Brain Tumor Segmentation From MRI Images Using Deep Learning

Submitted By

Nir P. Parikh

17MCEN09

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY

AHMEDABAD-382481

May 2019

Brain Tumor Segmentation From MRI Images Using Deep Learning

Major Project

Submitted in fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering (Networking Technologies)

Submitted By

Nir P. Parikh

(17MCEN09)

Guided By

Prof. Rupal Kapdi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2019

Certificate

This is to certify that the major project entitled **"Brain tumor segmentation from MRI images using deep learning"** submitted by **Nir P. Parikh (17MCEN09)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering (Netwoking Technologies) of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project part-II, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Rupal Kapdi
Guide & Associate Professor,
CSE Department,

Institute of Technology,
Nirma University, Ahmedabad.

Dr. Gaurang Raval
Associate Professor,
Coordinator M.Tech - CSE (Networking Technologies)
Institute of Technology,
Nirma University, Ahmedabad

Dr. Madhuri Bhavsar
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Nir P. Parikh**, **17MCEN09**, give undertaking that the Major Project entitled "**Brain tumor segmentation from MRI images using deep learning**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering (Networking Technologies)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:

Place:

Endorsed by

Prof. Rupal Kapi

(Signature of Guide)

Acknowledgements

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Rupal Kapdi**, Associate Professor, Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Madhuri Bhavsar**, Hon'ble Head of Computer Science And Engineering Department, Institute of Technology, Nirma University, Ahmedabad for her kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. Alka Mahajan**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Nir P. Parikh

17MCEN09

Abstract

Brain tumor is an abnormal growth of tissues inside the skull. Not all tumors are cancerous but still it affects to the nervous system. Early detection of a tumor can increase the chance of survival. That's why it is more important to identify tumor more accurately. Manual detection needs highly understanding about the tumor and experience and mainly it is highly dependant on human perspective. With the help of deep learning it is possible to develop a model which can identify brain tumor in a very early stage. Model can be trained through a large number of MRI images which helps to make it more accurate. Convolutional neural network is used to analysis visual images. It is useful to extract different features from given image and classify into different groups. U-Net architecture is implemented using CNN and specially designed for bio medical image segmentation. U-Net uses high number of parameters which leads to the over-fitting. It is also computationally intensive task. To overcome this issue, inception network is introduced inside the U-Net architecture. Using it, complete deep neural network can be formed to detect brain tumor. Glioma is a malignant type of tumor which directly affects nervous system. Current model is based on detection of glioma tumor. Training images and ground truth images are provided under Brain Tumor Segmentation Challenge. Data-sets are divided into HGG (High Grade Glioma) & LGG (Low Grade Glioma). Both are used to train model as well as test it.

Abbreviations

GM	Gray Matter.
WM	White Matter.
CT	Computerized Tomography.
MRI	Magnetic Resonance Imaging.
FLAIR	Fluid Attenuated Inversion Recovery.
CNN	Convolutional Neural Network.
RGB	Red Green Blue.
HGG	High Grade Glioma.
LGG	Low Grade Glioma.

–

Contents

List of Figures

Chapter 1

Introduction

1.1 Brain Tumor

Brain tumor occurs due to abnormal growth of tissues inside the body. Based on a location of a tumor it is divided into primary and secondary tumor. Primary tumor developed inside the brain. Other hand, secondary tumor developed inside the other body parts and then spread inside the brain. In both scenario it affects to the central nervous system which leads to the death. Tumor may be benign (not cancerous) or malignant (cancerous).

Human brain is divided into three parts (1) Gray matter, (2) White matter and (3) Cyrebrospinal fluid. Gray matter contains sensors for seeing, hearing, speech, emotions, movement, decision making many others. White matter transfer information between brain and other body parts. It also connect Gray matter internally. Cyrebrospinal fluid covers GM & WM and it also covers empty area inside the skull. It is used as a shock absorber. Basically brain is floating inside Cyrebrospinal fluid. Skull is a very rigid structure that is not expandable. So abnormal growth of a tissue becomes extreme problem.

Brain tumor is divided into four types:

- Grade I: It is a beginning of a tumor and very slow one. It looks like a normal growth of a tissue. Tumor is not a cancerous and known as benign.
- Grade II: It is same as Grade I but tumor can change from benign to malignant.
- Grade III: It is malignant tumor and growth is very faster.

- Grader IV: It is also malignant tumor but growth is abnormally fast. Tumor may be spread from brain to other parts of a body. It is highly risky tumor.

Location of a tumor, size and growth are key factors to diagnose brain tumor. Early detection can increase a chance of a survival. Brain tumor can be pictured using CT scan and MRI technologies. Those images are observed by experts or specialist. But manual diagnosis is based on a person's knowledge and experience in this area. And also it is very time consuming and non-reproducible.

1.2 MRI Images

If this type of observation can be done by a computer, it is more easiest way to identify a tumor as well as keep records of images. Computer can take images as a input and identify key objects from the images. These objects can be used to identify tumor. Various techniques are available for capturing the images.

- Computed Tomograph Scan: CT scan uses X-Ray to produce 3D image of body organ. It takes multiple X-Ray to generate volumetric image which shows bones, organs and tissues. But CT uses ionizing radiation through X-Ray which can increase chance of cancer.

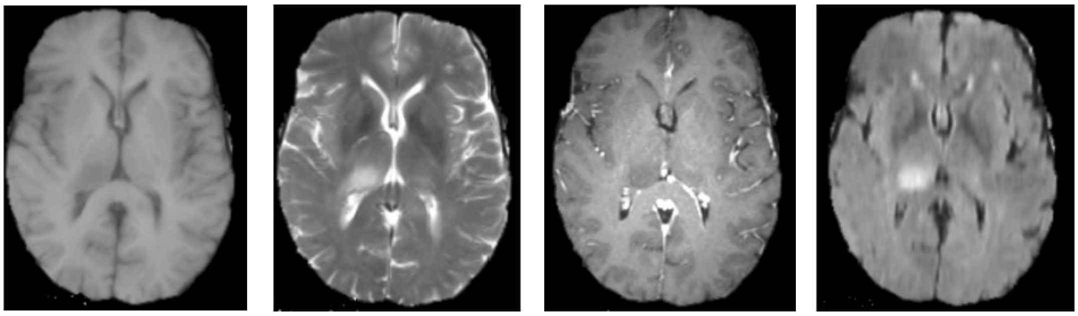
- Magnetic Resonance Imaging: MRI is a widely used technique for brain tumor identification. MRI uses magnetic field and radio pulse to generate 3D images. Depends on a magnetic field strength and radio frequency, four types of weighted images are generated. (1) T1, (2) T2, (3) T1C, (4) FLAIR. MRI image gives exact results even in the case of beginning of a tumor. Different weighted images are shown in figure 1.1a, figure 1.1b, figure 1.1c and figure 1.1d [1].

Advantages of MRI images,

- It generates high resolution images.

- It captures different intensity of images which is important to segmentation.

- It contains less noise.



(a) T1 (b) T2 (c) T1C (d) FLAIR

- Does not affect patients.

Disadvantages of MRI images,

- It is time consuming to capture a single image.
- It is very difficult to find out homogeneity in MRI images due to radio frequency and magnetic pulse.

MRI image can not be used directly for processing. It contains noise, non-brain tissues and some time image is not aligned. So it needs to be pre-processed before using to train model. Problems with MRI images,

- Tumor may originated anywhere. So to find out location of tumor is difficult.
- Abnormal growth of tissues can suppress normal tissues. So it will change the shape of brain parts.
- Every weighted image contains information. So without a single image it is very difficult to segment image.

1.3 Segmentation

Images needs to be segmented properly to extract features. Segmentation of an image is based on following factors.

- Application for which segmentation is needed.
- Image is taken for which body organs.
- Technique used to capture image.

Segmentation divides image with boundary and integrate region with same characteristic. There are some issues with segmentation which is equally important to resolve before processing.

- Due to different intensity, tissues in image can be overlapped by other one.
- Noise will added into images due to usage of sensors.
- Some organs are in motion ex. heart.
- In gray scale image, gray levels are very close to each other.

Before going for segmentation, these issues must be resolved. Noise can be removed from filtering, using restoration motion can be removed. Segmentation can be based on color, contrast, brightness, texture, gray scale. There are various conventional techniques to segment image.

- Threshold based detection: Threshold method is used to convert gray scale image into binary image. It divides image into small chunks and define a threshold value. Using this value it converts image into binary. Threshold values must be given initially.
- Edge based detection: Edge detection is based on identify boundary line inside the image region. Based on boundaries image is segmented into small junks.
- Region based detection: It will retrieve pixels with same characteristic and grouped together to form homogeneous region.
- Texture based detection: It is based on identification of texture inside the image. Edge, Region and Texture based detection needs seed point to start. Seed point must be specified before segmentation.
- Atlas based detection: It uses information of organs like shape, size, features and color to detect boundary. But atlas selection is very important to get exact segmentation.

All these methods need human interaction before segmentation process. So it is deeply based on human selection and knowledge. Deep learning can solve this issue. Using it, a network can be developed which learns from the experience to solve this type of issues. Deep learning gives accurate result with compare of conventional segmentation method. Convolutional neural network is deep learning model which is used to process 2D 3D images using supervised learning. Convolutional, detection and pooling operations are used to extract features from images at every layer. Convolutional operation required filter at every layer to extract features. But it is difficult to decide filter at every level. Inception module is introduced to solve this problem. Model will decide size of filter as per it's requirement.

1.4 Similarity Metrics

To verify a brain tumor detection system, no ground truth images are available. BraTS (Brain Tumor Segmentation) is a part of SBIA (Section of Bio-medical Image Analysis) and CBICA (Center of Bio-medical Image Computing and Analysis). It provides free MRI image database to use as a ground images. Using that accuracy for a model can be identified.

- Dice's similarity co-efficient:

$$DC = \frac{2|A \cap B|}{|A| + |B|} \tag{1.1}$$

  Value of DC must be between 0 and 1. Here A is ground truth image dataset and B is testing image set.

- Jaccard similarity: Similarity equation,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.2}$$

  Dissimilarity equation,

$$d(A, B) = 1 - J(A, B) \tag{1.3}$$

  Value of J(A,B) must be between 0 and 1. Here A is ground truth image dataset and B is testing image set.

1.5 Glioma Tumor Data-set

Glioma is one of the most common tumor. It occurs into brain or spinal code. It is malignant type of tumor. So early detection is highly useful. Brain Tumor Segmentation Challenge provides data-set for training. Data-set is divided into 210 number of HGG & 75 number of LGG. Each record contains four set of MRI images as an input T1, T1-weighted, T2 & Flair and one ground truth image for comparison. Dice co-efficient is used to measure accuracy of an model.

Chapter 2

Literature Survey

2.1 Literature Summary

Paper Title	Year	Type	Author	Summary
Brain Tumor Segmentation Towards a better life	2016	Article	Ms. Rupal R. Agravat & Dr. Mehul S. Raval	This article provides basic knowledge about brain tumors, its type, method to detect tumors, technique to capture visual images of brain tumor and provides different techniques for image segmentation. MRI image is used to identify brain tumor which is a gray scale image. For segmentation, many conventional methods are described but with the help of machine learning, segmentation would be more accurate.
Deep Learning for Automated Brain Tumor Segmentation in MRI Images	2017	Paper	Ms. Rupal R. Agravat & Dr. Mehul S. Raval	Paper contains information about Deep learning methods to segment image accurately. Convolutional Neural Network is mainly used for image processing. CNN contains convolution, maxpooling layer to reduce number of features. Filters are used to traverse image and extract information. Padding, strides , number of filter will affect the network accuracy. Using deep network will return more accurate results. This paper also introduced comparison between multiple CNN architecture.

Table 2.1: Literature summary

Paper Title	Year	Type	Author	Summary
Going Deeper with Convolutions	2015	Paper	Christian Szegedy , Wei Liu, Yangqing Jia , Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich	This paper has introduced new classification method Inception for deep learning. It is very difficult to identify filter size at every layer of CNN. To make it more simple, inception model uses 3 filters 1 X 1, 3 X 3 and 5 X 5 with one maxpooling layer. All are combined into one single inception layer. Detailed information is available about different version of inception network is available in paper. GoogLeNet is an implementation of inception layer. It contains 22 inception layer in stack order.
U-Net: Convolutional Networks for Biomedical Image Segmentation	2015	Paper	Olaf Ronneberger, Philipp Fischer, and Thomas Brox	U-Net is an implementation of CNN specially for image segmentation and feature extraction. Ii has 3 downsampling CNN layer, 2 bottleneck layer CNN and 4 upsampling CNN layer. Paper has introduction, working and implementation about U-Net architecture.

Table 2.2: Literature summary

Paper Title	Year	Type	Author	Summary
Brain Tumor Segmentation and Radiomics Survival Prediction: Contribution to the BraTS 2017 Challenge	2018	Paper	Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, Klaus H., Maier-Hein	This article provides basic knowledge about brain tumors, its type, method to detect tumor. MRI image is used to identify brain tumor which is a gray scale image. U-Net inspired deep neural network is proposed to detect tumor. Techniques and measures used to figure out efficiency were listed. Outputs were also attached.
Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks	2015	Paper	Hao Dong, Guang Yang, Fangde Liu, Yuanhan Mo, Yike Guo	This paper has introduced proposed architecture based on U-Net to detect malignant brain tumor. Data set was taken from BraTS 2015 challenge.

Table 2.3: Literature summary

Chapter 3

Convolutional Neural Network

3.1 Basics of CNN

Convolutional neural network is sequence of layers, and every layer transforms image into smaller resolution to generate number of features. It is used to process 2D and 3D images. In CNN 3 different layers are used. (1)Convolution, (2)Pooling and (3)Fully connected layers. [13]

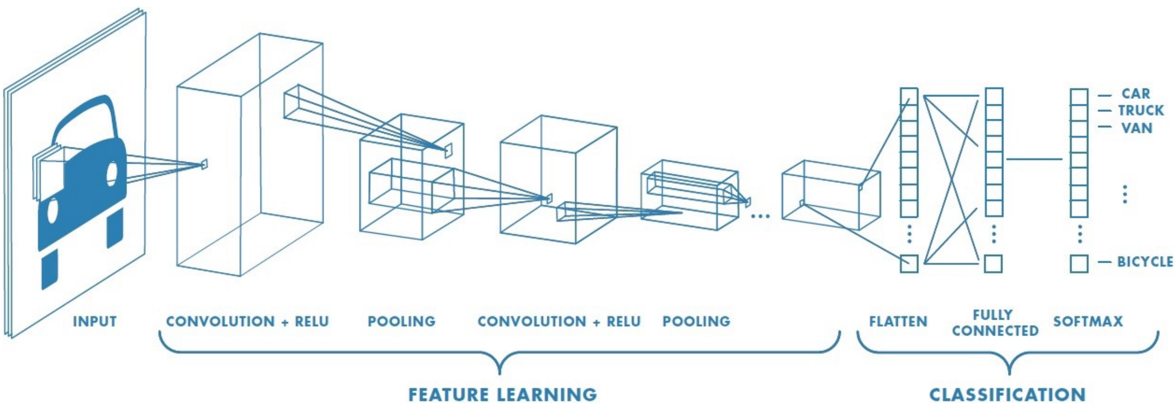


Figure 3.1: Convolutional Neural Network [13]

3.1.1 Image Processing

Every image is consider as a matrix of a pixel value. Based on image resolution, size of a matrix is defined as H X W X D (H=height, W=width, D=dimension). Every pixel is converted to a fix number based on RGB value. For large dimension images, number of features are extremely large. Using these numbers of feature to detect tumor is very complex computation task.

3.1.2 Convolution Function

To reduce number of features CNN is used. Convolutional function is used to extract features from images. For that it performs mathematical operation between small chunks of a image matrix and kernel or filter matrix. Size of a filter is depends upon requirement of a network but dimension must be same for input image and filter. Filter is used to identify pattern inside the image. Image size is H1 X W1 X D and N filters with the size of H2 X W2 X D then output image size is, [10]

$$(H1 - H2 + 1)X(W1 - W2 + 1)XN \tag{3.1}$$

3.1.3 Padding

In convolution function, at each layer size of an image is reduced and also it affects to the information inside the image. To maintain size of an image it is necessary to add padding bits outside the image matrix. Now, image size is H1 X W1 X D, N filters with the size of H2 X W2 X D and P padding is used. Output image size is, [10]

$$(H1 + 2P - H2 + 1)X(W1 + 2P - W2 + 1)XN \tag{3.2}$$

$$H1 + 2P - H2 + 1 = H1 \tag{3.3}$$

$$P = \frac{H2 - 1}{2} \tag{3.4}$$

Using equation (3.4), padding size can be identified to generate same size of output image. Convolution has two types of padding (1) VALID and (2) SAME. For VALID option, no padding is applied inside the convolution. And for SAME option, user has to specify value for padding.

3.1.4 Strided Convolution

Stride is used to define step size for kernel function while traversing the image. It is used to decrease the size of features. If image size is H1 X W1 X D, N filters with the size of

H2 X W2 X D, Padding is P and Stride is S, then output image size is,[10]

$$(\frac{H1 + 2P - H2}{S} + 1)X(\frac{W1 + 2P - W2}{S} + 1)XN \tag{3.5}$$

3.1.5 Pooling Layer

Pooling layer is periodically used between two CNN layers. It takes input image and applies some function. Most commonly it use max-pooling and average-pooling function to extract features. It is used to collaborate values which are geometrically close to each other and reduce size of parameters to avoid overfitting. It becomes easy to find object in every chunks of an image. Generally two types of pooling is used in CNN.

- Max Pooling: It extracts maximum values from the chunks and put it into single matrix. Stride and padding size is predefined and it remains constant during execution.
- Average Pooling: It finds average value from the chunks and put it into matrix.

Following image contains matrix with the size 4 X 4 X 1, padding=0 and strides=2. Maxpooling converts 4 X 4 X 1 size matrix into 2 X 2 X 1. [13]

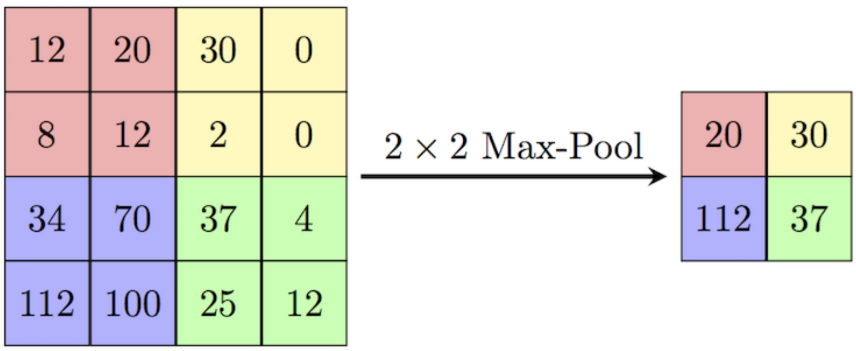


Figure 3.2: Maxpooling Layer[13]

Chapter 4

U-Net Architecture

4.1 Introduction about U-Net Architecture

U-Net architecture is implementation of fully connected CNN. It is mainly designed for medical image segmentation and detection. U-Net consist two network path Contracting path and Expensive path. Contracting path contains down sampling which is basic CNN architecture. Expensive path uses up sampling method to concate information with features from contracting path. Network between these two path is known as Bottleneck [14].

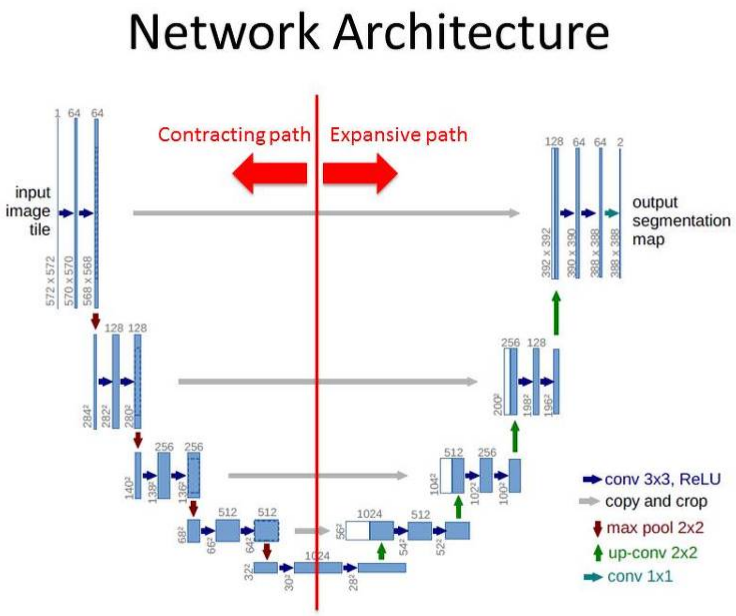


Figure 4.1: U-Net Architecture[14]

4.1.1 Contracting Path

Contracting path contains 3 blocks. Each block has 3 X 3 Convolution layer, 3 X 3 Convolution layer and 2 X 2 Maxpooling function. It is down-sampling method. Number of feature map is increase at every layer. It starts with 64, 128, 256 512 at each level.

4.1.2 Bottleneck

This part contains 2 Convolution module. It is network between Contracting and Expensive path.

4.1.3 Expensive Path

Expensive path contains 4 blocks. Each block has Deconvolution layer with stride 2, Concatenation with the feature map from Contracting path, 3 X 3 Convolution layer and 3 X 3 Convolution layer.

4.1.4 U-Net Architecture Design

U-Net consists 5 layers for down-sampling and 4 layers for up-sampling. Description of layer is given below.

- Downsampling
  - Layer 1
    - 3 X 3 Convolution Function
    - 3 X 3 Convolution Function
    - 2 X 2 Maxpooling Function
  - Layer 2
    - 3 X 3 Convolution Function
    - 3 X 3 Convolution Function
    - 2 X 2 Maxpooling Function
  - Layer 3
    - 3 X 3 Convolution Function
    - 3 X 3 Convolution Function
    - 2 X 2 Maxpooling Function

- Layer 4
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function
    * 2 X 2 Maxpooling Function
- Layer 5
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function

- Upsampling
  - Layer 1
    * 3 X 3 Deconvolution Function
    * Concatenation Function
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function
  - Layer 2
    * 3 X 3 Deconvolution Function
    * Concatenation Function
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function
  - Layer 3
    * 3 X 3 Deconvolution Function
    * Concatenation Function
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function
  - Layer 4
    * 3 X 3 Deconvolution Function
    * Concatenation Function
    * 3 X 3 Convolution Function
    * 3 X 3 Convolution Function
    * 1 X 1 Convolution Function

4.1.5 Mini U-Net Architecture

U-Net architecture has 5 layers for Down-sampling and 4 layers for Up-sampling. After 5 layers of Down-sampling U-Net generates 1024 features. To identify parameters for these much features is computationally intensive task. Another issue is, sometimes it leads to over-fitting. To resolve these issues Mini U-Net architecture is introduced here. It contains 3 layers for Down-sampling and 2 layers for Up-sampling.

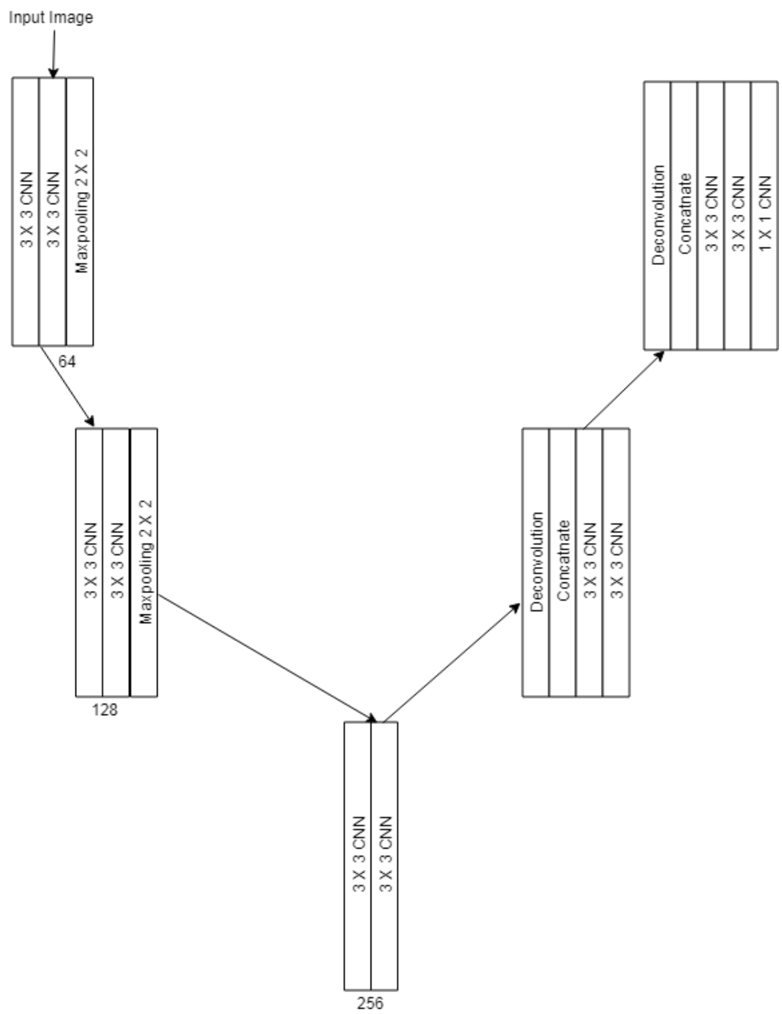


Figure 4.2: Mini U-Net Architecture

This architecture generates 256 features.

Chapter 5

Inception Network

5.1 Introduction about Inception Network

While using convolution layer, it is very difficult to decide which filter size is used to get a proper segmentation. So that issue is resolved in Inception network. Inception network uses 1 X 1, 3 X 3, 5 X 5 max-pooling layers together. Inception network reduces number of features generated by U-Net architecture. So number of parameters are decreased which finally reduce the complexity of network. Each inception layer contains three convolution function and one max-pooling function. After that results are concatenated and pass to next layer. This is naive inception model which is shown in figure 5.1. [12]

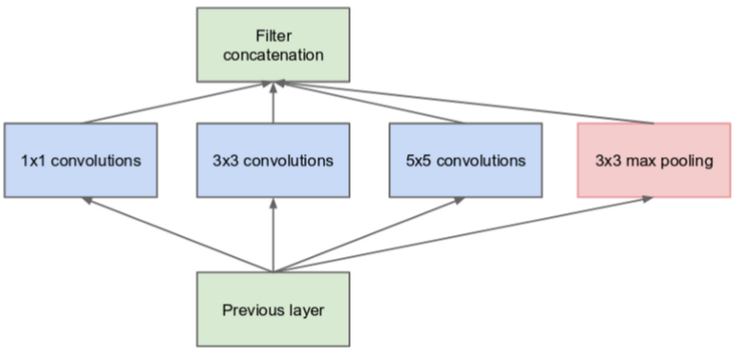


Figure 5.1: Naive Inception Network[12]

5.1.1 Dimension Reduce Inception Network

Deep neural network is computationally expensive task. To reduce the complexity and number of features extra 1 X 1 convolution is added before 3 X 3 and 5 X 5 convolution and max-pooling. Here 1 X 1 convolution function is bottleneck layer. That inception is known as dimension reduce inception network.[12]

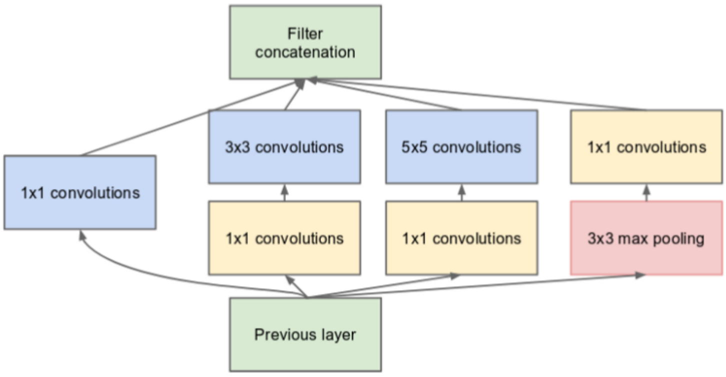


Figure 5.2: Inception Network[12]

5.1.2 GoogLeNet

Using dimension reduce inception network artificial neural network is developed which is known as GoogLeNet. It contains 9 inception module in stack manner. It is pretty deep classifier. [12]

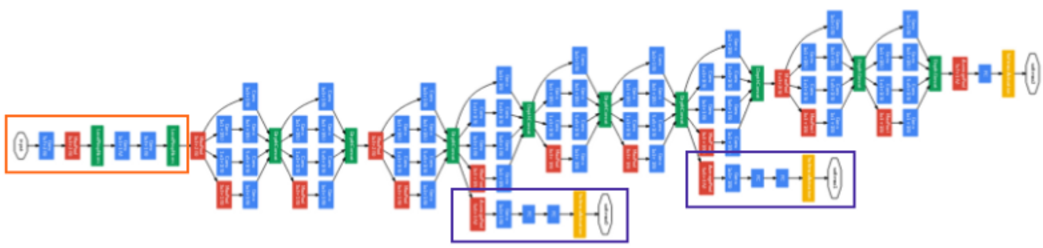


Figure 5.3: GoogLeNet[12]

5.1.3 Inception Network in Mini U-Net Architecture

Following image contains inception module implementation in Mini U-Net architecture. Each convolution layer is replaced with one inception module in down-sampling. Up-sampling is constant.

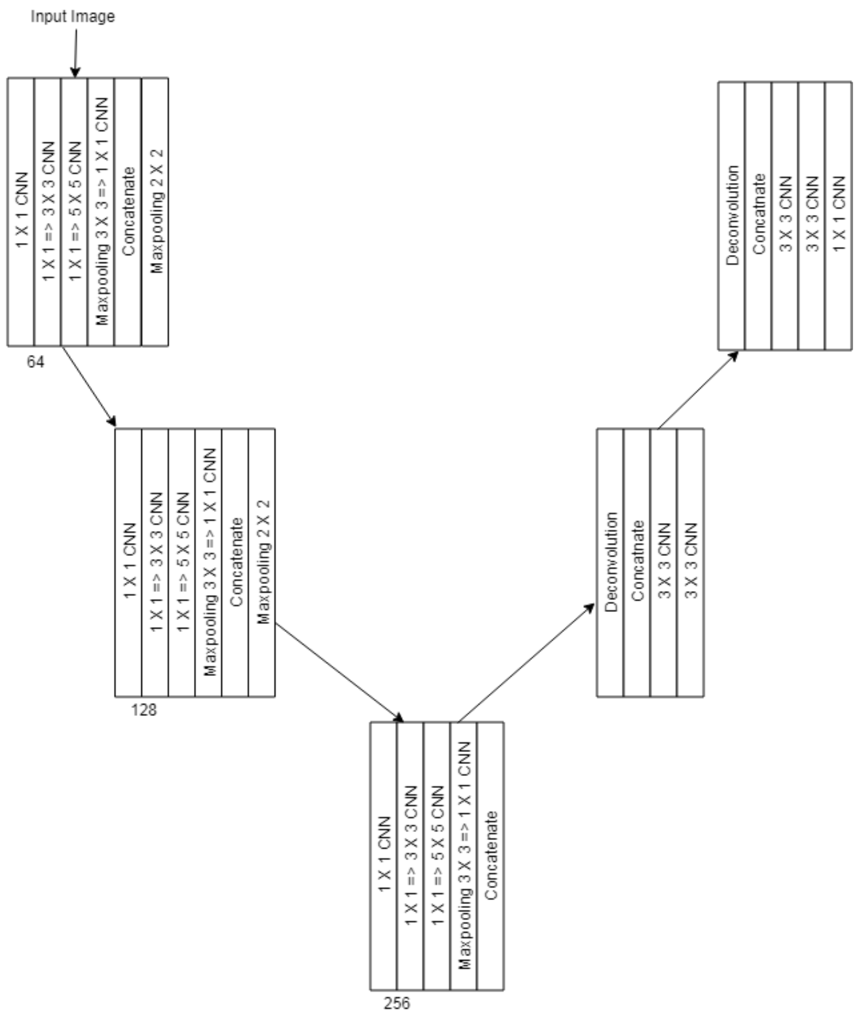


Figure 5.4: Inception Network in Mini U-Net Architecture

Inception in Mini U-Net consists 3 layers for down-sampling and 2 layers for up-sampling. Here between every 2 layer of inception, there is 2 X 2 Maxpooling layer is available to reduce size of image. Description of layer is given below.

- Downsampling
  - Layer 1
    - 1 X 1 Convolution Function
    - 1 X 1 Convolution Function - 3 X 3 Convolution Function
    - 1 X 1 Convolution Function - 5 X 5 Convolution Function
    - 3 X 3 Maxpooling Function - 1 X 1 Convolution Function
    - Concatenation Function (All above function's output)
    - 2 X 2 Maxpooling Function

- Layer 2
  - 1 X 1 Convolution Function
  - 1 X 1 Convolution Function - 3 X 3 Convolution Function
  - 1 X 1 Convolution Function - 5 X 5 Convolution Function
  - 3 X 3 Maxpooling Function - 1 X 1 Convolution Function
  - Concatenation Function (All above function's output)
  - 2 X 2 Maxpooling Function
- Layer 3
  - 1 X 1 Convolution Function
  - 1 X 1 Convolution Function - 3 X 3 Convolution Function
  - 1 X 1 Convolution Function - 5 X 5 Convolution Function
  - 3 X 3 Maxpooling Function - 1 X 1 Convolution Function
  - Concatenation Function (All above function's output)
  - 2 X 2 Maxpooling Function

- Upsampling
  - Layer 1
    - 3 X 3 Deconvolution Function
    - Concatenation Function
    - 3 X 3 Convolution Function
    - 3 X 3 Convolution Function
  - Layer 2
    - 3 X 3 Deconvolution Function
    - Concatenation Function
    - 3 X 3 Convolution Function
    - 3 X 3 Convolution Function
    - 1 X 1 Convolution Function

Chapter 6

Implementation of U-Net Architecture

6.1 U-Net Implementation

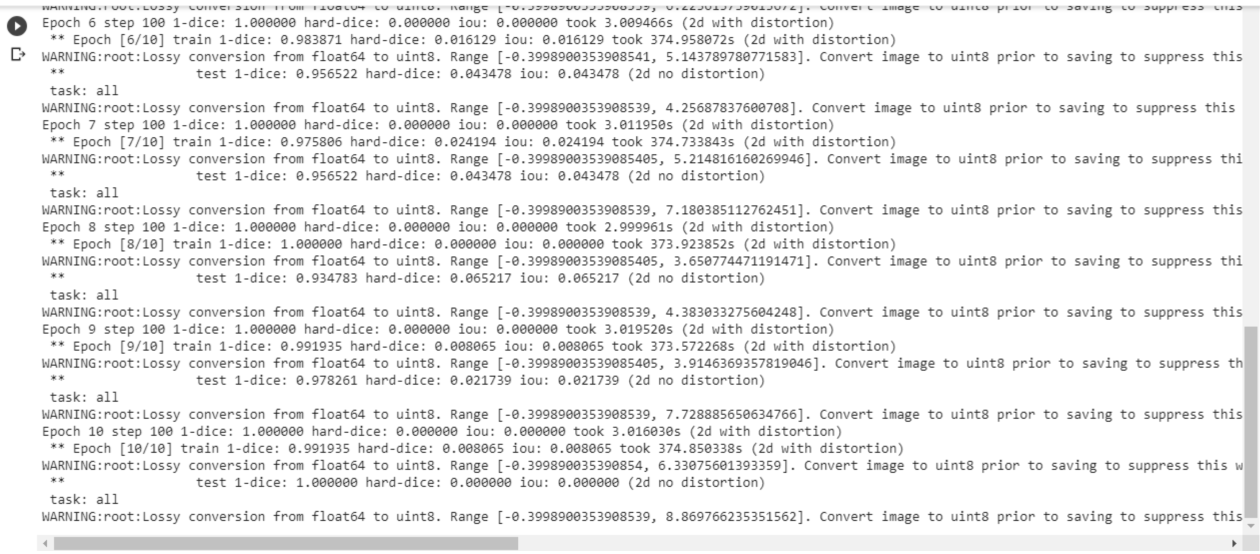
Following figures show implementation of U-Net architecture, execution and output.

```python
def u_net(x, is_train=False, reuse=False, n_out=1):
    _, nx, ny, nz = x.get_shape().as_list()
    with tf.variable_scope("u_net", reuse=reuse):
        tl.layers.set_name_reuse(reuse)
        inputs = InputLayer(x, name='inputs')
        conv1 = Conv2d(inputs, 64, (3, 3), act=tf.nn.relu, name='conv1_1')
        conv1 = Conv2d(conv1, 64, (3, 3), act=tf.nn.relu, name='conv1_2')
        pool1 = MaxPool2d(conv1, (2, 2), name='pool1')
        conv2 = Conv2d(pool1, 128, (3, 3), act=tf.nn.relu, name='conv2_1')
        conv2 = Conv2d(conv2, 128, (3, 3), act=tf.nn.relu, name='conv2_2')
        pool2 = MaxPool2d(conv2, (2, 2), name='pool2')
        conv3 = Conv2d(pool2, 256, (3, 3), act=tf.nn.relu, name='conv3_1')
        conv3 = Conv2d(conv3, 256, (3, 3), act=tf.nn.relu, name='conv3_2')
        pool3 = MaxPool2d(conv3, (2, 2), name='pool3')
        conv4 = Conv2d(pool3, 512, (3, 3), act=tf.nn.relu, name='conv4_1')
        conv4 = Conv2d(conv4, 512, (3, 3), act=tf.nn.relu, name='conv4_2')
        pool4 = MaxPool2d(conv4, (2, 2), name='pool4')
        conv5 = Conv2d(pool4, 1024, (3, 3), act=tf.nn.relu, name='conv5_1')
        conv5 = Conv2d(conv5, 1024, (3, 3), act=tf.nn.relu, name='conv5_2')

        up4 = DeConv2d(conv5, 512, (3, 3), (nx/8, ny/8), (2, 2), name='deconv4')
        up4 = ConcatLayer([up4, conv4], 3, name='concat4')
        conv4 = Conv2d(up4, 512, (3, 3), act=tf.nn.relu, name='uconv4_1')
        conv4 = Conv2d(conv4, 512, (3, 3), act=tf.nn.relu, name='uconv4_2')
        up3 = DeConv2d(conv4, 256, (3, 3), (nx/4, ny/4), (2, 2), name='deconv3')
        up3 = ConcatLayer([up3, conv3], 3, name='concat3')
        conv3 = Conv2d(up3, 256, (3, 3), act=tf.nn.relu, name='uconv3_1')
        conv3 = Conv2d(conv3, 256, (3, 3), act=tf.nn.relu, name='uconv3_2')
        up2 = DeConv2d(conv3, 128, (3, 3), (nx/2, ny/2), (2, 2), name='deconv2')
        up2 = ConcatLayer([up2, conv2], 3, name='concat2')
        conv2 = Conv2d(up2, 128, (3, 3), act=tf.nn.relu,  name='uconv2_1')
        conv2 = Conv2d(conv2, 128, (3, 3), act=tf.nn.relu, name='uconv2_2')
        up1 = DeConv2d(conv2, 64, (3, 3), (nx/1, ny/1), (2, 2), name='deconv1')
        up1 = ConcatLayer([up1, conv1] , 3, name='concat1')
        conv1 = Conv2d(up1, 64, (3, 3), act=tf.nn.relu, name='uconv1_1')
        conv1 = Conv2d(conv1, 64, (3, 3), act=tf.nn.relu, name='uconv1_2')
        conv1 = Conv2d(conv1, n_out, (1, 1), act=tf.nn.sigmoid, name='uconv1')
    return conv1

# def u_net(x, is_train=False, reuse=False, pad='SAME', n_out=2):
```

Figure 6.1: U-Net Code



```
Epoch 6 step 100 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 3.009466s (2d with distortion)
 ** Epoch [6/10] train 1-dice: 0.983871 hard-dice: 0.016129 iou: 0.016129 took 374.958072s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908541, 5.143789780771583]. Convert image to uint8 prior to saving to suppress this
 **               test 1-dice: 0.956522 hard-dice: 0.043478 iou: 0.043478 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908539, 4.25687837600708]. Convert image to uint8 prior to saving to suppress this
Epoch 7 step 100 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 3.011950s (2d with distortion)
 ** Epoch [7/10] train 1-dice: 0.975806 hard-dice: 0.024194 iou: 0.024194 took 374.733843s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.39989003539085405, 5.214816160269946]. Convert image to uint8 prior to saving to suppress thi
 **               test 1-dice: 0.956522 hard-dice: 0.043478 iou: 0.043478 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908539, 7.180385112762451]. Convert image to uint8 prior to saving to suppress this
Epoch 8 step 100 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 2.999961s (2d with distortion)
 ** Epoch [8/10] train 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 373.923852s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.39989003539085405, 3.650774471191471]. Convert image to uint8 prior to saving to suppress thi
 **               test 1-dice: 0.934783 hard-dice: 0.065217 iou: 0.065217 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908539, 4.383033275604248]. Convert image to uint8 prior to saving to suppress this
Epoch 9 step 100 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 3.019520s (2d with distortion)
 ** Epoch [9/10] train 1-dice: 0.991935 hard-dice: 0.008065 iou: 0.008065 took 373.572268s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.39989003539085405, 3.9146369357819046]. Convert image to uint8 prior to saving to suppress th
 **               test 1-dice: 0.978261 hard-dice: 0.021739 iou: 0.021739 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908539, 7.728885650634766]. Convert image to uint8 prior to saving to suppress this
Epoch 10 step 100 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 took 3.016030s (2d with distortion)
 ** Epoch [10/10] train 1-dice: 0.991935 hard-dice: 0.008065 iou: 0.008065 took 374.850338s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.399890035390854, 6.33075601393359]. Convert image to uint8 prior to saving to suppress this w
 **               test 1-dice: 1.000000 hard-dice: 0.000000 iou: 0.000000 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3998900353908539, 8.869766235351562]. Convert image to uint8 prior to saving to suppress this
```

Figure 6.2: U-Net Execution

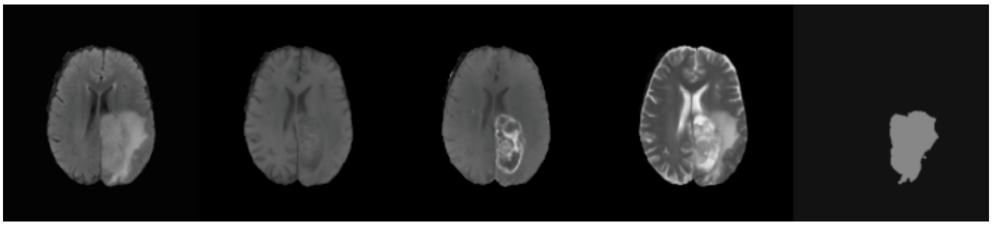


Figure 6.3: U-Net Output

6.2 Implementation of Inception Network

Following figures show implementation of Inception network in Mini U-Net architecture, execution and output. Figure 6.4 shows inception model inside the U-Net architecture. Between two layer of inception one 2 X 2 max-pool layer is available. Down-sampling is same. Definition of each inception layer is available in 6.5. Each function contains architecture which is explained earlier. Implementation also contains method to find accuracy for prediction. 6.8 shows 100 epocs to train model and dice co-efficient for predicted value.

```
def u_net(x, is_train=False, reuse=False, n_out=1):
    _, nx, ny, nz = x.get_shape().as_list()
    with tf.variable_scope("u_net", reuse=reuse):
        tl.layers.set_name_reuse(reuse)
        inputs = InputLayer(x, name='inputs')
        inception1=inception_model1(inputs,64)
        pool1 = MaxPool2d(inception1, (2, 2), strides=(1,1), padding='SAME', name='pool1')
        inception2=inception_model2(pool1,128)
        pool2 = MaxPool2d(inception2, (2, 2), strides=(1,1), padding='SAME', name='pool2')
        inception3=inception_model3(pool2,256)

        up2 = DeConv2d(inception3, 128, (3, 3), (1, 1), name='deconv2')
        up2 = ConcatLayer([up2, inception2], 3, name='concat2')
        conv2 = Conv2d(up2, 128, (3, 3), act=tf.nn.relu,  name='uconv2_1')
        conv2 = Conv2d(conv2, 128, (3, 3), act=tf.nn.relu, name='uconv2_2')
        up1 = DeConv2d(conv2, 64, (3, 3), (1, 1), name='deconv1')
        up1 = ConcatLayer([up1, inception1], 3, name='concat1')
        conv1 = Conv2d(up1, 64, (3, 3), act=tf.nn.relu, name='uconv1_1')
        conv1 = Conv2d(conv1, 64, (3, 3), act=tf.nn.relu, name='uconv1_2')
        conv1 = Conv2d(conv1, n_out, (1, 1), act=tf.nn.sigmoid, name='uconv1')
    return conv1
```

Figure 6.4: Inception Implementation

```
def inception_model1(input,size):
    inception1_1_1=Conv2d(input,(size),(1,1),strides=(1,1),act=tf.nn.relu, name='conv1_1_1')

    inception1_2_1=Conv2d(input,(size),(1,1),padding='SAME',act=tf.nn.relu, name='conv1_2_1')
    inception1_2_3=Conv2d(inception1_2_1,(size),(3,3),strides=(1,1),padding='SAME',act=tf.nn.relu, name='conv1_2_3')

    inception1_3_1=Conv2d(input,(size),(1,1),padding='SAME',act=tf.nn.relu, name='conv1_3_1')
    inception1_3_3=Conv2d(inception1_3_1,(size),(5,5),strides=(1,1),padding='SAME',act=tf.nn.relu, name='conv1_3_3')

    maxpooling1=MaxPool2d(input,(3,3), strides=(1,1), padding='SAME', name='pool1_1')
    conv1_1=Conv2d(maxpooling1,(size),(1,1),strides=(1,1),padding='SAME',act=tf.nn.relu, name='maxconv1_1_1')

    concate1=ConcatLayer([inception1_1_1, inception1_2_3, inception1_3_3, conv1_1], 3, name='concat1_1')
    return concate1

def inception_model2(input,size):
    inception2_1_1=Conv2d(input,(size),(1,1),strides=(1,1),act=tf.nn.relu, name='conv2_1_1')

    inception2_2_1=Conv2d(input,(size),(1,1),padding='SAME',act=tf.nn.relu, name='conv2_2_1')
    inception2_2_3=Conv2d(inception2_2_1,(size),(3,3),strides=(1,1),padding='SAME',act=tf.nn.relu, name='conv2_2_3')

    inception2_3_1=Conv2d(input,(size),(1,1),padding='SAME',act=tf.nn.relu, name='conv2_3_1')
    inception2_3_3=Conv2d(inception2_3_1,(size),(5,5),strides=(1,1),padding='SAME',act=tf.nn.relu, name='conv2_3_3')

    maxpooling2=MaxPool2d(input,(3,3), strides=(1,1), padding='SAME', name='pool2_1')
    conv2_1=Conv2d(maxpooling2,(size),(1,1),strides=(1,1),padding='SAME',act=tf.nn.relu, name='maxconv2_1_1')

    concate2=ConcatLayer([inception2_1_1, inception2_2_3, inception2_3_3, conv2_1], 3, name='concat2_1')
    return concate2
```

Figure 6.5: Inception Implementation

```
Epoch 28 step 300 1-dice: 0.029291 hard-dice: 0.937792 iou: 0.882870 took 1.068373s (2d with distortion)
Epoch 28 step 400 1-dice: 0.044008 hard-dice: 0.929128 iou: 0.867637 took 1.038202s (2d with distortion)
Epoch 28 step 500 1-dice: 0.041614 hard-dice: 0.923102 iou: 0.857186 took 1.037458s (2d with distortion)
Epoch 28 step 600 1-dice: 0.028145 hard-dice: 0.934291 iou: 0.876684 took 1.039185s (2d with distortion)
Epoch 28 step 700 1-dice: 0.035301 hard-dice: 0.934495 iou: 0.877044 took 1.072124s (2d with distortion)
Epoch 28 step 800 1-dice: 0.201446 hard-dice: 0.722969 iou: 0.566132 took 1.059175s (2d with distortion)
Epoch 28 step 900 1-dice: 0.018003 hard-dice: 0.950453 iou: 0.905584 took 1.046387s (2d with distortion)
Epoch 28 step 1000 1-dice: 0.084693 hard-dice: 0.872340 iou: 0.773585 took 1.057204s (2d with distortion)
 ** Epoch [28/30] train 1-dice: 0.072954 hard-dice: 0.889368 iou: 0.810367 took 1539.809535s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.1831204593181611, 1.2930699742223122]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.304095 hard-dice: 0.651206 iou: 0.513046 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.183120459318161, 1.1169229745864868]. Convert image to uint8 prior to saving to suppress this warning.
Epoch 29 step 100 1-dice: 0.051508 hard-dice: 0.917335 iou: 0.847293 took 1.049278s (2d with distortion)
Epoch 29 step 200 1-dice: 0.063754 hard-dice: 0.889714 iou: 0.801338 took 1.050601s (2d with distortion)
Epoch 29 step 300 1-dice: 0.032095 hard-dice: 0.933938 iou: 0.876064 took 1.063587s (2d with distortion)
Epoch 29 step 400 1-dice: 0.044658 hard-dice: 0.923381 iou: 0.857667 took 1.043298s (2d with distortion)
Epoch 29 step 500 1-dice: 0.050451 hard-dice: 0.899797 iou: 0.817847 took 1.051616s (2d with distortion)
Epoch 29 step 600 1-dice: 0.022127 hard-dice: 0.950012 iou: 0.904784 took 1.067438s (2d with distortion)
Epoch 29 step 700 1-dice: 0.408593 hard-dice: 0.635037 iou: 0.465241 took 1.051144s (2d with distortion)
Epoch 29 step 800 1-dice: 0.104715 hard-dice: 0.858154 iou: 0.751550 took 1.045000s (2d with distortion)
Epoch 29 step 900 1-dice: 0.041646 hard-dice: 0.914310 iou: 0.842146 took 1.041014s (2d with distortion)
Epoch 29 step 1000 1-dice: 0.084431 hard-dice: 0.863603 iou: 0.759948 took 1.063796s (2d with distortion)
 ** Epoch [29/30] train 1-dice: 0.073827 hard-dice: 0.886522 iou: 0.806913 took 1544.096103s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.1831204593181611, 4.234838485396721]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.273730 hard-dice: 0.688187 iou: 0.554385 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.183120459318161, 2.2723281383514404]. Convert image to uint8 prior to saving to suppress this warning.
Epoch 30 step 100 1-dice: 0.050373 hard-dice: 0.904411 iou: 0.825503 took 1.068631s (2d with distortion)
Epoch 30 step 200 1-dice: 0.612223 hard-dice: 0.228883 iou: 0.129231 took 1.067773s (2d with distortion)
Epoch 30 step 300 1-dice: 0.054584 hard-dice: 0.905538 iou: 0.827383 took 1.059849s (2d with distortion)
Epoch 30 step 400 1-dice: 0.061839 hard-dice: 0.888555 iou: 0.799459 took 1.077518s (2d with distortion)
Epoch 30 step 500 1-dice: 0.049993 hard-dice: 0.918673 iou: 0.849580 took 1.034741s (2d with distortion)
Epoch 30 step 600 1-dice: 0.019908 hard-dice: 0.955248 iou: 0.914330 took 1.039684s (2d with distortion)
Epoch 30 step 700 1-dice: 0.151548 hard-dice: 0.764808 iou: 0.619181 took 1.050097s (2d with distortion)
Epoch 30 step 800 1-dice: 0.067969 hard-dice: 0.884973 iou: 0.793679 took 1.040563s (2d with distortion)
Epoch 30 step 900 1-dice: 0.031631 hard-dice: 0.932815 iou: 0.874090 took 1.066303s (2d with distortion)
Epoch 30 step 1000 1-dice: 0.039958 hard-dice: 0.918799 iou: 0.849794 took 1.065421s (2d with distortion)
 ** Epoch [30/30] train 1-dice: 0.066452 hard-dice: 0.893274 iou: 0.818007 took 1540.275133s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.18312045931816112, 1.9025724331866813]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.258468 hard-dice: 0.707703 iou: 0.576074 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.183120459318161, 3.487290143966675]. Convert image to uint8 prior to saving to suppress this warning.
mehulr@mehulr-HP:~/u-net$
```

Figure 6.6: Inception Output

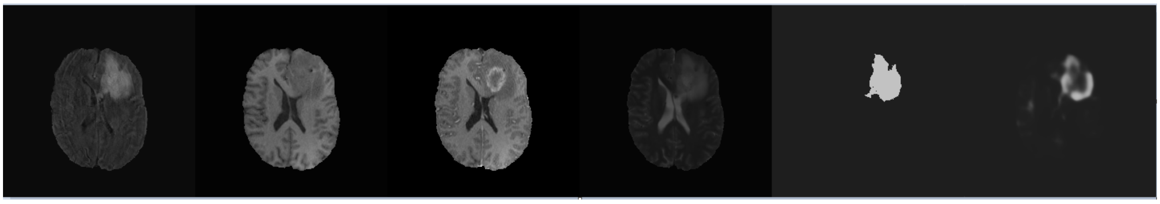


Figure 6.7: Inception Output

```
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3844873607158661, 3.790135622024536]. Convert image to uint8 prior to saving to suppress this warning.
 ** Epoch [97/100] train 1-dice: 0.870968 hard-dice: 0.129032 iou: 0.129032 took 89.406882s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.38448736071586626, 3.014359101406046]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.854839 hard-dice: 0.145161 iou: 0.145161 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3844873607158661, 1.9971837997436523]. Convert image to uint8 prior to saving to suppress this warning.
 ** Epoch [98/100] train 1-dice: 0.870968 hard-dice: 0.129032 iou: 0.129032 took 88.717803s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.38448736071586626, 2.630432604121291]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.854839 hard-dice: 0.145161 iou: 0.145161 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3844873607158661, 3.8454248905181885]. Convert image to uint8 prior to saving to suppress this warning.
 ** Epoch [99/100] train 1-dice: 0.903226 hard-dice: 0.096774 iou: 0.096774 took 88.620923s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.38448736071586626, 4.017180501194413]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.903226 hard-dice: 0.096774 iou: 0.096774 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3844873607158661, 5.614681243896484]. Convert image to uint8 prior to saving to suppress this warning.
 ** Epoch [100/100] train 1-dice: 0.838710 hard-dice: 0.161290 iou: 0.161290 took 88.608794s (2d with distortion)
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.38448736071586626, 3.6262858205415647]. Convert image to uint8 prior to saving to suppress this warning.
 **                test 1-dice: 0.822581 hard-dice: 0.177419 iou: 0.177419 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float64 to uint8. Range [-0.3844873607158661, 3.624267816543579]. Convert image to uint8 prior to saving to suppress this warning.
For Single Patient:
 **                test 1-dice: 0.881720 hard-dice: 0.118280 iou: 0.118280 (2d no distortion)
 task: all
WARNING:root:Lossy conversion from float32 to uint8. Range [0.0, 1279.0]. Convert image to uint8 prior to saving to suppress this warning.
```

Figure 6.8: Inception Network Accuracy for a Patient

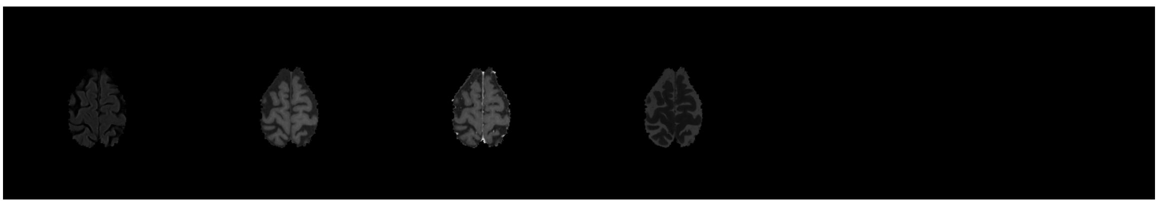


Figure 6.9: Inception Network Output for a Patient

Chapter 7

Conclusion

Brain is a very rigid structure inside the human body. Abnormal growth can damage central nervous system. Early detection can increase the chance of survival. Automatic tumor detection system can be more efficient in terms of time and effort. Using more training data set with deep learning, more accurate system can be developed. Convolution neural network is specially designed for image processing. MRI images of brain tumor can be analyzed by trained CNN and detect tumor. CNN uses different size of filters to traverse image and fetch features. These features are used to detect object from images. Different size of filters generate different number of features. U-Net architecture is commonly used for bio-medical image processing. Current architecture is computationally very intensive and it generates large number of features which leads to over-fitting. To overcome these issues, mini U-Net architecture is introduced which reduce number of features. Inception model applies different possible size of filters and concatenate results to get more accurate result. Proposed model reduced complexity of network and provided good accuracy which is necessary to make a good brain tumor detection system.

Bibliography

[1] Ms. Rupal R. Agravat, Dr. Mehul S. Raval, Deep Learning for Automated Brain Tumor Segmentation in MRI Images, School of Engineering and Applied Science, Ahmedabad University, 2017.

[2] Ms. Rupal R. Agravat, Dr. Mehul S. Raval, Brain Tumor Segmentation Towards a better life, Computer Societly of India Communications, Volume No. 40, Issue No. 9, December 2016.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, Computer Science Department and BIOSS Centre for Biological Signalling Studies, 2015.

[4] Srgio Pereira*, Adriano Pinto, Victor Alves, and Carlos A. Silva*, Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images, IEEE TRANSACTIONS ON MEDICAL IMAGING, VOL. 35, NO. 5, MAY 2016.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going Deeper with Convolutions, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[6] C. Anitha, S. Gowsalya, Brain Tumor Segmentation using Convolutional Neural Networks in MRI Images, International Journal of Computational Research and Development, Volume 1, Issue 2, 2016.

[7] Deepika R., Mr. J.Koeski Rajan, Brain Tumor Segmentation using Convolutional Neural Networks in MRI Images, International Conference on Energy Efficient Technologies for SustainabilityICEETS18, 2018.

[8] Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, Klaus H., Maier-Hein, Brain Tumor Segmentation and Radiomics Survival Prediction: Contribution to the BraTS 2017 Challenge, Springer, 2018.

[9] Hao Dong, Guang Yang, Fangde Liu, Yuanhan Mo, Yike Guo, Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks , Springer, 2017.

[10] Convolutional Neural Networks,
`https://www.coursera.org/learn/convolutional-neural-networks`

[11] Convolutional Neural Networks for Visual Recognition,
`http://cs231n.github.io/convolutional-networks/`

[12] A Simple Guide to the Versions of the Inception Network,
`https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202`

[13] An intuitive guide to Convolutional Neural Networks,
`https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050`

[14] U-Net,
`http://deeplearning.net/tutorial/unet.html`