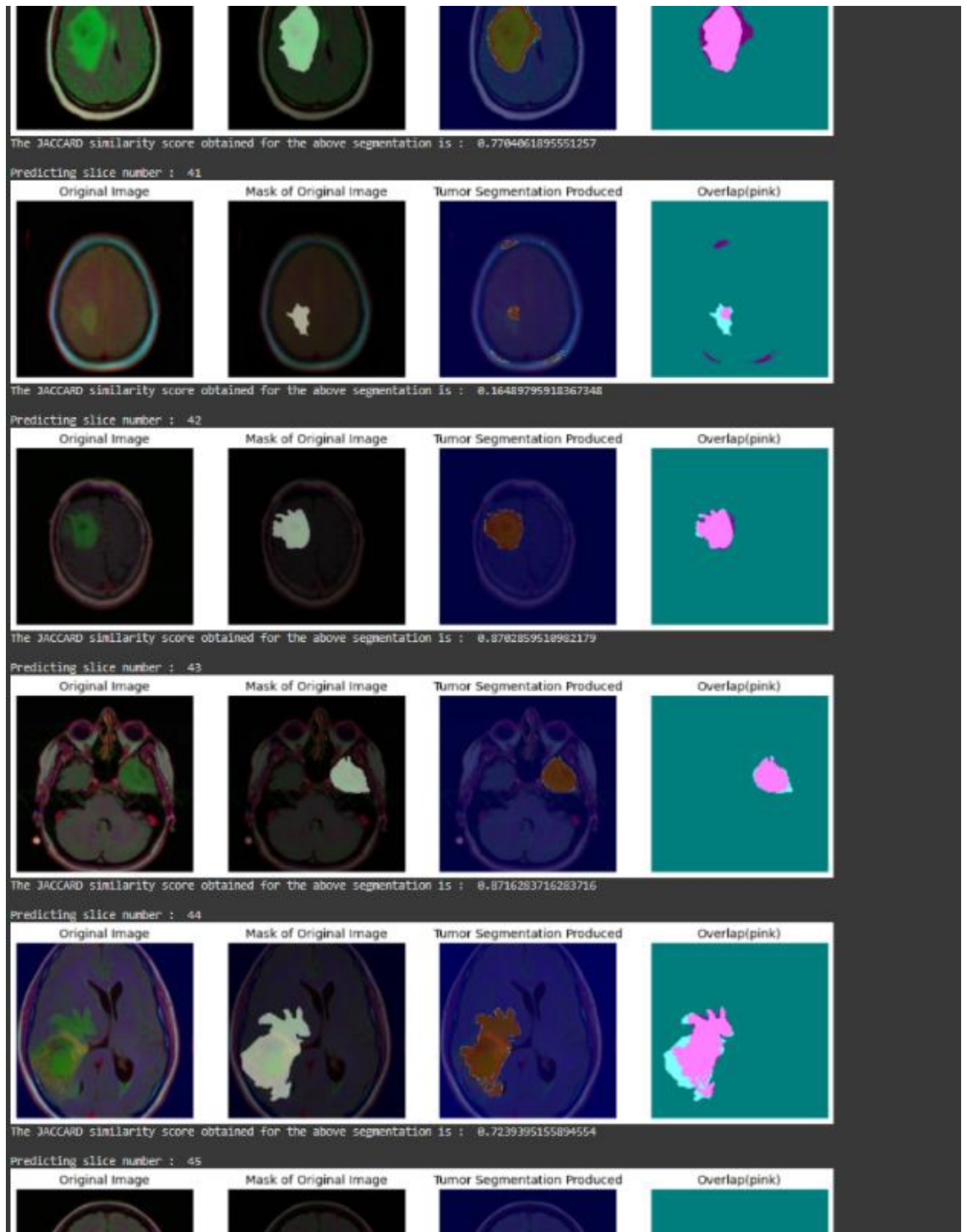


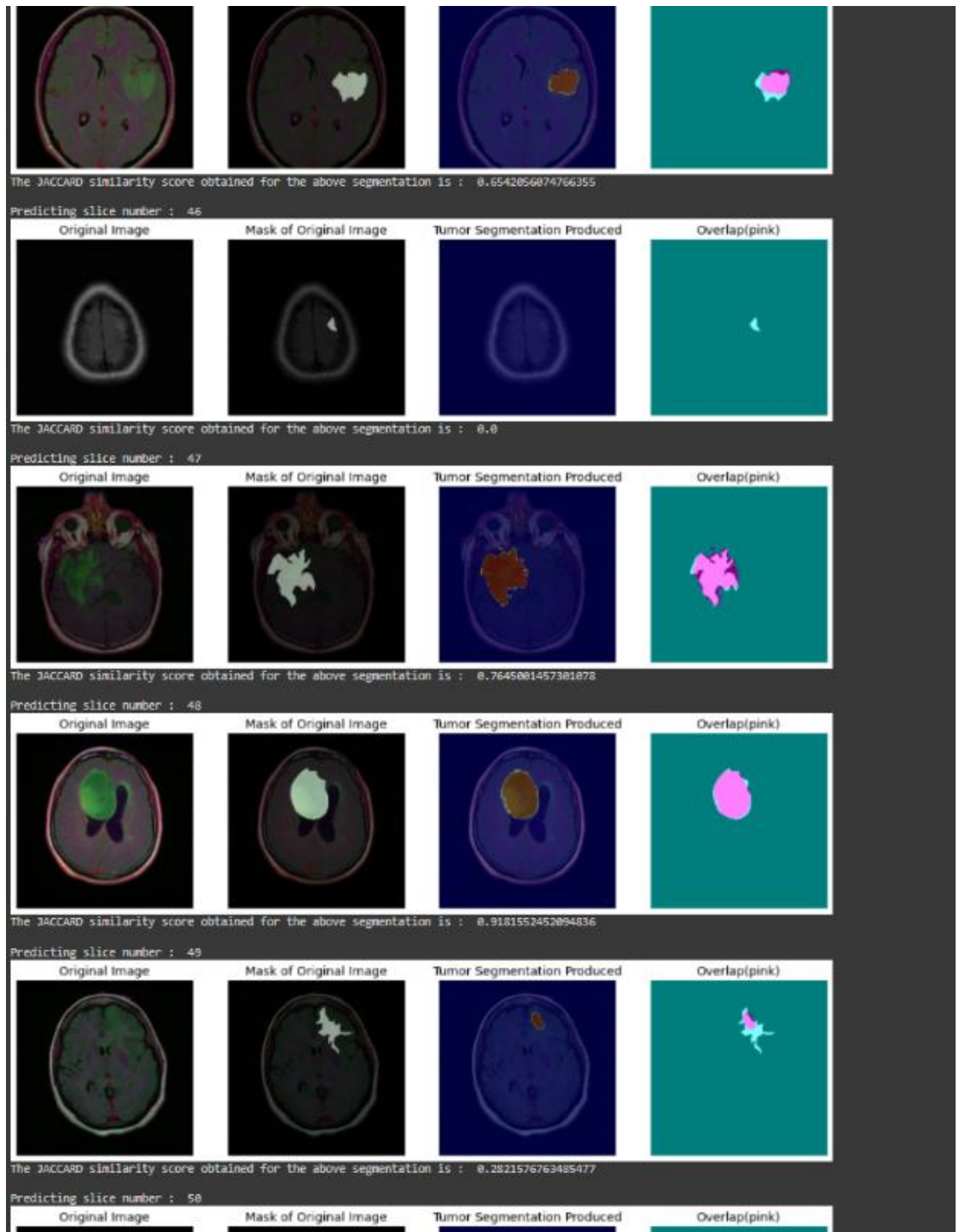


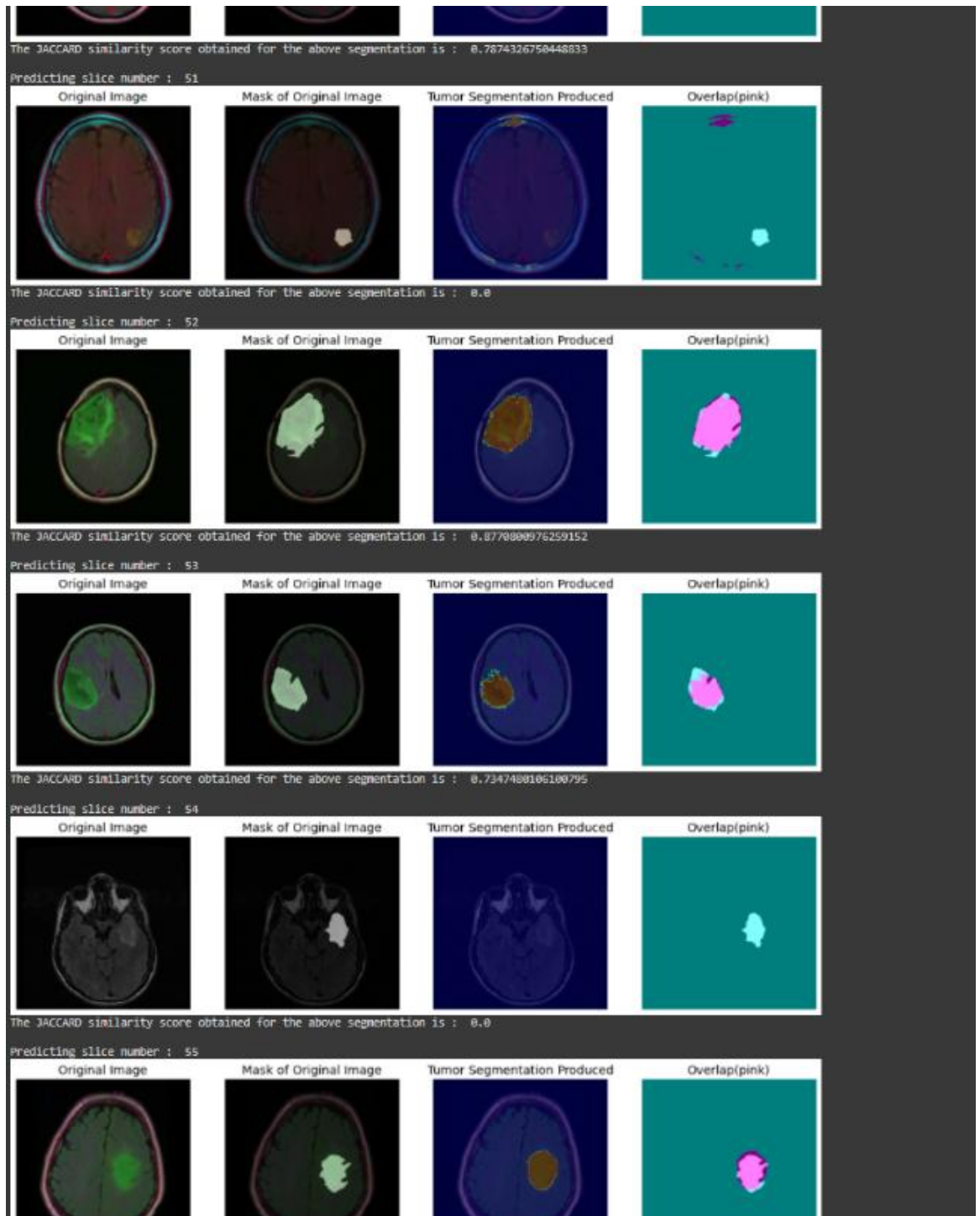


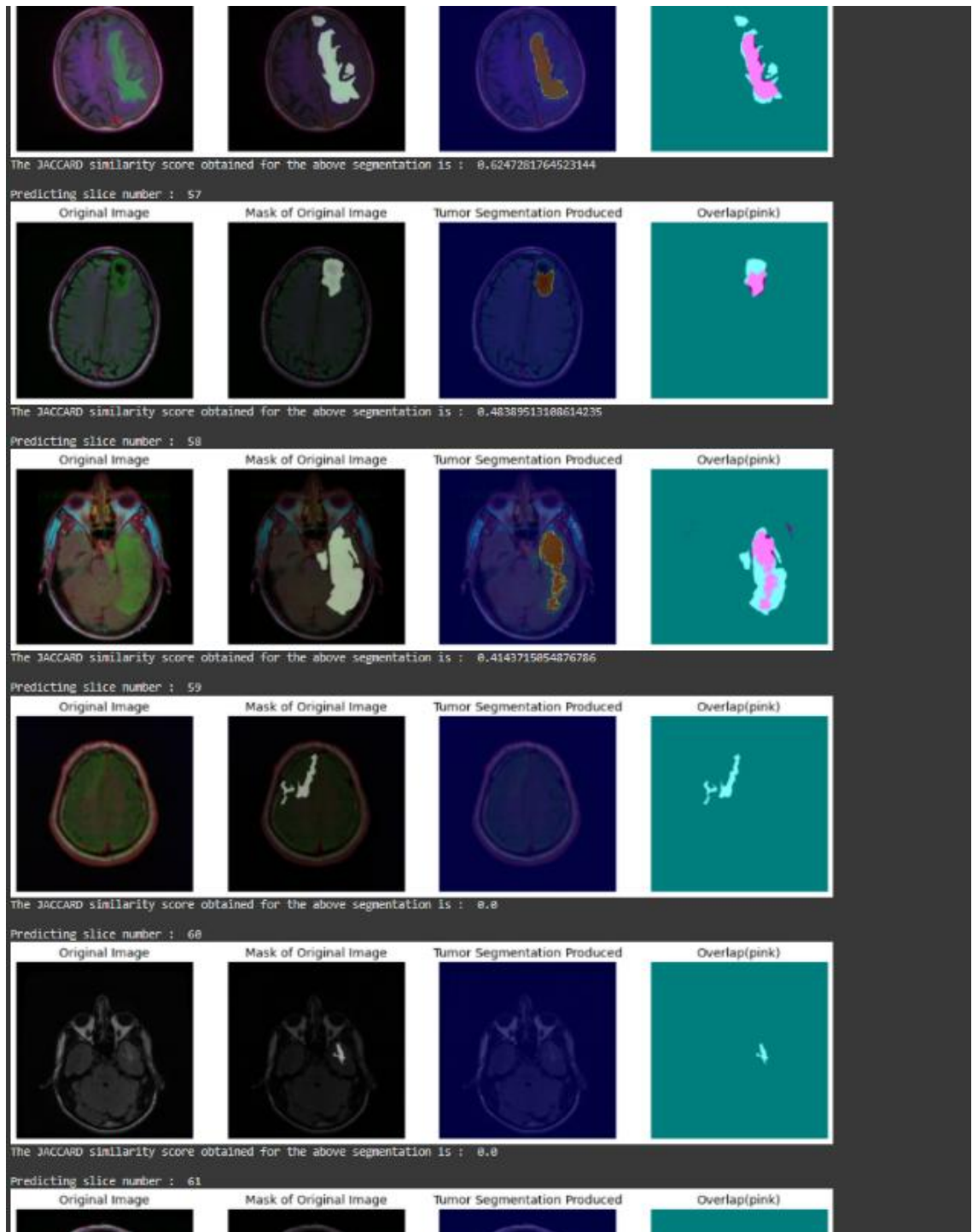


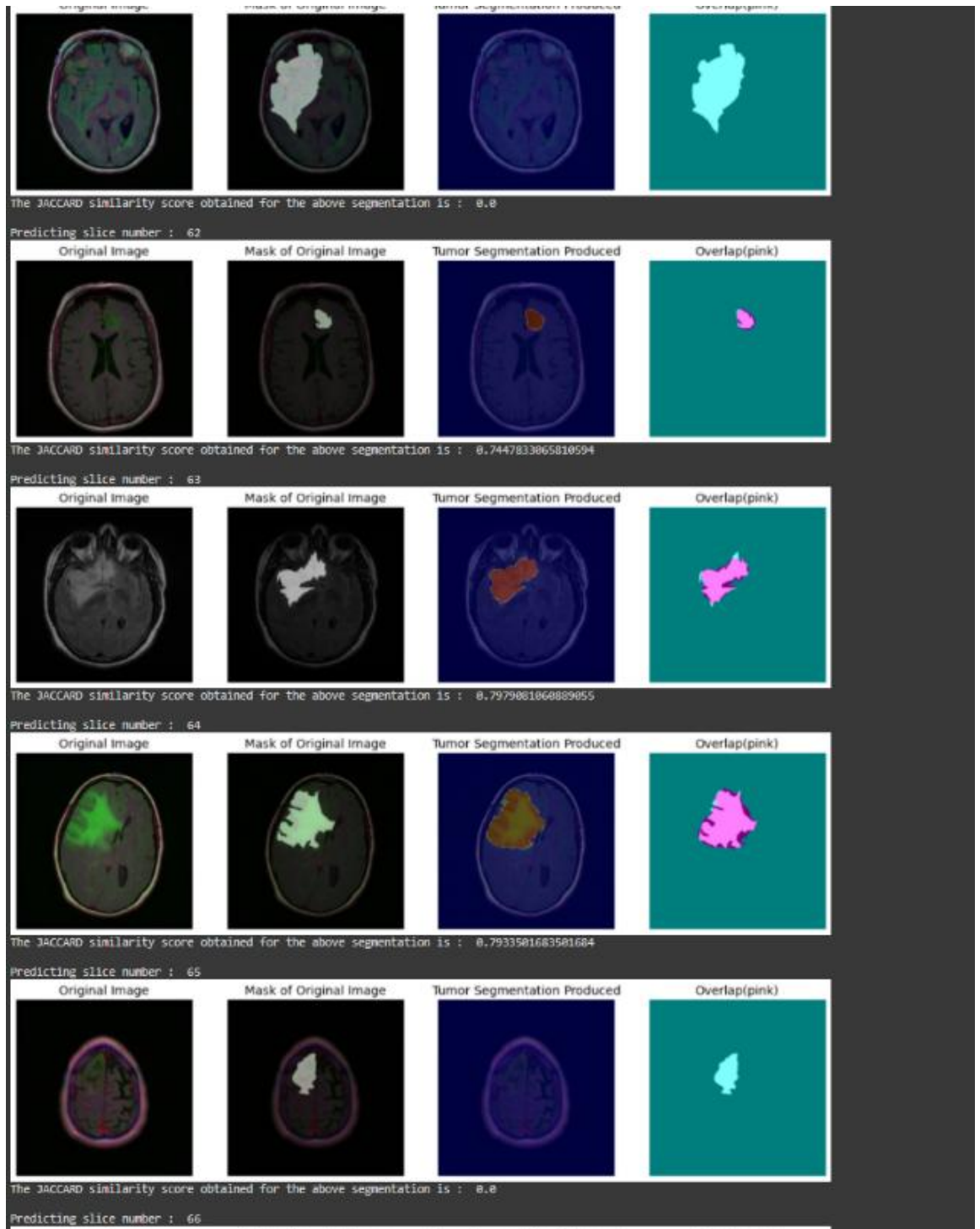


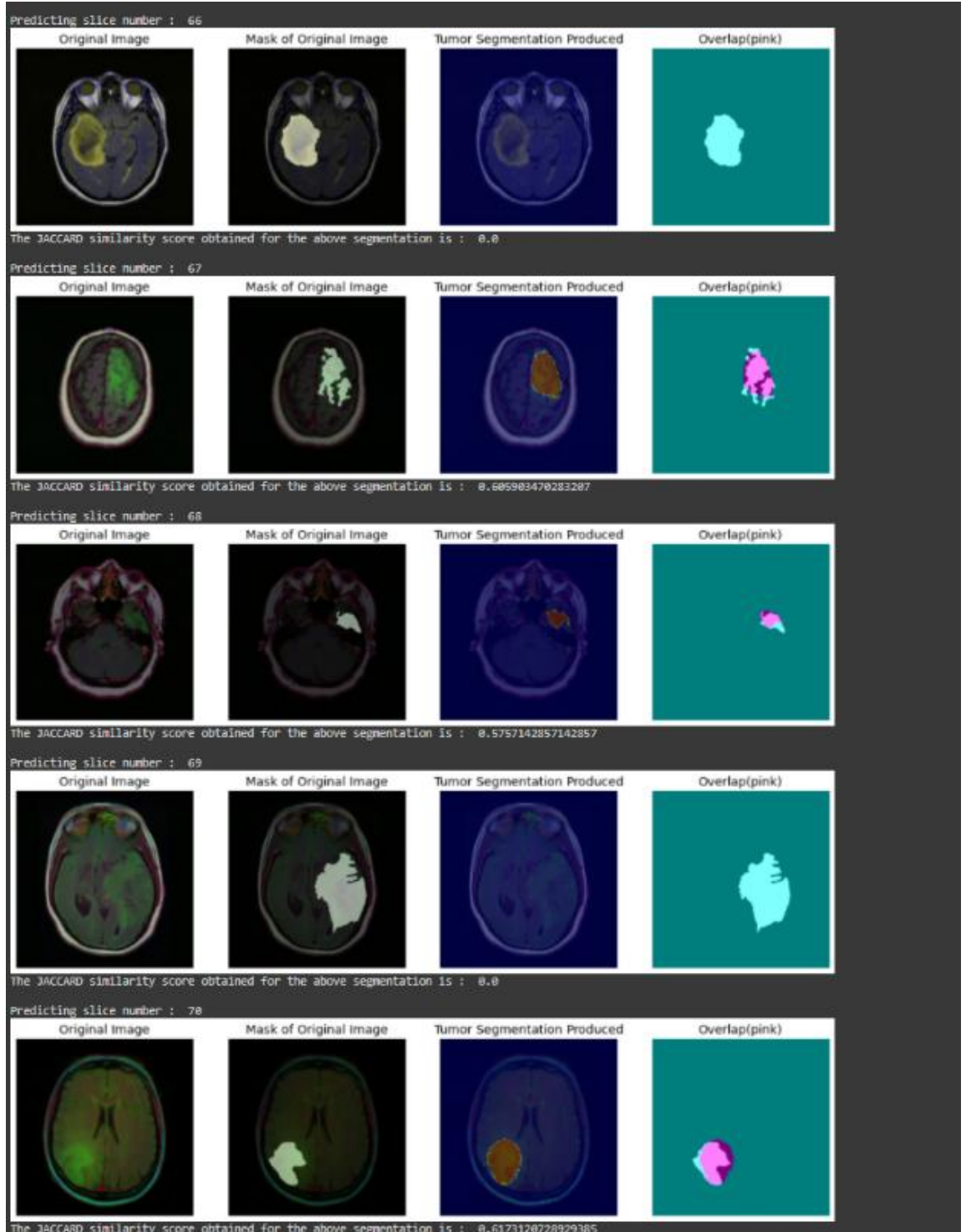


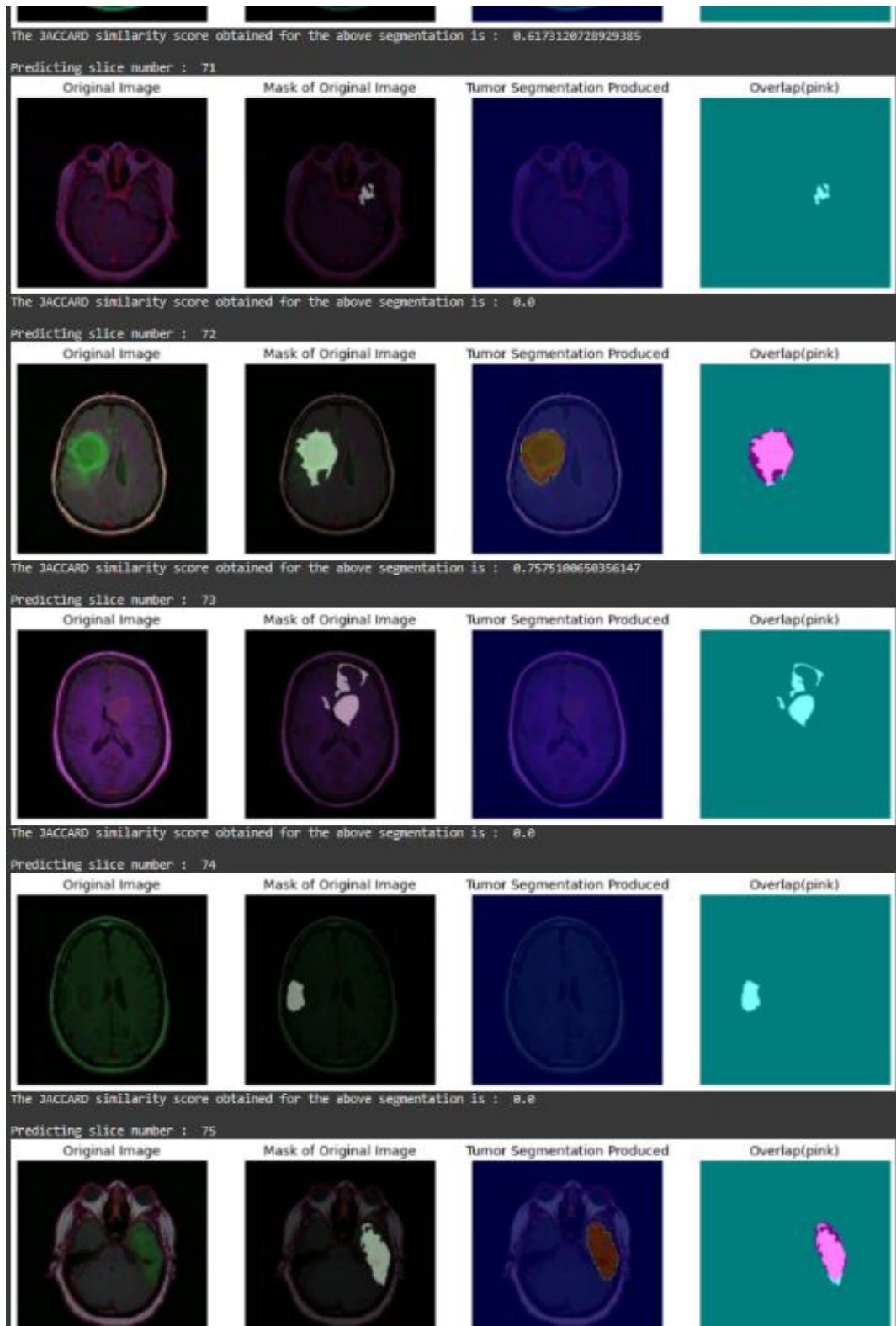


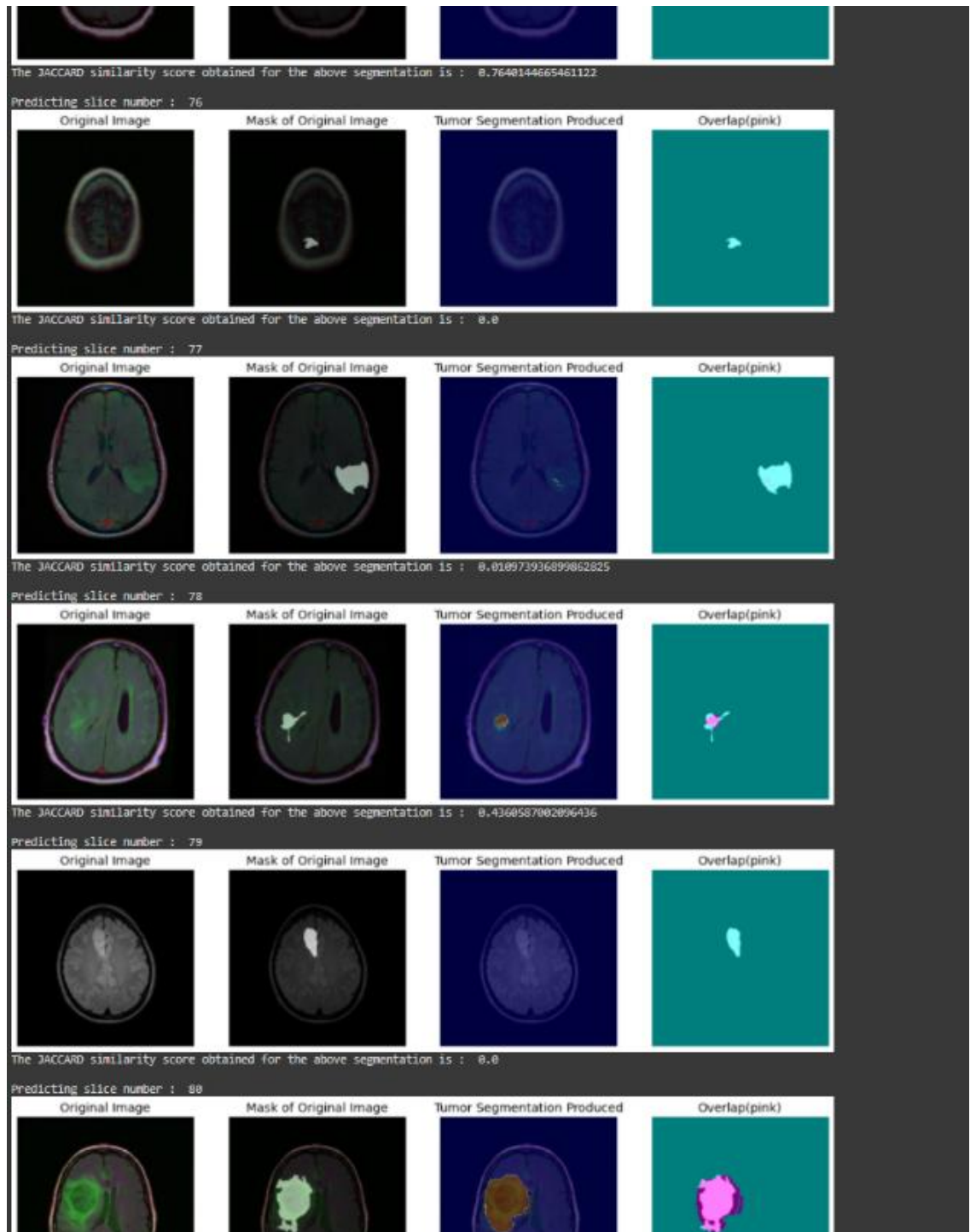


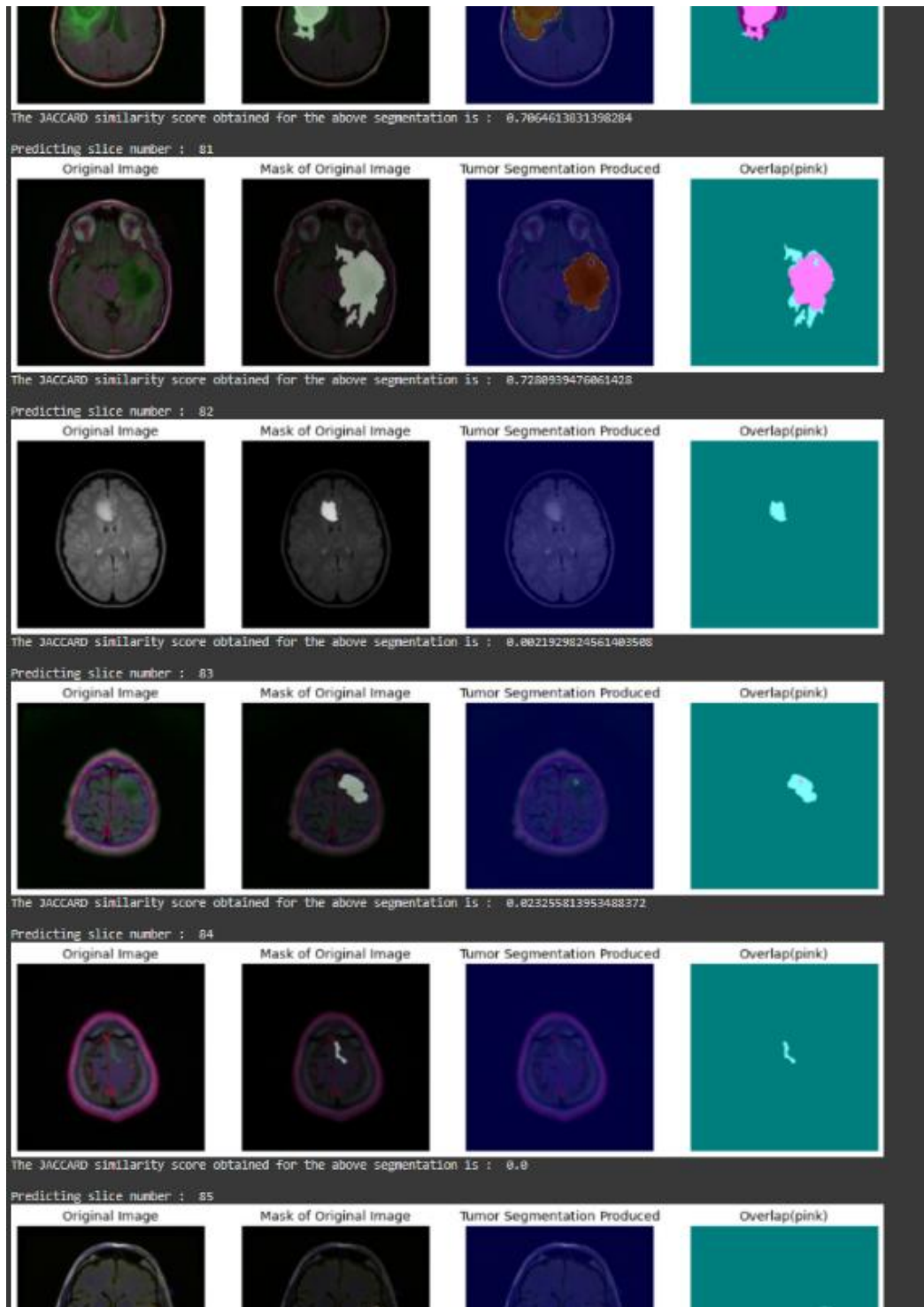


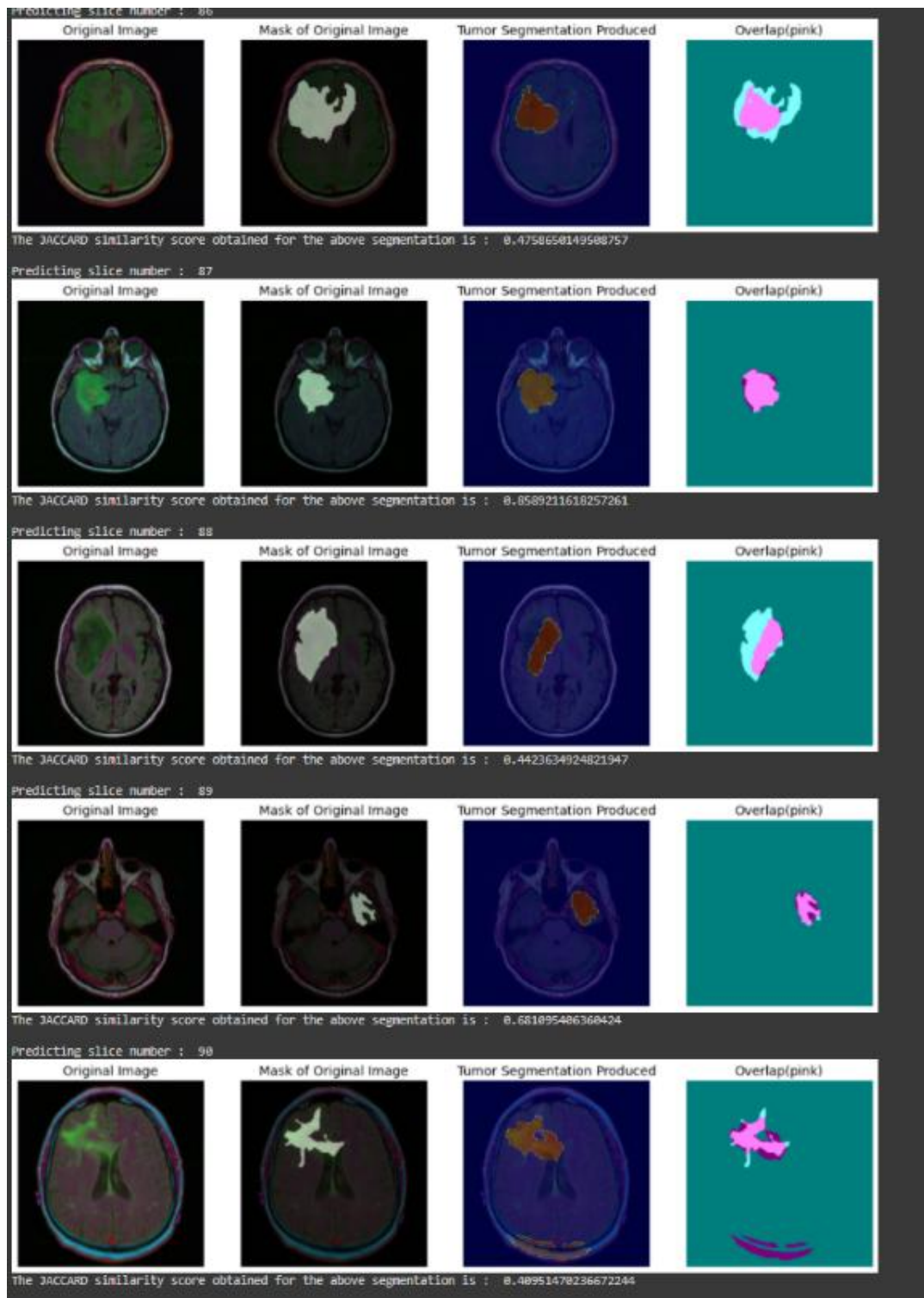


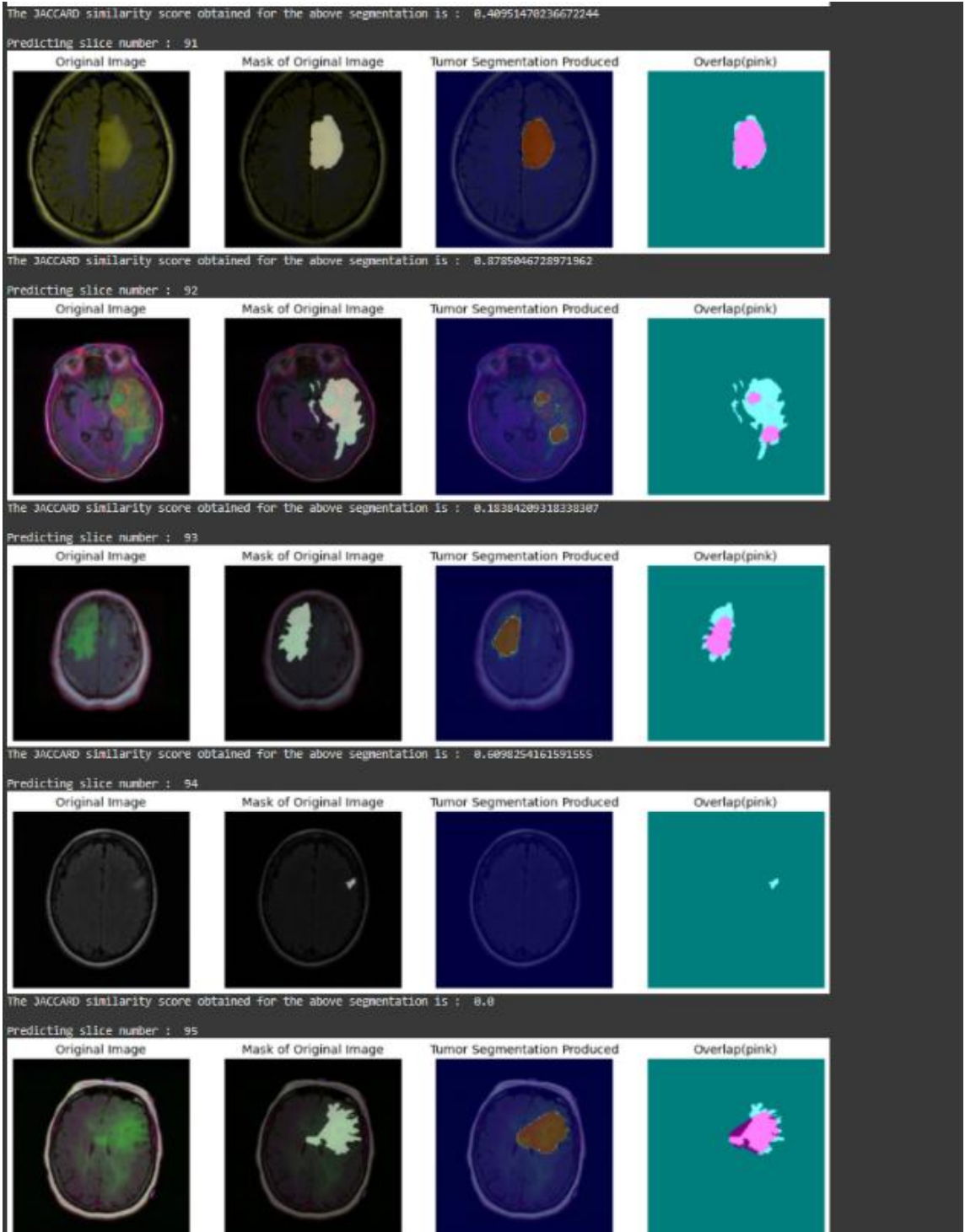


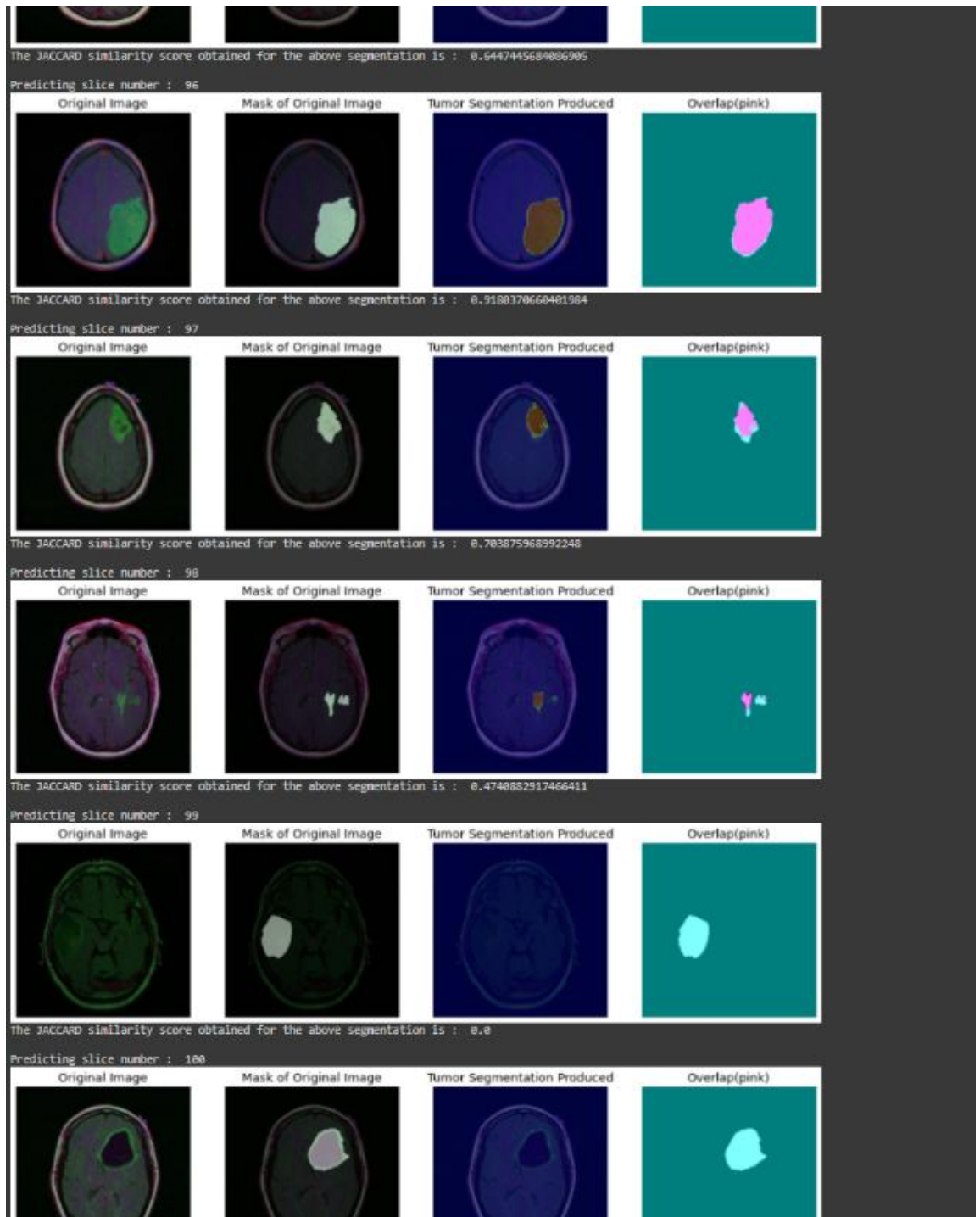


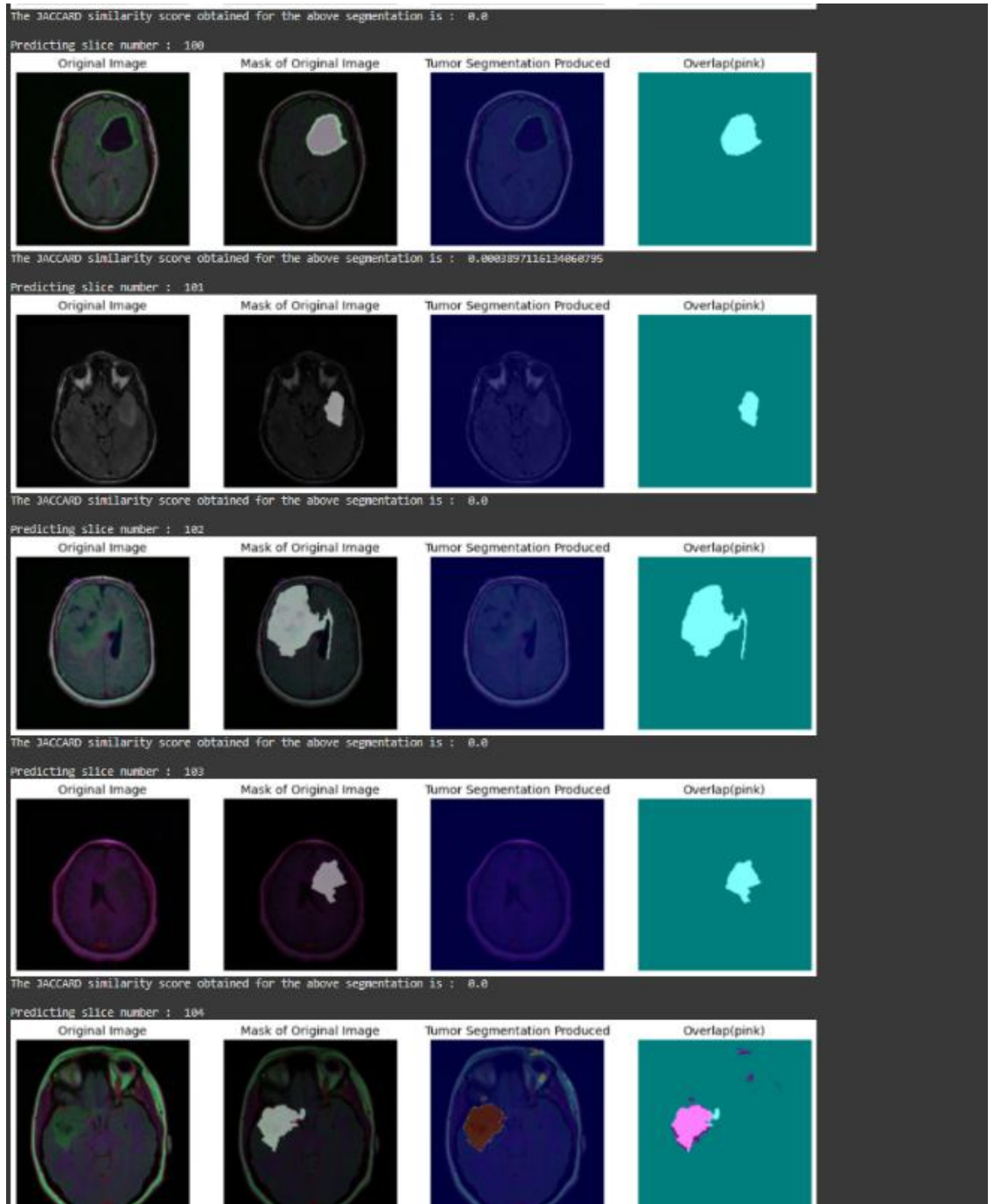


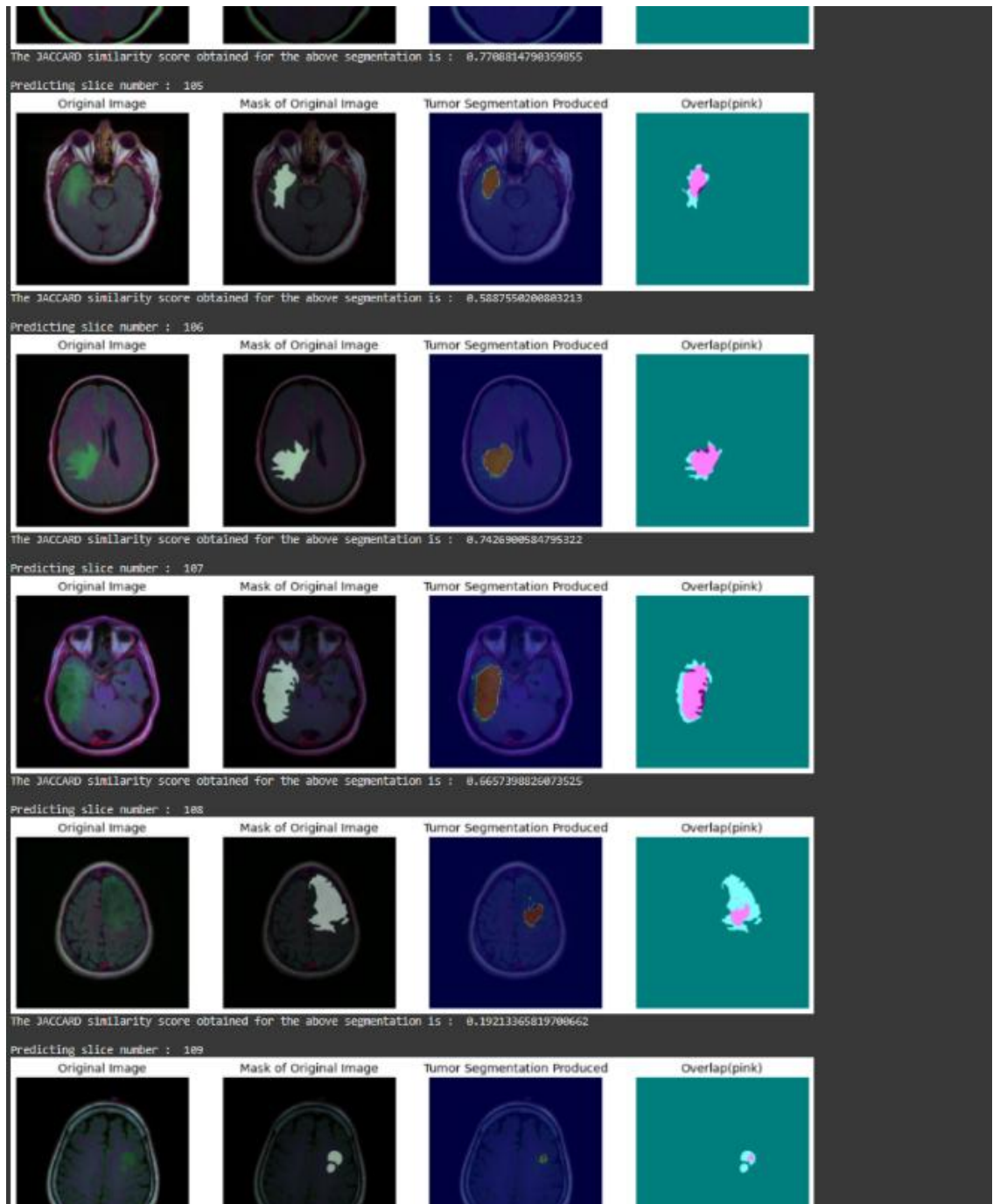


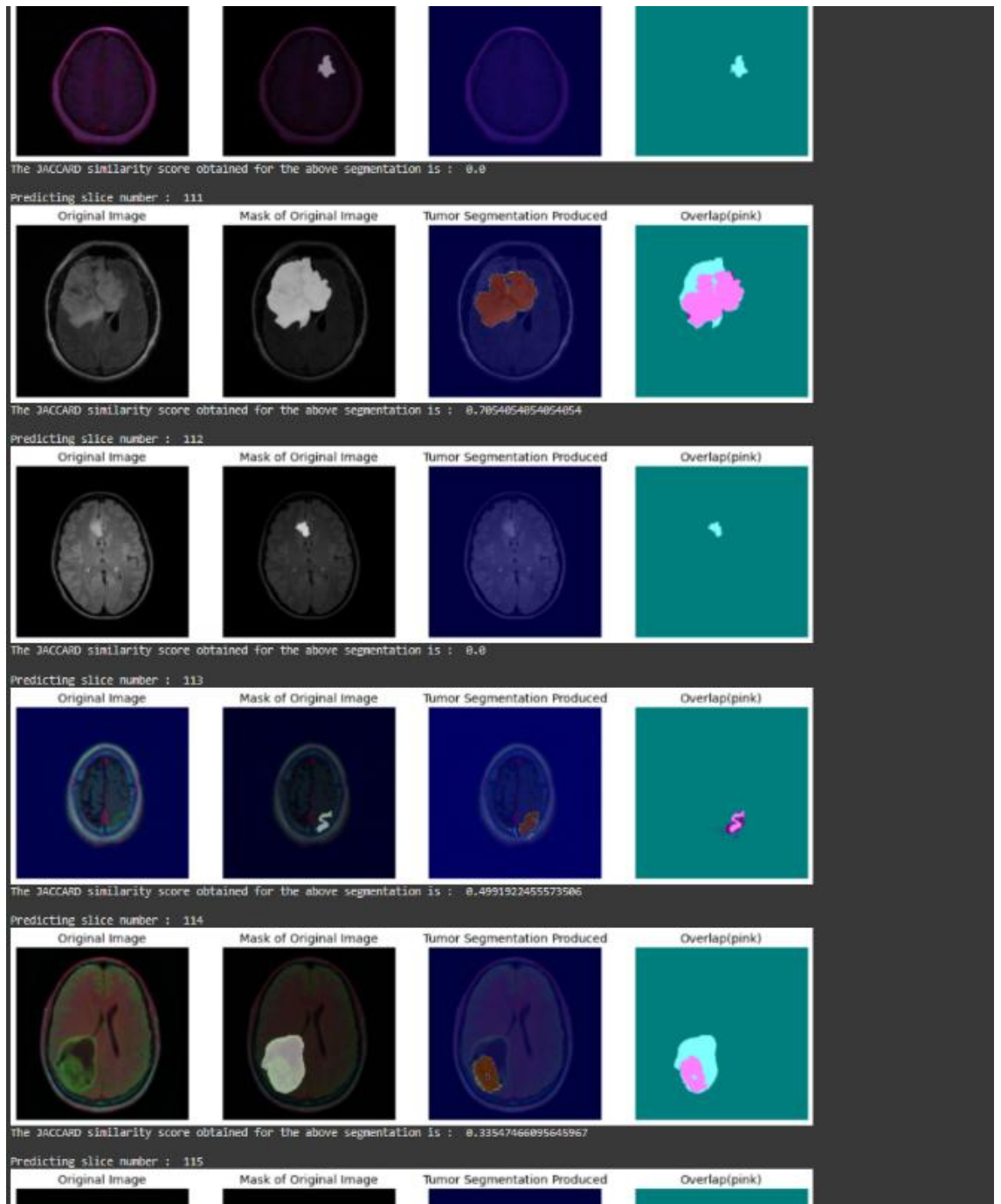


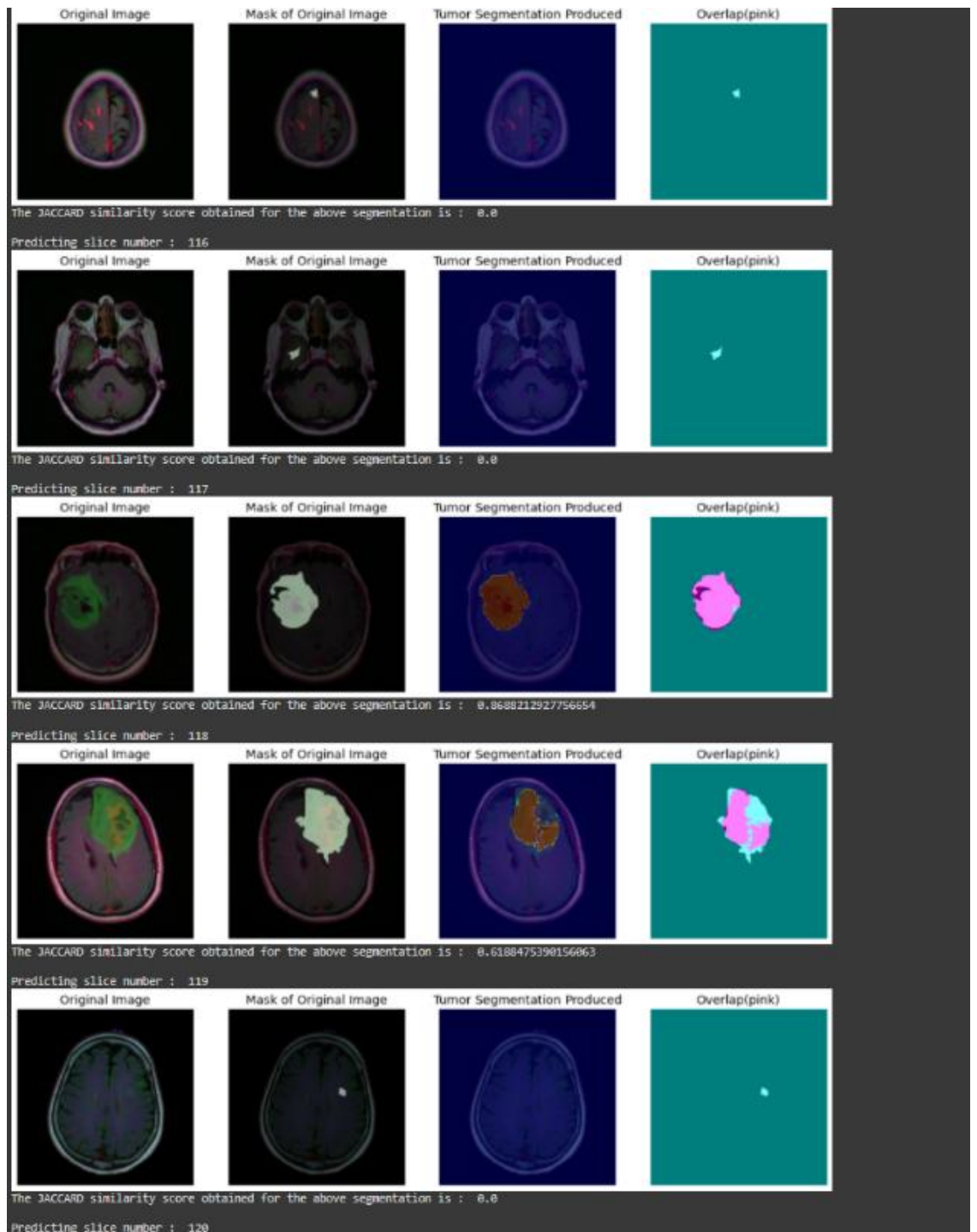


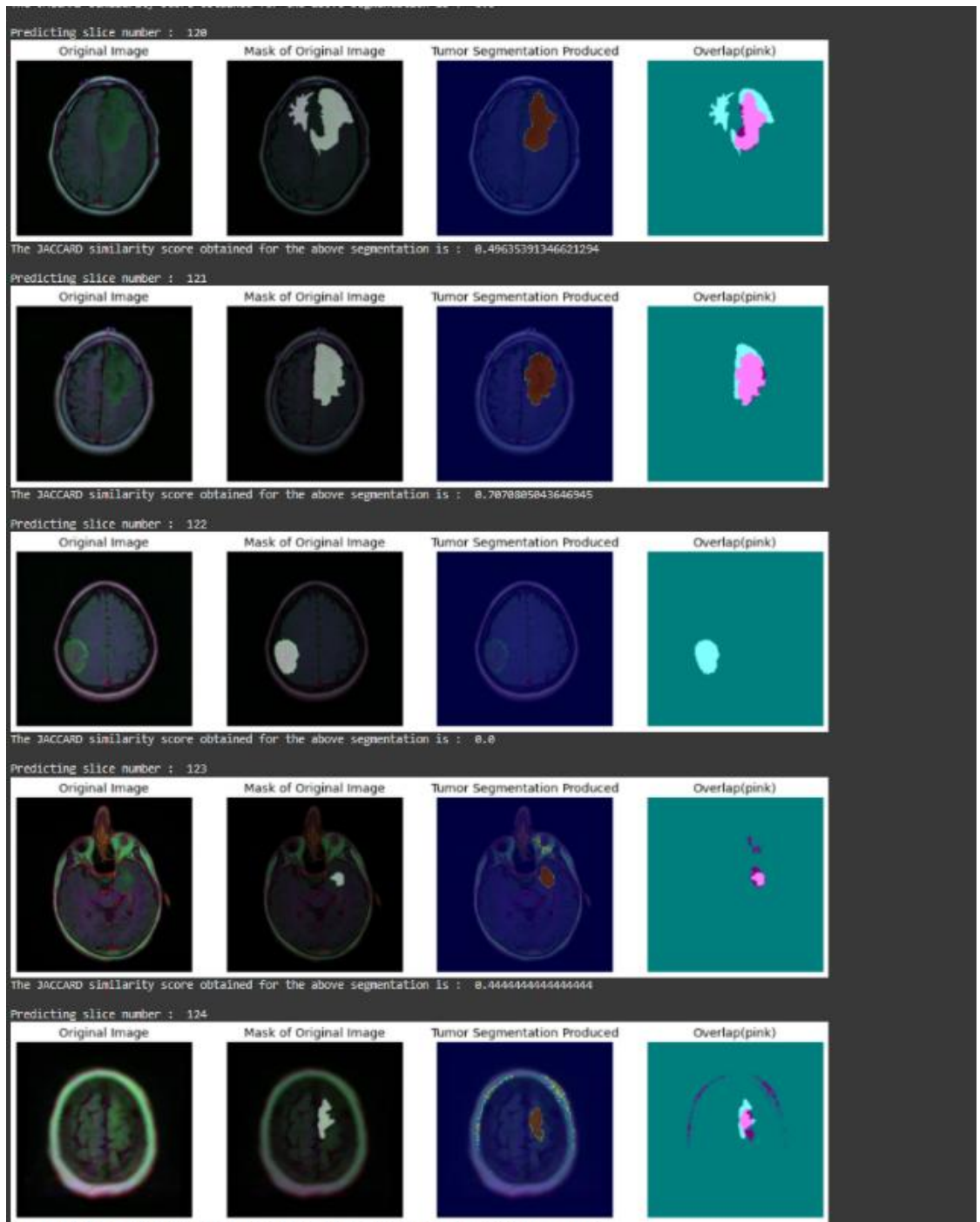


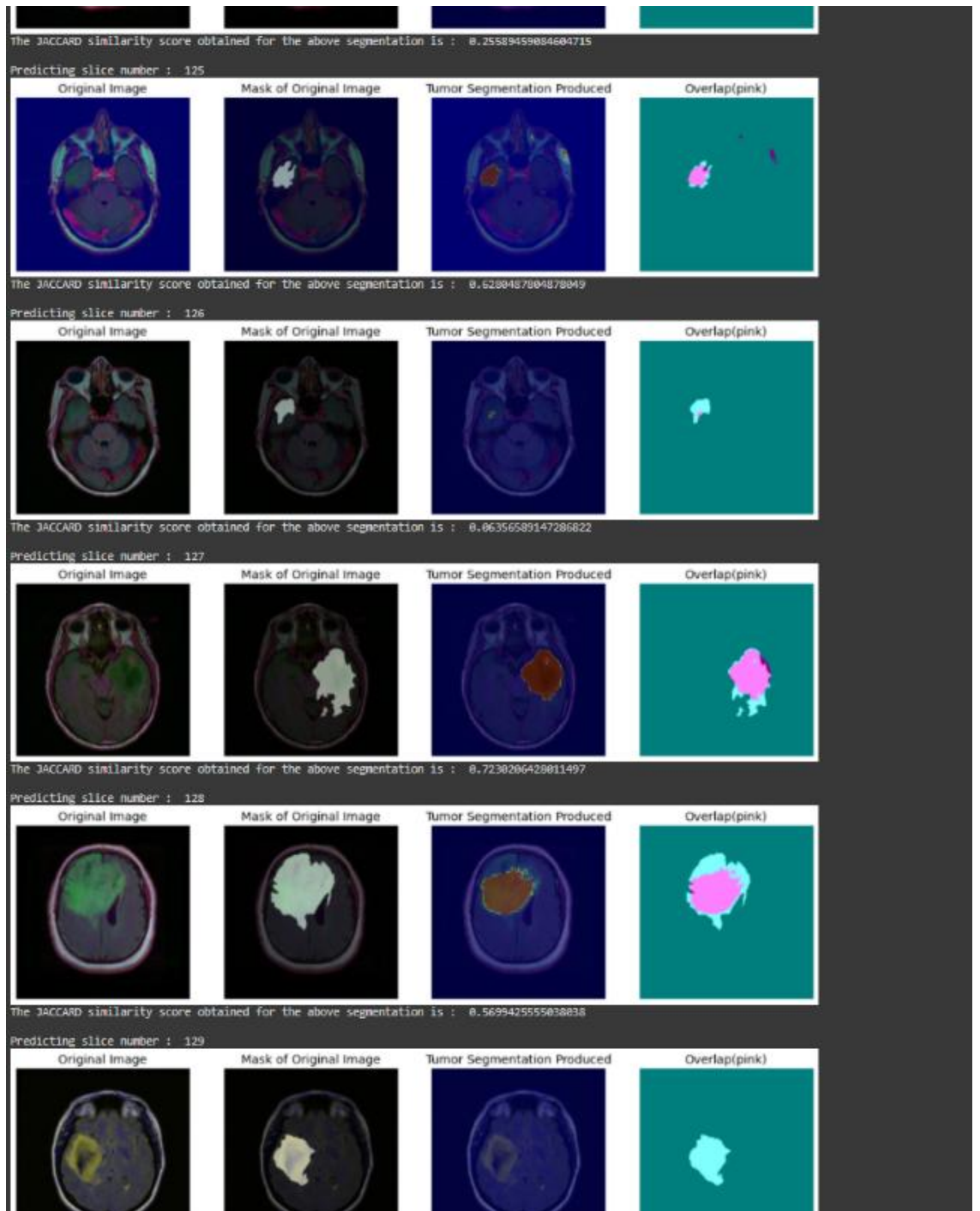


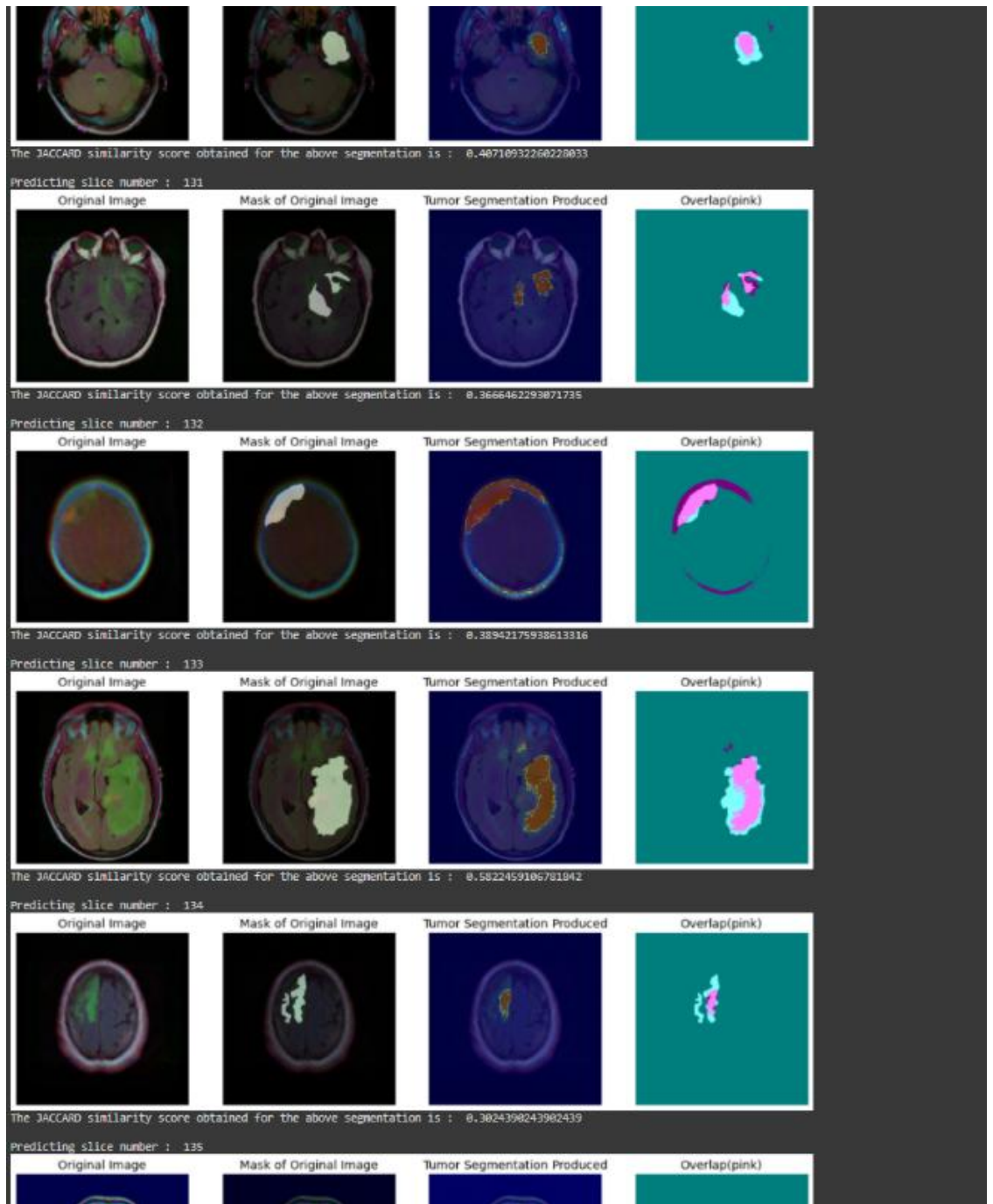


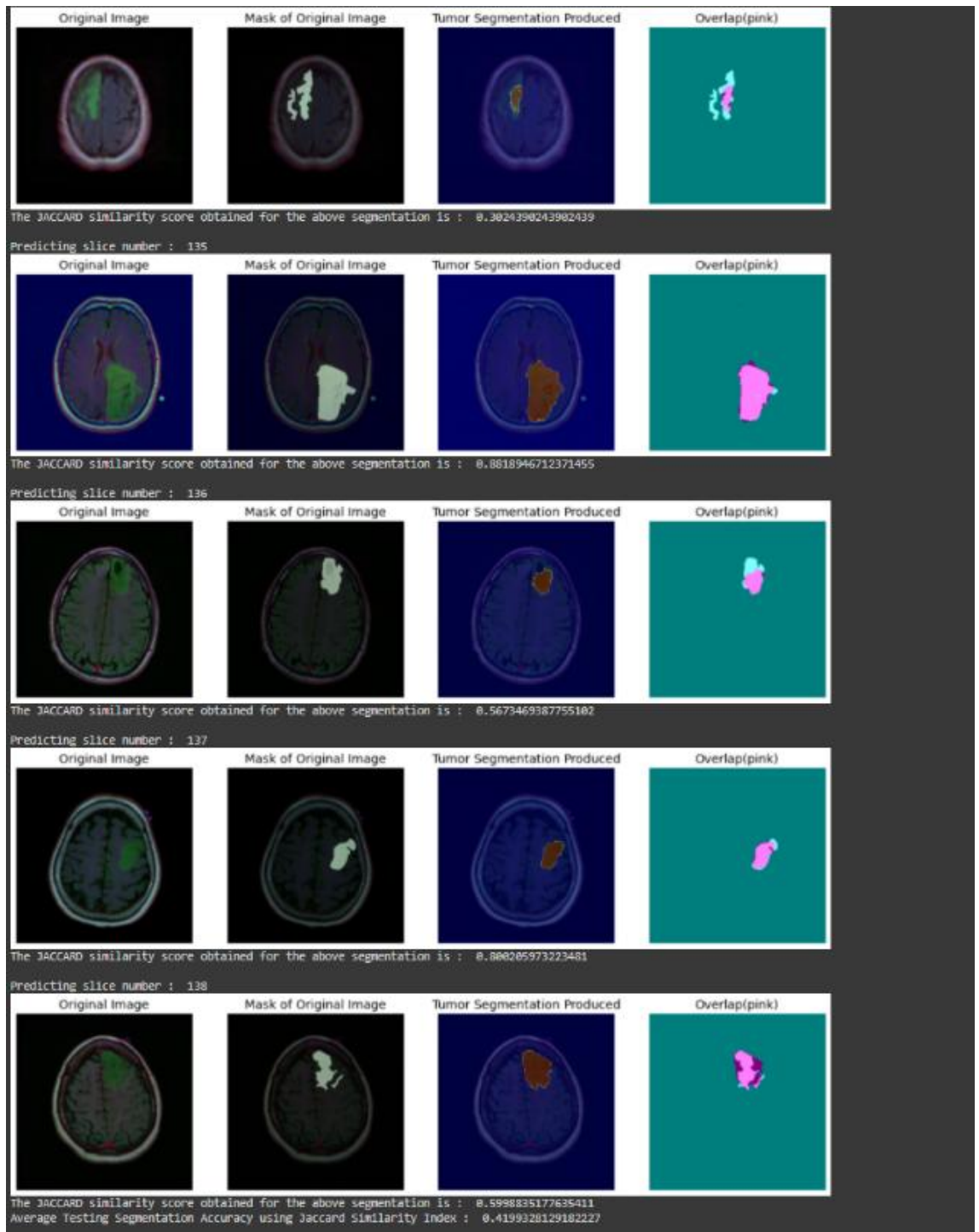




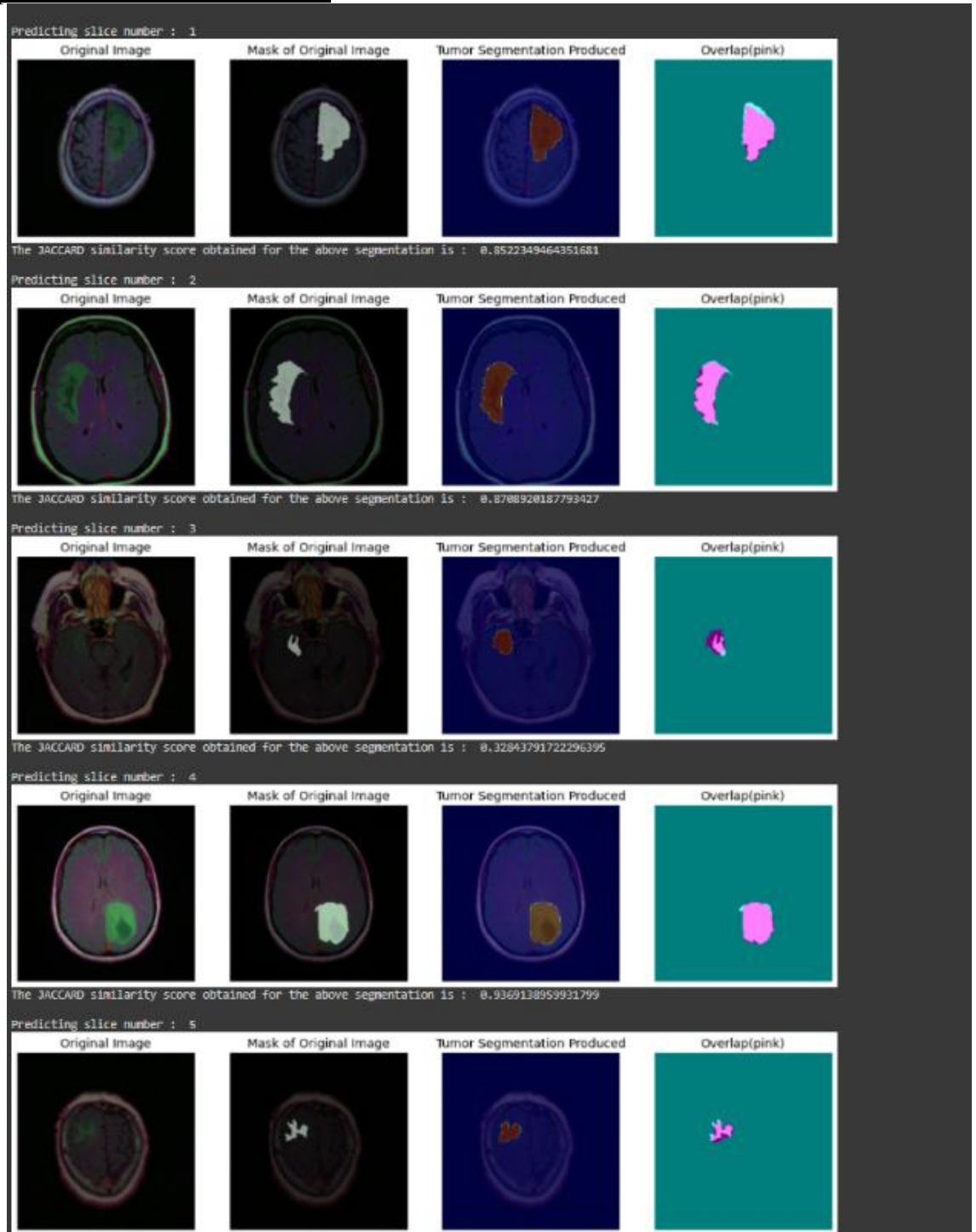


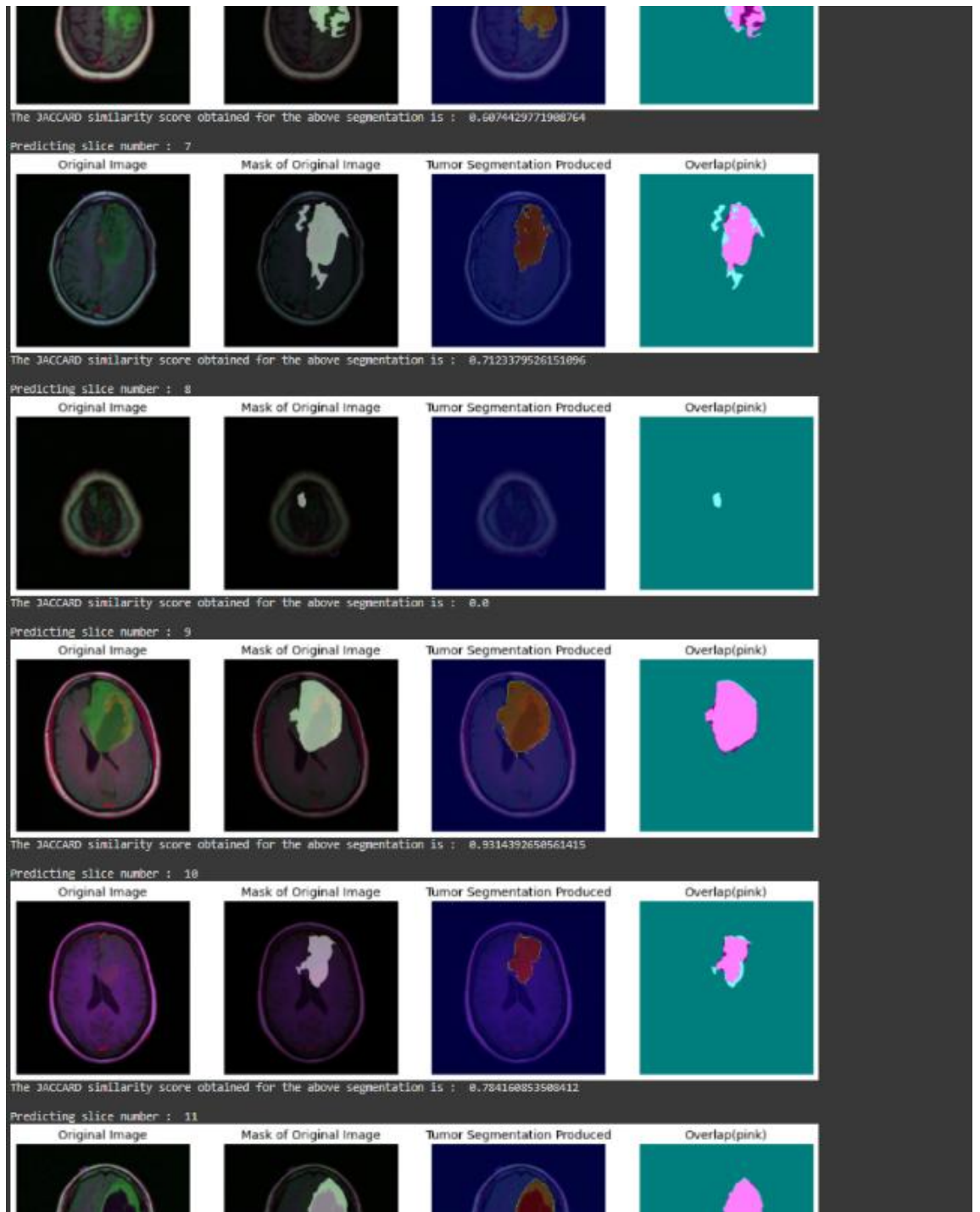


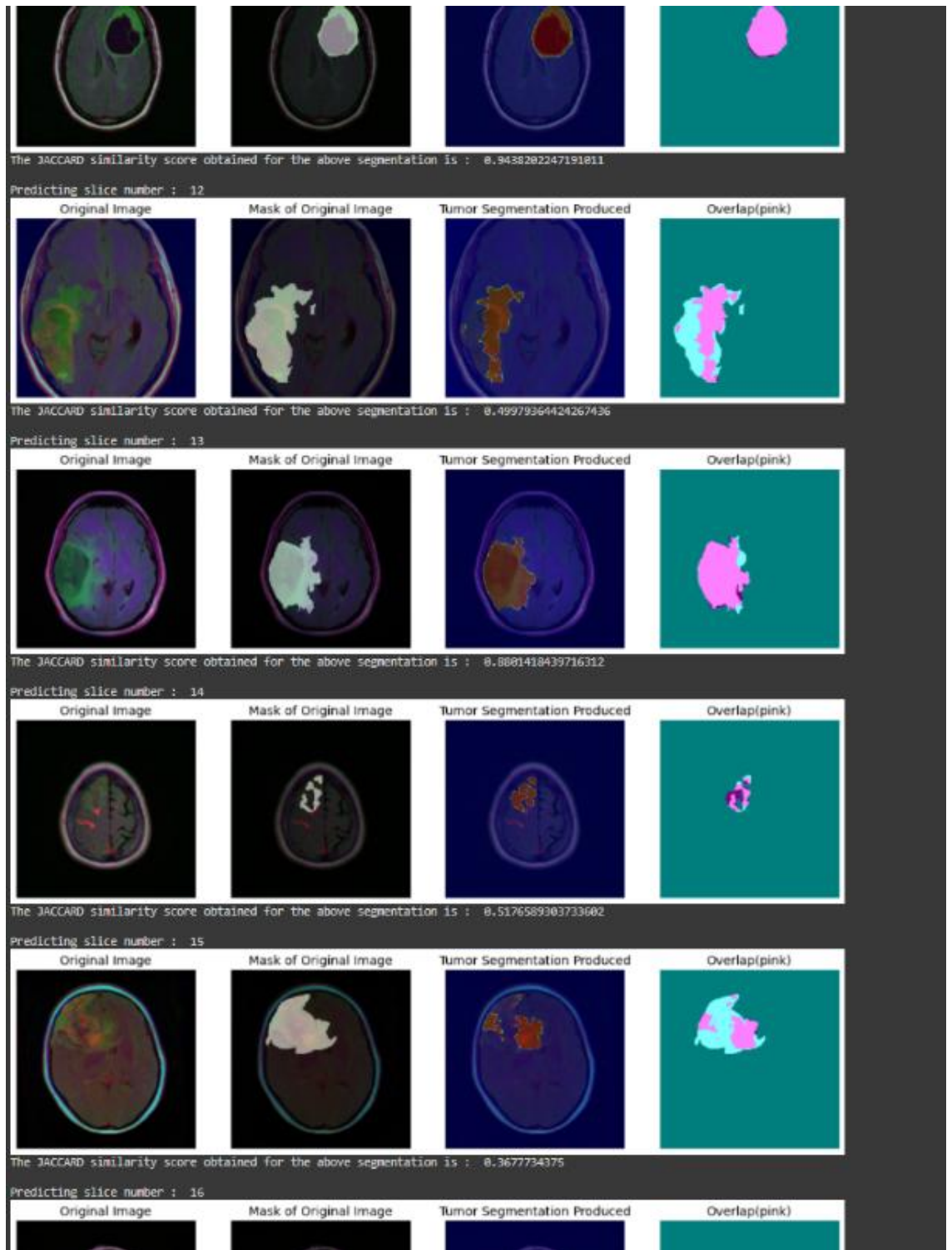


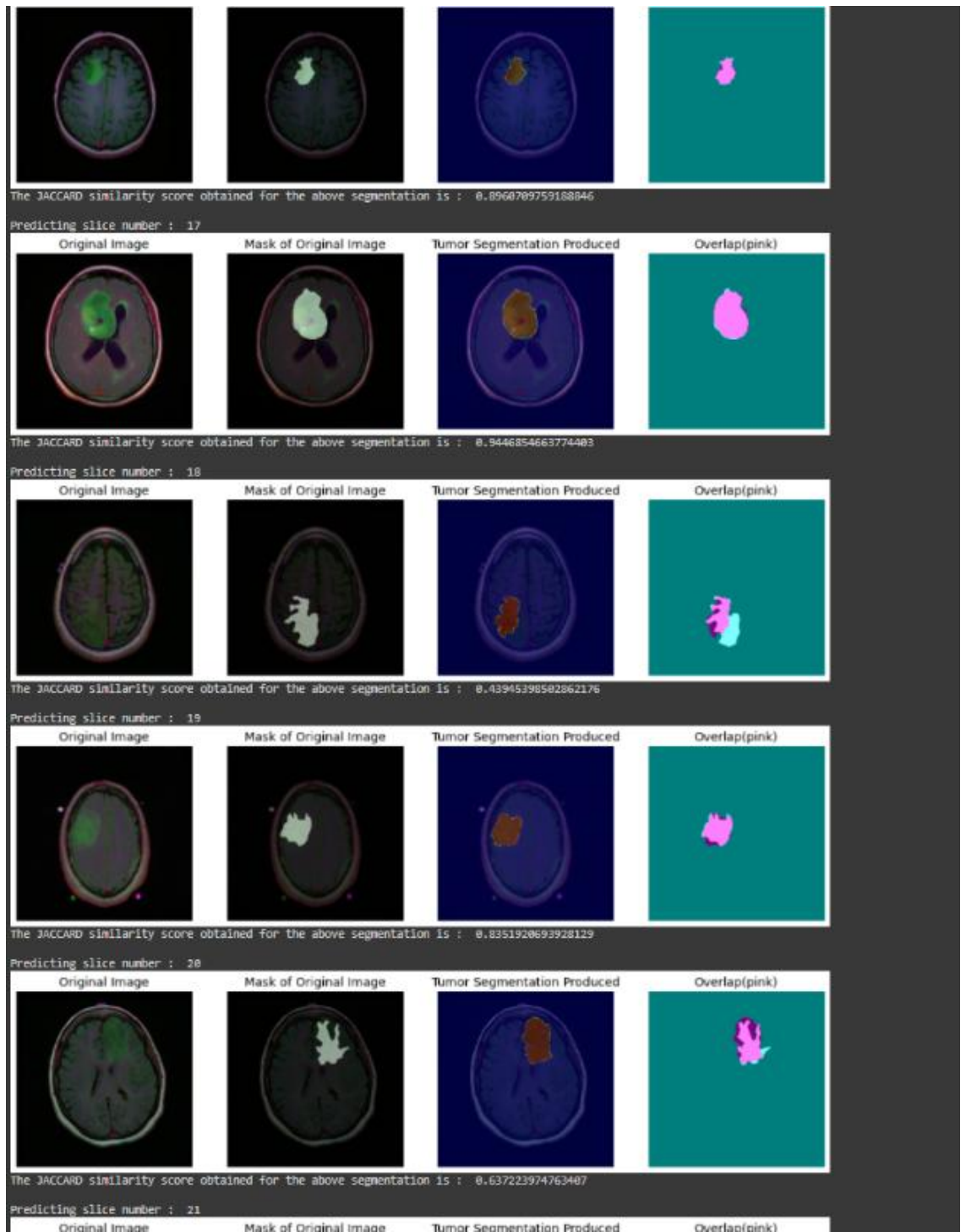


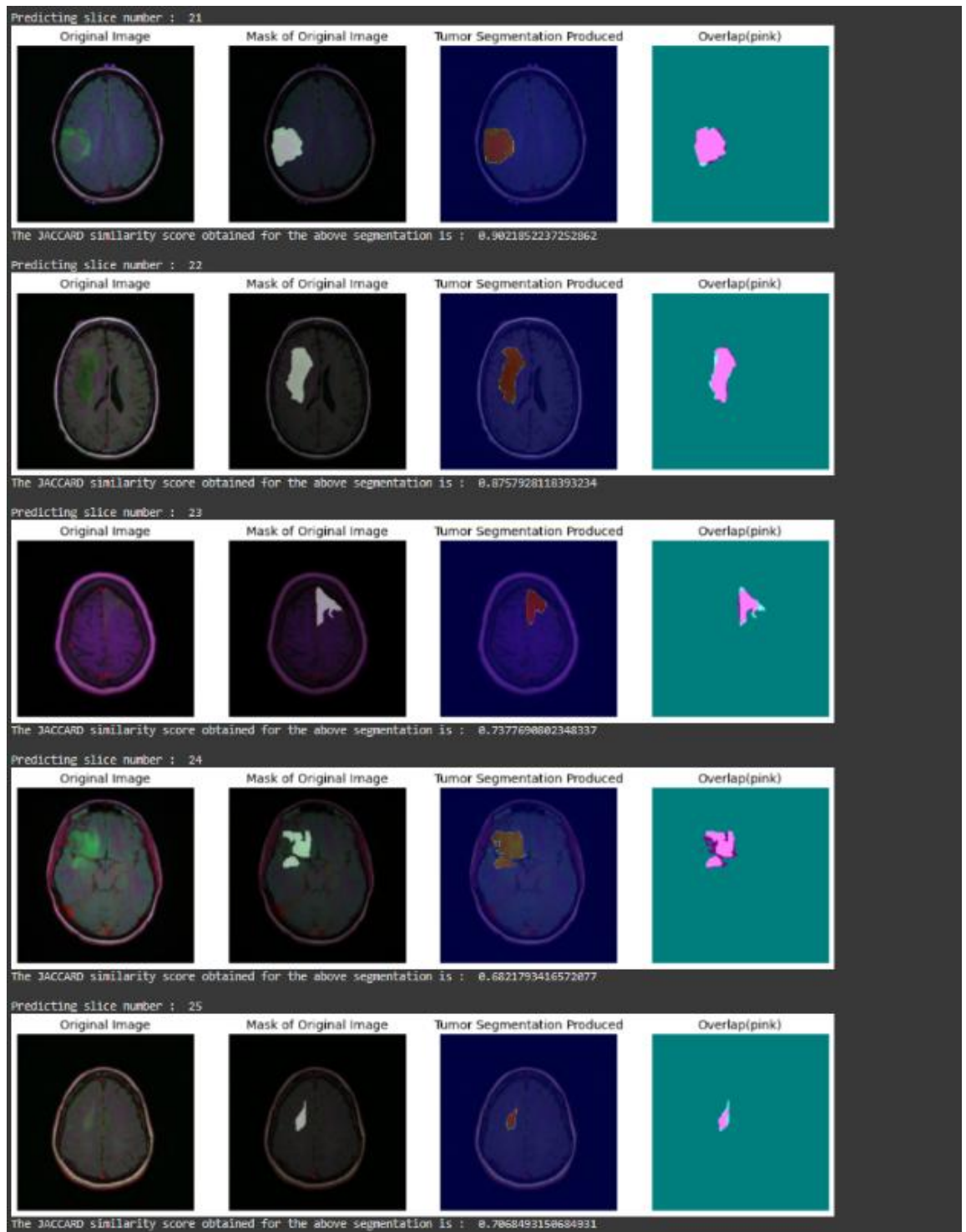
6. Experimental Model No. 13

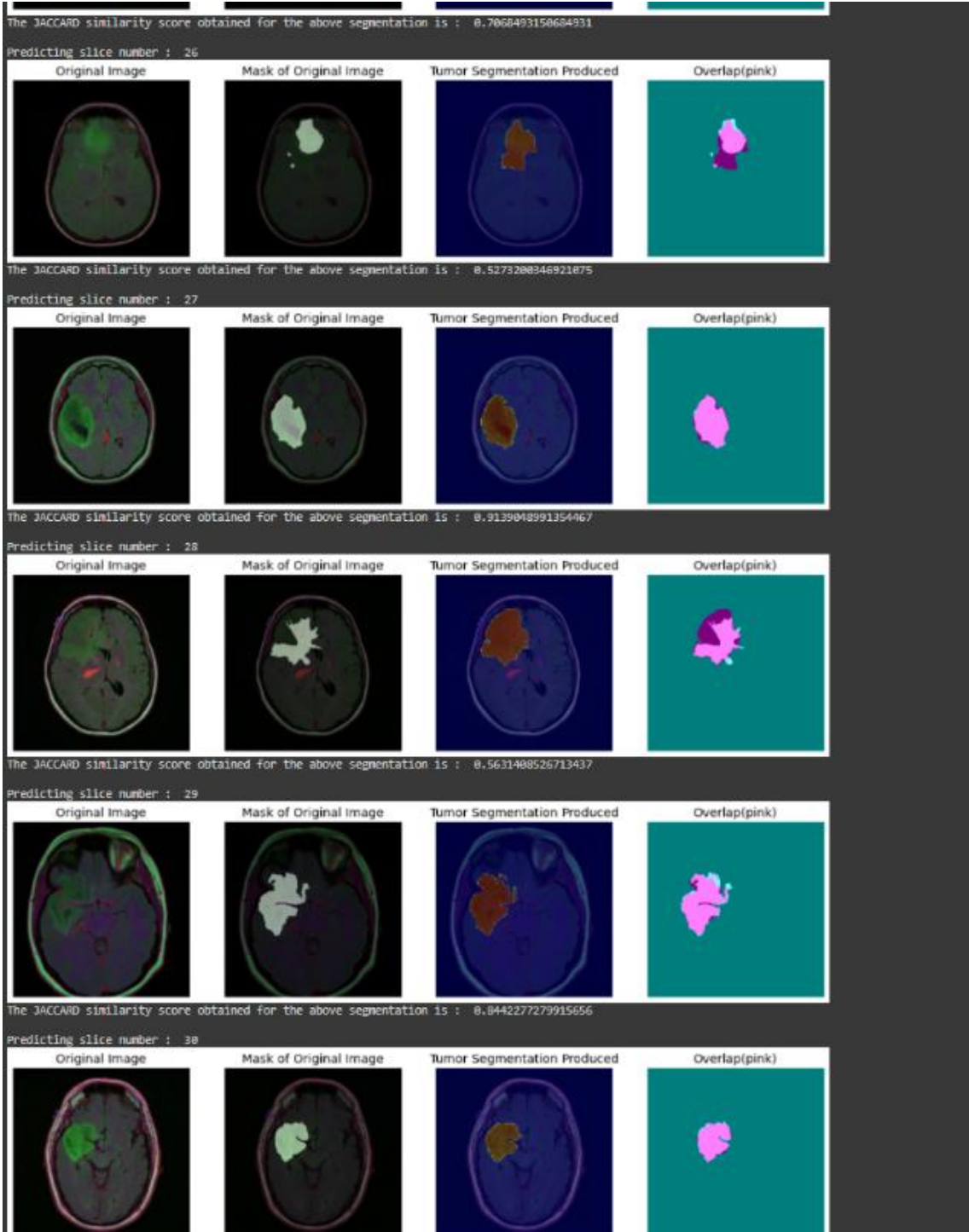


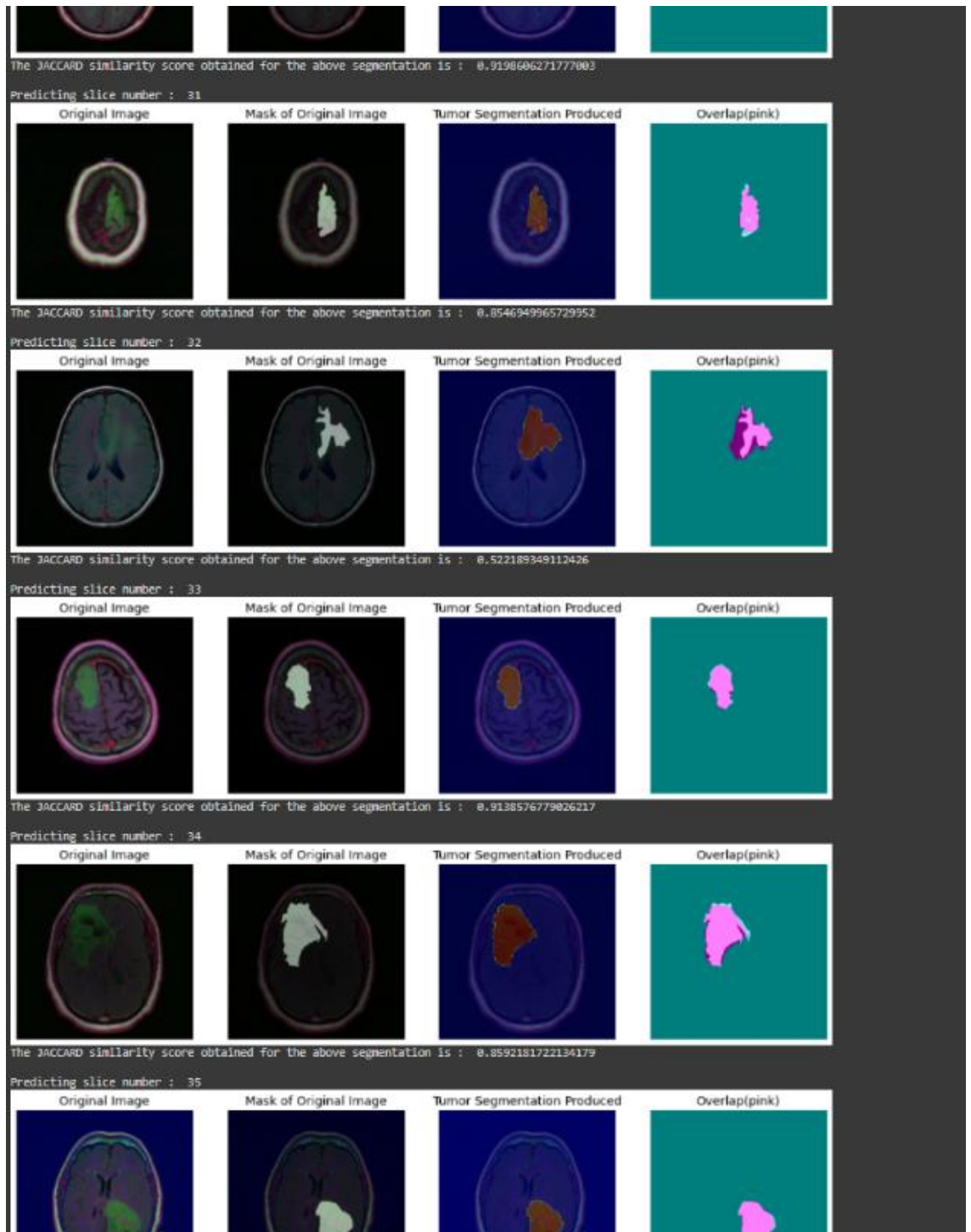


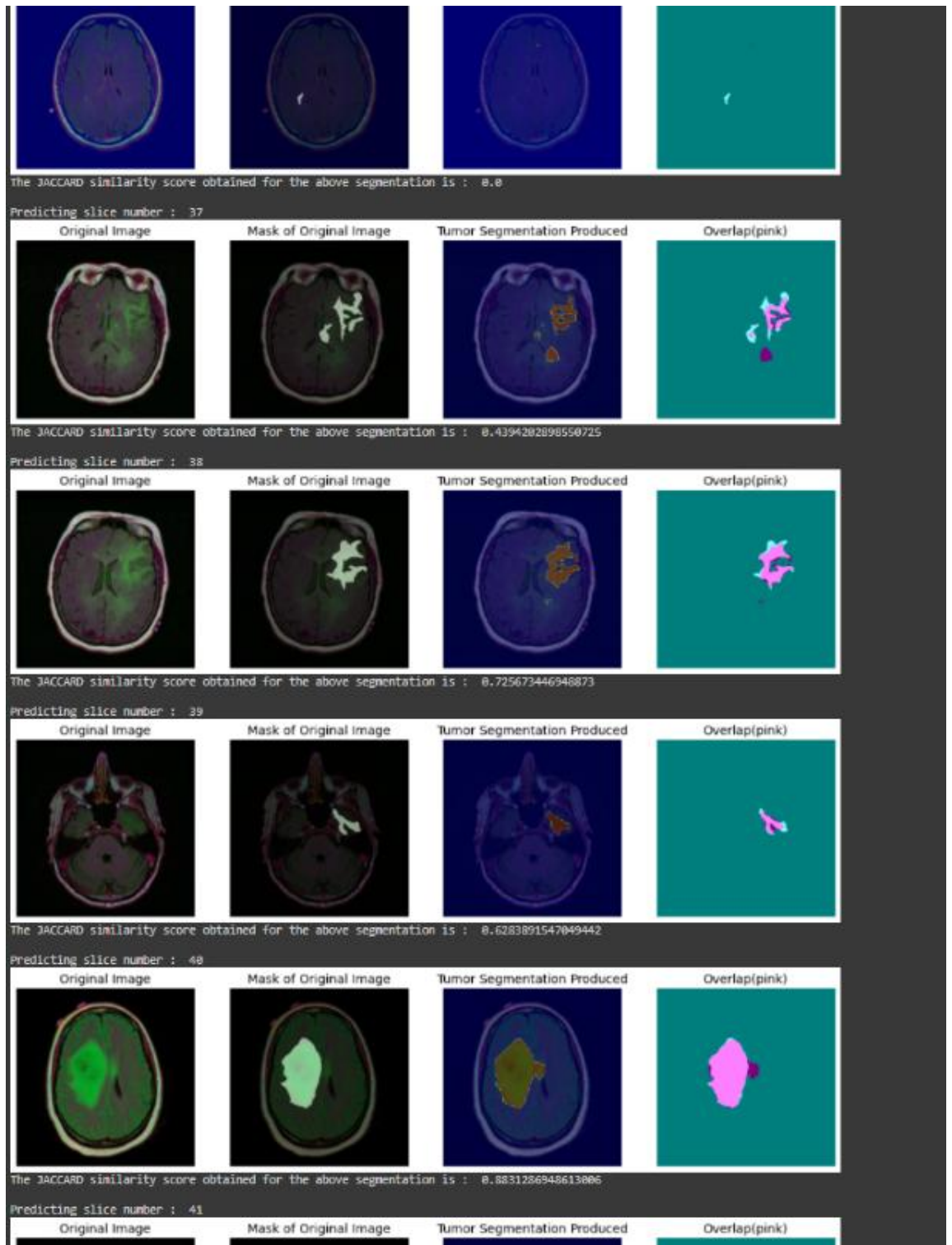


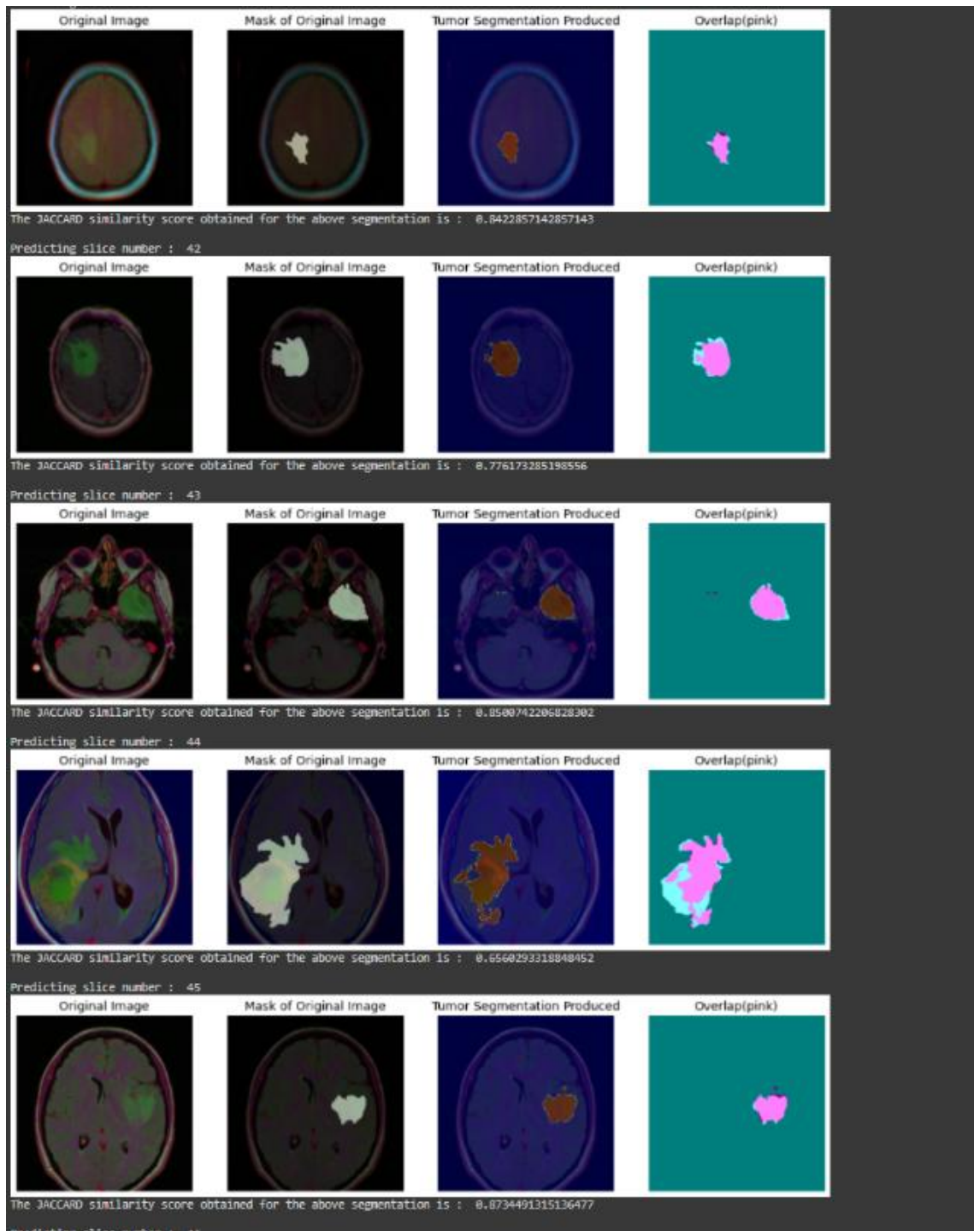


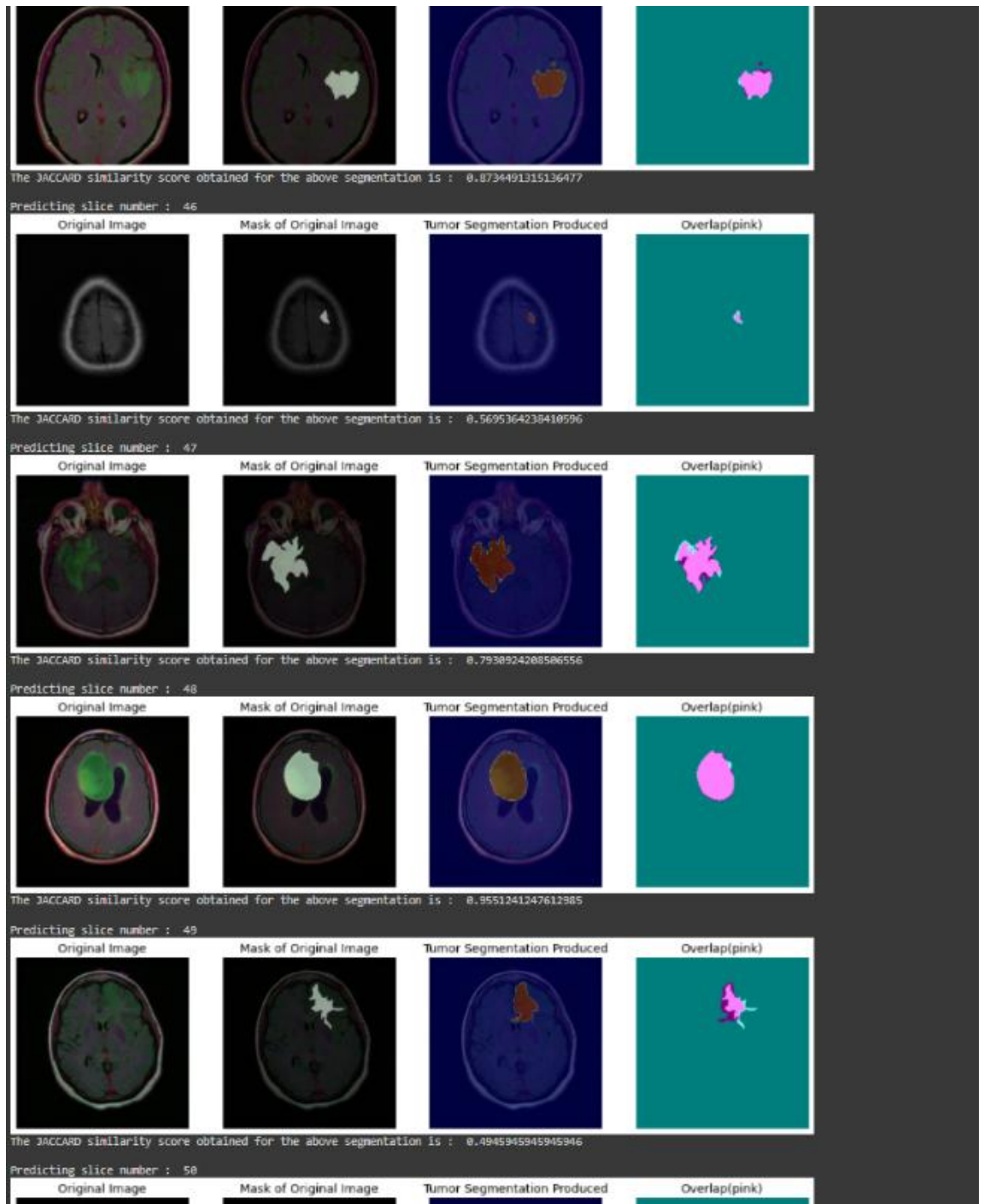


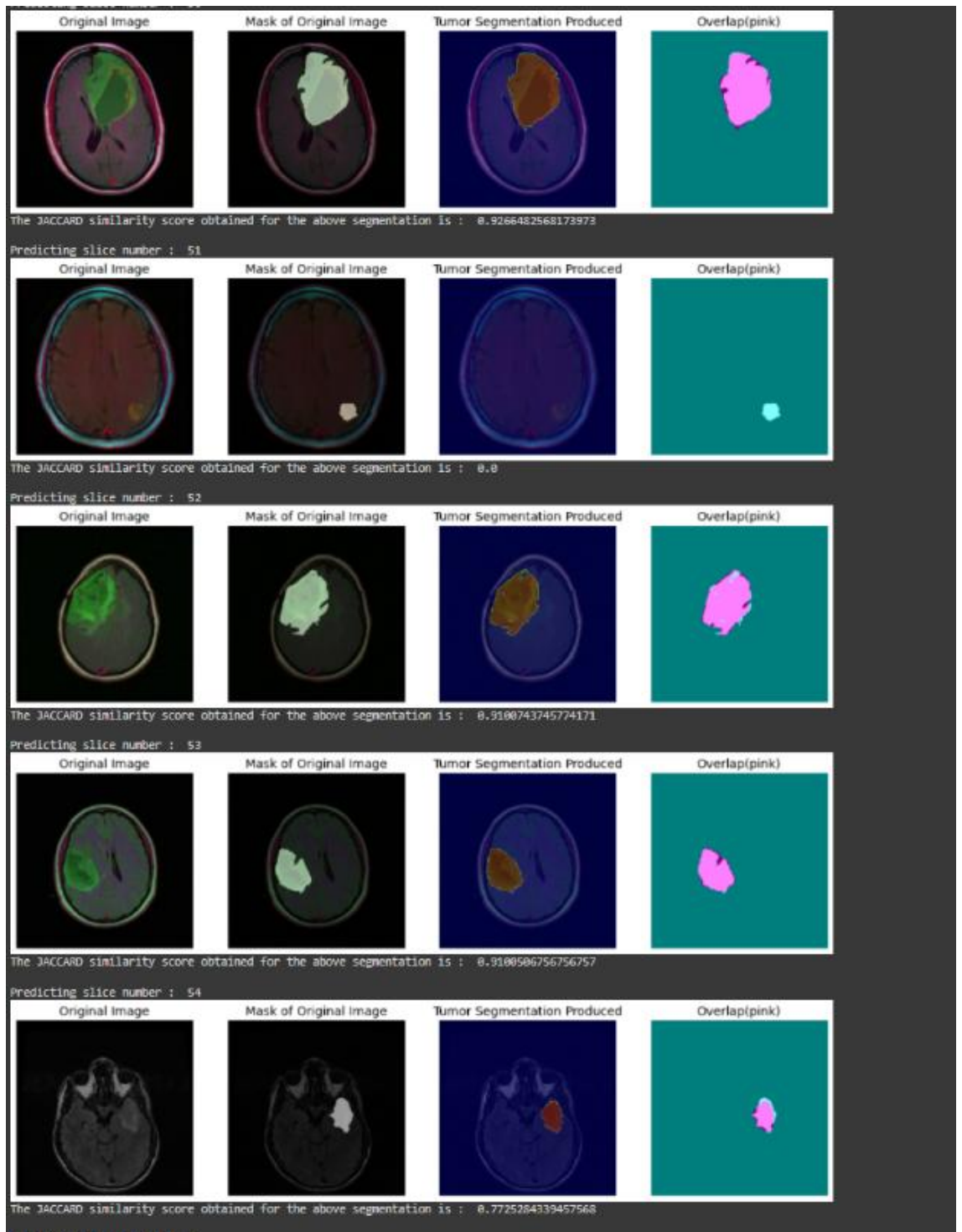


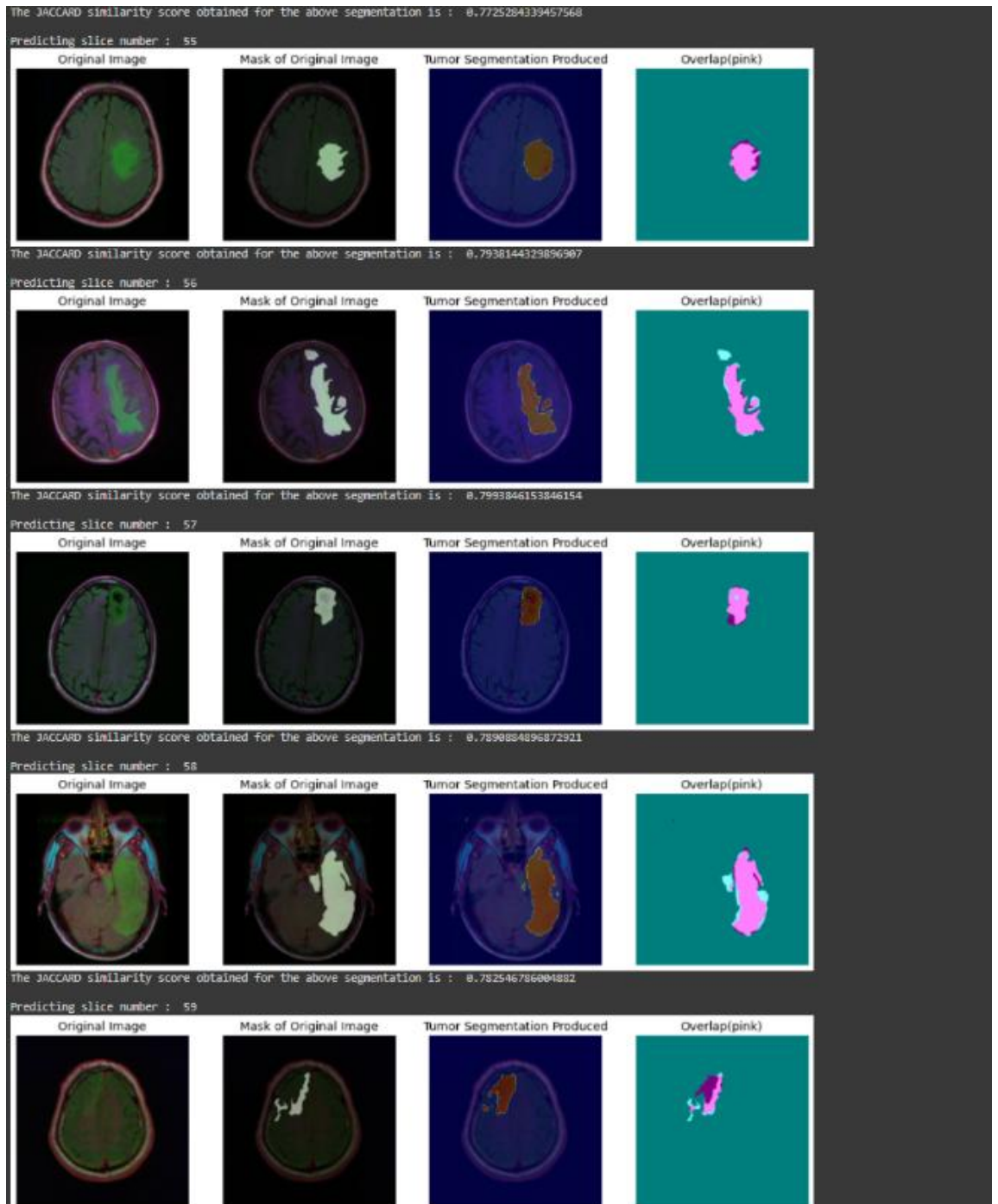


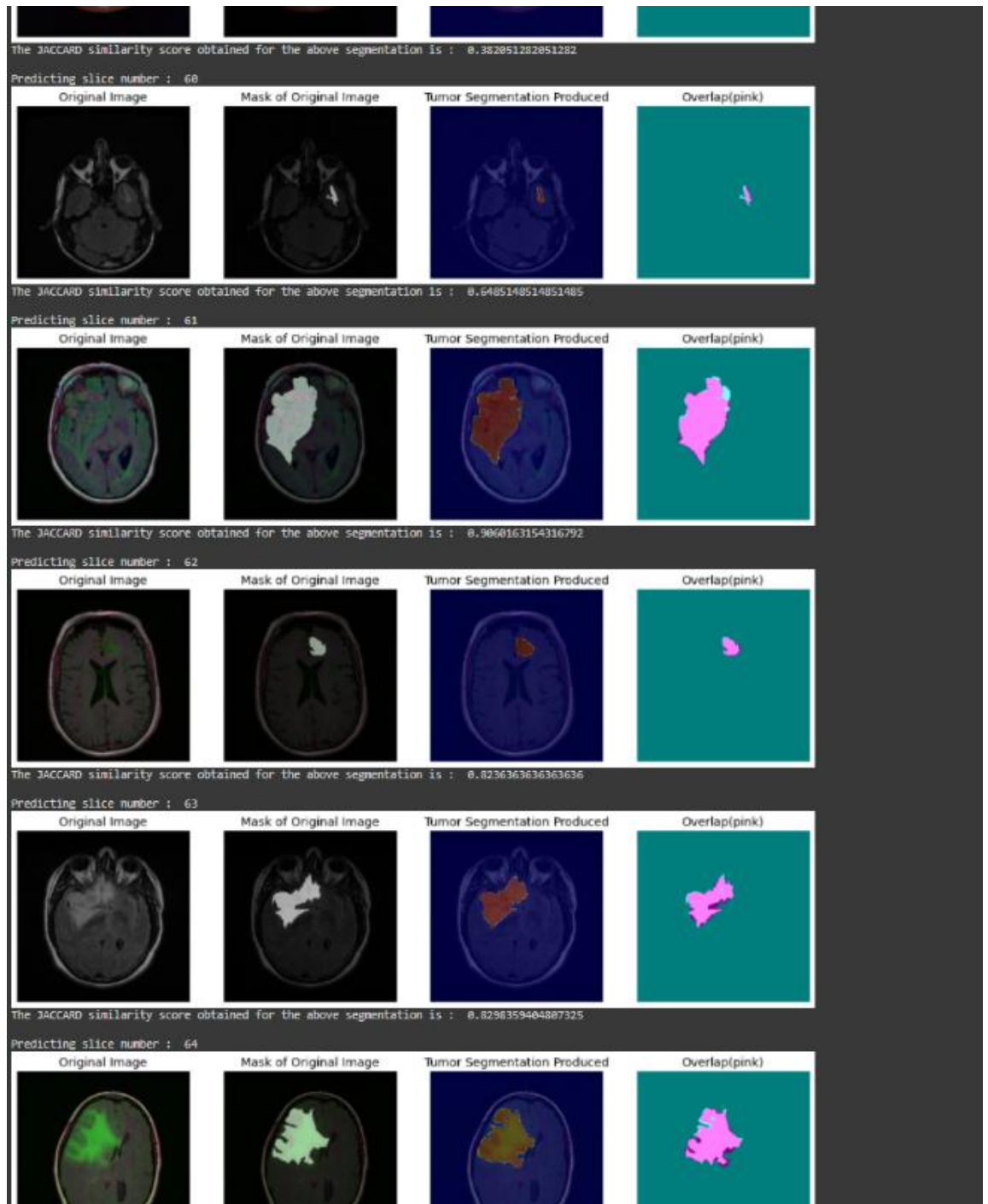


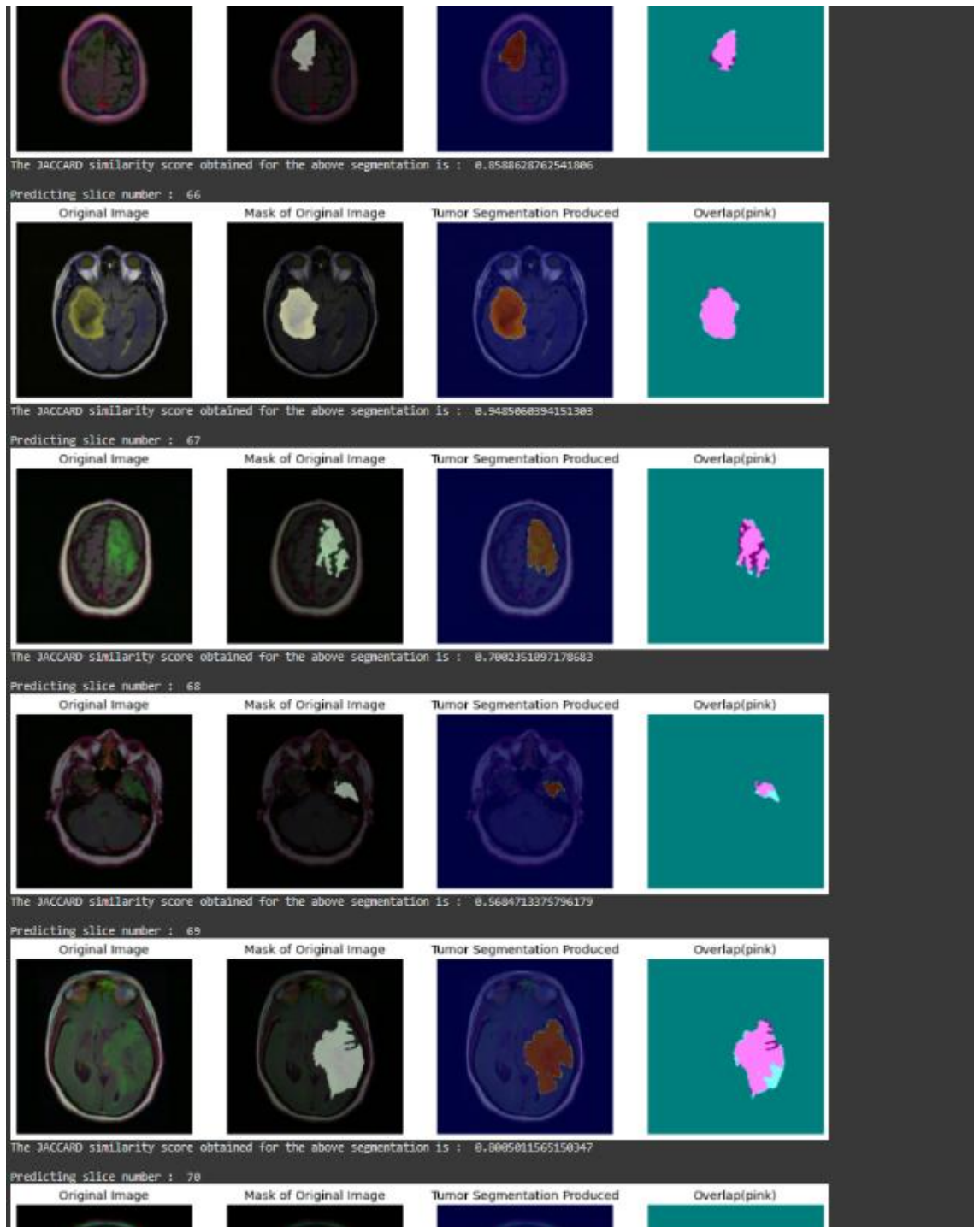


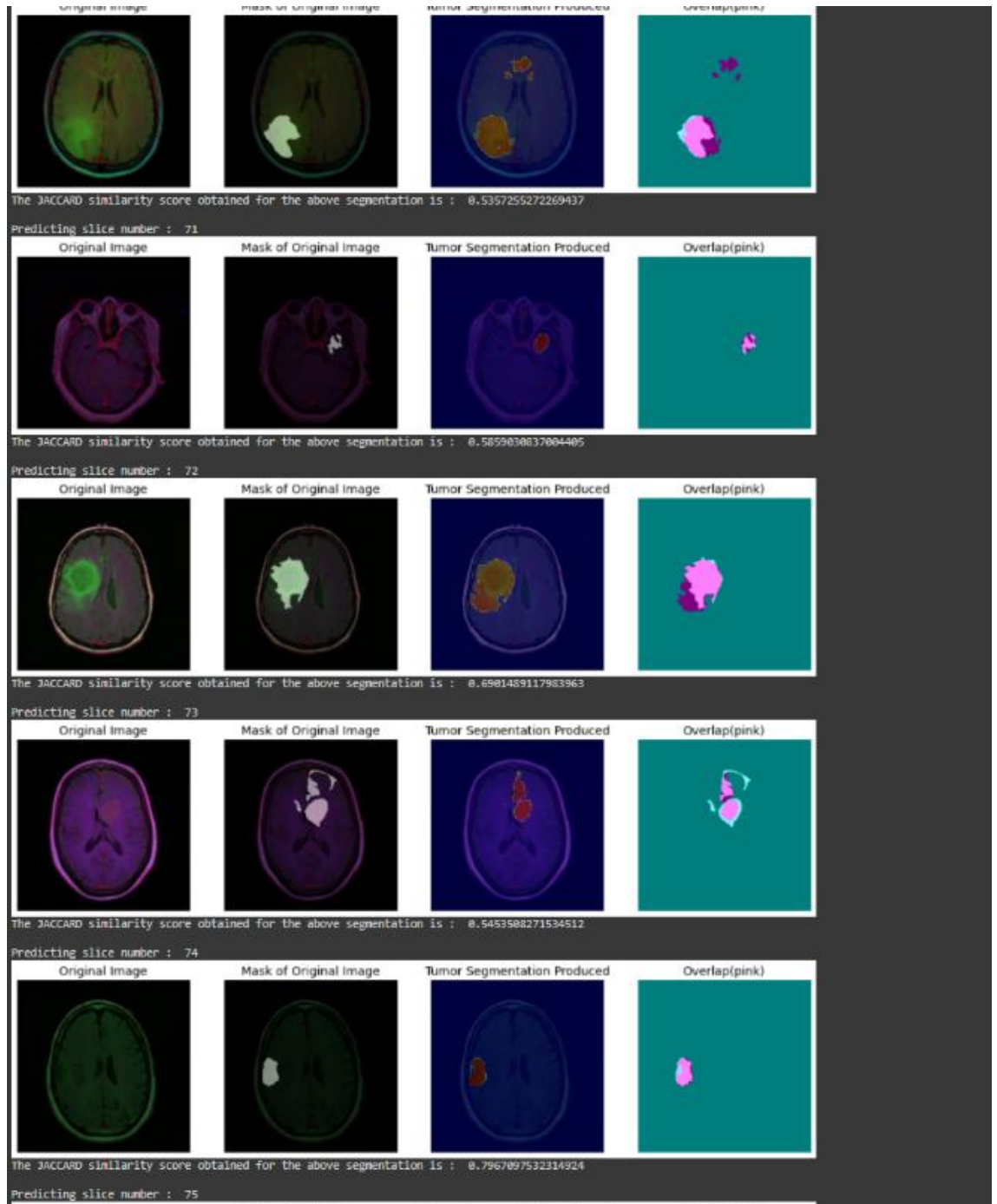


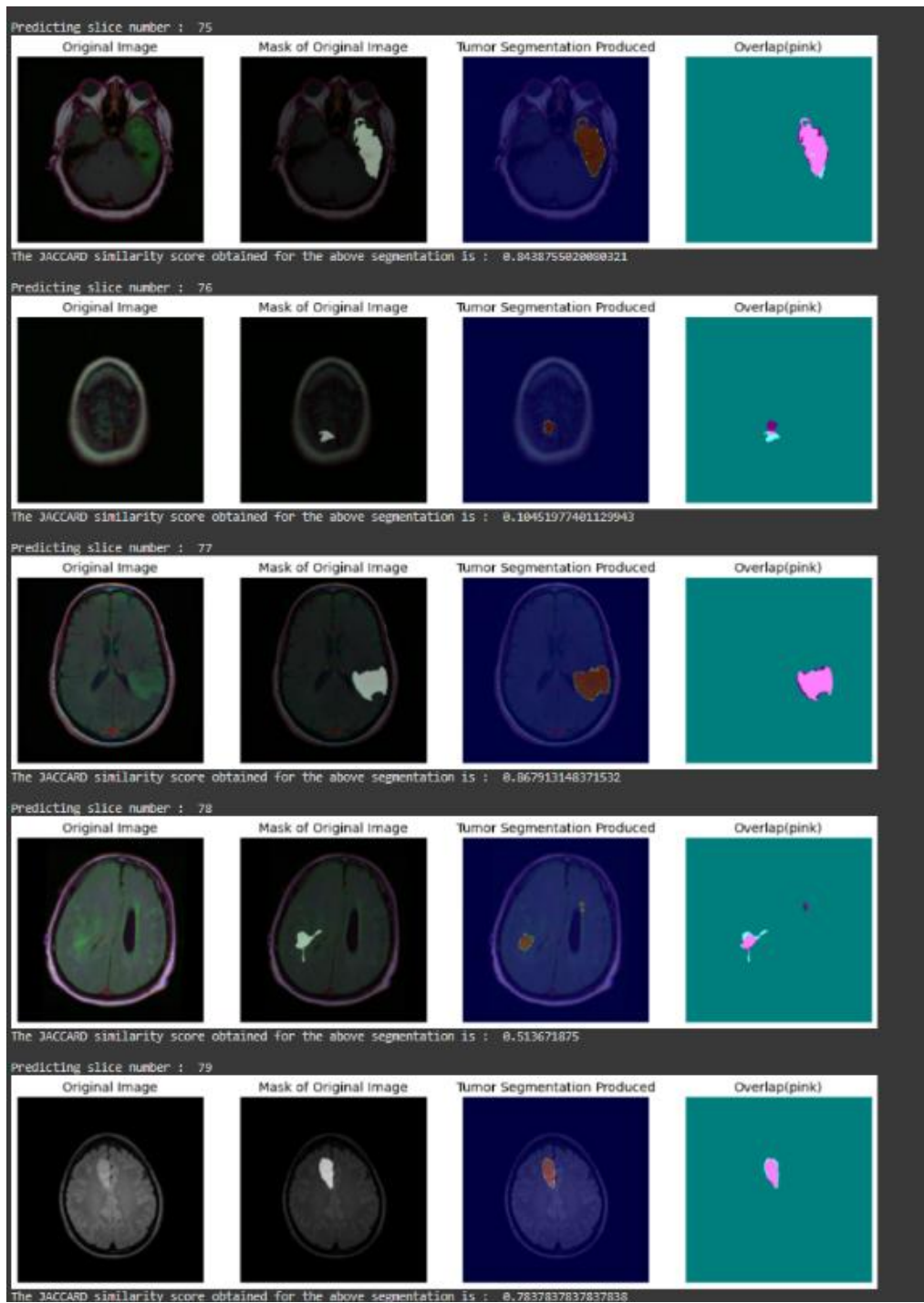


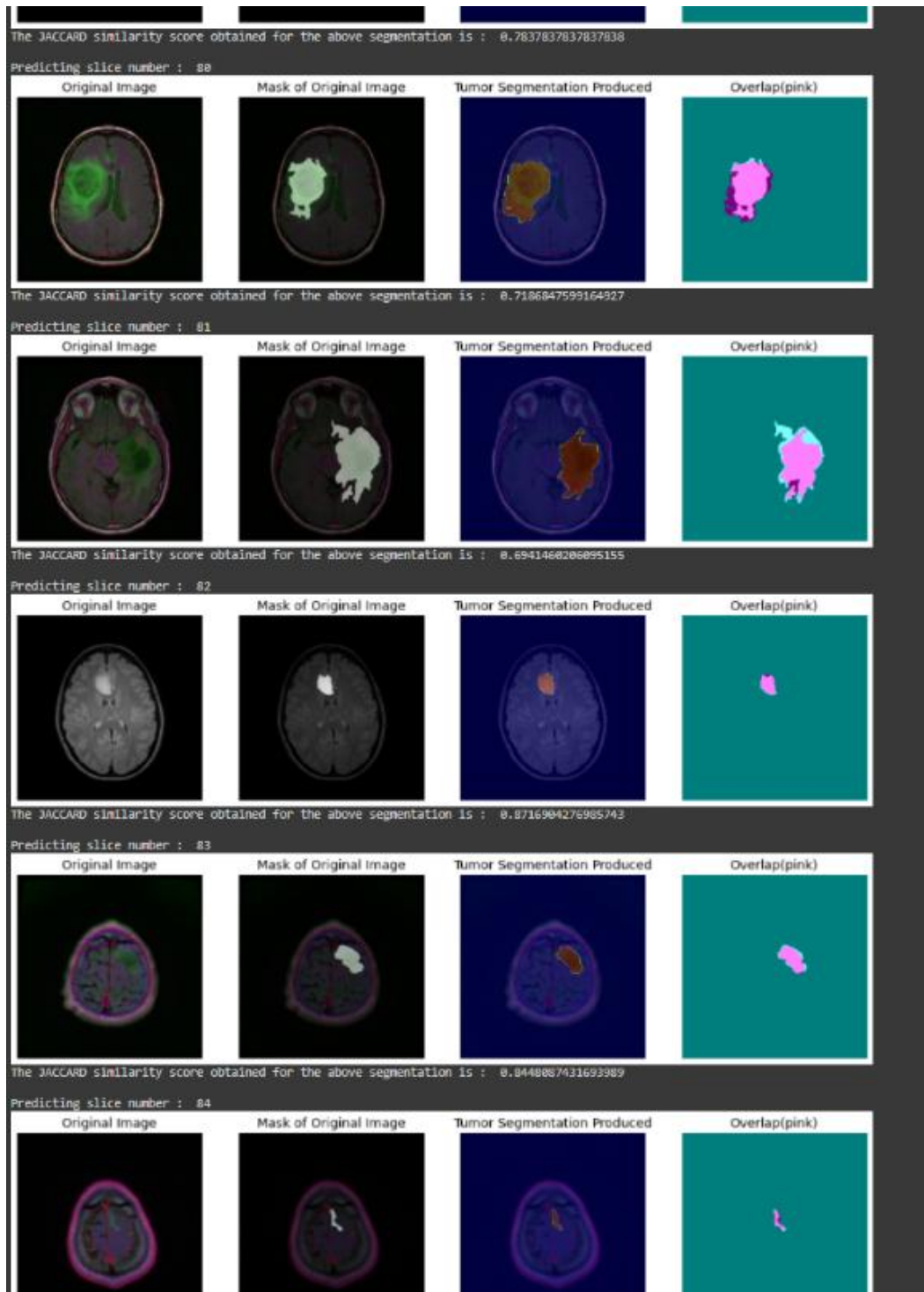


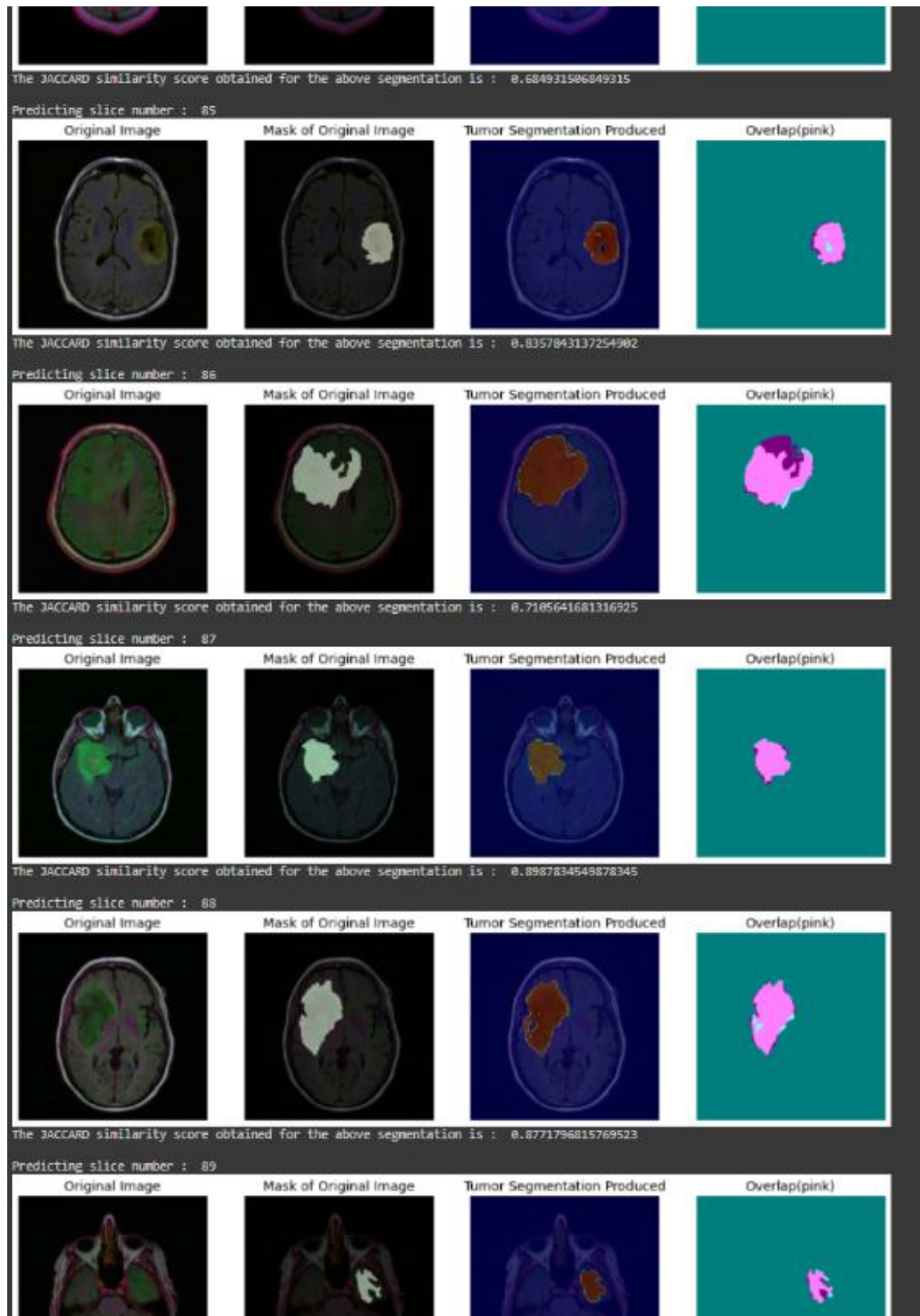


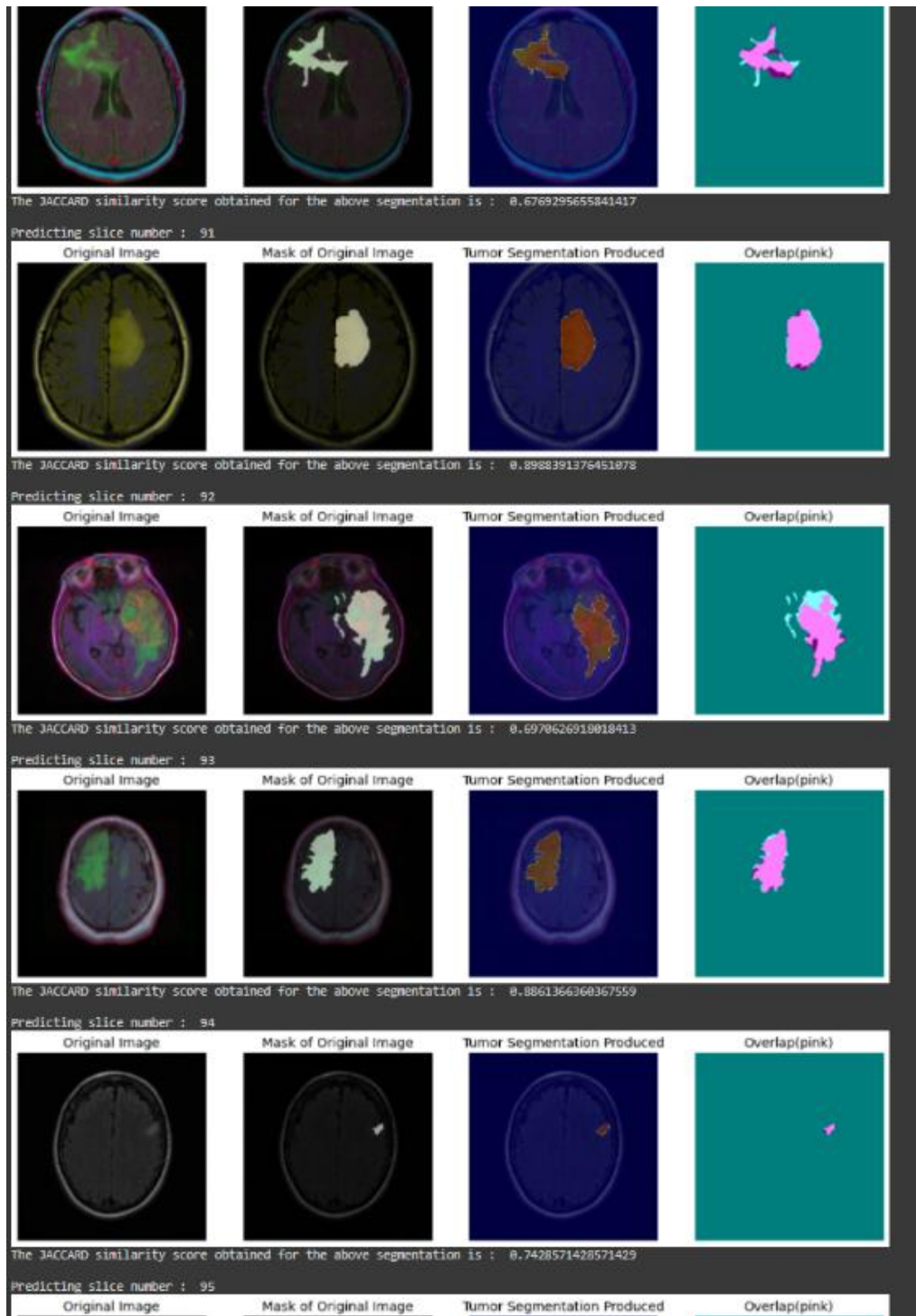


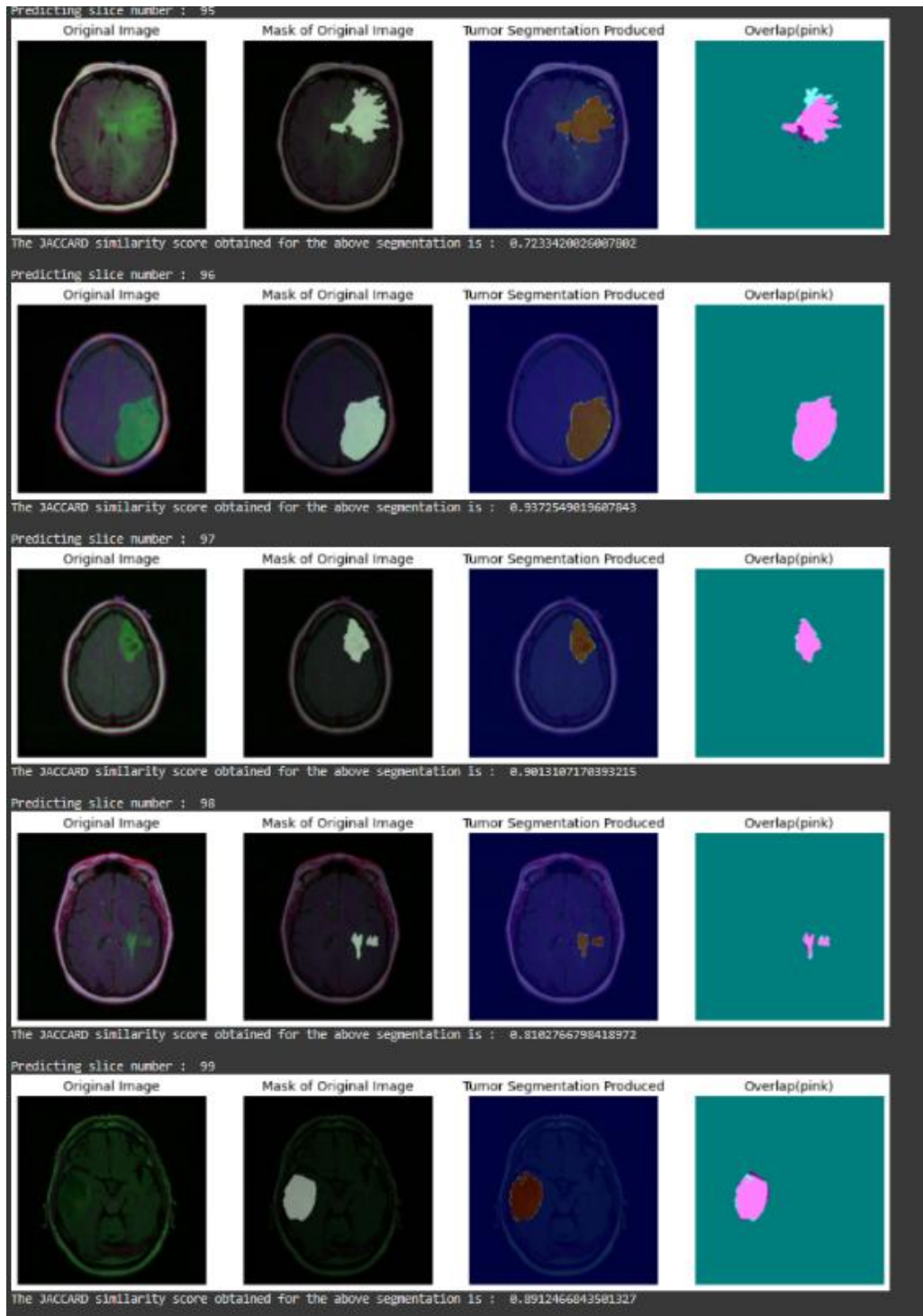


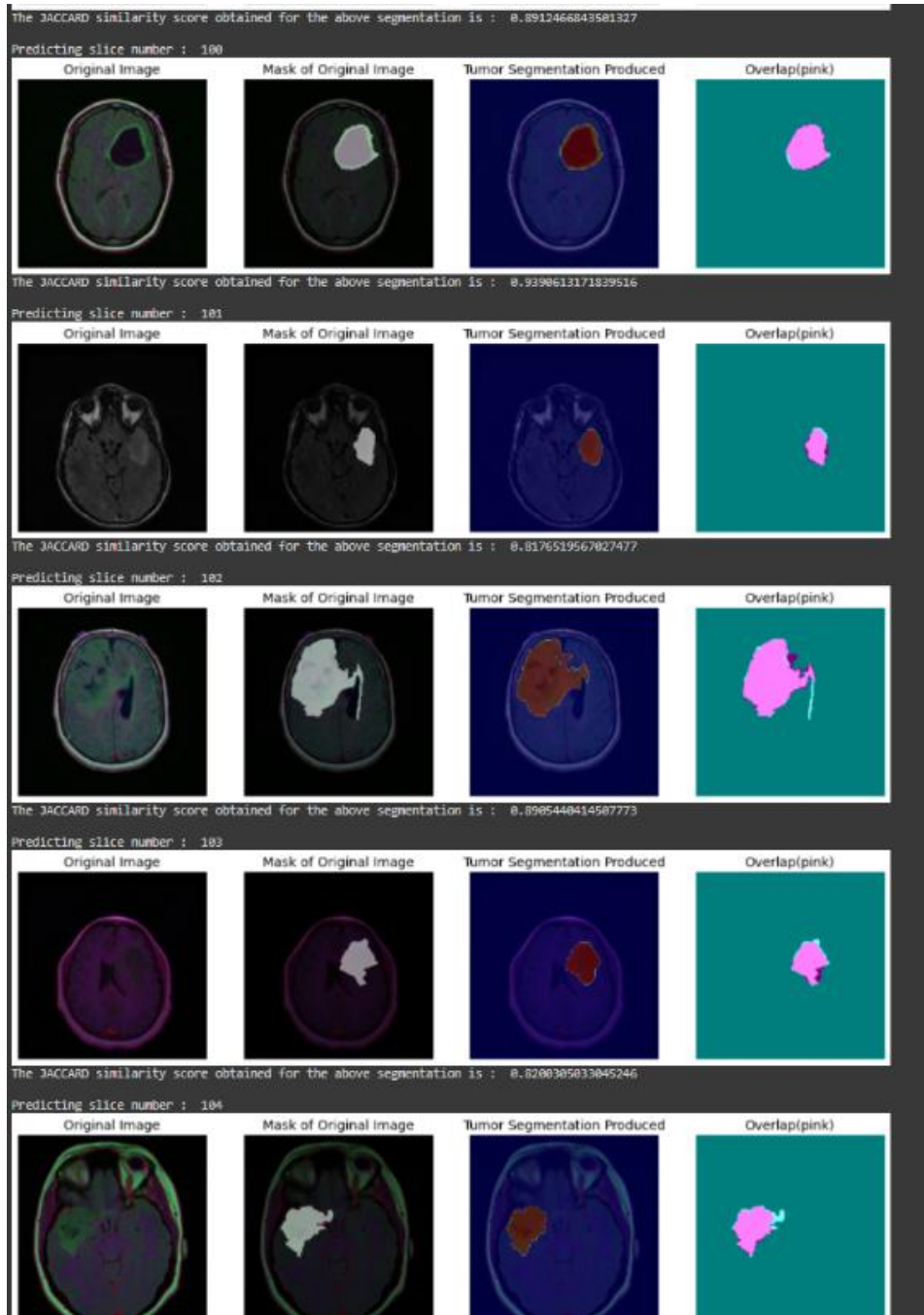


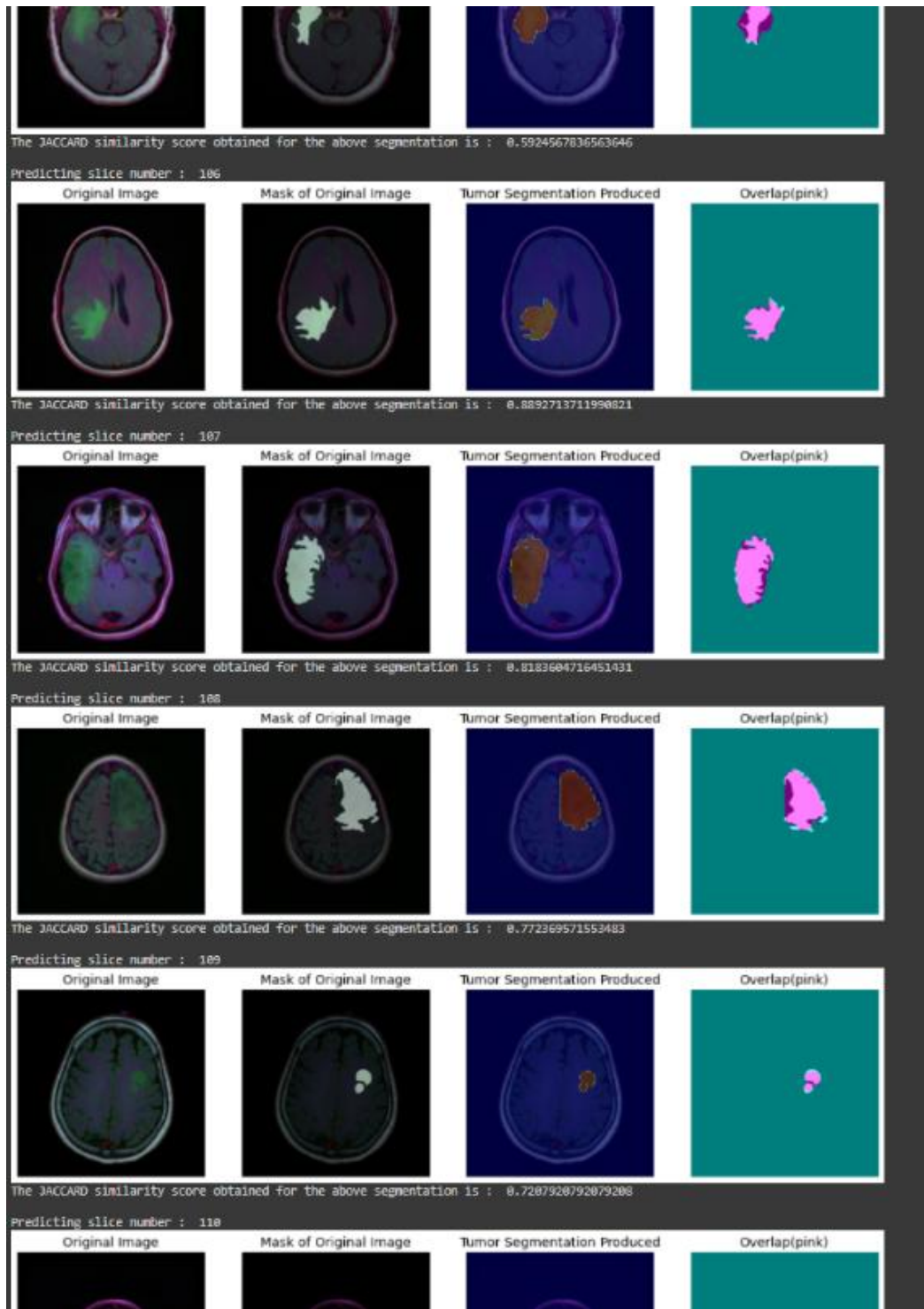


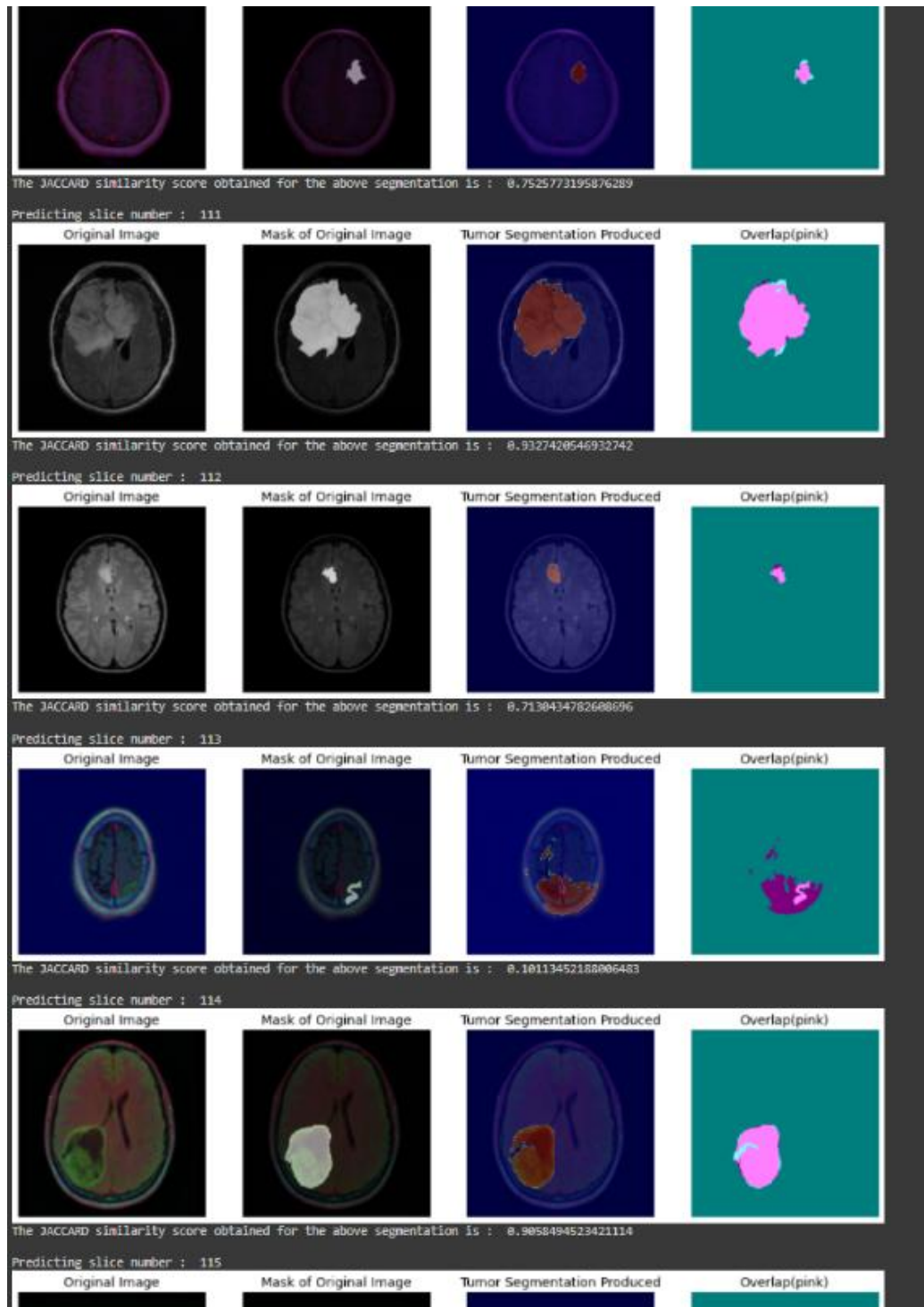


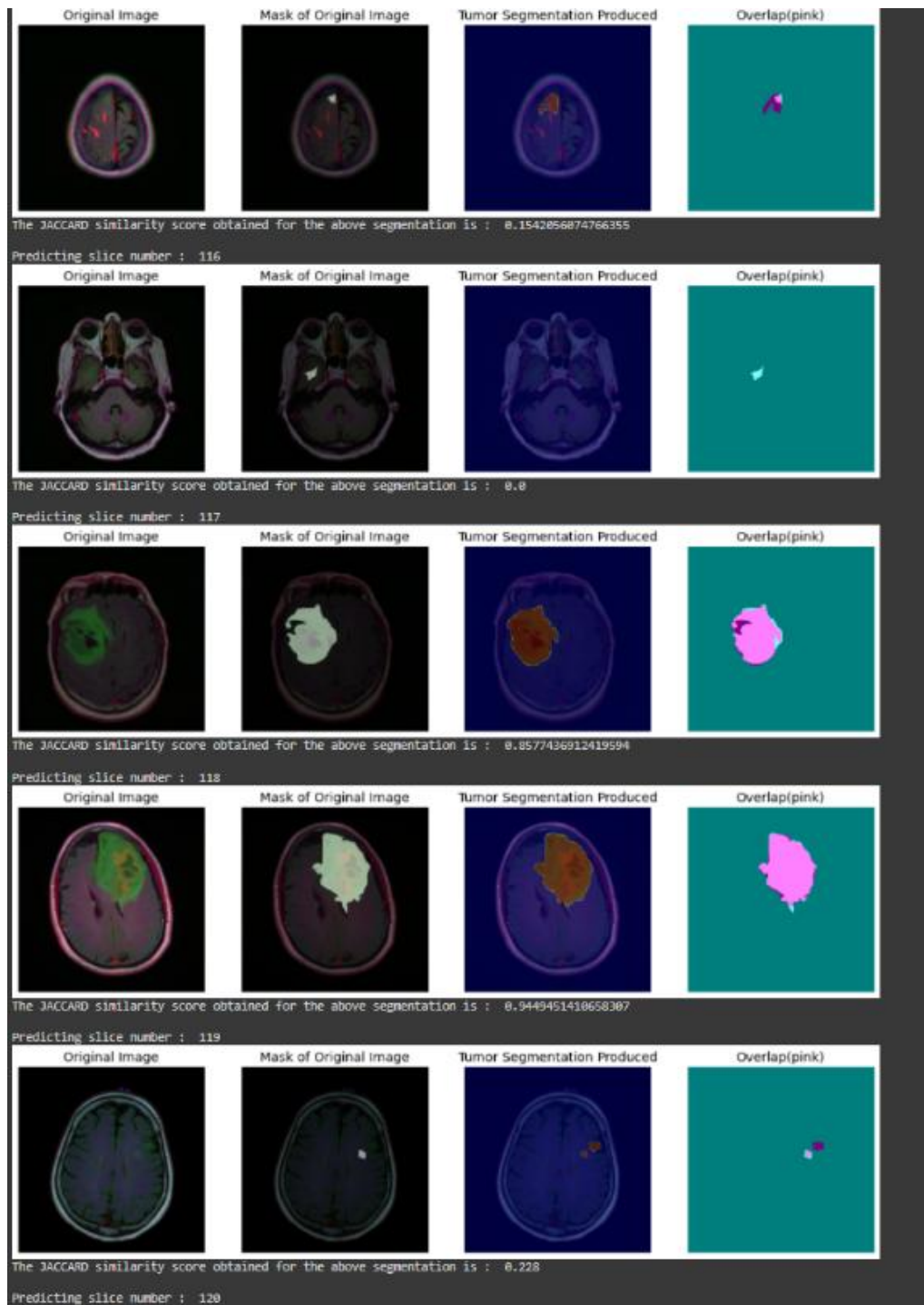


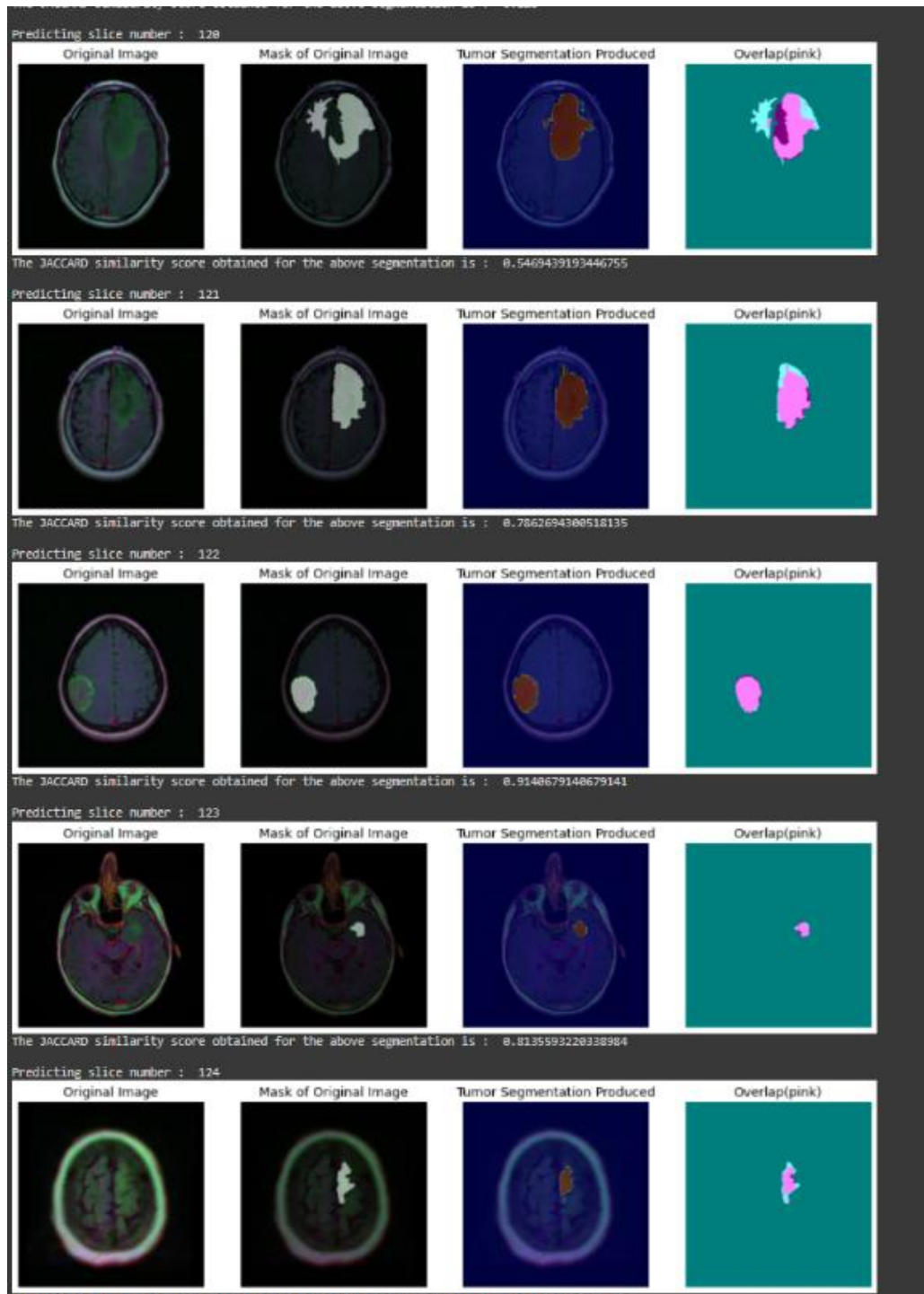


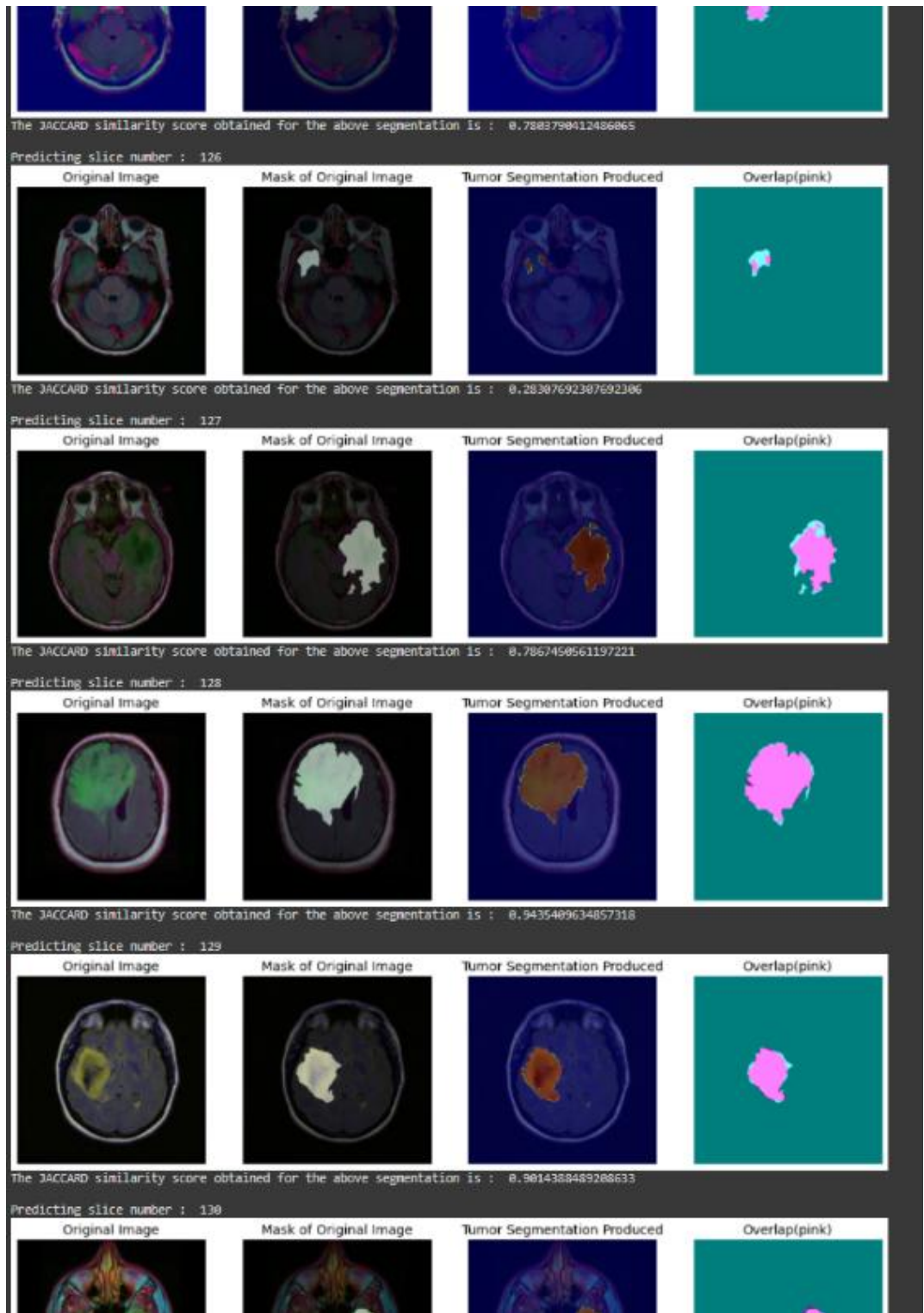


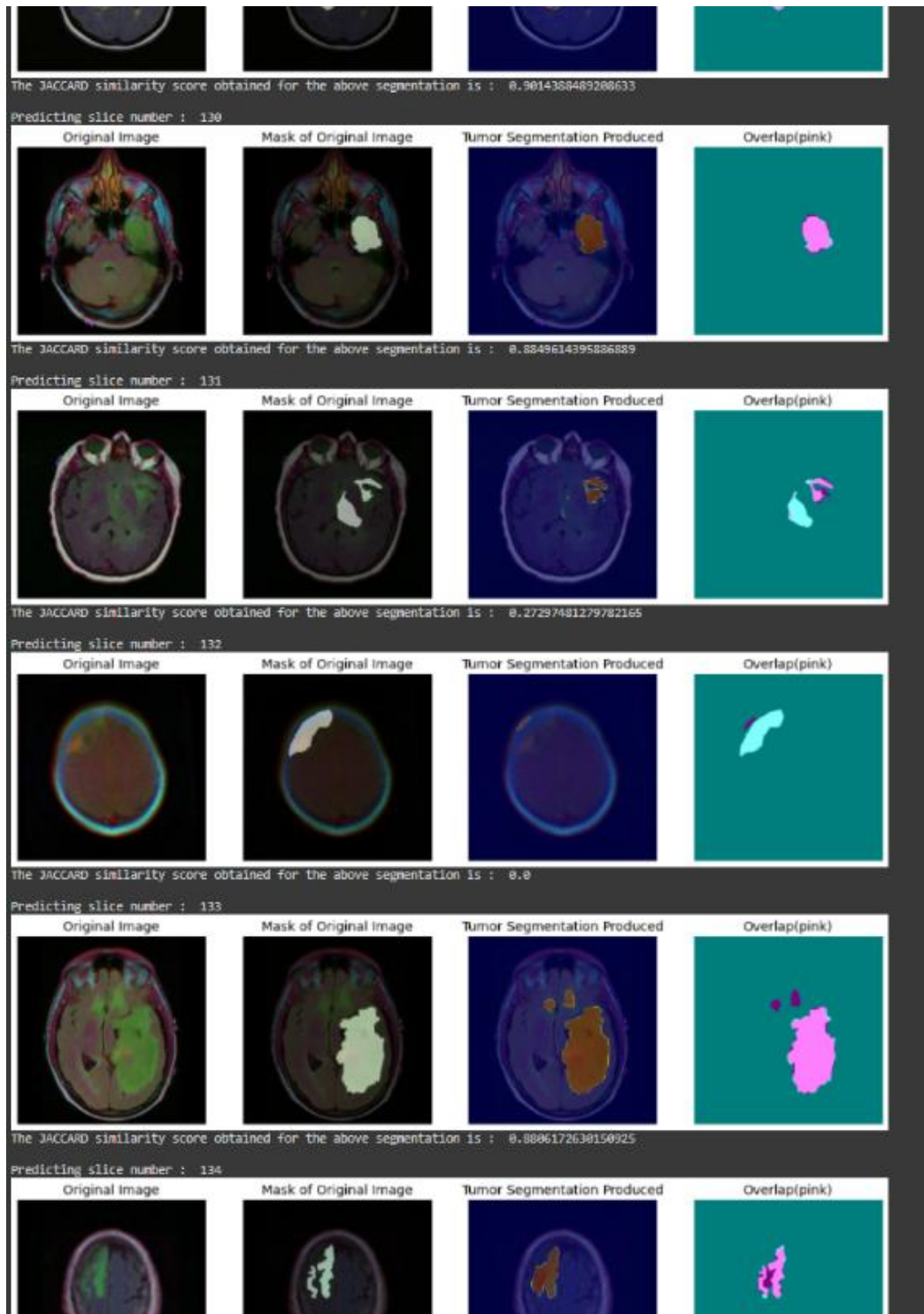


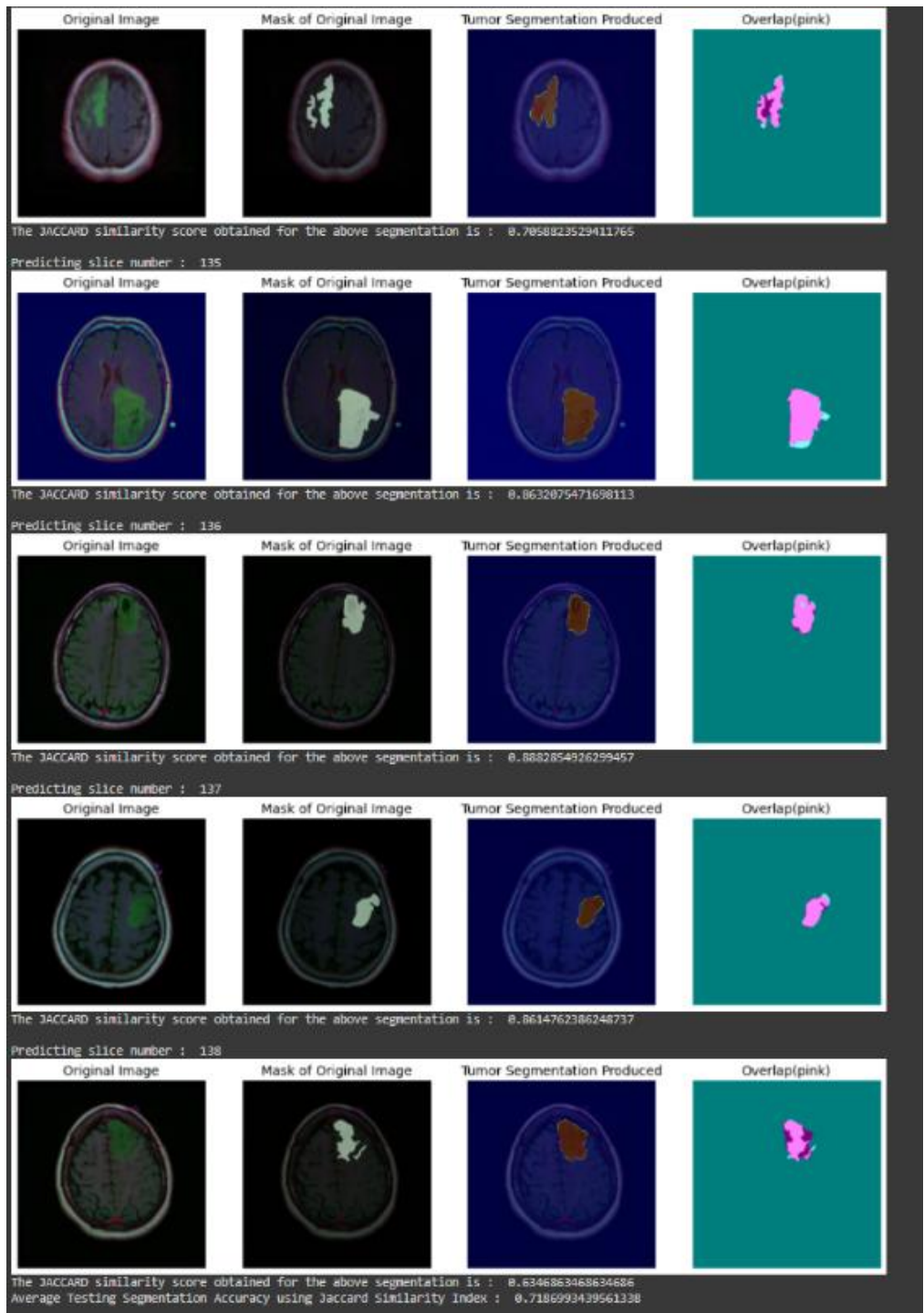




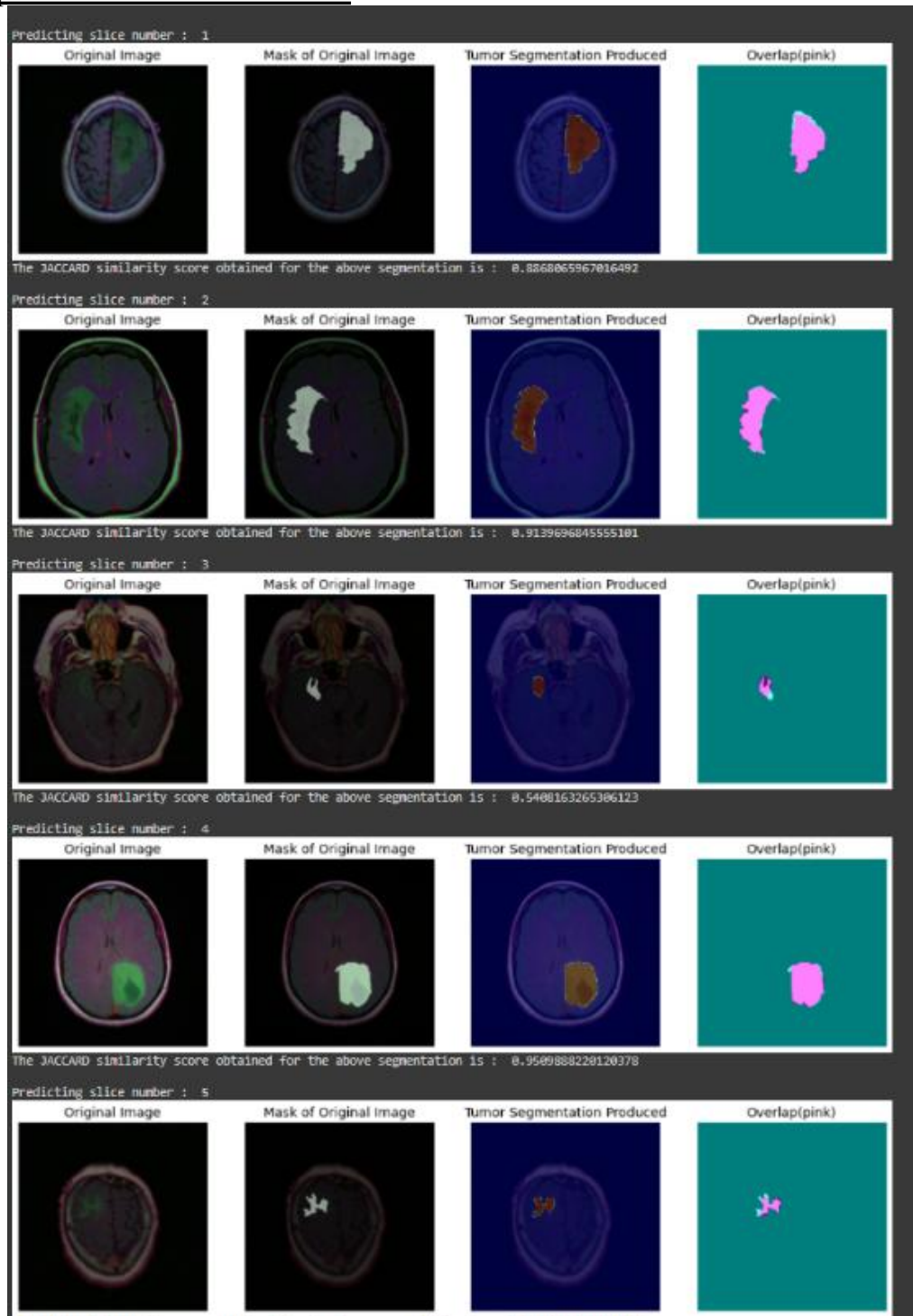


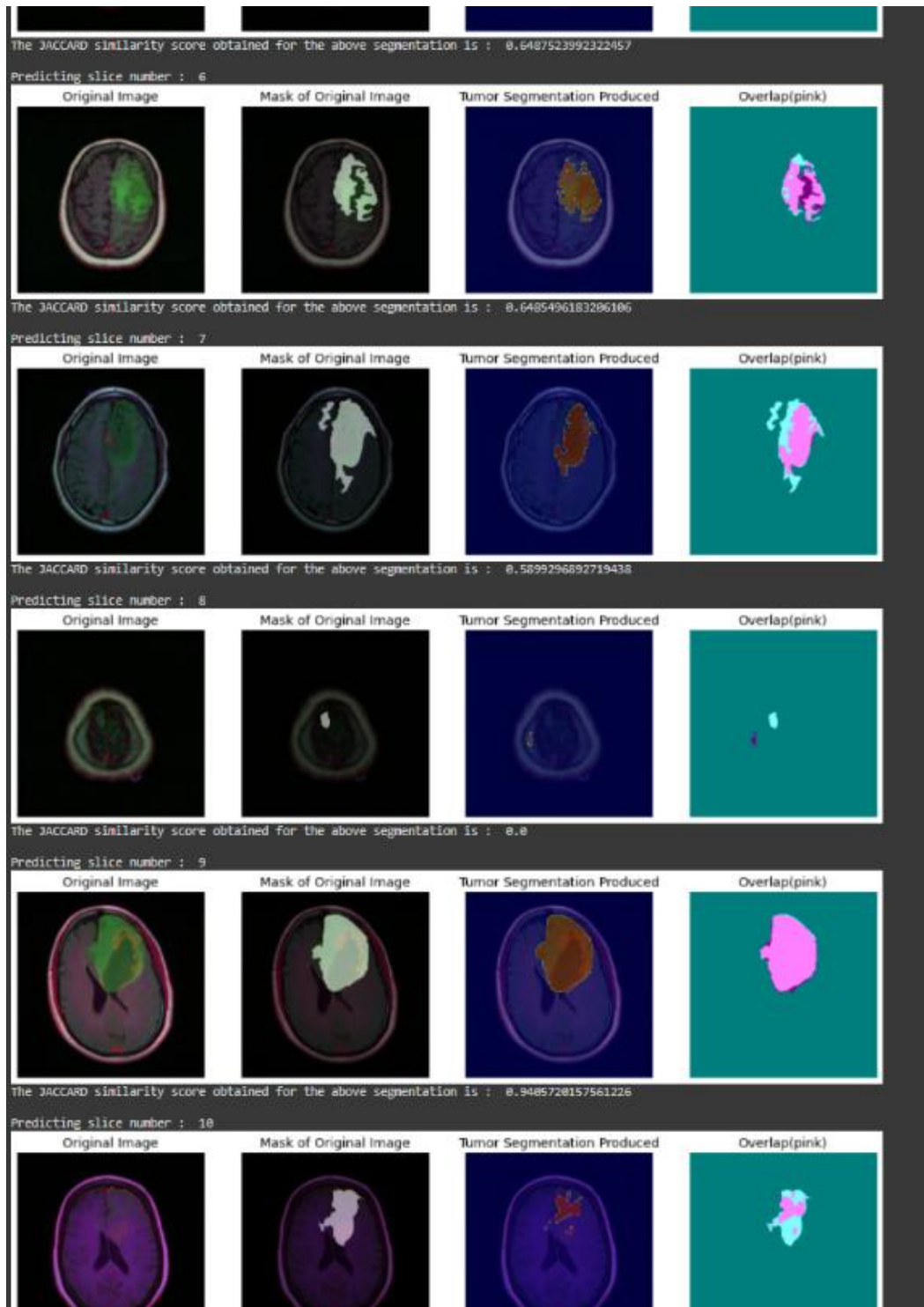


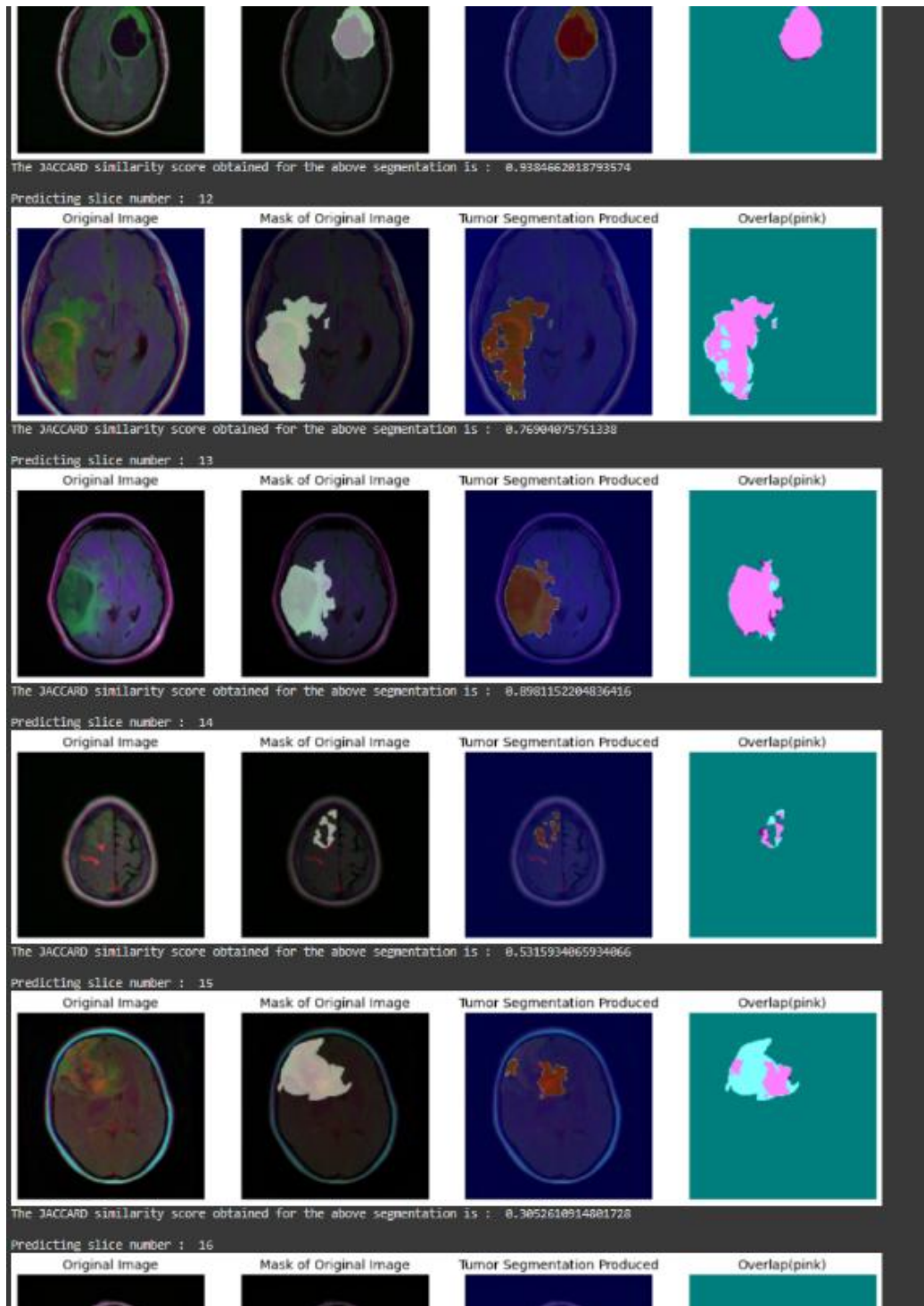


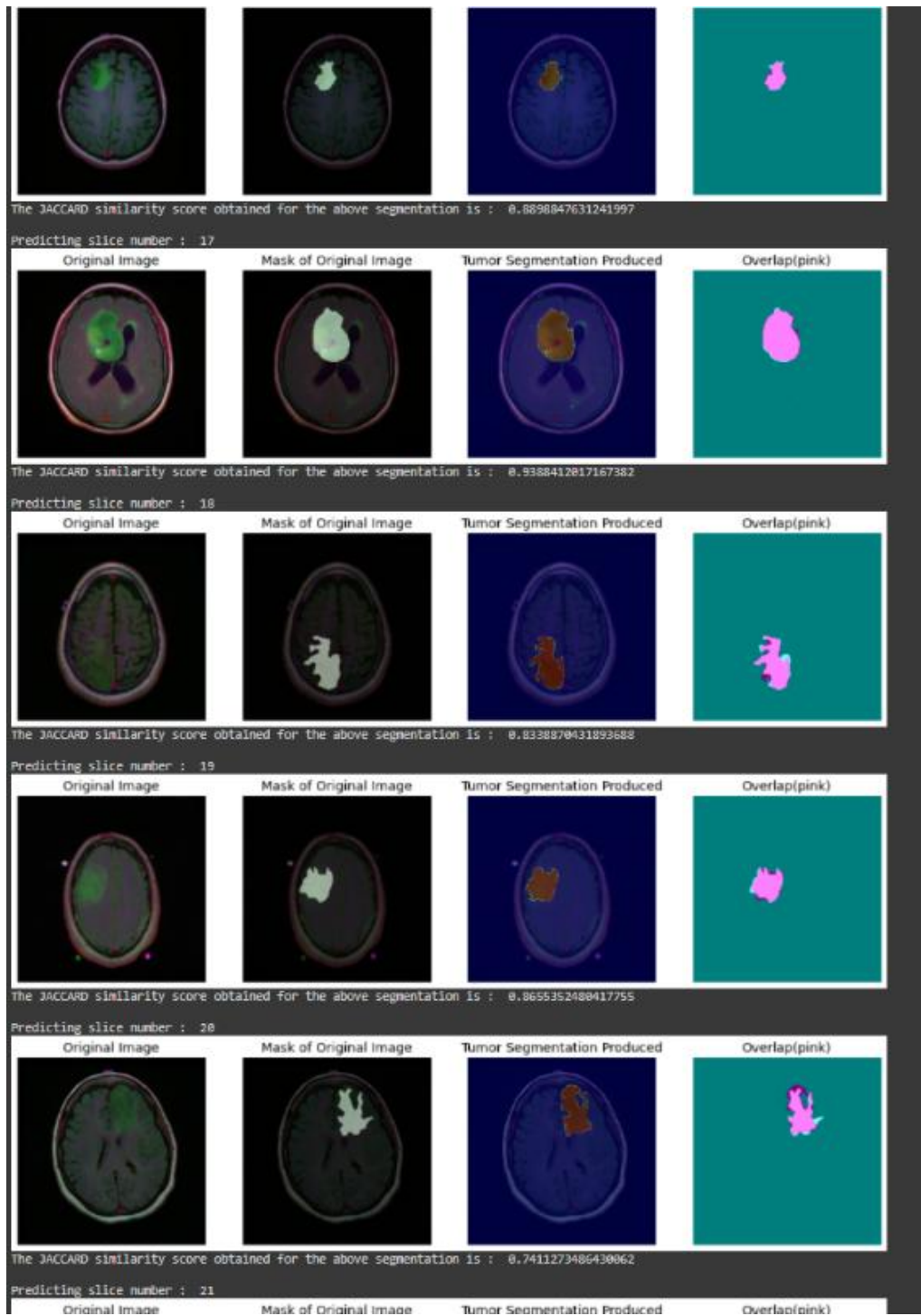


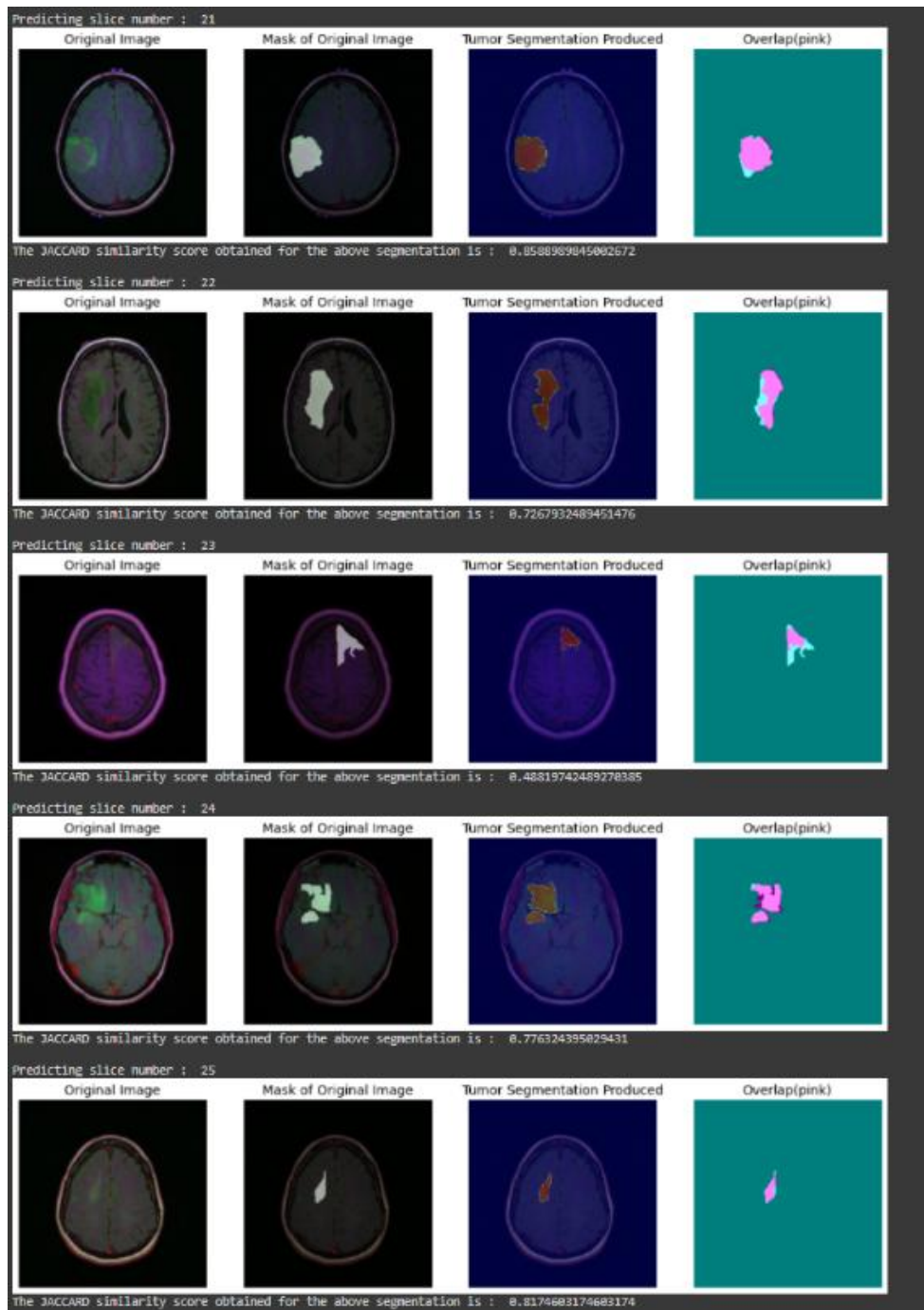
7. Experimental Model No. 18

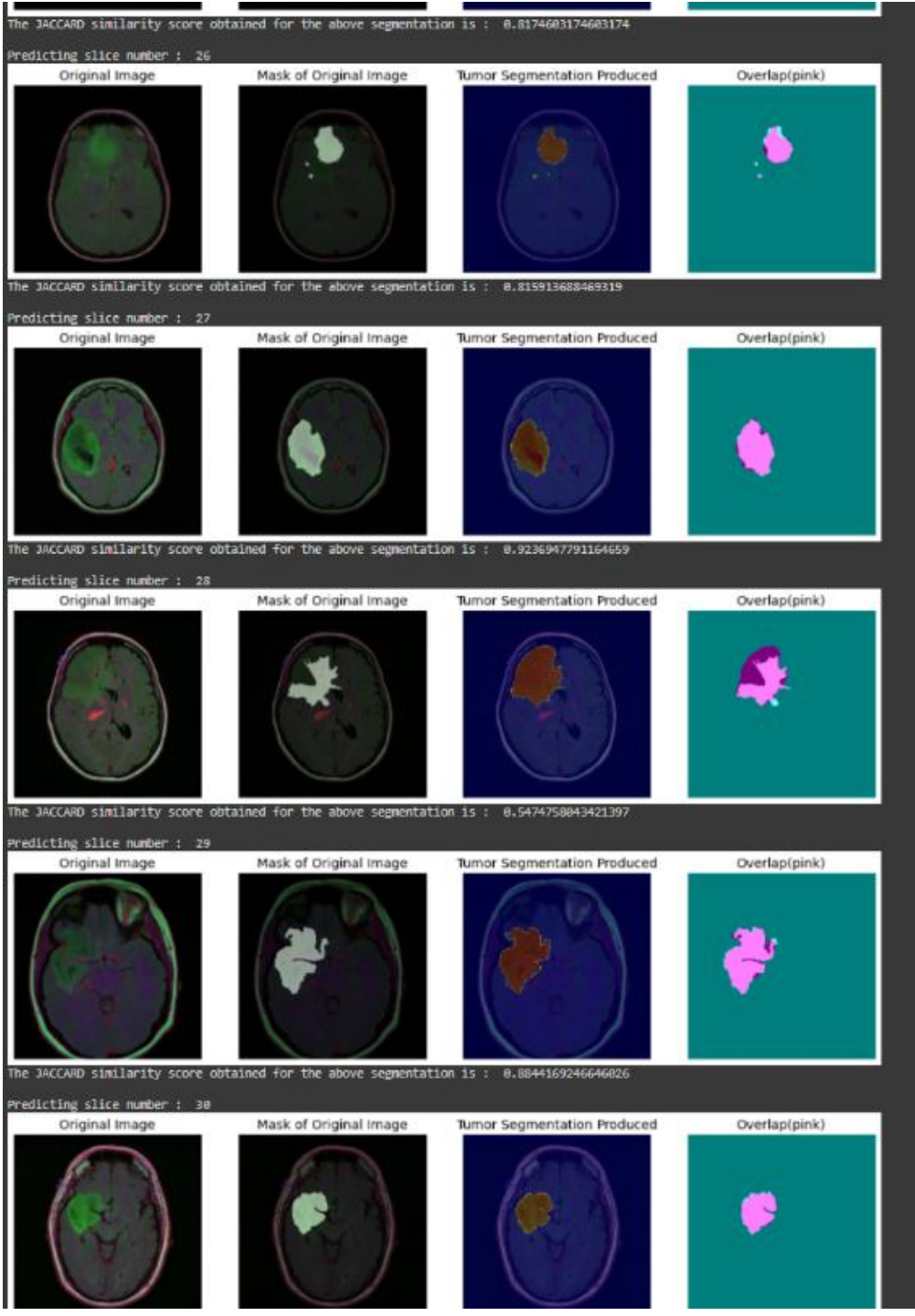


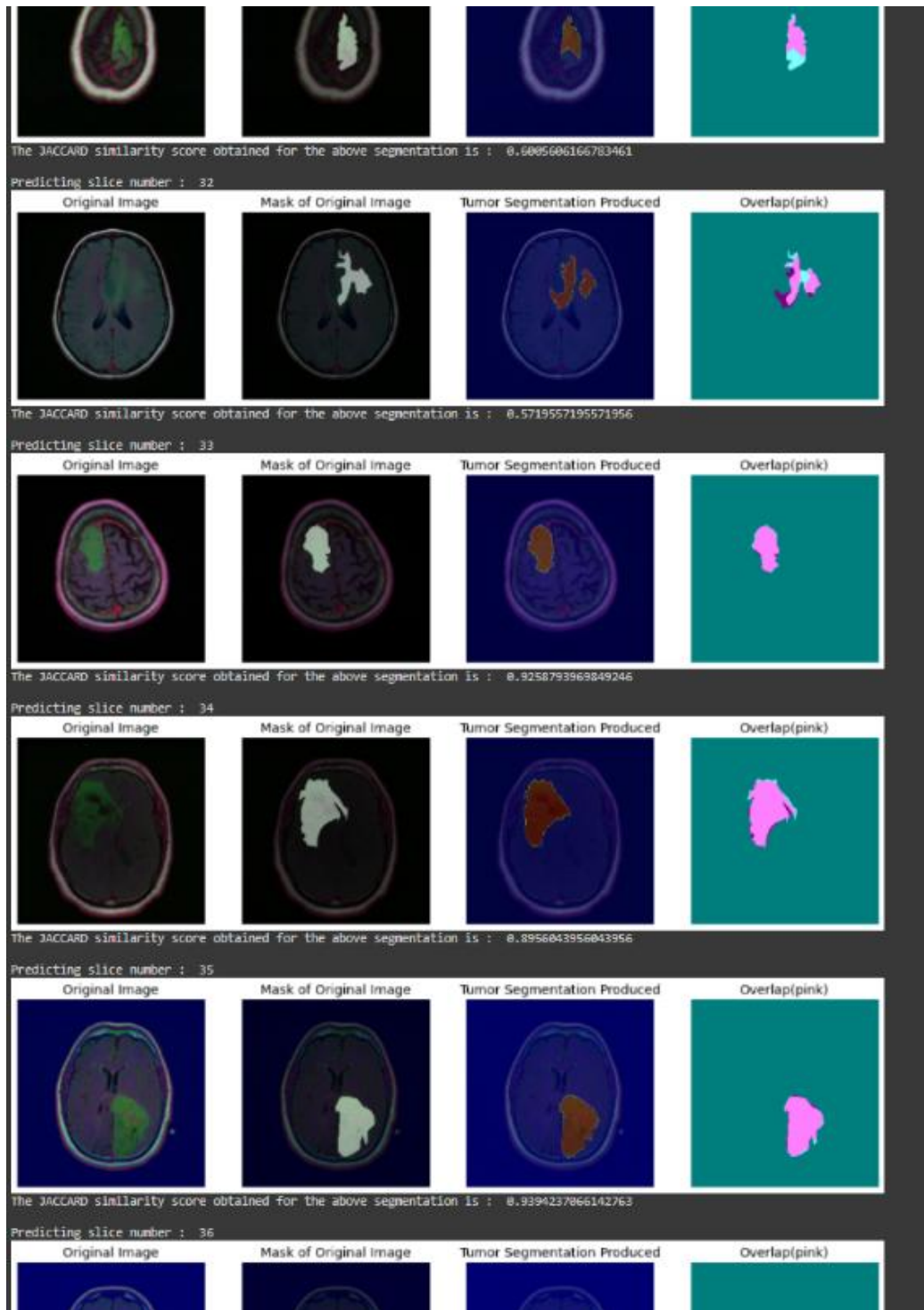


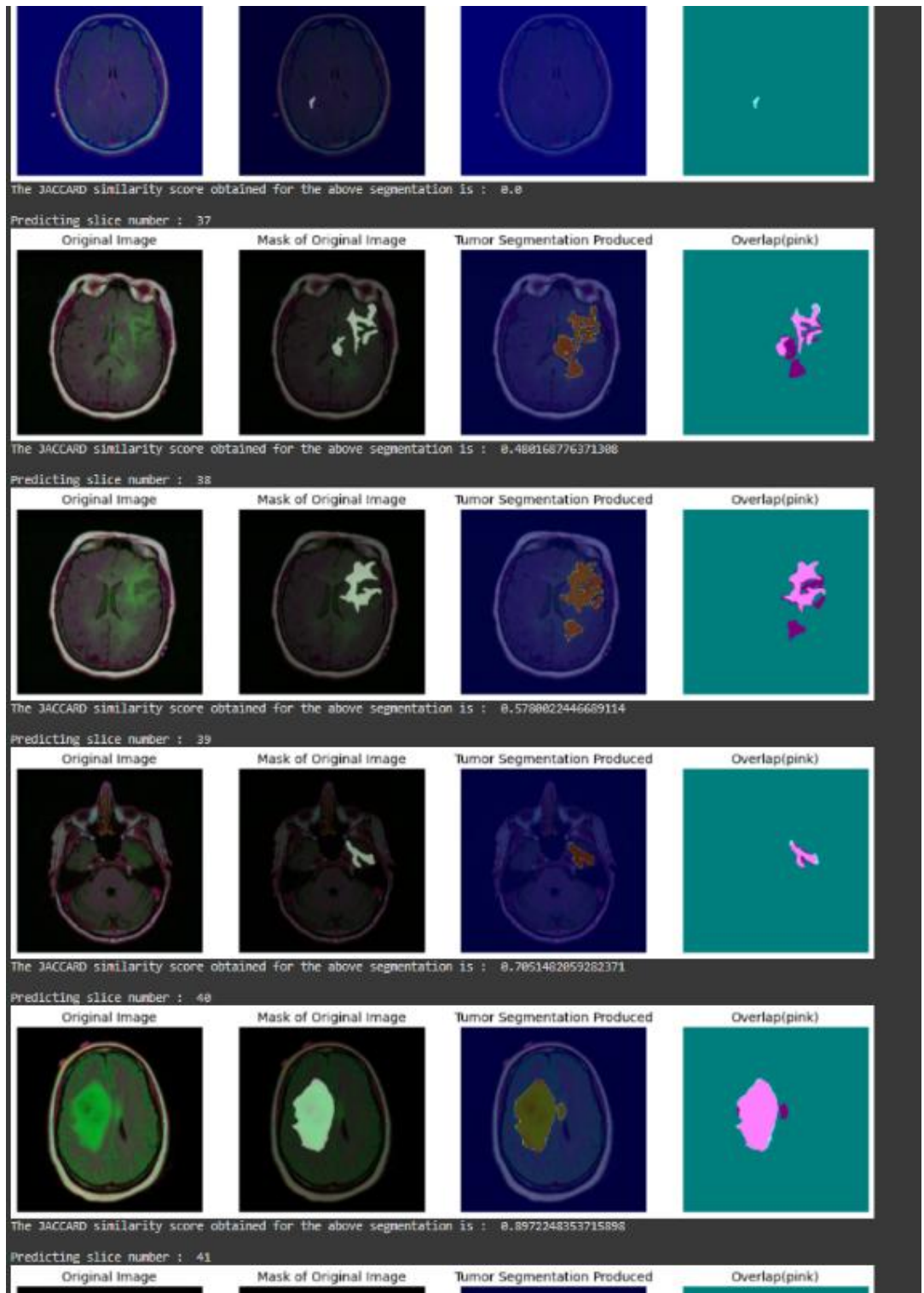


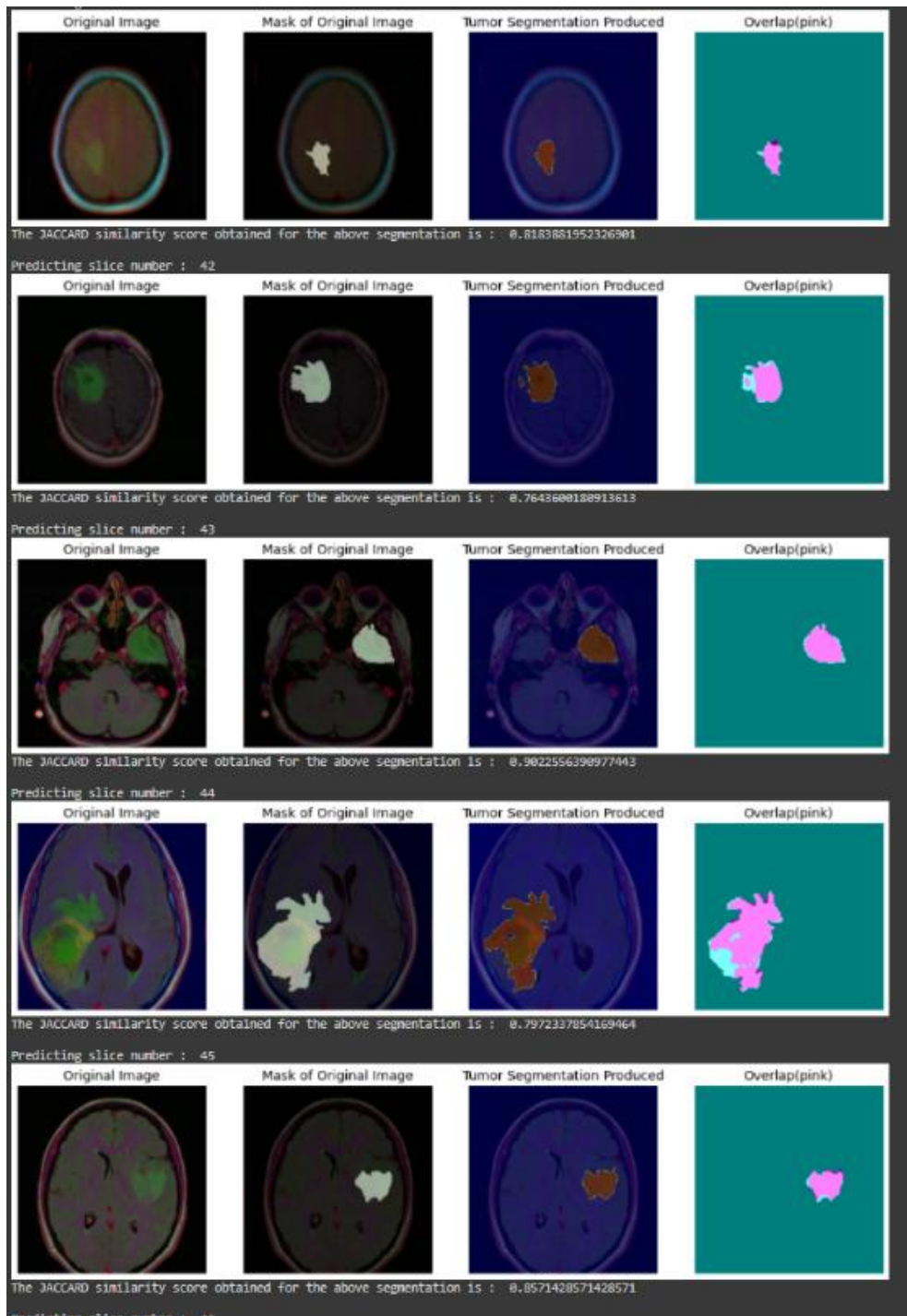


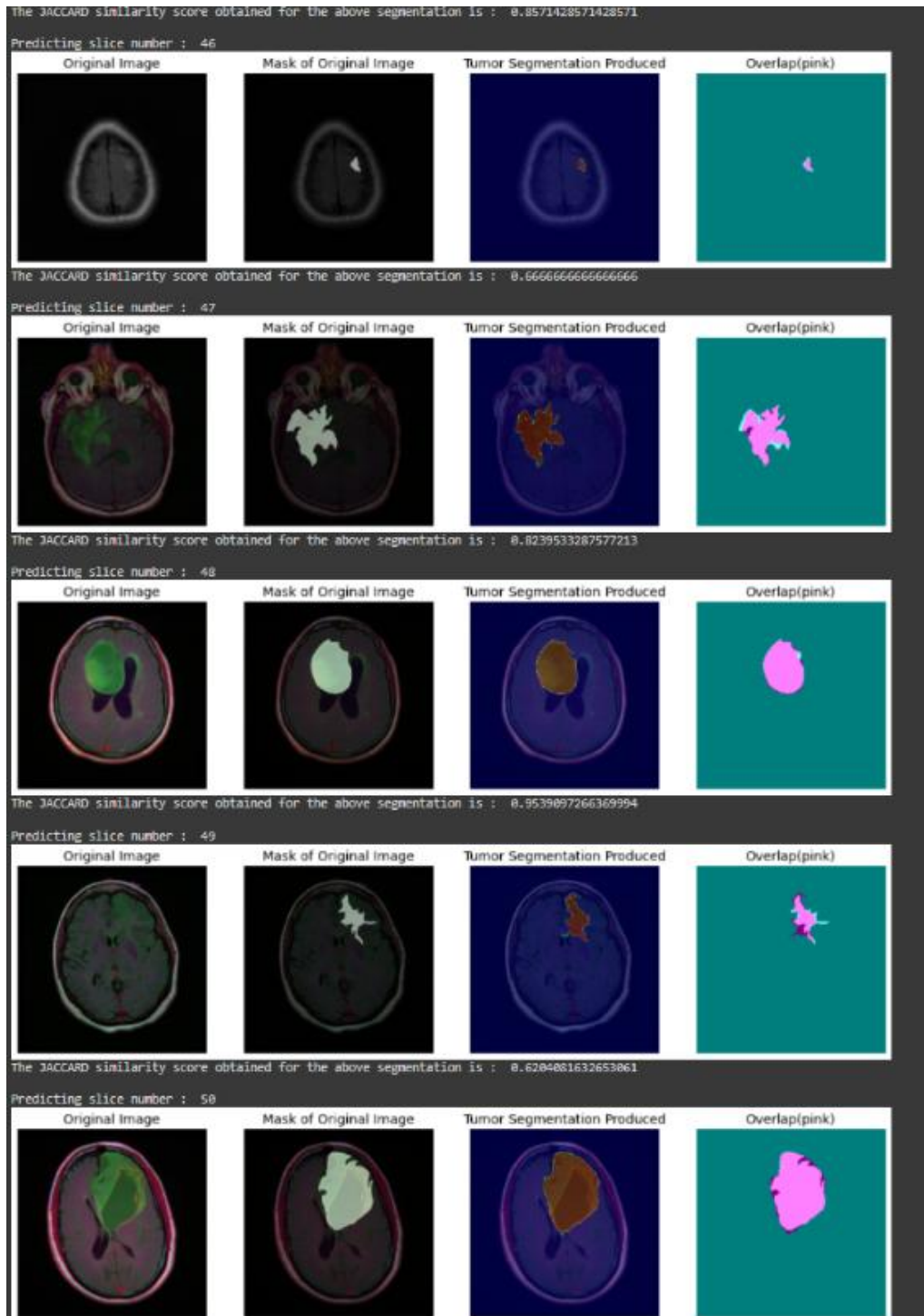


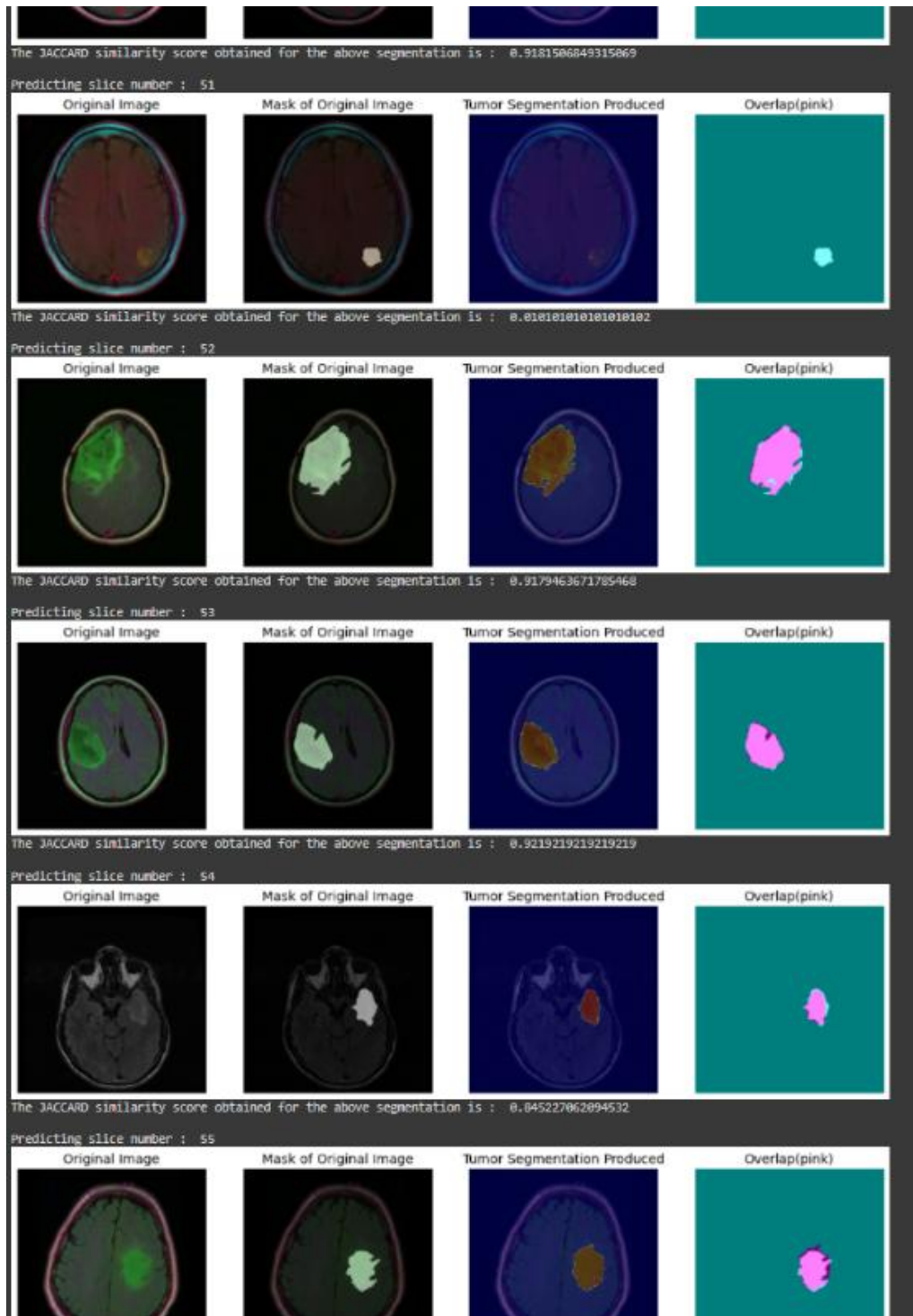


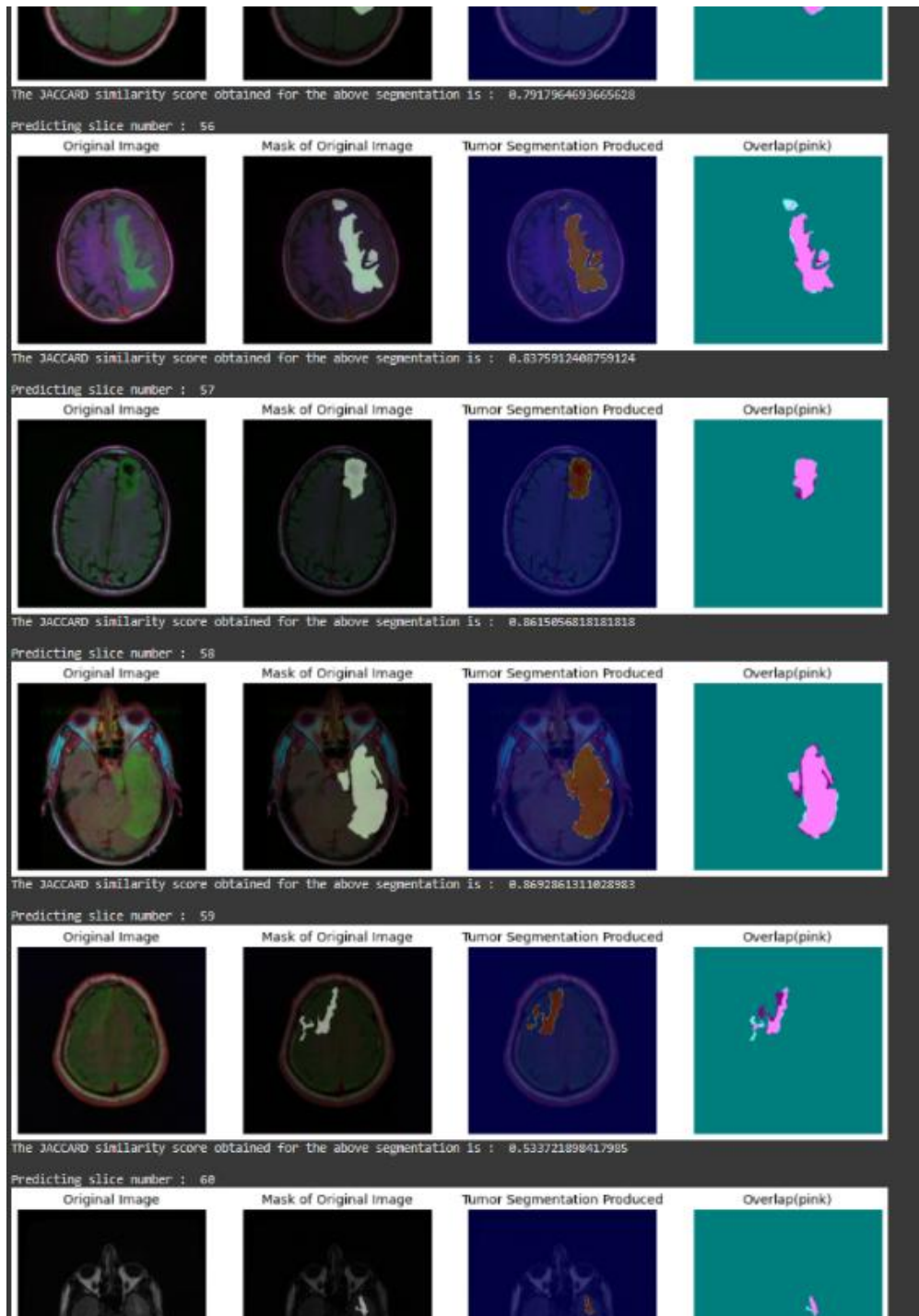


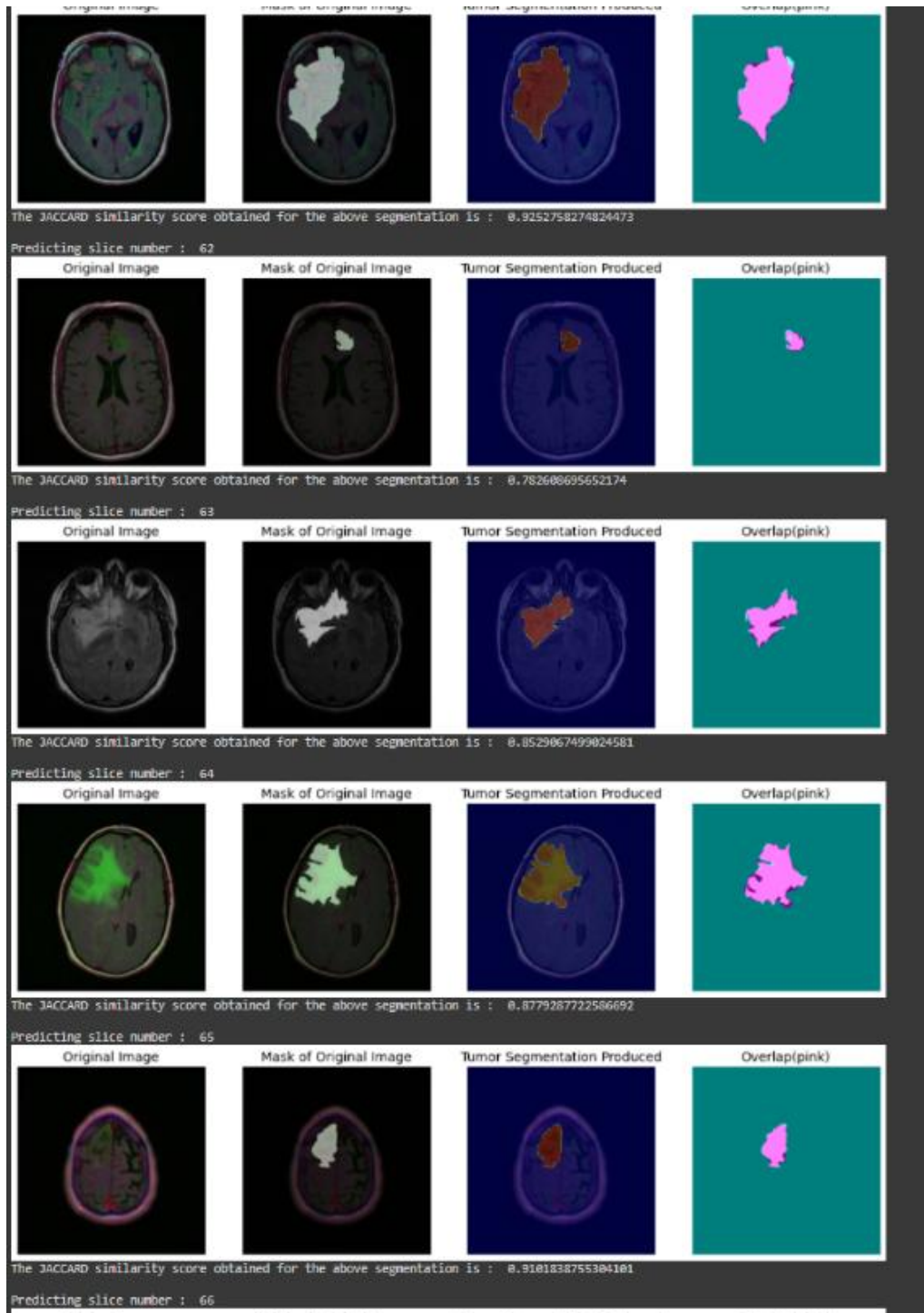


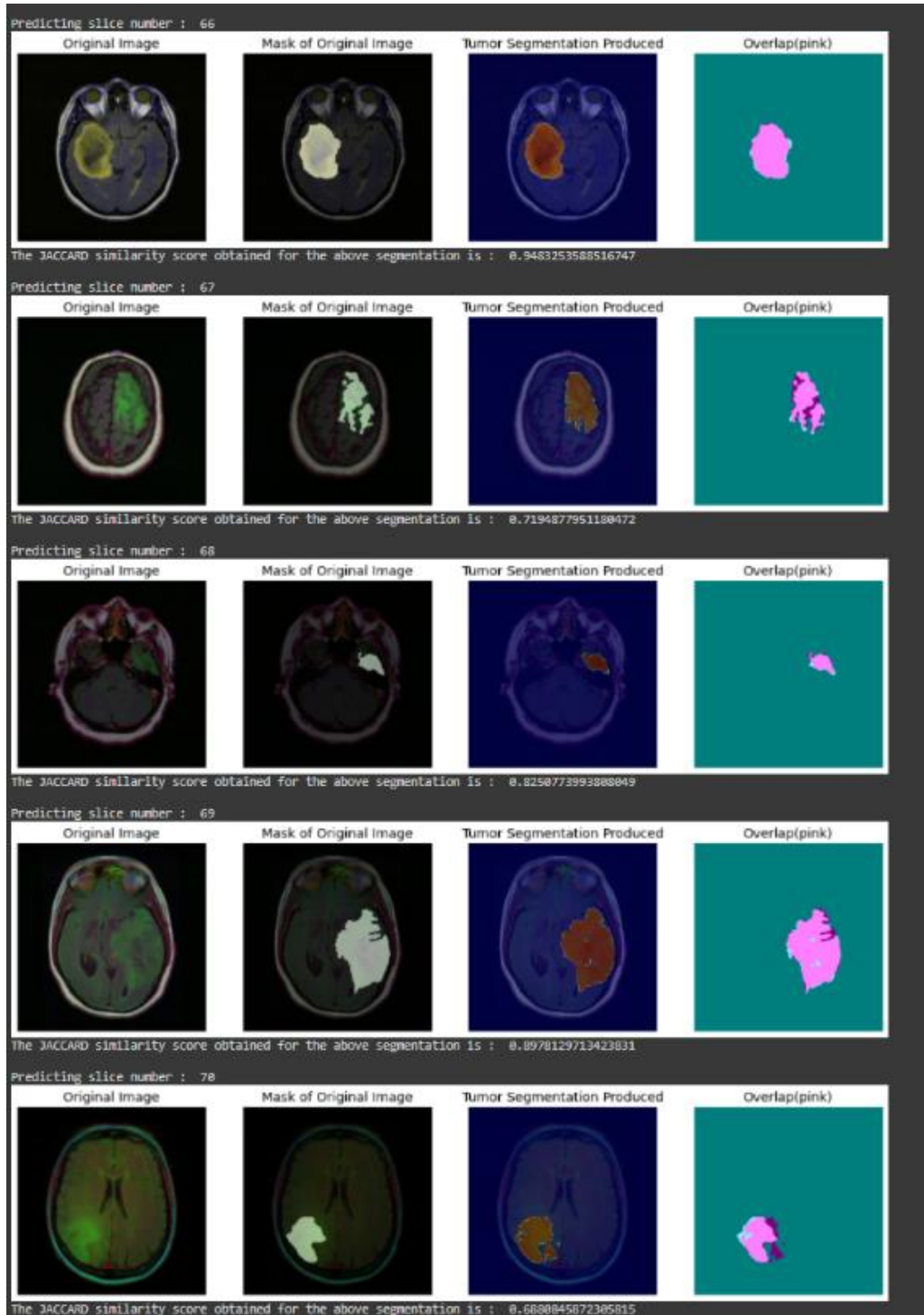


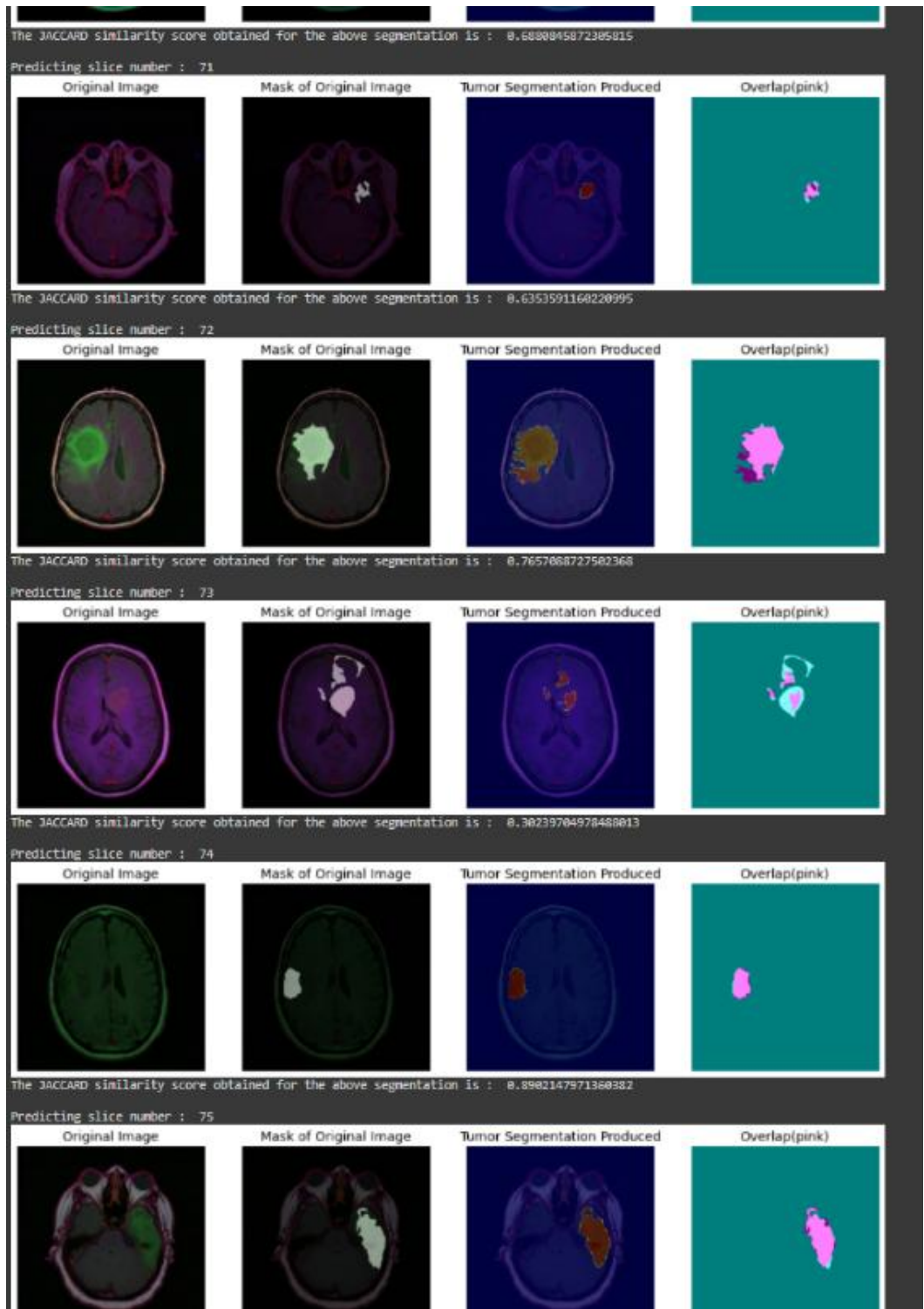


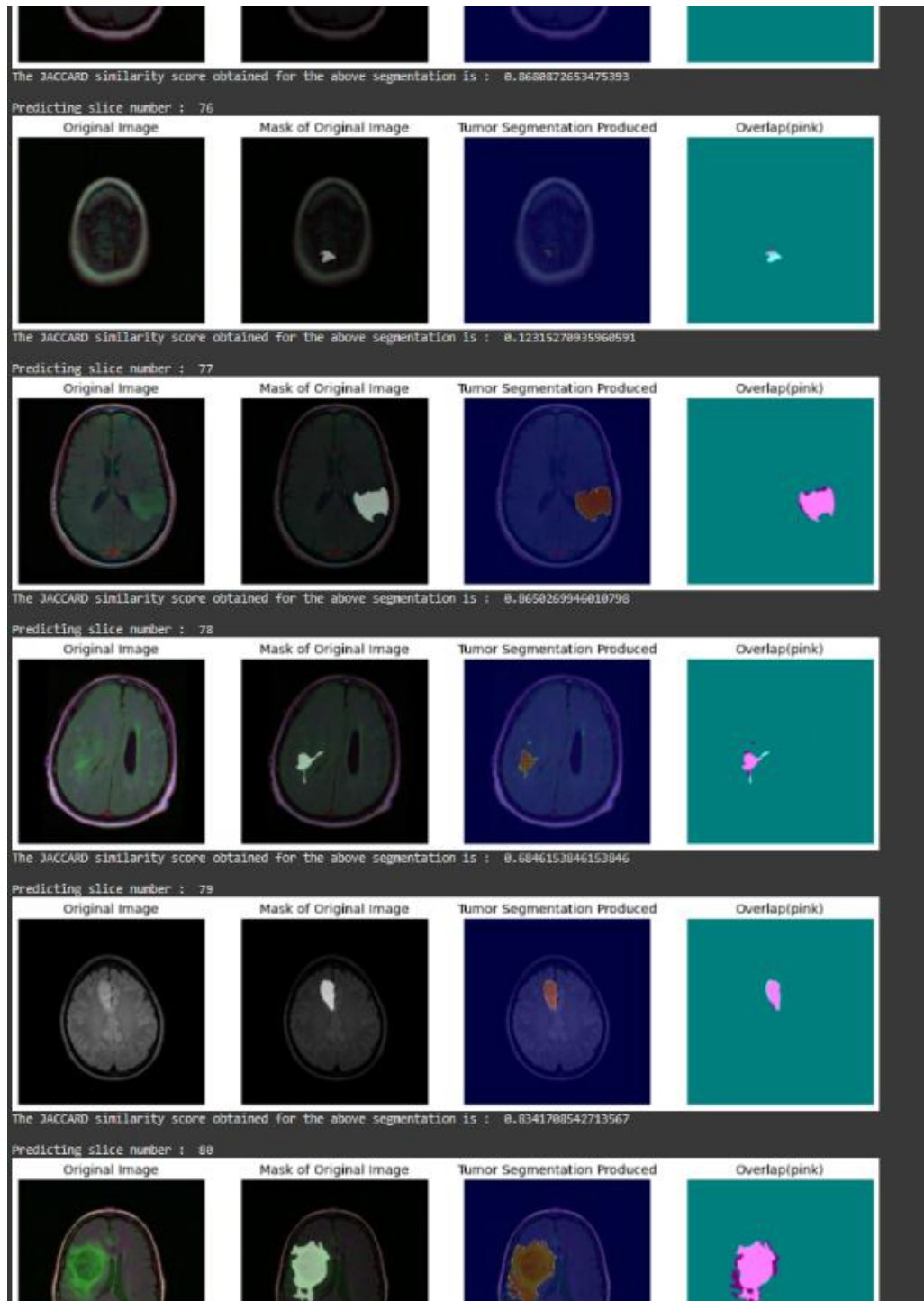


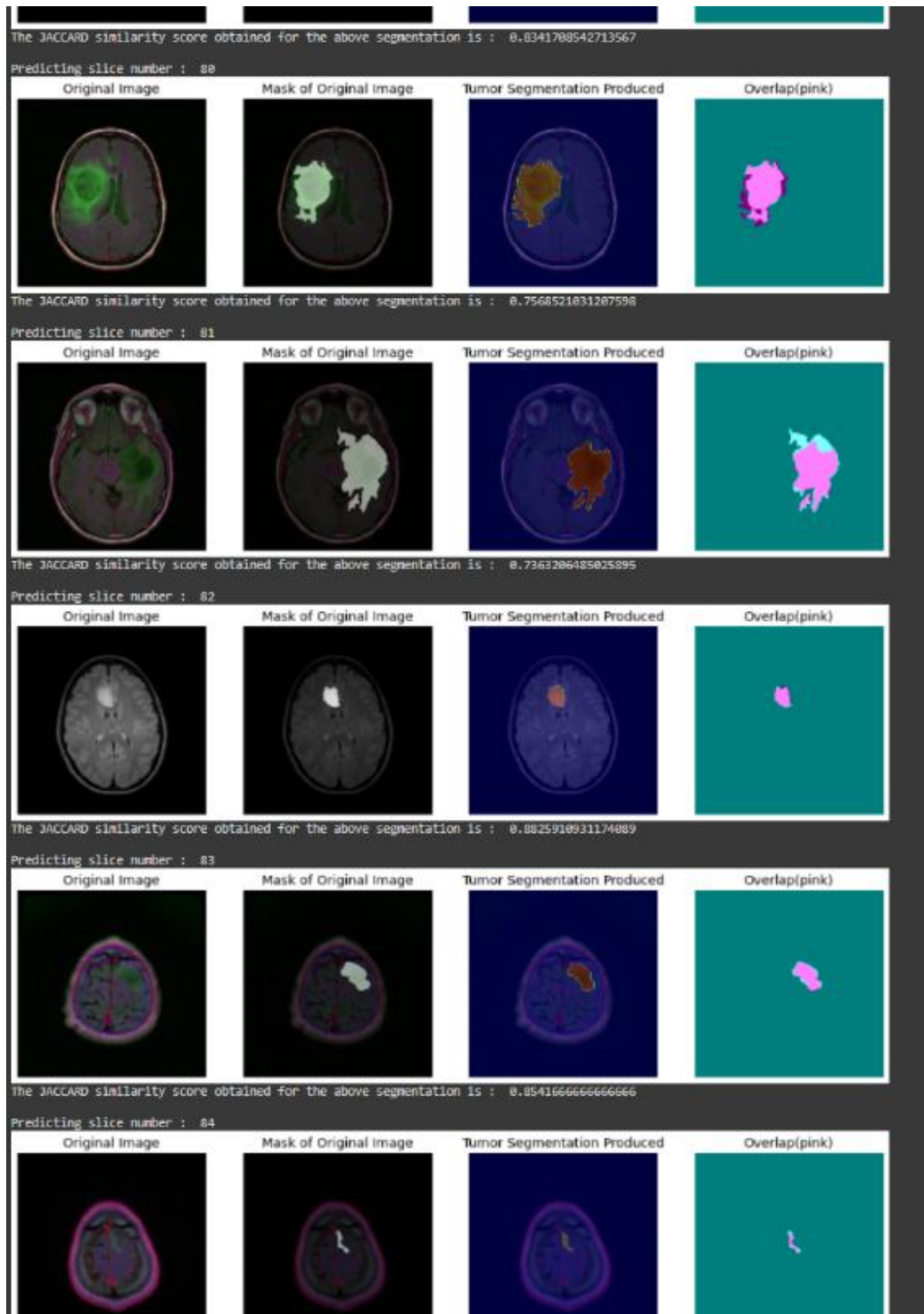


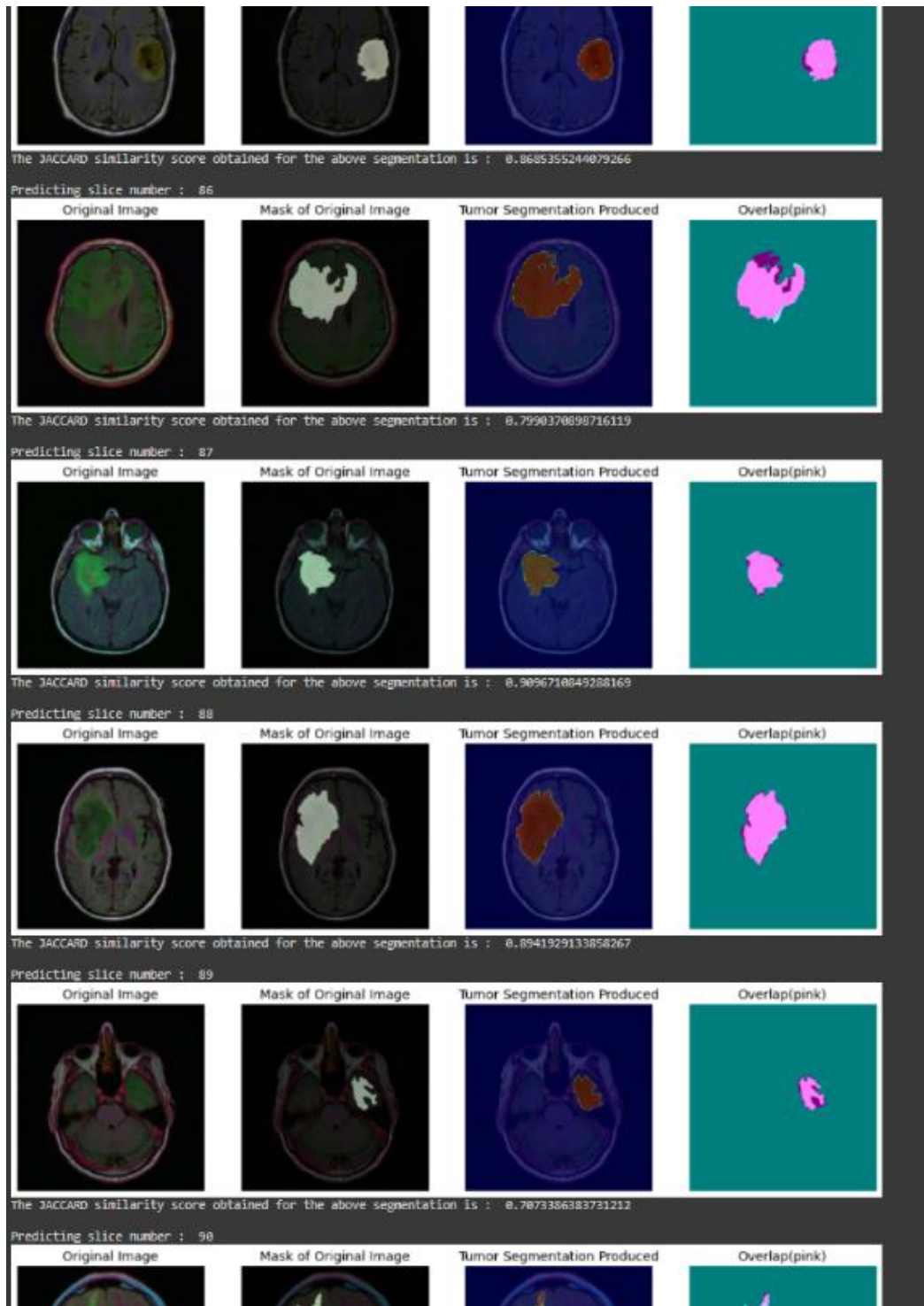


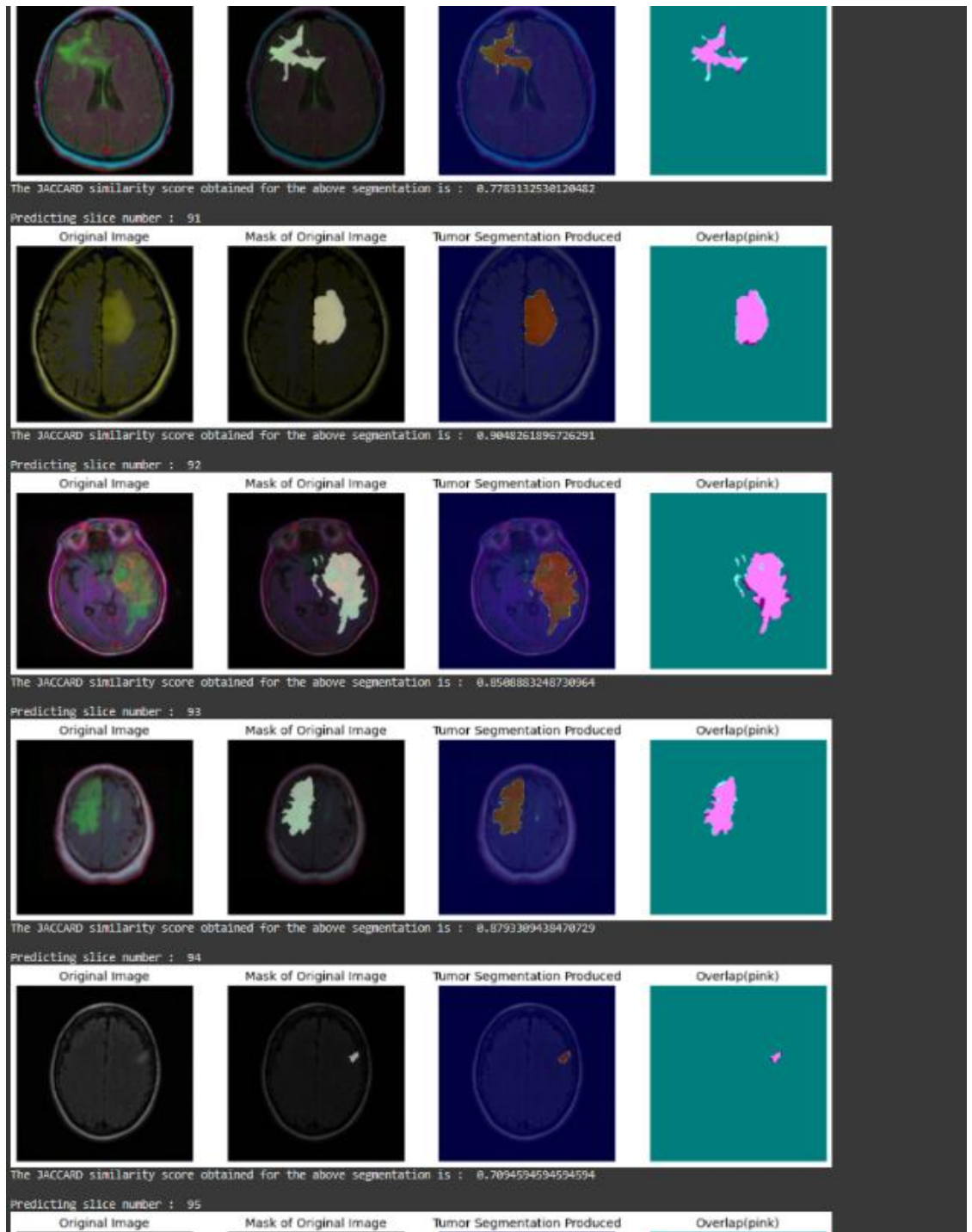


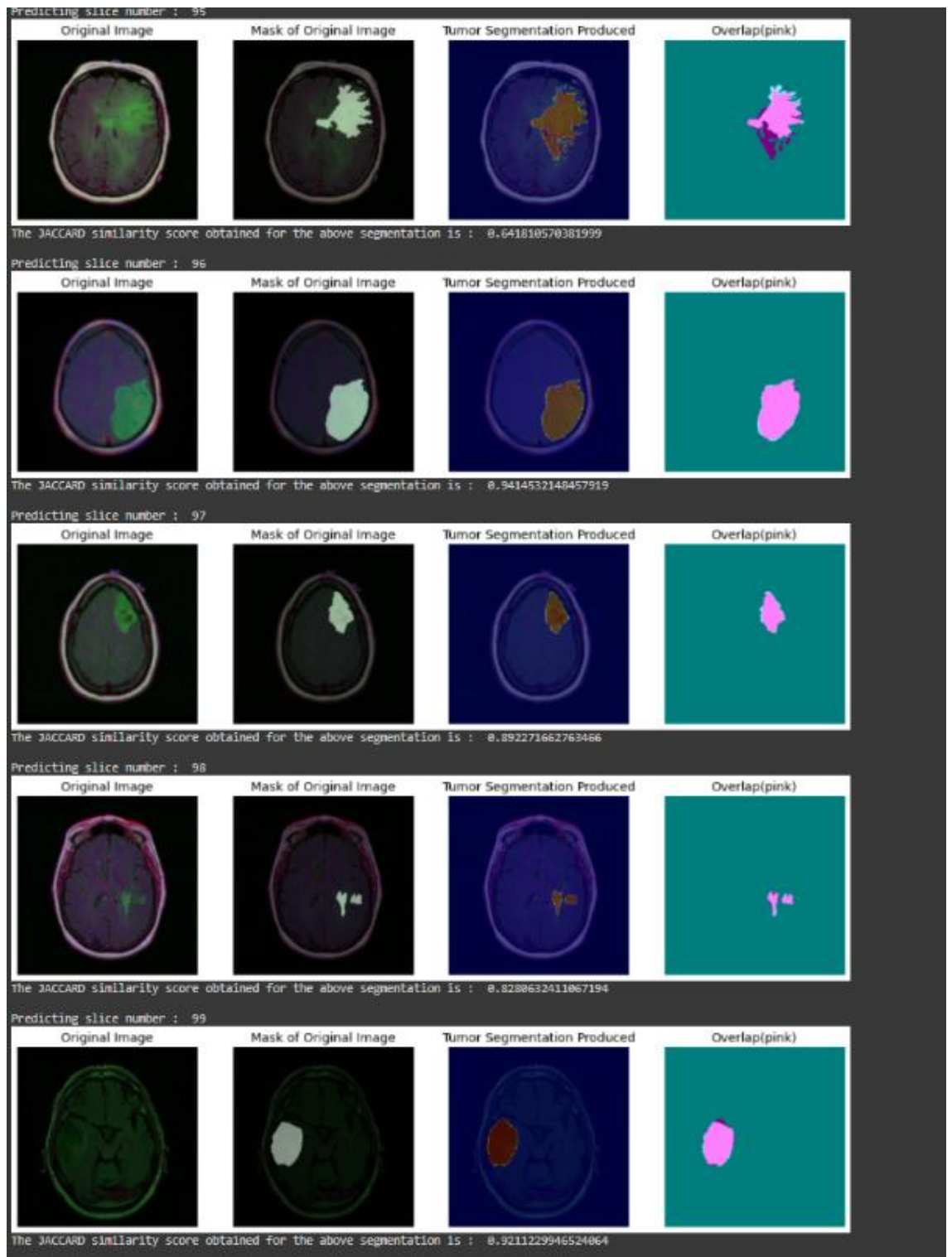


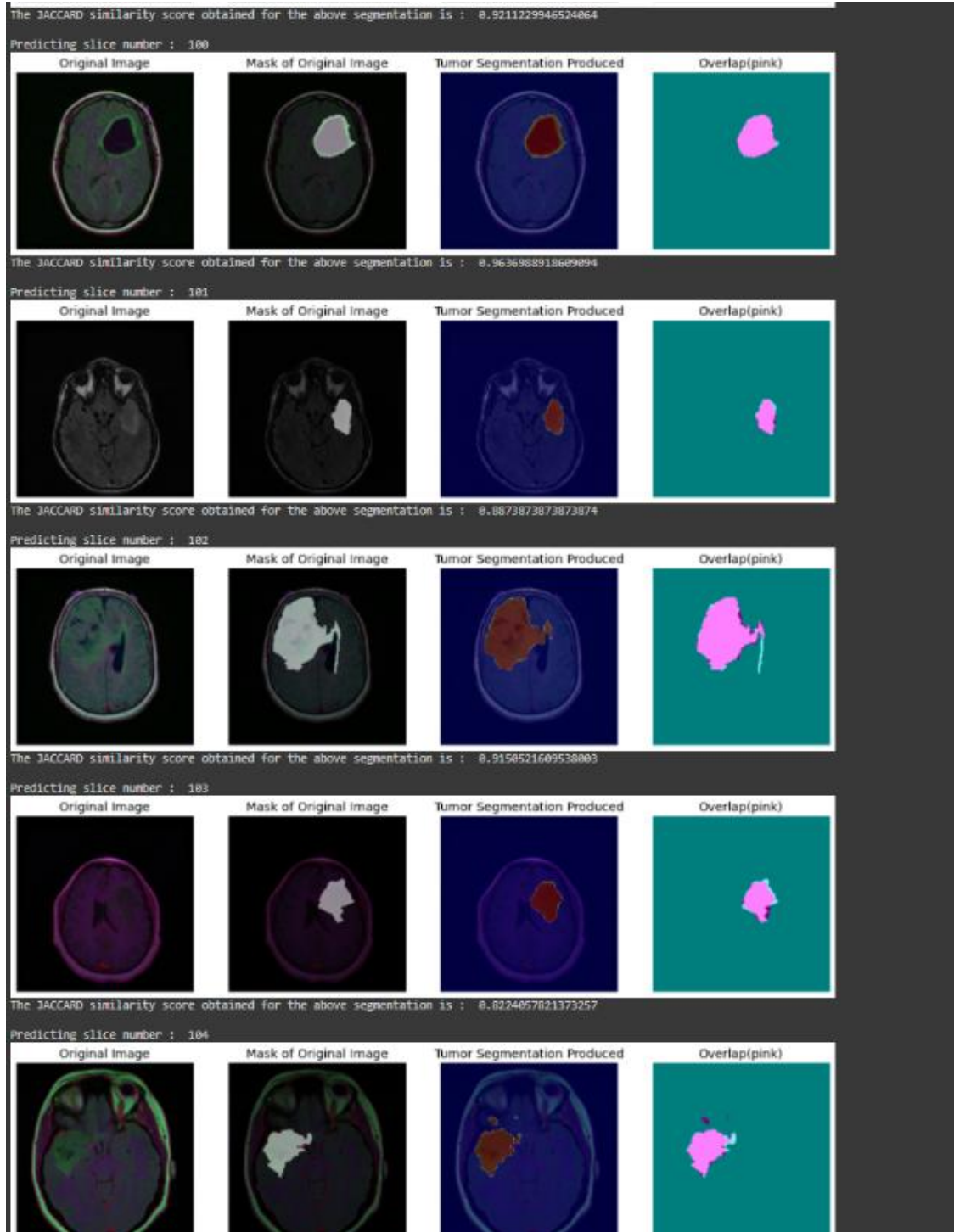


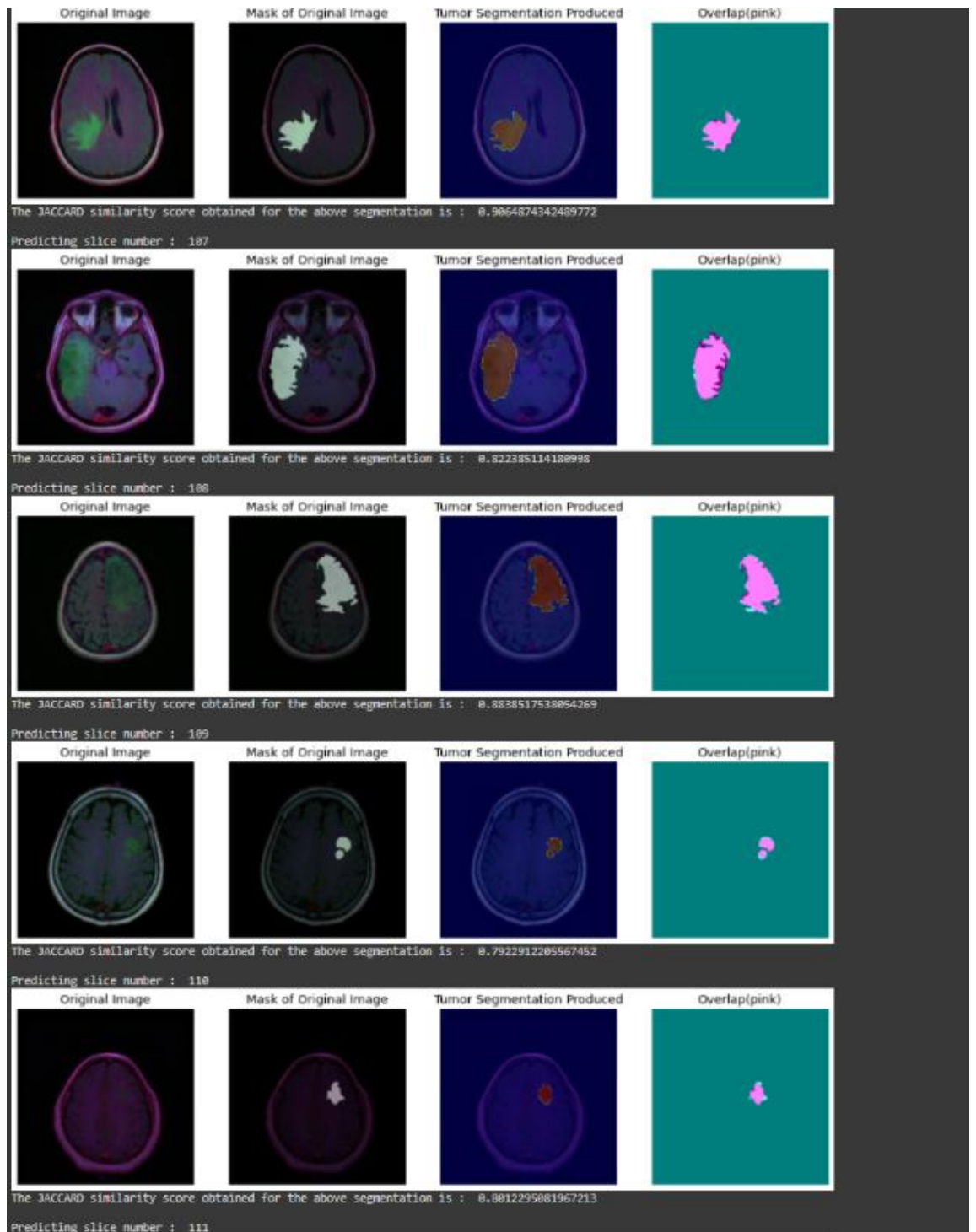


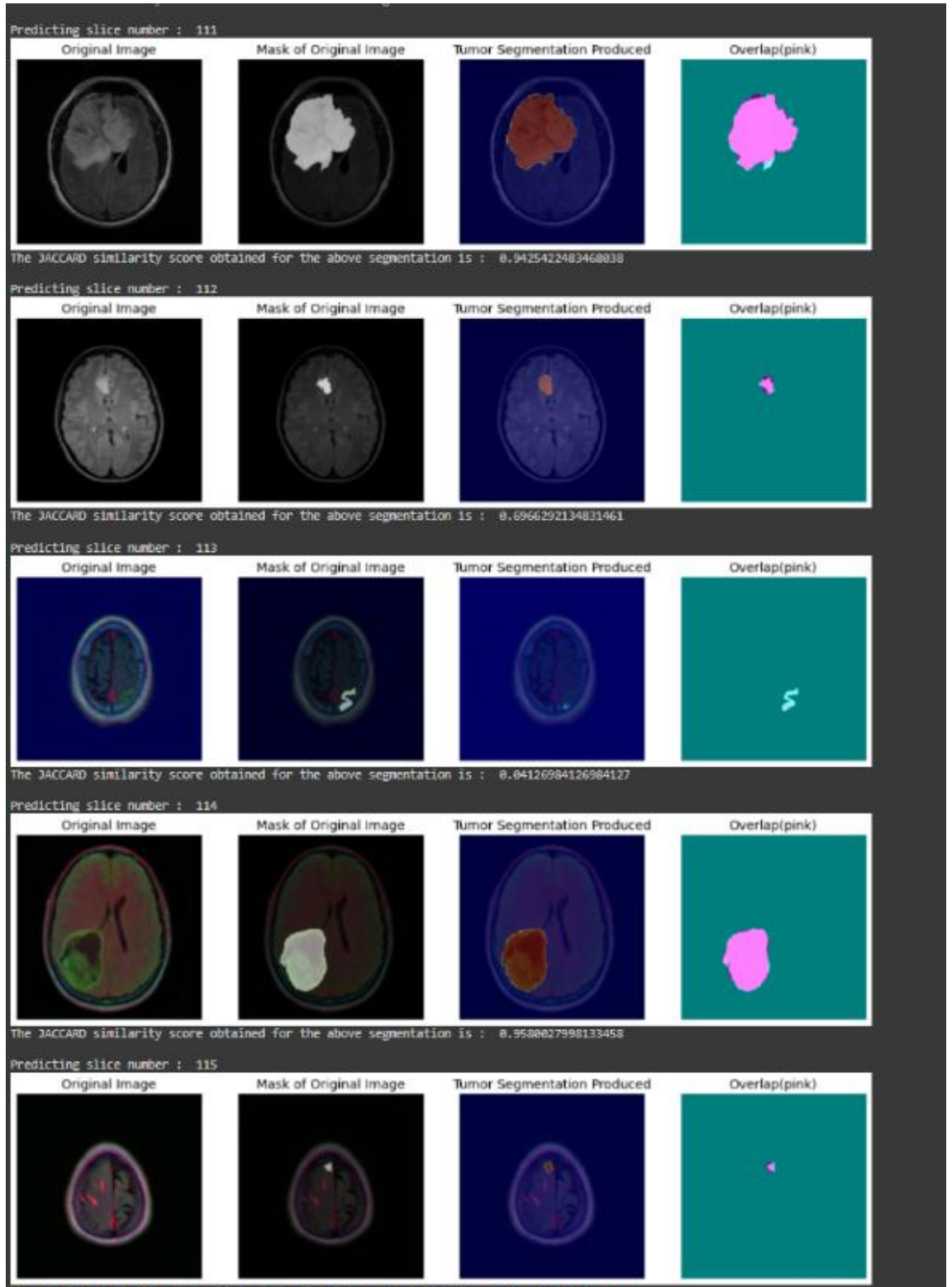


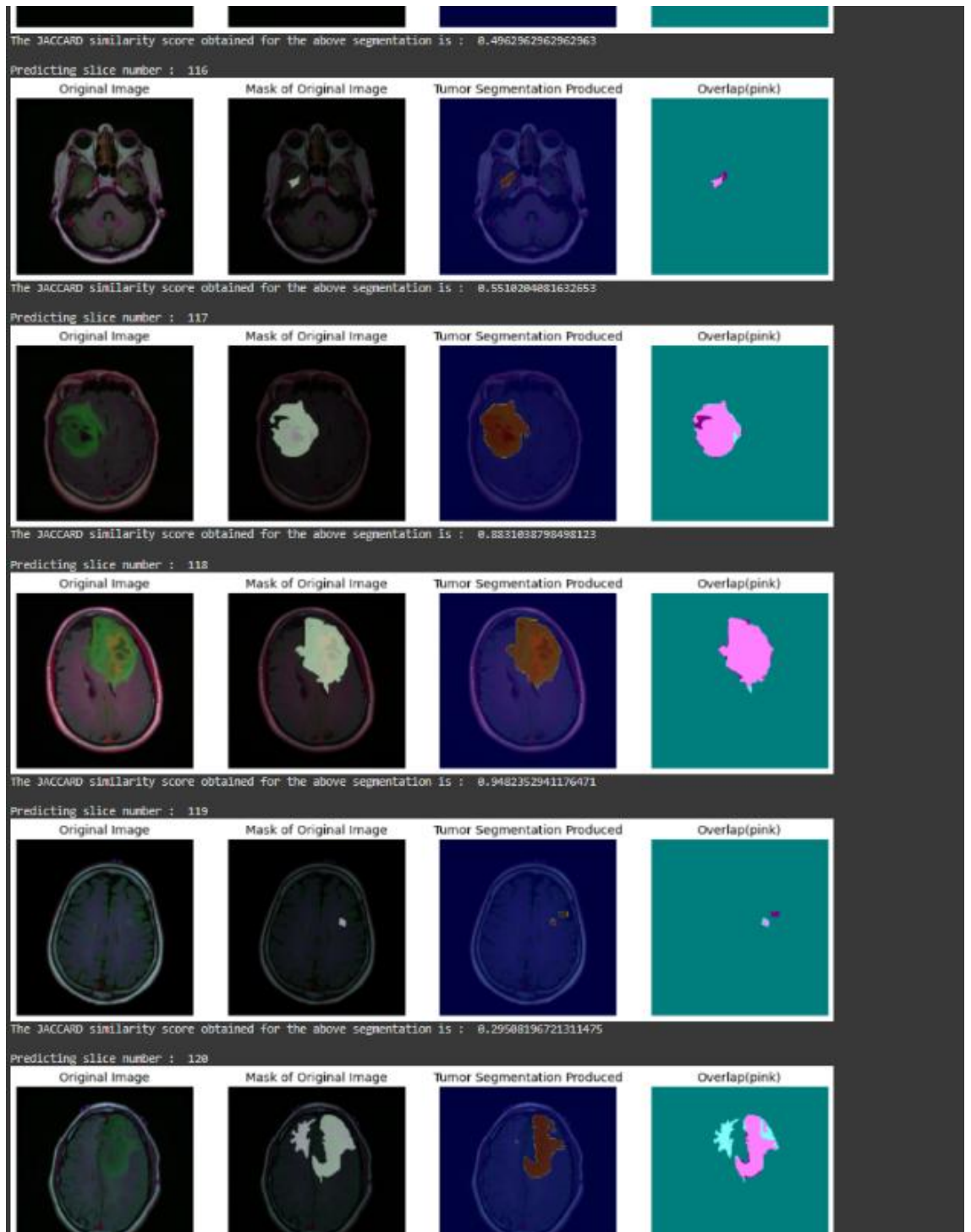


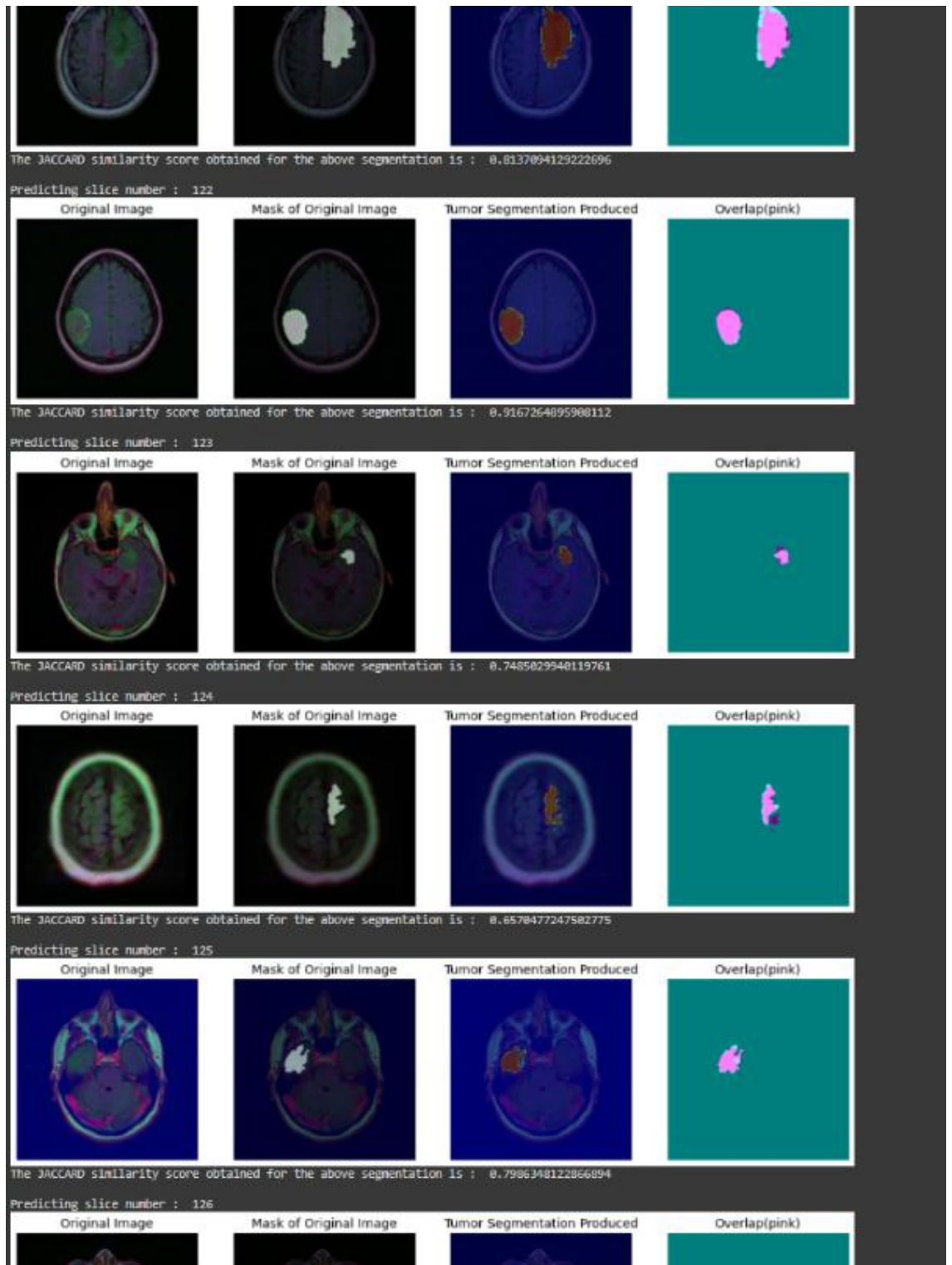


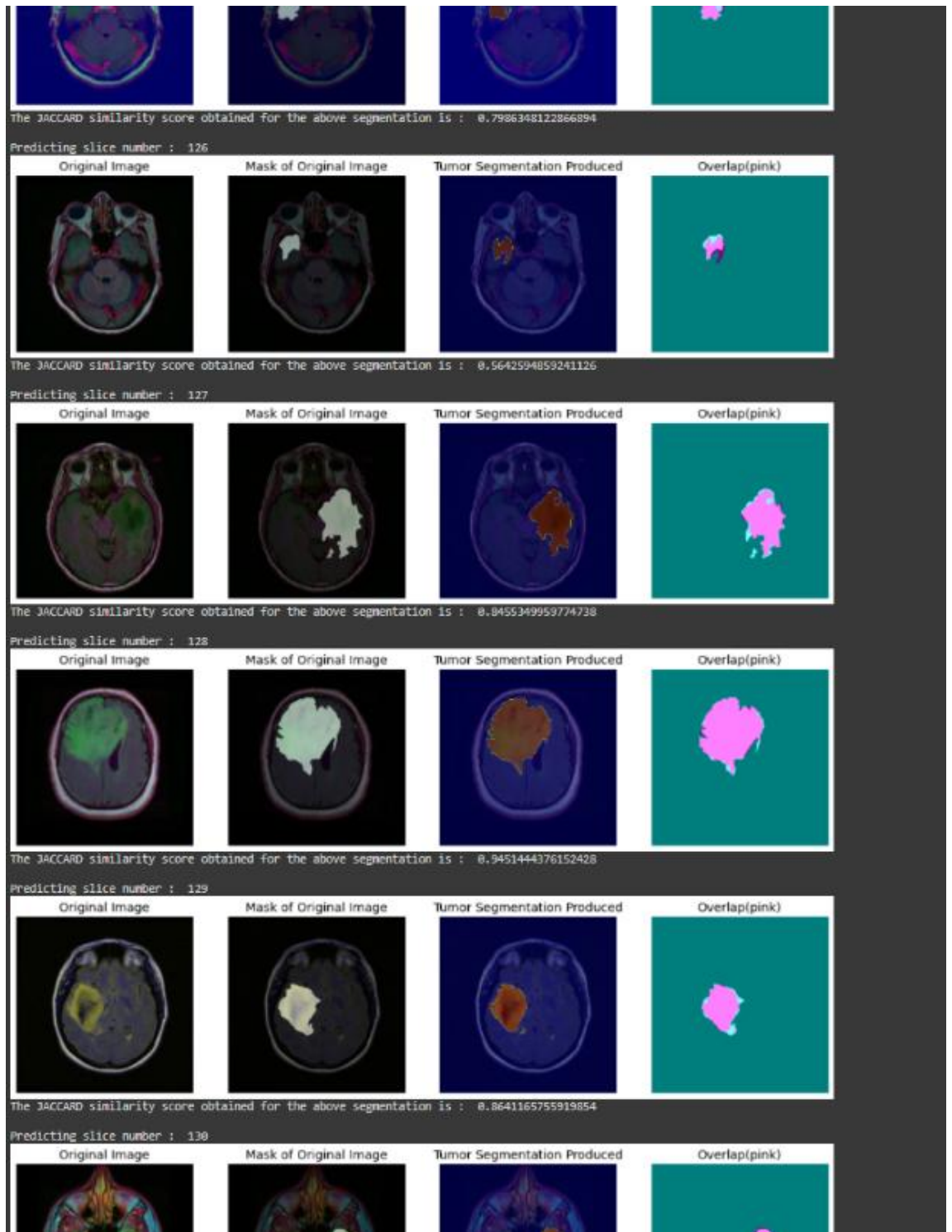


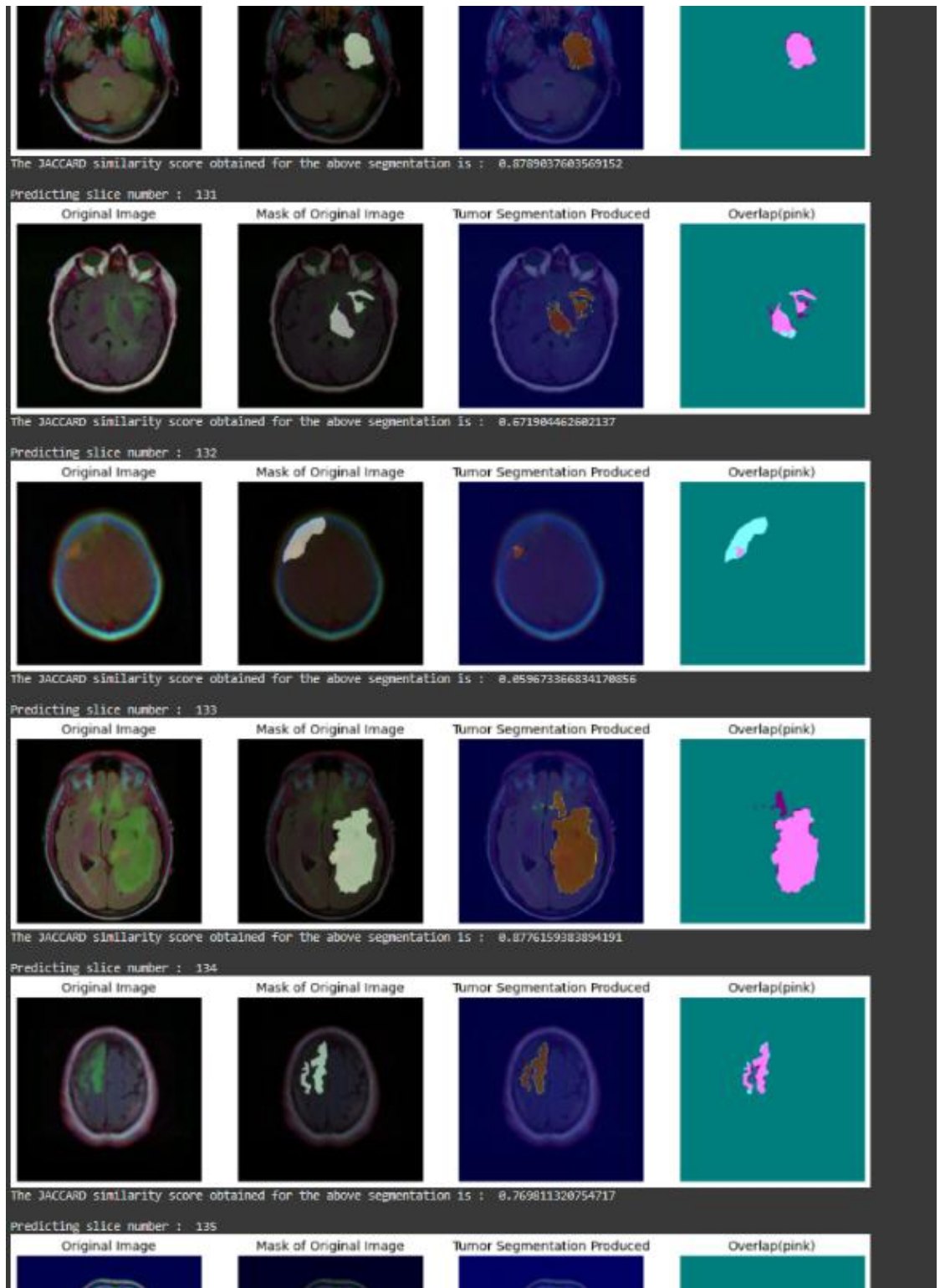


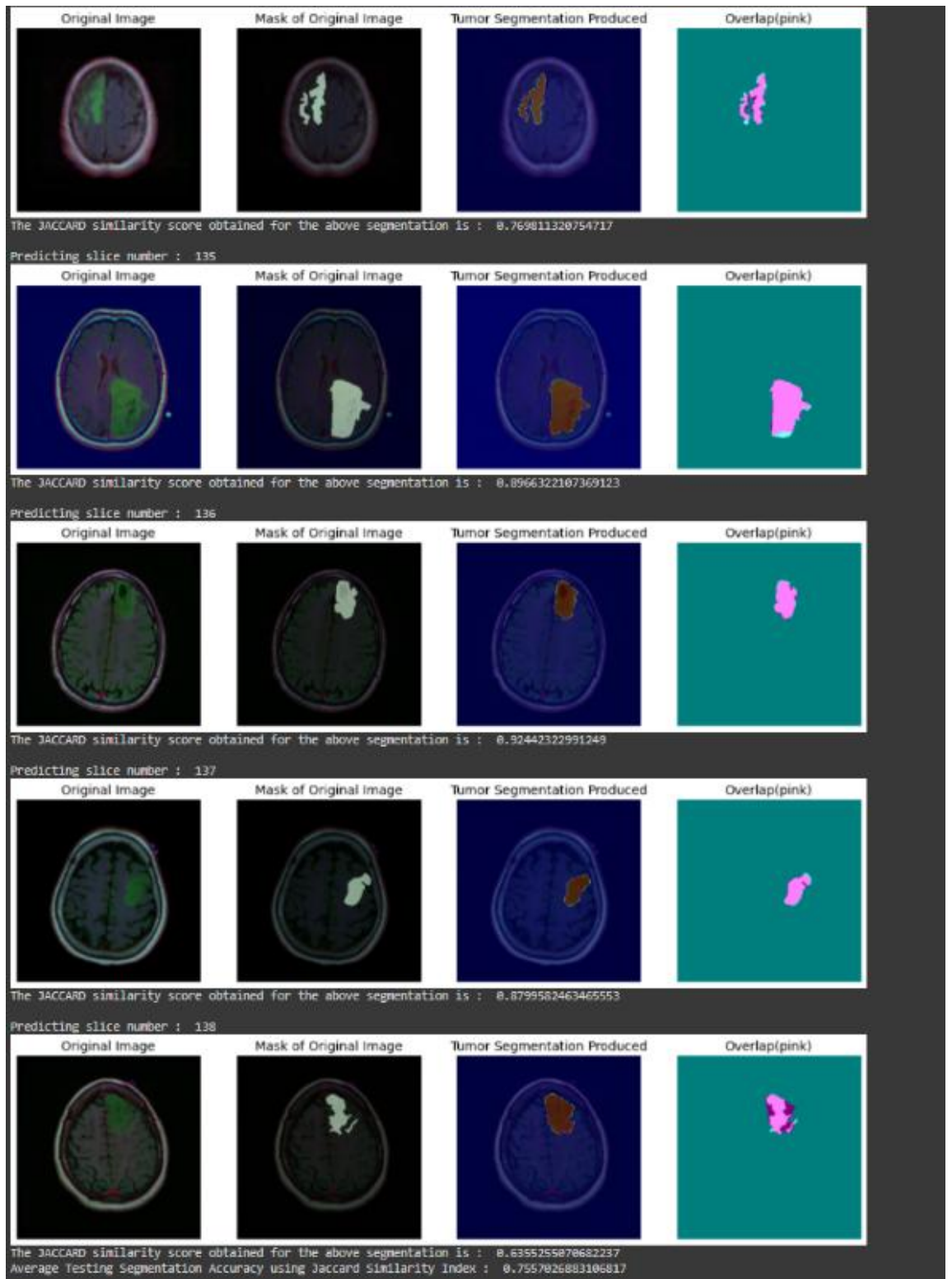












FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 2
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model research and further training

2. WORK TO BE DONE

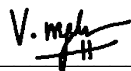
To build more models based on the researched content

3. PROBLEMS ENCOUNTERED

Colab usage limit

4. SELF EVALUATION OF THE PROGRESS

Grown more knowledgeable on deep learning concepts



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 4
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model testing and validation

2. WORK TO BE DONE

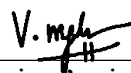
To test on more models

3. PROBLEMS ENCOUNTERED

Colab usage limit and difficult implementations

4. SELF EVALUATION OF THE PROGRESS

Grown more knowledgeable on deep learning concepts



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 6
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Further model testing and validation and refinemenet

Finish up the preparation of second dataset

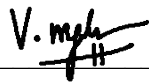
2. WORK TO BE DONE

To test on more models

3. PROBLEMS ENCOUNTERED

Colab usage limit and difficult implementations

4. SELF EVALUATION OF THE PROGRESS



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 8
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model deployment

2. WORK TO BE DONE

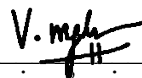
To make the web page more beautiful

3. PROBLEMS ENCOUNTERED

Difficulty in implementing in Flask

4. SELF EVALUATION OF THE PROGRESS

Grown more knowledgeable on Flask website concepts



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 10
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model deployment progress

2. WORK TO BE DONE

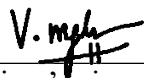
To make the web page more beautiful

3. PROBLEMS ENCOUNTERED


Difficulty in implementing in Flask

4. SELF EVALUATION OF THE PROGRESS

Grown more knowledgeable on Flask website concepts



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3,Y4	Study week no.: 12
Student Name & ID: Darshan A/L Suresh, 16ACB06423	
Supervisor: Dr. Mogana Vadiveloo	
Project Title: Brain Tumor Image Segmentation using Deep Learning Approach	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Model final deployment and web app design finalization

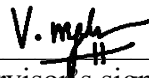
2. WORK TO BE DONE

Finish up report writing

3. PROBLEMS ENCOUNTERED

4. SELF EVALUATION OF THE PROGRESS

Grown more knowledgeable on preparing concise reports



Supervisor's signature





Student's signature

POSTER

BY DARSHAN SURESH

BRAIN TUMOR IMAGE SEGMENTATION USING DEEP LEARNING APPROACH

Deployed as a web application



UNET+SEGUNET ENSEMBLE MODEL

SUPERVISED BY DR. MOGANA

INTRODUCTION

Brain tumor segmentation is hard no more! With this web based application, brain tumor segmentation of 2D MR images can be done easily and swiftly with minimal intervention.

This application also provides built-in computation of Jaccard score of the predictions for those that are learning or researching about the brain tumor MR images!


OBJECTIVES

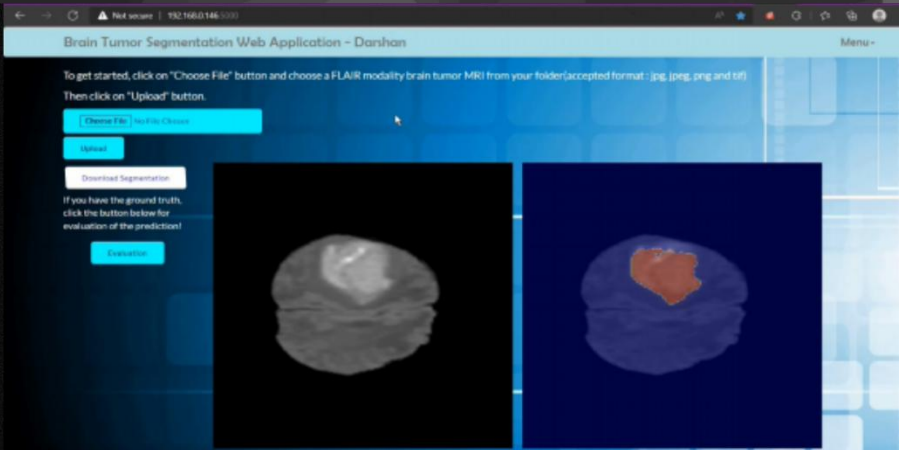
- To develop a brain tumor segmentation model using deep learning approach
- To improve the tumor segmentation using ensemble learning with combined predictions from uNet+segUnet
- To deploy the proposed ensemble model as a web app using Flask

FEATURES OF THIS BRAIN TUMOR SEGMENTATION WEB APP

Automatic! EASY TO USE WEB APP! Satisfactory tumor segmentation!

Demo





PLAGIARISM CHECK RESULT

BRAIN TUMOR IMAGE SEGMENTATION USING DEEP LEARNING APPROACH

ORIGINALITY REPORT

4%	2%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.utar.edu.my Internet Source	<1%
2	"Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)", Springer Science and Business Media LLC, 2013 Publication	<1%
3	"Medical Image Computing and Computer Assisted Intervention – MICCAI 2019", Springer Science and Business Media LLC, 2019 Publication	<1%
4	link.springer.com Internet Source	<1%
5	www.coursehero.com Internet Source	<1%
6	www.frontiersin.org Internet Source	<1%

7	proceedings.mlr.press Internet Source	<1 %
8	www.kaggle.com Internet Source	<1 %
9	"Machine Learning in Medical Imaging", Springer Science and Business Media LLC, 2021 Publication	<1 %
10	Alexander Y. Sun, Bridget R. Scanlon, Zizhan Zhang, David Walling, Soumendra N. Bhanja, Abhijit Mukherjee, Zhi Zhong. "Combining Physically-Based Modeling and Deep Learning for Fusing GRACE Satellite Data: Can We Learn from Mismatch?", Water Resources Research, 2019 Publication	<1 %
11	Saroj Kumar Chandra, Manish Kumar Bajpai. "Brain tumor detection and segmentation using mesh-free super-diffusive model", Multimedia Tools and Applications, 2019 Publication	<1 %
12	norma.ncirl.ie Internet Source	<1 %
13	ijasre.net Internet Source	<1 %
14	www.intel.com Internet Source	<1 %
		<1 %
15	"Medical Image Computing and Computer- Assisted Intervention – MICCAI 2012", Springer Science and Business Media LLC, 2012 Publication	<1 %
16	Heng-Yu Chi, Wen-Huang Cheng, Ming-Syan Chen, Arvin Wen Tsui. "Chapter 18 MOSRO: Enabling Mobile Sensing for Real-Scene Objects with Grid Based Structured Output Learning", Springer Science and Business Media LLC, 2014 Publication	<1 %
17	umpir.ump.edu.my Internet Source	<1 %
18	es.scribd.com Internet Source	<1 %
19	eprints.utm.my Internet Source	<1 %
20	mobt3ath.com Internet Source	<1 %
21	Hala M. Abdelmigid, Mohammed Baz, Mohammed A. AlZain, Jehad F. Al-Amri et al. "Spatiotemporal Deep Learning Model for	<1 %

Prediction of Taif Rose Phenotyping",
Agronomy, 2022

Publication

22	daneshyari.com Internet Source	<1 %
23	docs.lib.purdue.edu Internet Source	<1 %
24	"Machine Learning and Knowledge Discovery in Databases", Springer Science and Business Media LLC, 2020 Publication	<1 %
25	Lecture Notes in Computer Science, 2014. Publication	<1 %
26	Lingxi Xie, Qihang Yu, Yuyin Zhou, Yan Wang, Elliot K. Fishman, Alan L. Yuille. "Recurrent Saliency Transformation Network for Tiny Target Segmentation in Abdominal CT Scans", IEEE Transactions on Medical Imaging, 2020 Publication	<1 %
27	R.K. Al Seyab, Yi Cao. "Differential recurrent neural network based predictive control", Computers & Chemical Engineering, 2008 Publication	<1 %
28	Smita Tiwari, Shivani Goel, Arpit Bhardwaj. "MIDNN- a classification approach for the EEG based motor imagery tasks using deep neural network", Applied Intelligence, 2021	<1 %

Publication

29	deepai.org Internet Source	<1 %
30	theses.whiterose.ac.uk Internet Source	<1 %
31	theses.gla.ac.uk Internet Source	<1 %
32	"Data Intelligence and Cognitive Informatics", Springer Science and Business Media LLC, 2021 Publication	<1 %

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION
TECHNOLOGY (KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	16ACB06423
Student Name	DARSHAN A/L SURESH
Supervisor Name	DR. MOGANA A/P VADIVELOO

TICK (✓)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✗	Front Plastic Cover (for hardcopy)
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✗	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 22/4/2022