“Brain Tumor Detection and Segmentation”

By

Karthik Bala Manokaran

This project is submitted to the Gannon University graduate faculty in partial fulfillment for the degree Master of Science in the Department of Computer and Information Science.

Major: Data Science

Approved:

Dr. Richard Matovu, Ph.D. Main Supervisor’s name	*Dr. Md Tajmilur Rahman, Ph.D.* 2nd Supervisor’s name
Dr. Joshua Nwokeji, Ph.D. 3rd Supervisor’s name	*Dr. Mei-Huei Tang, Ph.D.* Chair, Computer & Information Science Department

Gannon University

Erie, Pennsylvania 16541

December 2022

Acknowledgements

I would like to thank various professors and organizations for their steady support throughout my graduate studies. Firstly, I want to thank my supervisor, Professor Dr. Richard Matovu, for his insightful comments, helpful information, practical advice, and exploring ideas, all of which have aided me throughout my project and the writing of this report. I was able to complete the project because of his vast knowledge, extensive experience, and professional expertise in Software Development. This endeavor would not have been possible without his help and supervision. Aside from my adviser, I would like to thank the remainder of our project committee: Prof. Dr. Md. Tajmilur Rahman and Prof. Dr. Joshua Nwokeji for their support, insights, and suggestions on how to solve the problem. I would also like to express our gratitude to Gannon University for allowing us into the graduate program. I also owe a debt of gratitude to Gannon University's Faculty of Computer and Information Science for inspiring us and providing us with valuable input, allowing us to complete the project, and earning our master's degree with excellence.

Abstract

Correct and early brain tumor identification is crucial for a successful treatment. In this project we propose an effective method for classifying and dividing brain tumor tissues using machine learning. Convolutional neural networks are used in our system for feature extraction and classification. By generating new features from the old ones via feature extraction, we lower the number of features in the dataset. Here, we use CNN to identify the different tissue types. The feature maps' spatial resolution is decreased using the pooling layer. The number of parameters required for image processing is reduced by this layer. The goal of this project is to give the radiologist and doctor a second view on the diagnosis. These methods make it very simple for specialists to undertake tumor detection. This study aims to diagnose brain tumors using MRI images. We create two parts of this process; one is classification and other one is segmentation.

Table of Contents

List Of Figures

Chapter 1

Introduction

American cancer society states that 0.7 million people living with primary cancer in United states in that there are 88,970 people were diagnosed in 2022. 61 years is average age for primary brain tumor. Primary brain tumors arise in more than 100 different varieties, each with a unique presentation, course of therapy, and prognosis. The brain is the physical part that is most important and significant. One of the most frequent reasons for brain dysfunction is a brain tumor. A tumor is nothing more than an abnormal cell growth. Brain failure occurs when the nutrients meant for healthy cells and tissues is completely consumed by expanding brain tumor cells. When abnormal brain cells first formed, a brain tumor began to develop. The two primary categories of brain tumors are benign tumors and malignant tumors. Malignant tumors can be divided into primary tumors, which begin inside the brain, and secondary tumors, or cerebrum metastatic tumors, which have spread from another location. Brain tumor symptoms vary widely and are entirely based on the location of the tumor in the brain. Headache, eye problems, nausea, convulsions, and phycological changes are some of the most typical symptoms. Usually, the headache is worse first thing in the morning and goes away with vomiting. Other symptoms include difficulty speaking, walking, or sensing. As the illness advances, consciousness may occur.

1.1 Problem Definition

A tumor may progress to cancer, a major cause of death that accounts for 11% of all fatalities globally. The global incidence of cancer is rising at an alarming rate. Considering this, early tumor detection is crucial. providing doctors with high-quality tools to help them identify tumors and their causes saving the patient's time and offering a suitable cure as soon as possible. It would be simpler to get rapid consultation as a result. The capability to recognize and classify various tumor forms is the main aim of brain tumor detection. It can identify tumors from photos and provide a positive or negative response, which might be helpful when we need to know whether a tumor is present or absent. The Convolution Neural Network Algorithm is used in this study's system to identify tumor blocks and categorize the type of tumor in MRI data from different patients.

1.2 Project Overview

The primary goal of a brain tumor detection system is to identify tumor cells in the brain by differentiating between regular and irregular pixels on MRI images using arithmetical and texture-based features. The main objective of the application is to find tumors. The construction of this application had as its main objectives the protection of endangered human life and the prompt delivery of adequate therapy. Both doctors and patients can benefit from this application. Although manual identification takes longer, it is more accurate and useful for the patient who can choose early treatment options such as prescription of medicine or therapy treatments. This program was developed to deal with those problems. It's a simple software to use.

1.3 Background

1.3.1 What is Magnetic Resonance Imaging (MRI)?

MRI is a type of medical imaging used in radiology to create images of the body's anatomy and physiological functions. To produce images of the body's organs, MRI scanners employ powerful magnetic fields, magnetic field gradients, and radio waves. The difference between MRI and CT and PET scans is that MRI does not employ X-rays or ionizing radiation. Nuclear magnetic resonance is used in medicine through MRI (NMR). In addition, NMR can be utilized for imaging in NMR spectroscopy, among other NMR applications [2]. The posterior cranial fossa, which houses the brainstem and the cerebellum, may be seen more clearly with MRI than with CT, making it the preferred investigative tool for neurological tumors. MRI is the greatest option for many central nervous system disorders, such as dementia, cerebrovascular illness, infectious diseases, Alzheimer's disease, and epilepsy, due to the contrast between the grey and white matter that it provides. Researchers are able to examine both the functional and anatomical abnormalities of the brain in psychological illnesses since the images are captured milliseconds apart and demonstrate how the brain reacts to various stimuli. In addition, guided stereotactic surgery and radiosurgery employing the N-localizer are utilized to treat brain tumors, arteriovenous malformations, and other surgically curable disorders using MRI.

1.3.2 What is Image Processing?

Image processing is a technique used to modify or enhance an image or to draw out some relevant information from it. It is a kind of signal processing where the input is an image and the output can either be another image or features or characteristics related to that image. Image processing is one of the technologies that is currently expanding quickly.

Figure 1: MRI

Within the fields of engineering and computer science as well, it serves as a primary research focus. Basically, image processing involves the following three steps:

● Importing the picture using software for image capture.

● Examining and modifying the picture.

● An output based on image analysis that may include a transformed image or report as a result.

1.3.3 What is Machine learning?

A branch of artificial intelligence known as machine learning (ML) gives computers the capacity to autonomously learn from experience and advance without explicit programming. The creation of software that can access data and utilize it to learn for itself is the main goal of machine learning. Machine learning is built on statistical analysis, which uses a variety of statistical techniques to draw conclusions from the data. The software industry utilizes ML in many different areas.

In order to find patterns in data and improve future decisions based on the test sets that we supply, the learning process starts with observations or data, such as examples, firsthand experience, or instruction.

Machine learning algorithms are classified as follows:

● In order to anticipate future events, **supervised** machine learning algorithms can apply what they have learnt in the past to fresh data using labeled examples. The learning algorithm creates an inferred function to forecast the values of the outputs starting from the examination of a known training dataset. After adequate training, the system may provide targets for any new input. Additionally, the learning algorithm can discover faults and repair them by comparing its output with the desired, appropriate result.

● When the data used to train is neither classified nor labeled, **unsupervised** machine learning algorithms are employed. Unsupervised learning investigates how systems might extrapolate a function from unlabeled data to describe a hidden structure. Although the system is unable to determine the proper output, it explores the data and can infer hidden structures from unlabeled data using datasets.

● **Semi-supervised** machine learning techniques use both labeled and unlabeled data for training, often a small quantity of labeled data and a big amount of unlabeled data, and hence fall midway between supervised and unsupervised learning. The learning accuracy of systems that employ this technique can be greatly increased. Semi-supervised learning is typically used when the collected labeled data needs knowledgeable and pertinent resources to train from. Otherwise, it usually doesn't call for extra resources to get unlabeled data.

● A learning method known as **Reinforcement** machine learning algorithms interacts with the environment by taking actions and identifying rewards or errors. The most important aspects of reinforcement learning are trial-and-error searching and delayed rewards. With the help of this technique, machines and software agents may automatically select the best course of action in a given situation to enhance performance. The reinforcement signal, or simple reward feedback, is necessary for the agent to learn which behavior is better.

Chapter 2.0

Literature Review

Today, an estimated 700,000 people in the United States are living with a primary brain tumor, and approximately 88,970 more will be diagnosed in 2022. Brain tumors can be deadly, significantly impact quality of life, and change everything for a patient and their loved ones. They do not discriminate, inflicting men, women, and children of all races and ethnicities [1]. semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as dense prediction [2]. To predict the accuracy from the model histogram features were extracted from the denoised MRI images, and then combined with patients' age information to train a prediction model for the overall survival time of the brain tumor patients. [3] Particularly for GBM, which necessitates the detection of all potentially spreading malignant regions, the proposed method can assist medical staff members such as surgeons and radiologists in making a diagnosis of brain cancer from an MRI scan. This technique uses a maximum entropy threshold in conjunction with Naive Bayes classification to identify brain tumors [4]. The superiority of metaheuristic algorithms can be supported by their applicability, or the ability to be applied to any problem that is formulated as a function optimization problem; hybridization, or the ability to combine these methods to create more reliable techniques; and flexibility, or the ability to scale these algorithms to fit the needs of the problem [5]. The input slices are enhanced and de-noised using the Weiner filter and various wavelet bands. With the help of Potential Field (PF) clustering, tumor pixel subsets are discovered. Furthermore, in Fluid Attenuated Inversion Recovery (Flair) and T2 MRI, the tumor zone is isolated using a global threshold and several mathematical morphological techniques. Local Binary Pattern (LBP) and Gabor Wavelet Transform (GWT) features are combined for precise categorization [6] [3]. The study suggests an automatic segmentation technique that uses CNNs (Convolution Neural Networks), which establish tiny 3 3 kernels. Segmentation and classification are achieved with just this one technique. CNN (a machine learning technology) is based on layer-based neural networks for results classification. The study suggests a novel method for detecting tumors based on high level features retrieved from CNNs using the Hough transform methodology. A collection of FC (completely connected) layers are used to segment the tumors that are being found, and then FCs are used to classify the segmented mask. The suggested method produces results that are guaranteed to meet the accepted benchmarks for medical images. ConvNet, also known as CNN (Convolutional Neural Network), is a deep machine learning technique used to analyze the Image [7]. Across all the technology in the field using CNN would be best possible option to complete this particular task which we took here [2] [7]. The simplest way to identify and categorize items in pictures is by hand, although this process takes time and may be inconsistent owing to human error. Modern hardware advancements like enhanced GPUs (graphics processing units) enable the execution of complicated algorithms that learn from enormous image datasets with effective extraction and generalization capabilities and, as a result, can perform extremely accurate object categorization [8]. Despite the fact that there are numerous approximate methods for dealing with CNN, most concur on the key phases required to follow the entire CNN process and obtain the desired outcome [2] [8].

Chapter 3.0

Methodology

Here the progress of our project overview given below, top one classification model which will identify the tumor present or not. Segmentation model present below which helps to identify the tumor region.

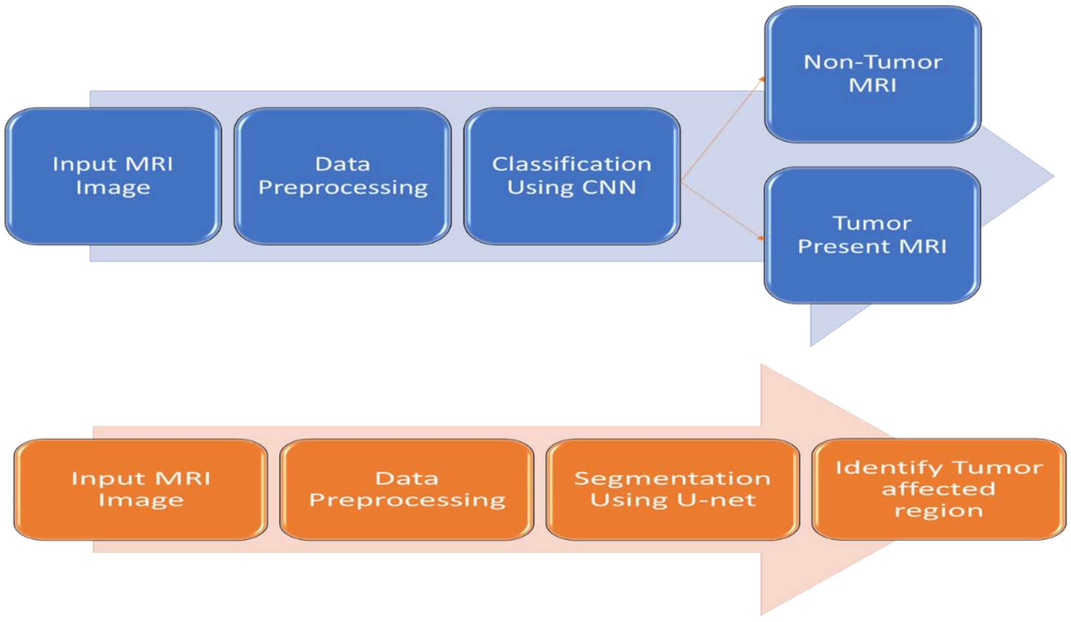
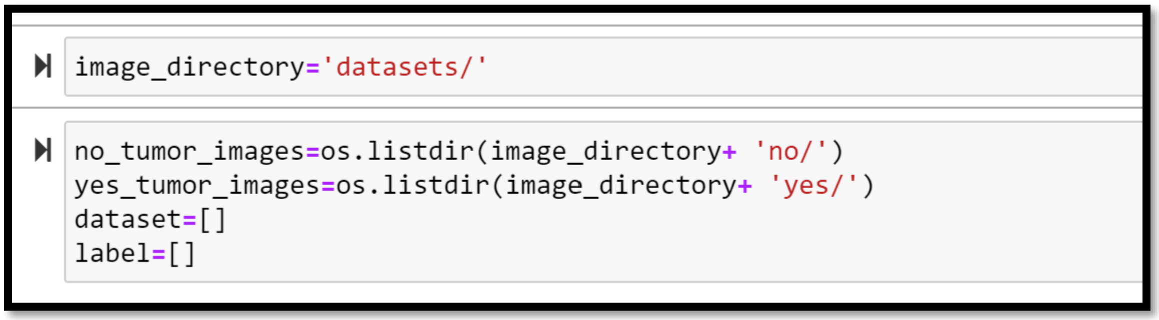


Figure 2 Methodology

3.1 Data Preprocessing for Detection:



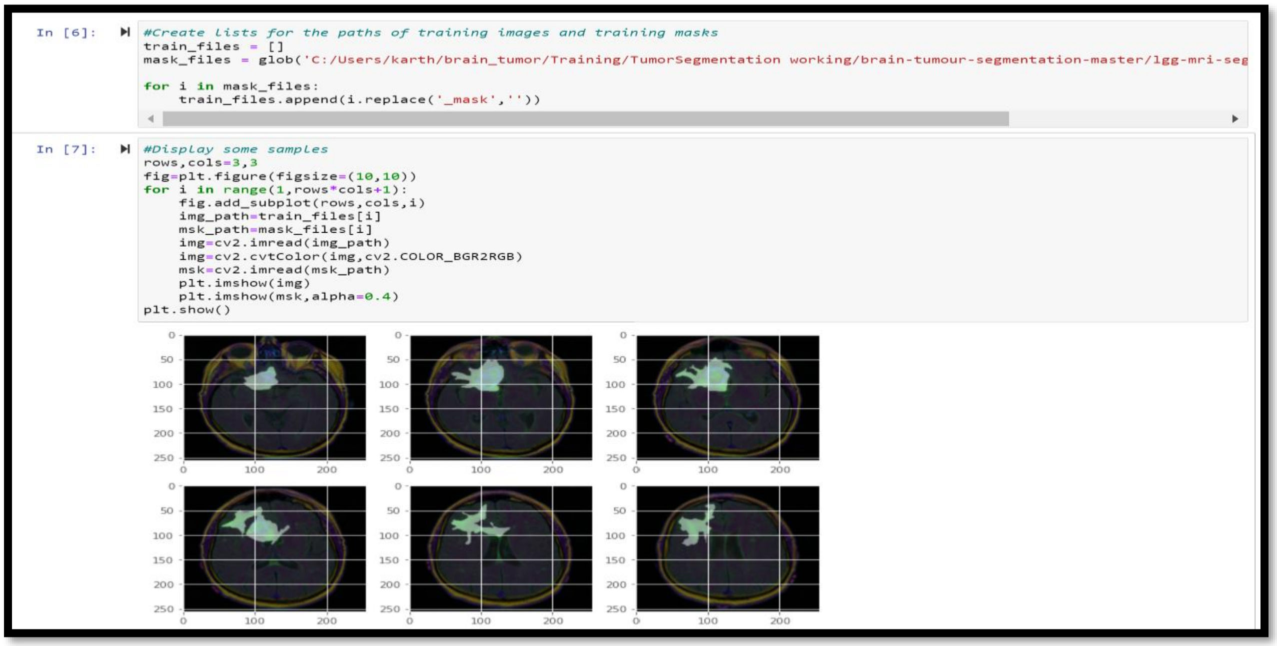
```
image_directory='datasets/'
```

```
no_tumor_images=os.listdir(image_directory+ 'no/')
yes_tumor_images=os.listdir(image_directory+ 'yes/')
dataset=[]
label=[]
```

Figure 3 Data preprocessing for detection

For dataset we are giving the label no folder as no_tumor_images , yes folder as yes_tumor_images

3.2 Data Preprocessing for Segmentation:



```
In [6]:  #Create lists for the paths of training images and training masks
         train_files = []
         mask_files = glob('C:/Users/karth/brain_tumor/Training/TumorSegmentation working/brain-tumour-segmentation-master/lgg-mri-seg

         for i in mask_files:
             train_files.append(i.replace('_mask',''))
```

```
In [7]:  #Display some samples
         rows,cols=3,3
         fig=plt.figure(figsize=(10,10))
         for i in range(1,rows*cols+1):
             fig.add_subplot(rows,cols,i)
             img_path=train_files[i]
             msk_path=mask_files[i]
             img=cv2.imread(img_path)
             img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
             msk=cv2.imread(msk_path)
             plt.imshow(img)
             plt.imshow(msk,alpha=0.4)
         plt.show()
```

Figure 4 Data preprocessing for segmentation

3.3 Libraries:

The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules. Most of the Python Libraries are written in the C programming language. The Python standard library consists of more than 200 core modules. All these work together to make Python a high-level programming language. Python Standard Library plays a very important role. Without it, the programmers can't have access to the functionalities of Python. But other than this, there are several other libraries in Python that make a programmer's life easier. Let's have a look at some of the commonly used libraries:

A) **TensorFlow**: This library was developed by Google in collaboration with the Brain Team. It is an open-source library used for high-level computations. It is also used in machine learning and deep learning algorithms. It contains a large number of tensor operations. Researchers also use this Python library to solve complex computations in Mathematics and Physics.

B) **Matplotlib**: This library is responsible for plotting numerical data. And that's why it is used in data analysis. It is also an open-source library and plots high-defined figures like pie charts, histograms, scatterplots, graphs, etc.

C) **Pandas**: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.

D) **Numpy**: The name "Numpy" stands for "Numerical Python". It is the commonly used library. It is a popular machine learning library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Even libraries like TensorFlow use Numpy internally to perform several operations on tensors. Array Interface is one of the key features of this library.

E) **Scikit-learn**: It is a famous Python library to work with complex data. Scikit-learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with Numpy and SciPy.

F) **PyTorch**: PyTorch is the largest machine learning library that optimizes tensor computations. It has rich APIs to perform tensor computations with strong GPU acceleration. It also helps to solve application issues related to neural networks.

3.4 Classification Using CNN:

Using CNN creating model that helps us to Classify the MRI Image then the final output gives the result of the MRI that has the tumor or not. About the classification model explained in chapter 4.

3.5 Segmentation Using U-Net:

Using U-Net creating model that helps us to find tumor region of the MRI Image. About the Segmentation model explained in chapter 4.

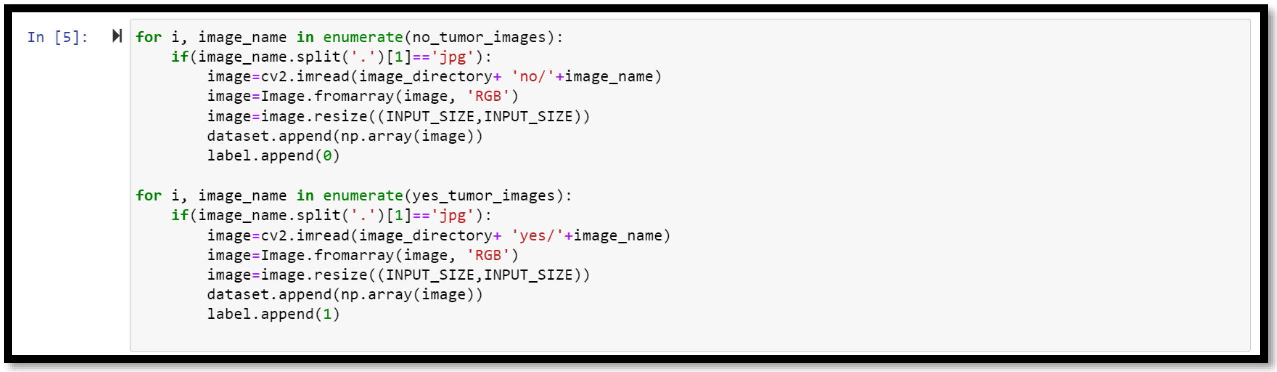
Chapter 4.0

Model Build up

4.1 Classification Model

4.1.1 Label for Classification Model

Create for loop that creates a class for classification label. Image doesn't have the tumor will be label 0 and label 1 is for the image does have the tumor.



```
In [5]:  for i, image_name in enumerate(no_tumor_images):
             if(image_name.split('.')[1]=='jpg'):
                 image=cv2.imread(image_directory+ 'no/'+image_name)
                 image=Image.fromarray(image, 'RGB')
                 image=image.resize((INPUT_SIZE,INPUT_SIZE))
                 dataset.append(np.array(image))
                 label.append(0)

         for i, image_name in enumerate(yes_tumor_images):
             if(image_name.split('.')[1]=='jpg'):
                 image=cv2.imread(image_directory+ 'yes/'+image_name)
                 image=Image.fromarray(image, 'RGB')
                 image=image.resize((INPUT_SIZE,INPUT_SIZE))
                 dataset.append(np.array(image))
                 label.append(1)
```

Figure 5 Labeling

4.1.2 Train test split for Classification Model

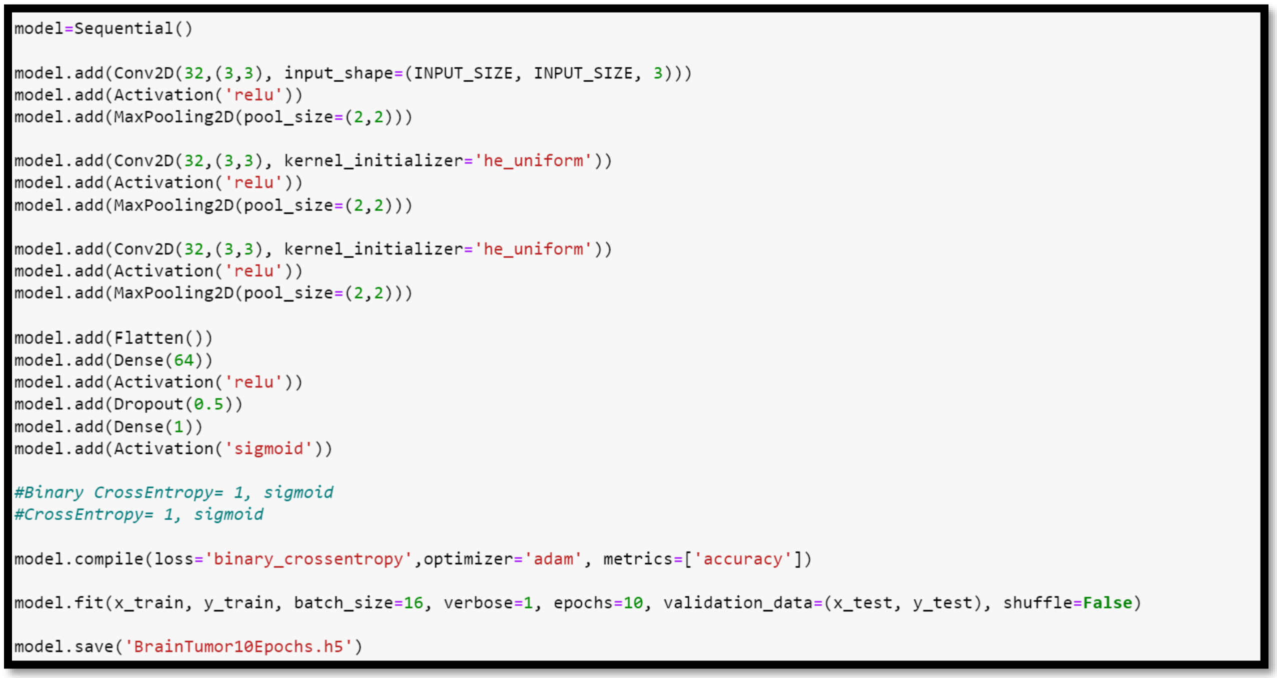
Syntax:

sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state =None, shuffle=True, stratify=None)

4.1.3 Classification Model build up

As its name suggests it is one of the models that is used to investigate varied types of neural networks where the model gets in one input as feedback and expects an output as desired. Keras API and library is incorporated with a sequential model to judge the entire simple model not the complex kind of model. It passes on the data and flows in sequential order from top to bottom approach till the data reaches at end of the model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

Generally, all layers in Keras need to know the shape of their inputs in order to be able to create their weights. It creates its weights the first time it is called on an input, since the shape of the weights depends on the shape of the inputs. Naturally, this also applies to Sequential models. When you instantiate a Sequential model without an input shape, it isn't "built": it has no weights (and calling model. Weights results in an error stating just this). The weights are created when the model first sees some input data.

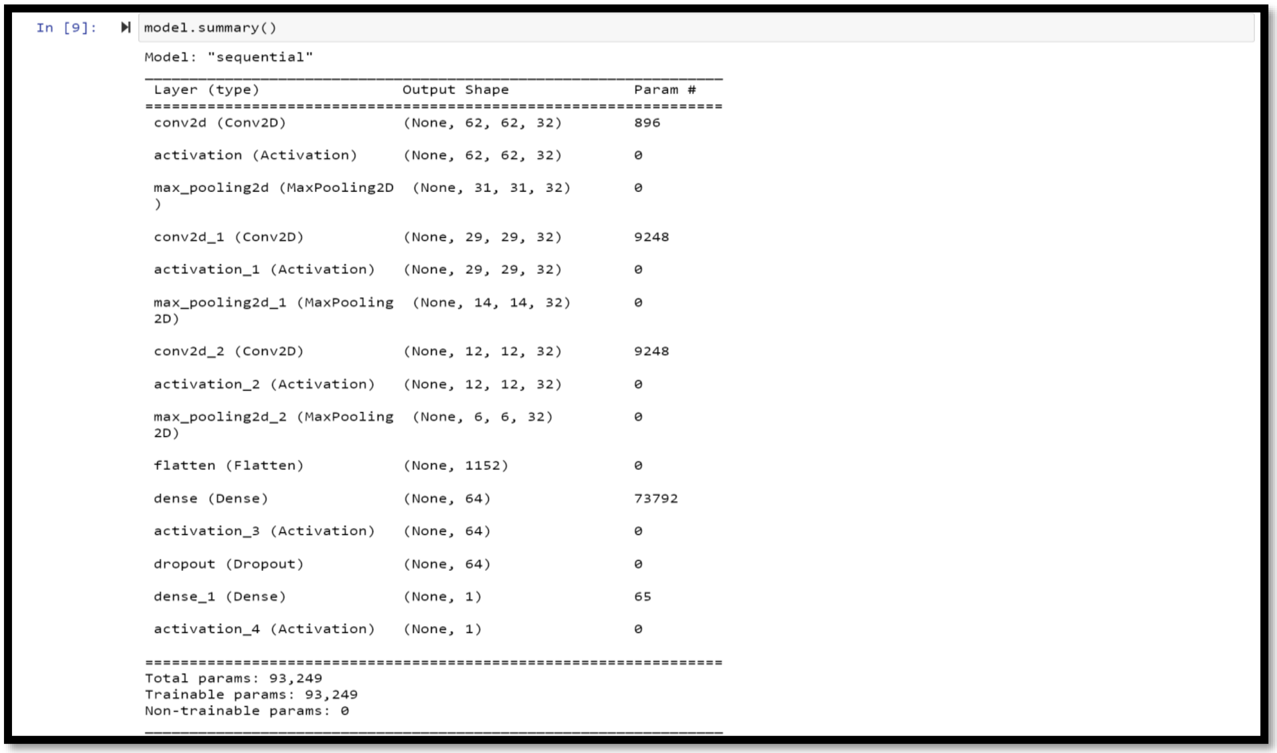


```
model=Sequential()

model.add(Conv2D(32,(3,3), input_shape=(INPUT_SIZE, INPUT_SIZE, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3), kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32,(3,3), kernel_initializer='he_uniform'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

#Binary CrossEntropy= 1, sigmoid
#CrossEntropy= 1, sigmoid

model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=16, verbose=1, epochs=10, validation_data=(x_test, y_test), shuffle=False)

model.save('BrainTumor10Epochs.h5')
```

Figure 6 Sequential model

However, it can be very useful when building a Sequential model incrementally to be able to display the summary of the model so far, including the current output shape. In this case, you should start your model by passing an Input object to your model, so that it knows its input shape from the start.

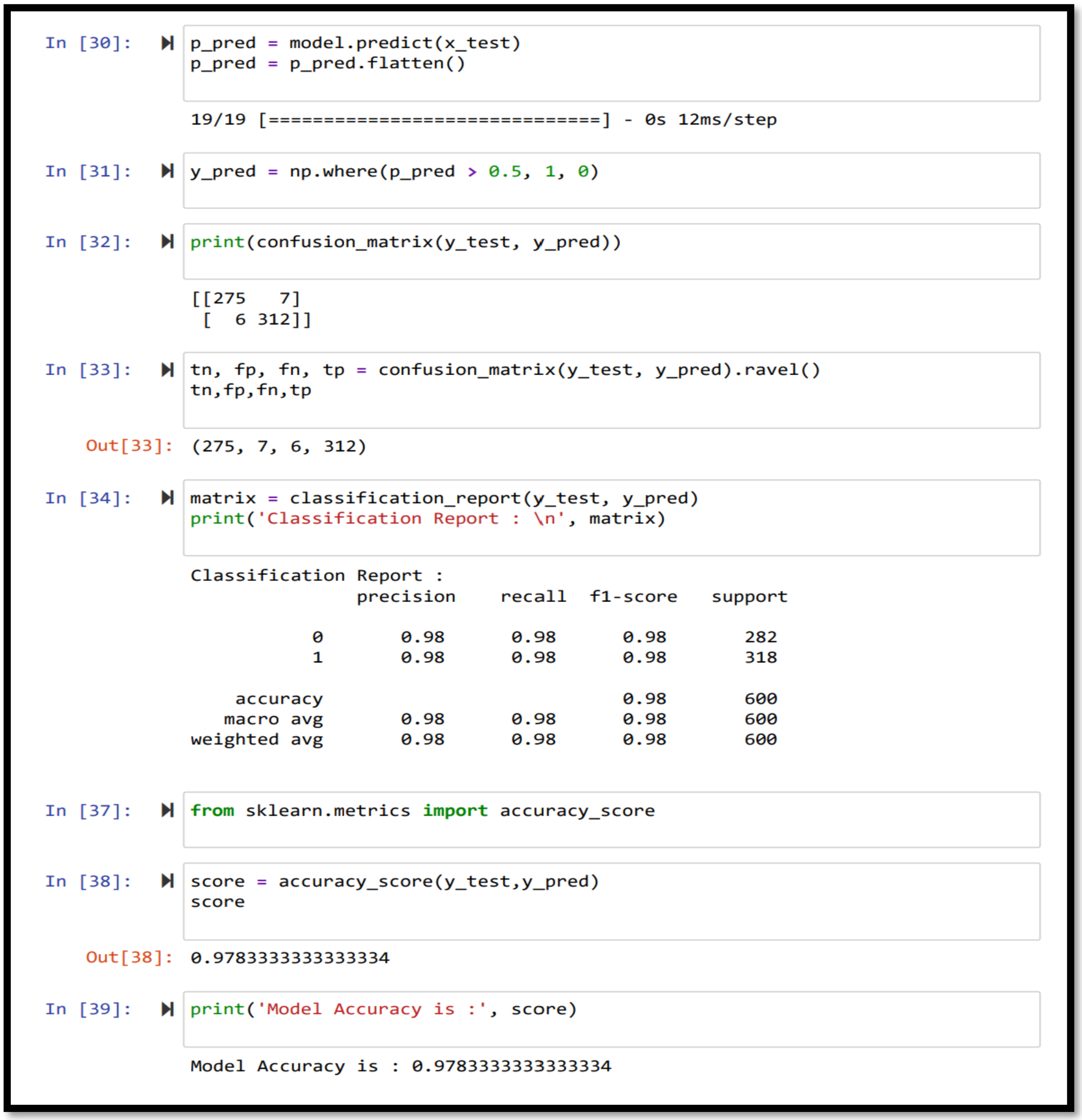
Model Summary:



```
In [9]:  model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896       
                                                                 
 activation (Activation)     (None, 62, 62, 32)        0         
                                                                 
 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248      
                                                                 
 activation_1 (Activation)   (None, 29, 29, 32)        0         
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 12, 12, 32)        9248      
                                                                 
 activation_2 (Activation)   (None, 12, 12, 32)        0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 6, 6, 32)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dense (Dense)               (None, 64)                73792     
                                                                 
 activation_3 (Activation)   (None, 64)                0         
                                                                 
 dropout (Dropout)           (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 1)                 65        
                                                                 
 activation_4 (Activation)   (None, 1)                 0         
                                                                 
=================================================================
Total params: 93,249
Trainable params: 93,249
Non-trainable params: 0
_________________________________________________________________
```

Figure 7 Sequential model summary

4.1.4 Detection Model Evaluation:



```
In [30]:    p_pred = model.predict(x_test)
            p_pred = p_pred.flatten()

            19/19 [==============================] - 0s 12ms/step

In [31]:    y_pred = np.where(p_pred > 0.5, 1, 0)

In [32]:    print(confusion_matrix(y_test, y_pred))

            [[275   7]
             [  6 312]]

In [33]:    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
            tn,fp,fn,tp

Out[33]:    (275, 7, 6, 312)

In [34]:    matrix = classification_report(y_test, y_pred)
            print('Classification Report : \n', matrix)

            Classification Report :
                           precision    recall  f1-score   support

                       0       0.98      0.98      0.98       282
                       1       0.98      0.98      0.98       318

                accuracy                           0.98       600
               macro avg       0.98      0.98      0.98       600
            weighted avg       0.98      0.98      0.98       600


In [37]:    from sklearn.metrics import accuracy_score

In [38]:    score = accuracy_score(y_test,y_pred)
            score

Out[38]:    0.9783333333333334

In [39]:    print('Model Accuracy is :', score)

            Model Accuracy is : 0.9783333333333334
```

Figure 8 Model Evaluation

Our Brain tumor detection model accuracy is 97.83 which is good on classification models.

4.2 Segmentation Model overview

4.2.1 Segmentation Model Introduction

Image segmentation is a computer vision task in which we label specific regions of an image according to what's being shown. More specifically, the goal of semantic image segmentation is to label each pixel of an image with a corresponding class of what is being represented. Because we're predicting for every pixel in the image, this task is commonly referred to as dense prediction. [2]

One important thing to note is that we're not separating instances of the same class; we only care about the category of each pixel. In other words, if you have two objects of the same category in your input image, the segmentation map does not inherently distinguish these as separate objects. There exists a different class of models, known as instance segmentation models, which do distinguish between separate objects of the same class.

Segmentation models are useful for a variety of tasks, including:

- Autonomous vehicles

We need to equip cars with the necessary perception to understand their environment so that self-driving cars can safely integrate into our existing roads.

- Medical image diagnostics

Machines can augment analysis performed by radiologists, greatly reducing the time required to run diagnostic tests. [2]

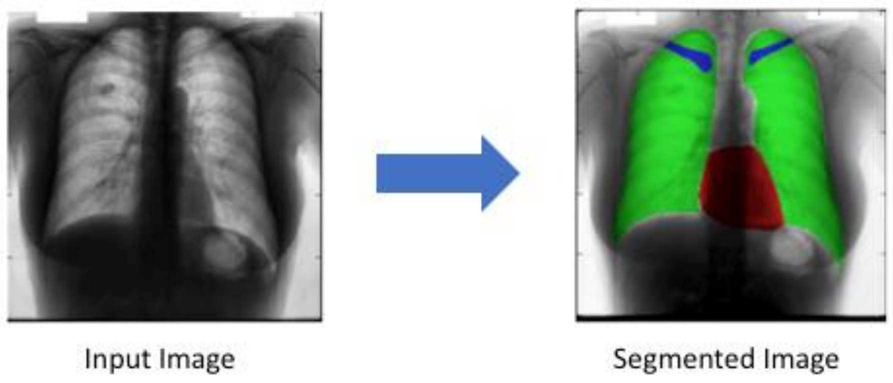


Figure 9 Medical Imaging

4.2.2 Constructing an architecture

A naive approach towards constructing a neural network architecture for this task is to simply stack a few convolutional layers (with same padding to preserve dimensions) and output a final segmentation map. This directly learns a mapping from the input image to its corresponding segmentation through the successive transformation of feature mappings; however, it's quite computationally expensive to preserve the full resolution throughout the network.

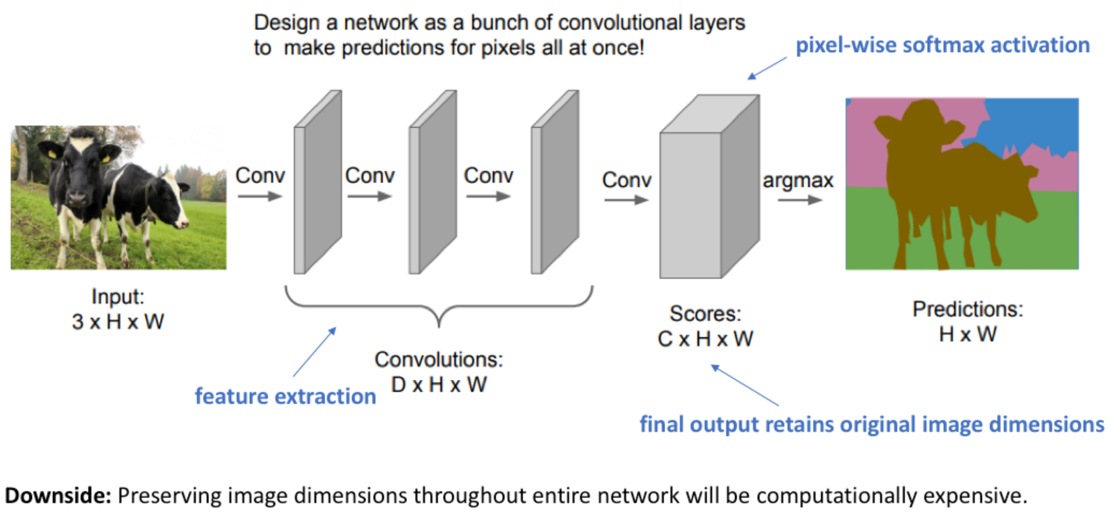
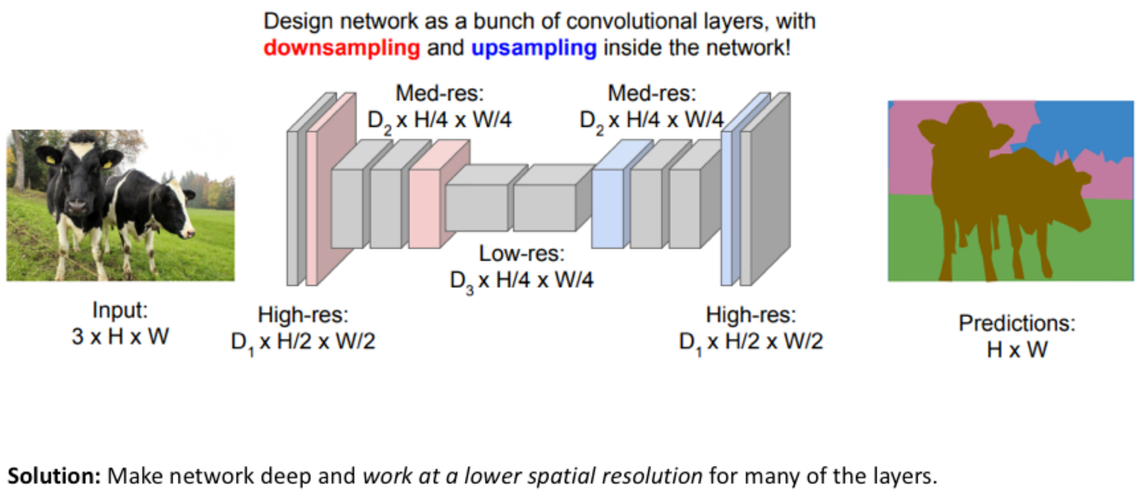

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!
pixel-wise softmax activation
Conv
Conv
Conv
Conv
argmax
Input:
3 x H x W
Convolutions:
D x H x W
feature extraction
Scores:
C x H x W
Predictions:
H x W
final output retains original image dimensions
Downside: Preserving image dimensions throughout entire network will be computationally expensive.

Figure 10 Convolutional layers prediction

Recall that for deep convolutional networks, earlier layers tend to learn low-level concepts while later layers develop more high-level (and specialized) feature mappings. In order to maintain expressiveness, we typically need to increase the number of feature maps (channels) as we get deeper in the network. [2]

This didn't necessarily pose a problem for the task of image classification, because for that task we only care about what the image contains (and not where it is located). Thus, we could alleviate computational burden by periodically downsampling our feature maps through pooling or strided convolutions (ie. compressing the spatial resolution) without concern. However, for image segmentation, we would like our model to produce a full-resolution semantic prediction.

One popular approach for image segmentation models is to follow an encoder/decoder structure where we down sample the spatial resolution of the input, developing lower-resolution feature mappings which are learned to be highly efficient at discriminating between classes, and the up sample the feature representations into a full-resolution segmentation map.


Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network!
Med-res: D_2 x H/4 x W/4
Med-res: D_2 x H/4 x W/4
Input: 3 x H x W
Low-res: D_3 x H/4 x W/4
High-res: D_1 x H/2 x W/2
High-res: D_1 x H/2 x W/2
Predictions: H x W

Solution: Make network deep and *work at a lower spatial resolution* for many of the layers.

Figure 11 Conv Downsampling and upsampling

4.2.3 Fully convolutional networks:

The approach of using a "fully convolutional" network trained end-to-end, pixels-to-pixels for the task of image segmentation was introduced by Long et al. The full network, as shown below, is trained according to a pixel-wise cross entropy loss.

However, because the encoder module reduces the resolution of the input by a factor of 32, the decoder module struggles to produce fine-grained segmentations (as shown below).

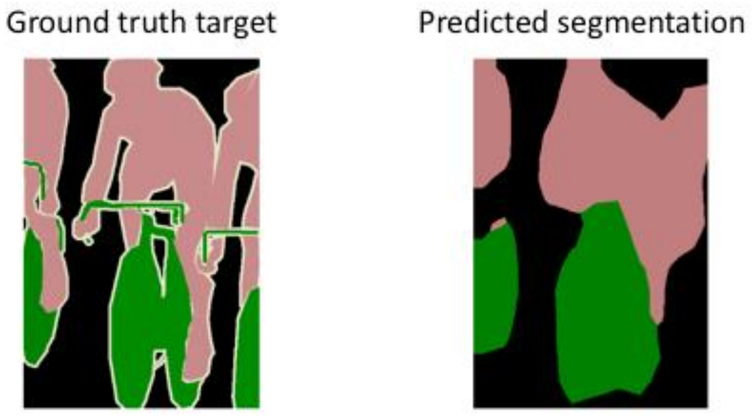


Figure 12 Grained segmentation

Semantic segmentation faces an inherent tension between semantics and location: global information resolves what while local information resolves where... Combining fine layers and coarse layers lets the model make local predictions that respect global structure. [2]

4.2.4 Adding skip connections:

Slowly up sampling (in stages) the encoded representation, adding "skip connections" from earlier layers and summing these two feature maps.

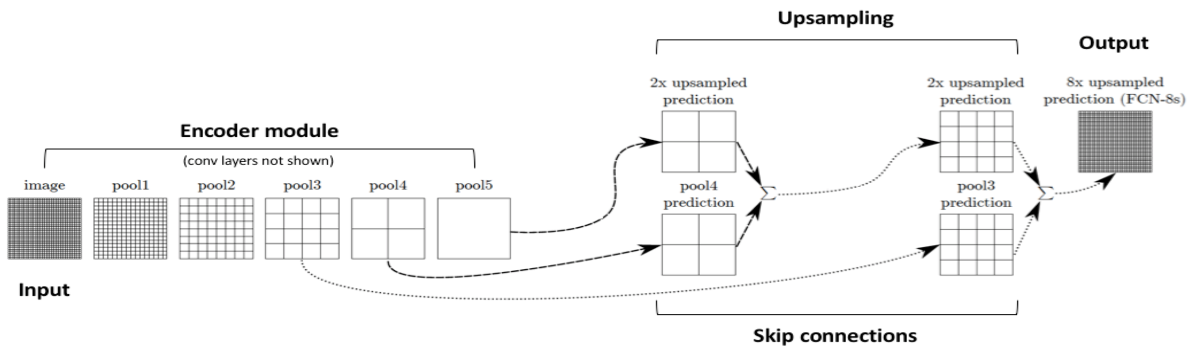


Figure 13 Skip connection addition

These skip connections from earlier layers in the network (prior to a downsampling operation) should provide the necessary detail in order to reconstruct accurate shapes for segmentation boundaries. Indeed, we can recover more fine-grain detail with the addition of these skip connections.

improve upon the "fully convolutional" architecture primarily through expanding the capacity of the decoder module of the network. More concretely, they propose the U-Net architecture which "consists of a contracting path to capture context and a symmetric expanding path that enables precise localization." This simpler architecture has grown to be very popular and has been adapted for a variety of segmentation problems.

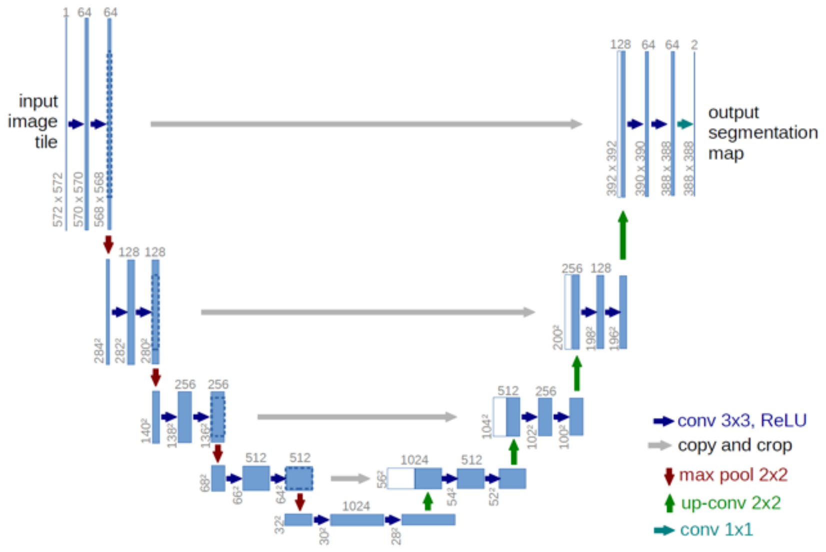


Figure 14 U-Net Architecture

data augmentation ("randomly mirroring and “jittering” the images by translating them up to 32 pixels") did not result in a noticeable improvement in performance, Ronneberger et al. (U-Net paper) credit data augmentations ("random elastic deformations of the training samples") as a key concept for learning. It appears as if the usefulness (and type) of data augmentation depends on the problem domain. **[2]**

Advanced U-Net variants

The standard U-Net model consists of a series of convolution operations for each "block" in the architecture. As I discussed in my post on common convolutional network architectures, there exist a number of more advanced "blocks" that can be substituted in for stacked convolutional layers.

Swap out the basic stacked convolution blocks in favor of **residual blocks**. This residual block introduces short skip connections (within the block) alongside the existing long skip connections (between the corresponding feature maps of encoder and decoder modules) found in the standard U-Net structure. They report that the short skip connections allow for faster convergence when training and allow for deeper models to be trained.

Expanding on this, Jegou et al. proposed the use of **dense blocks**, still following a U-Net structure, arguing that the "characteristics of DenseNets make them a very good fit for semantic segmentation as they *naturally induce skip connections and multi-scale supervision*." These dense blocks are useful as they carry low level features from previous layers directly alongside higher level features from more recent layers, allowing for highly efficient feature reuse.

One very important aspect of this architecture is the fact that the upsampling path does not have a skip connection between the input and output of a dense block. The authors note that because the "upsampling path increases the feature maps spatial resolution, the linear growth in the number of features would be too memory demanding." Thus, only the output of a dense block is passed along in the decoder module.

4.2.5 Defining a loss function:

The most used loss function for the task of image segmentation is a pixel-wise cross entropy loss. This loss examines each pixel individually, comparing the class predictions (depth-wise pixel vector) to our one-hot encoded target vector.

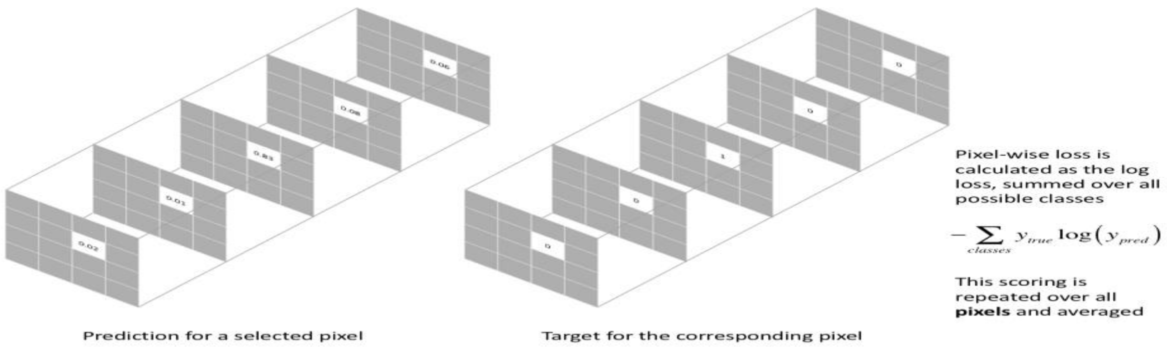

0.06
0.08
0.83
0.01
0.02
0
0
1
0
0
Pixel-wise loss is calculated as the log loss, summed over all possible classes
$-\sum_{classes} y_{true} \log(y_{pred})$
This scoring is repeated over all **pixels** and averaged
Prediction for a selected pixel
Target for the corresponding pixel

Figure 15 Targeted pixel

Because the cross entropy loss evaluates the class predictions for each pixel vector individually and then averages over all pixels, we're essentially asserting equal learning to each pixel in the image. This can be a problem if your various classes have unbalanced representation in the image, as training can be dominated by the most prevalent class. (FCN paper) discuss weighting this loss for each output channel in order to counteract a class imbalance present in the dataset.

This loss weighting scheme helped their U-Net model segment cells in biomedical images in a *discontinuous* fashion such that individual cells may be easily identified within the binary segmentation map.

Another popular loss function for image segmentation tasks is based on the Dice coefficient, which is essentially a measure of overlap between two samples. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap. The Dice coefficient was originally developed for binary data, and can be calculated as:

$$Dice = \frac{2\,|A \cap B|}{|A| + |B|}$$

Figure 16 Dice coefficient

where |A∩B| represents the common elements between sets A and B, and |A| represents the number of elements in set A (and likewise for set B).

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate |A∩B||A∩B| as the element-wise multiplication between the prediction and target mask, and then sum the resulting matrix.

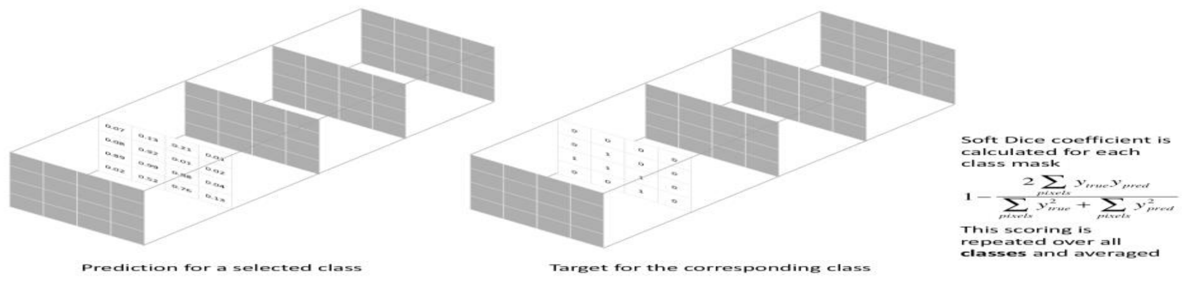

Prediction for a selected class
Target for the corresponding class
Soft Dice coefficient is calculated for each class mask
$1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2}$
This scoring is repeated over all **classes** and averaged

Figure 17 Targeted class

4.3 Segmentation Model working:

4.3.1 Segmentation Model load from Pytorch:

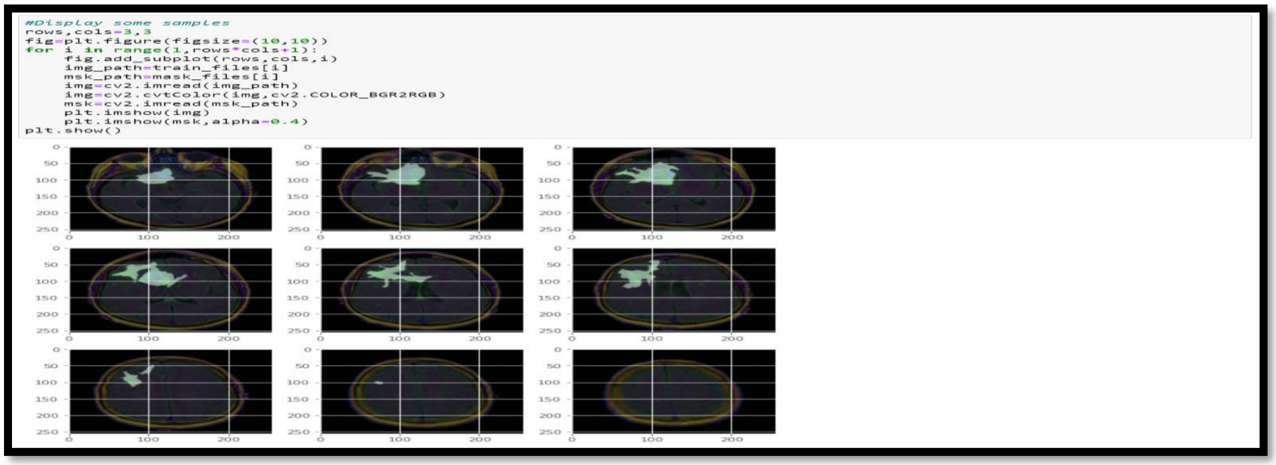
A state_dict is an integral entity if you are interested in saving or loading models from PyTorch. Because state_dict objects are Python dictionaries, they can be easily saved, updated, altered, and restored, adding a great deal of modularity to PyTorch models and optimizers. Note that only layers with learnable parameters (convolutional layers, linear layers, etc.) and registered buffers (batchnorm's running_mean) have entries in the model's state_dict. Optimizer objects (torch.optim) also have a state_dict, which contains information about the optimizer's state, as well as the hyperparameters used. In this recipe, we will see how state_dict is used with a simple model.

In PyTorch, the learnable parameters (i.e. weights and biases) of a torch.nn.Module model are contained in the model's parameters (accessed with model.parameters()). A state_dict is simply a Python dictionary object that maps each layer to its parameter tensor.

To check Compute Unified Device Architecture (CUDA) is available and can be used by one device. Create paths to training images and training masks list

4.3.2 Data visualization Segmentation Model:

Sample images to preview using plt shown below here.



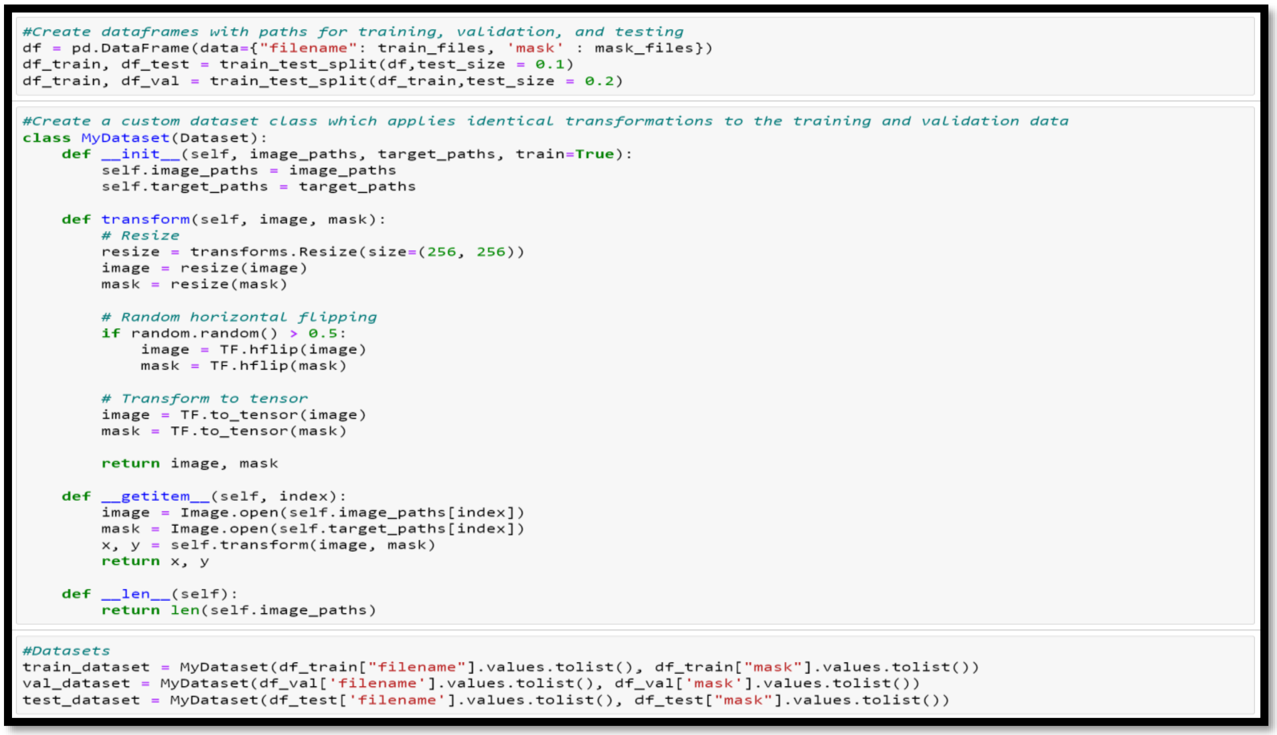
```
#Display some samples
rows,cols=3,3
fig=plt.figure(figsize=(10,10))
for i in range(1,rows*cols+1):
    fig.add_subplot(rows,cols,i)
    img_path=train_files[i]
    msk_path=mask_files[i]
    img=cv2.imread(img_path)
    img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    msk=cv2.imread(msk_path)
    plt.imshow(img)
    plt.imshow(msk,alpha=0.4)
plt.show()
```

Figure 18 Data Visualization

4.3.3 Segmentation Dataset:

With the help of pandas create data frames with paths for training validation and testing. Create a

Class which applies identical transformation to the training and validation data.



```
#Create dataframes with paths for training, validation, and testing
df = pd.DataFrame(data={"filename": train_files, 'mask' : mask_files})
df_train, df_test = train_test_split(df,test_size = 0.1)
df_train, df_val = train_test_split(df_train,test_size = 0.2)
```

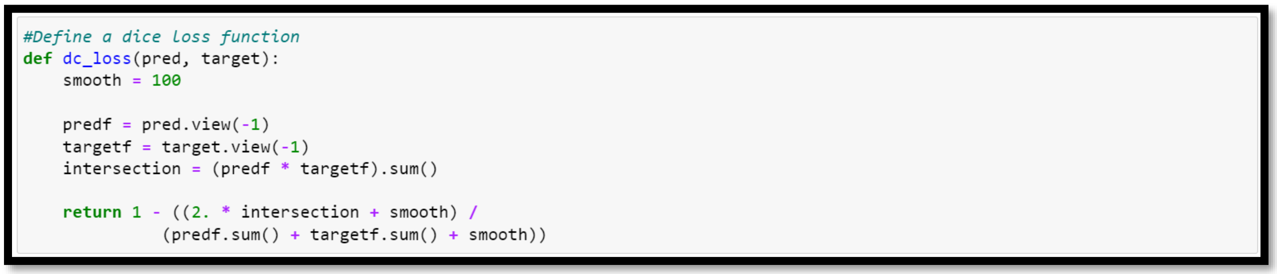
```
#Create a custom dataset class which applies identical transformations to the training and validation data
class MyDataset(Dataset):
    def __init__(self, image_paths, target_paths, train=True):
        self.image_paths = image_paths
        self.target_paths = target_paths

    def transform(self, image, mask):
        # Resize
        resize = transforms.Resize(size=(256, 256))
        image = resize(image)
        mask = resize(mask)

        # Random horizontal flipping
        if random.random() > 0.5:
            image = TF.hflip(image)
            mask = TF.hflip(mask)

        # Transform to tensor
        image = TF.to_tensor(image)
        mask = TF.to_tensor(mask)

        return image, mask

    def __getitem__(self, index):
        image = Image.open(self.image_paths[index])
        mask = Image.open(self.target_paths[index])
        x, y = self.transform(image, mask)
        return x, y

    def __len__(self):
        return len(self.image_paths)
```

```
#Datasets
train_dataset = MyDataset(df_train["filename"].values.tolist(), df_train["mask"].values.tolist())
val_dataset = MyDataset(df_val['filename'].values.tolist(), df_val['mask'].values.tolist())
test_dataset = MyDataset(df_test['filename'].values.tolist(), df_test["mask"].values.tolist())
```

Figure 19 Train Test Split

4.3.4 Dice Loss Function:

The **Dice coefficient** is widely used metric in computer vision community to calculate the similarity between two images. Later in 2016, it has also been adapted as loss function known as Dice Loss [10]. It adds a weight to FP (false positives) and FN (false negatives) with the help of β coefficient.

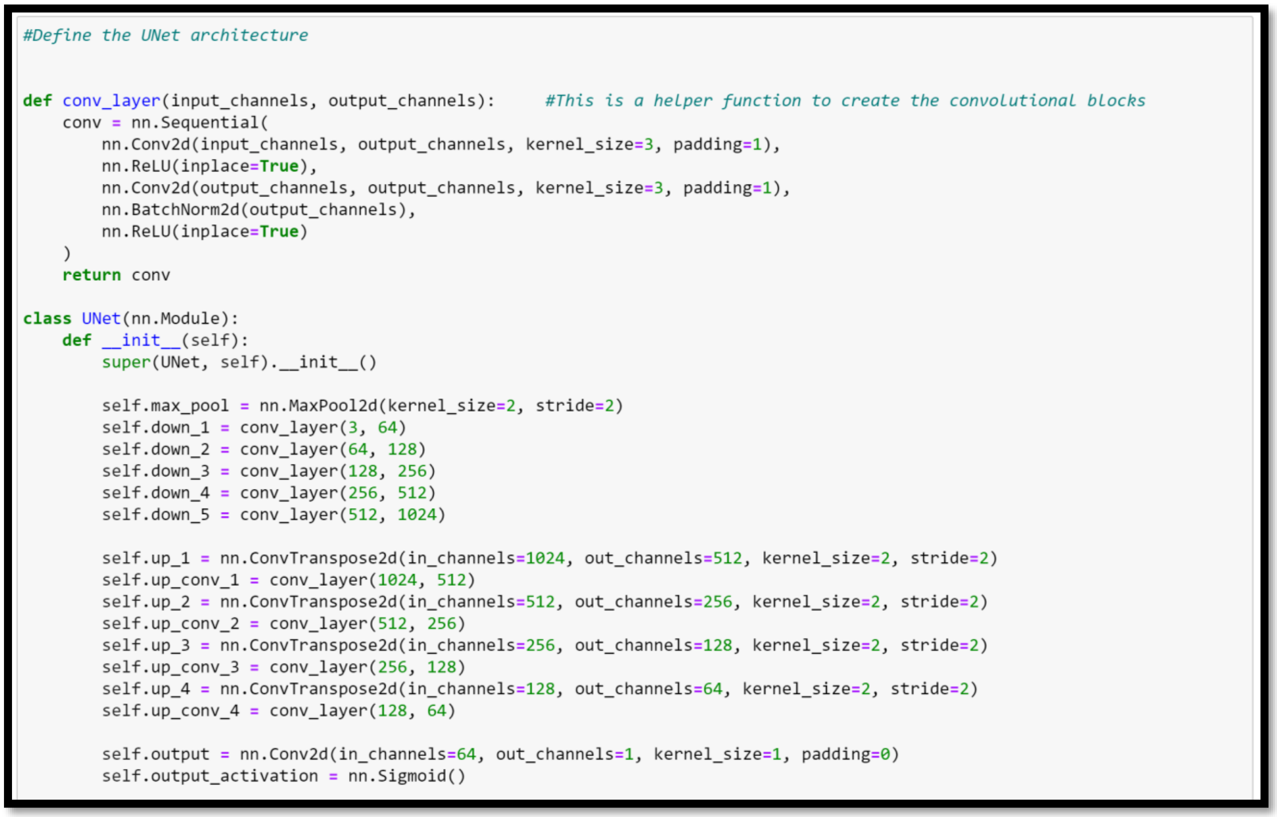


```
#Define a dice loss function
def dc_loss(pred, target):
    smooth = 100

    predf = pred.view(-1)
    targetf = target.view(-1)
    intersection = (predf * targetf).sum()

    return 1 - ((2. * intersection + smooth) /
              (predf.sum() + targetf.sum() + smooth))
```

Figure 20 Loss Function

4.3.5 2d U-Net architecture:

U-Net is a significant accomplishment in the field of deep learning, it is equally essential to understand the previous methods that were employed for solving such kind of similar tasks.



```
#Define the UNet architecture


def conv_layer(input_channels, output_channels):     #This is a helper function to create the convolutional blocks
    conv = nn.Sequential(
        nn.Conv2d(input_channels, output_channels, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(output_channels, output_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(output_channels),
        nn.ReLU(inplace=True)
    )
    return conv

class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        self.max_pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.down_1 = conv_layer(3, 64)
        self.down_2 = conv_layer(64, 128)
        self.down_3 = conv_layer(128, 256)
        self.down_4 = conv_layer(256, 512)
        self.down_5 = conv_layer(512, 1024)

        self.up_1 = nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=2, stride=2)
        self.up_conv_1 = conv_layer(1024, 512)
        self.up_2 = nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=2, stride=2)
        self.up_conv_2 = conv_layer(512, 256)
        self.up_3 = nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=2, stride=2)
        self.up_conv_3 = conv_layer(256, 128)
        self.up_4 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=2, stride=2)
        self.up_conv_4 = conv_layer(128, 64)

        self.output = nn.Conv2d(in_channels=64, out_channels=1, kernel_size=1, padding=0)
        self.output_activation = nn.Sigmoid()
```

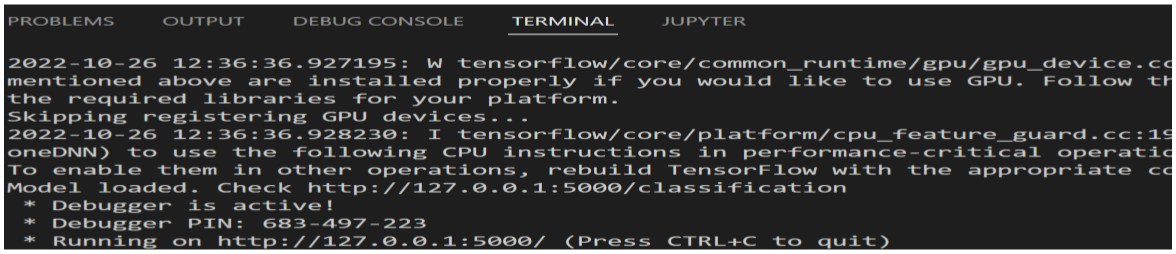
Figure 21 U-Net Architecture

Chapter 5

Deployment

5.1 Detection Model:

The main python file is connected to both Html pages. The model details are imported through a java script, CSS file and dumping the model into the Jupyter notebook. Here the project will be deployed through the flask.



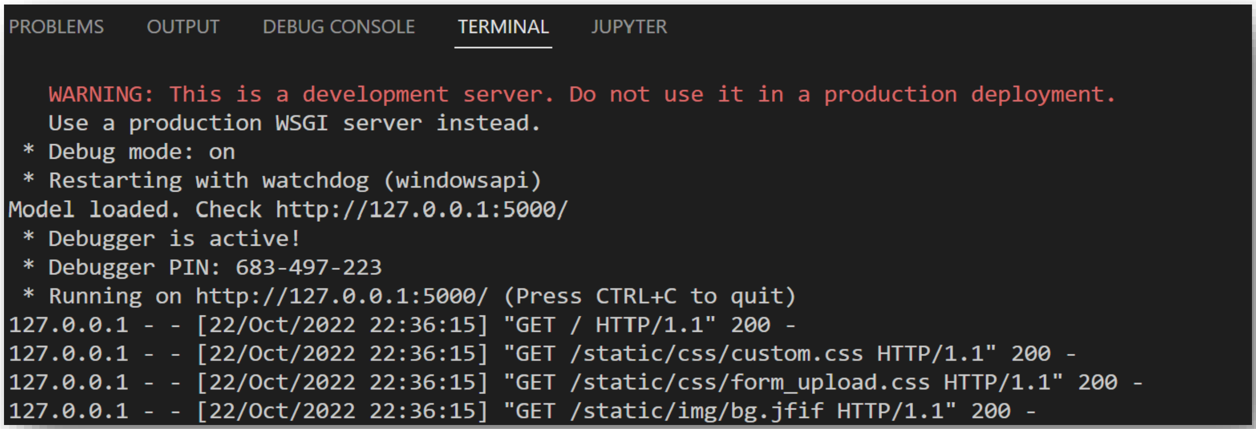
```
2022-10-26 12:36:36.927195: W tensorflow/core/common_runtime/gpu/gpu_device.cc
mentioned above are installed properly if you would like to use GPU. Follow th
the required libraries for your platform.
Skipping registering GPU devices...
2022-10-26 12:36:36.928230: I tensorflow/core/platform/cpu_feature_guard.cc:19
oneDNN) to use the following CPU instructions in performance-critical operatio
To enable them in other operations, rebuild TensorFlow with the appropriate co
Model loaded. Check http://127.0.0.1:5000/classification
 * Debugger is active!
 * Debugger PIN: 683-497-223
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figure 22 Detection model Deployment

Project files like the main python file, both Html files, java script, CSS file, and App.py should be in one single folder and need to be opened on VS Code. Running the app.py in vscode should take you to run on a local machine with a link.

5.2 Segmentation Model:

The main python file is connected to both Html pages. The model details are imported through a pickle file and dumping the model into the Jupyter notebook. Here the project will be deployed through the flask.



```
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
Model loaded. Check http://127.0.0.1:5000/
 * Debugger is active!
 * Debugger PIN: 683-497-223
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [22/Oct/2022 22:36:15] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Oct/2022 22:36:15] "GET /static/css/custom.css HTTP/1.1" 200 -
127.0.0.1 - - [22/Oct/2022 22:36:15] "GET /static/css/form_upload.css HTTP/1.1" 200 -
127.0.0.1 - - [22/Oct/2022 22:36:15] "GET /static/img/bg.jfif HTTP/1.1" 200 -
```

Figure 23 Segmentation model Deployment

Project files like the main python file, both Html files, pkl file, and App.py should be in one single folder and need to be opened on VS Code. Running the app.py in vscode should take you to run on a local machine with a link.

5.3 Detect and Segment:

5.3.1 Detection of Brain Tumor:

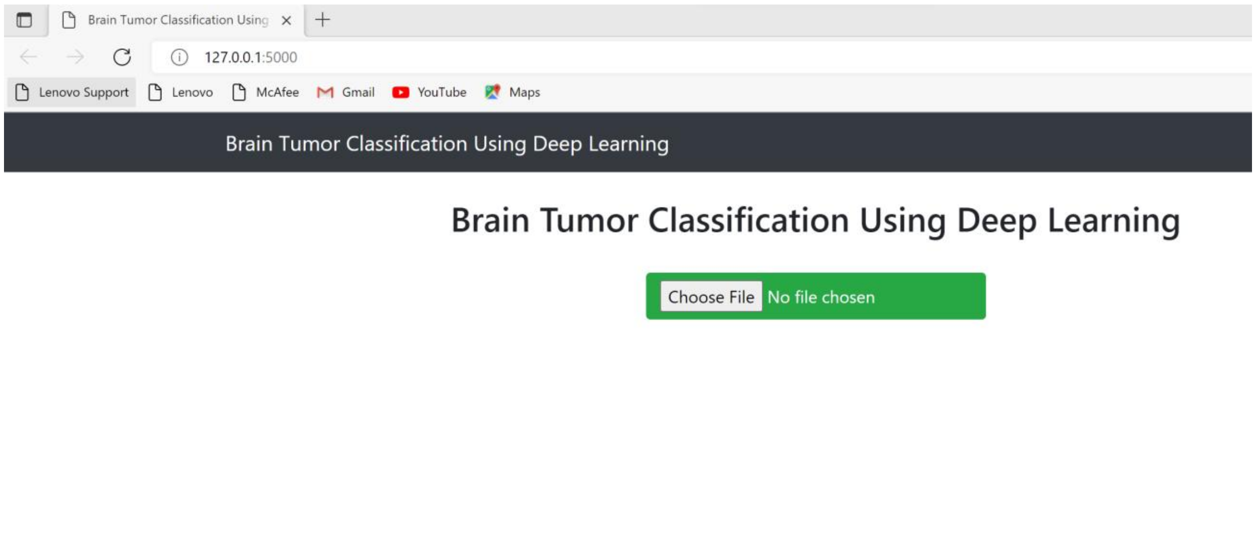
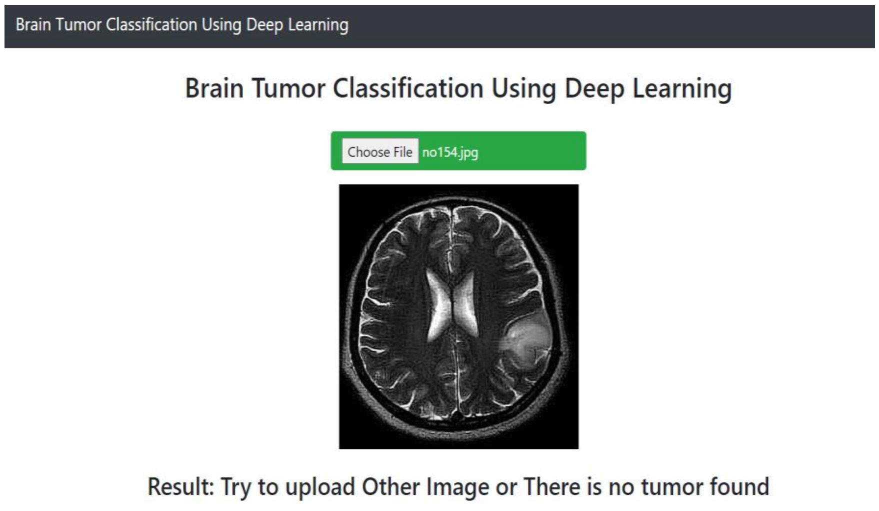

Brain Tumor Classification Using
127.0.0.1:5000
Lenovo Support
Lenovo
McAfee
Gmail
YouTube
Maps
Brain Tumor Classification Using Deep Learning
Brain Tumor Classification Using Deep Learning
Choose File
No file chosen

Figure 24 Detection model upload

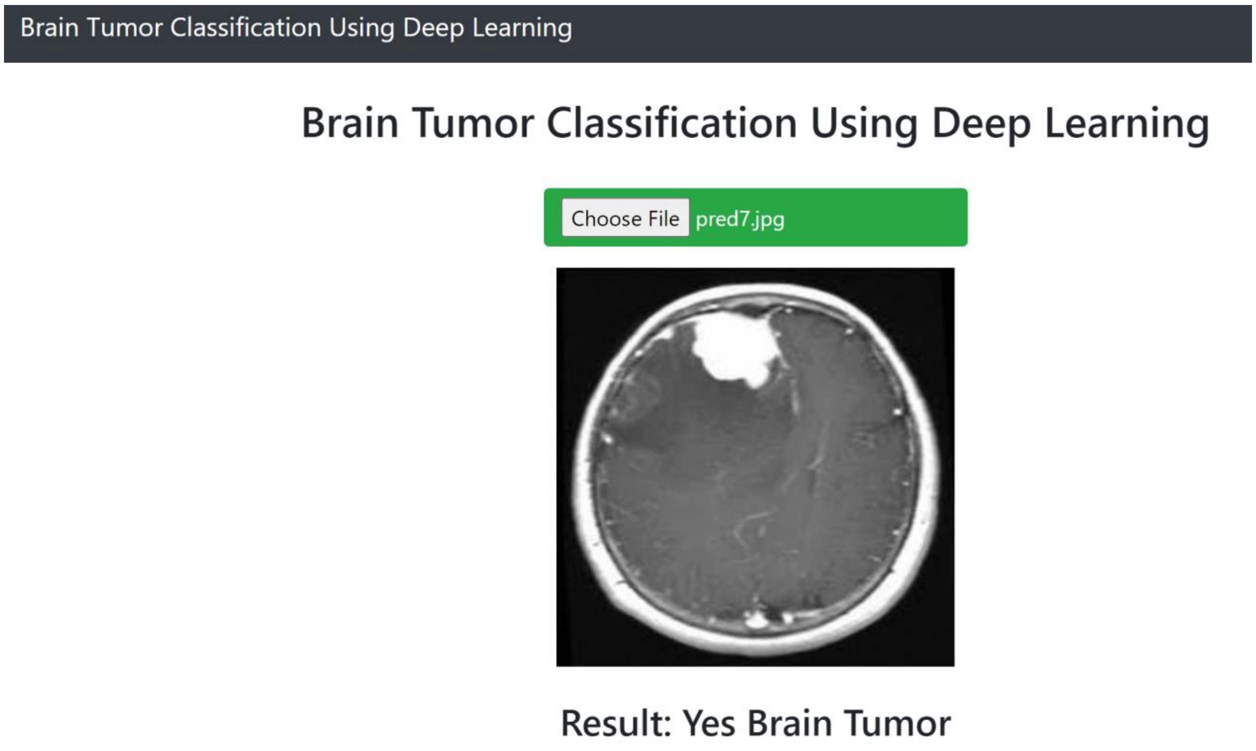
The result looks like below on the classification


Brain Tumor Classification Using Deep Learning
Brain Tumor Classification Using Deep Learning
Choose File
no154.jpg

Result: Try to upload Other Image or There is no tumor found

Figure 25 Detection model result 1

Tumor present case on below


Brain Tumor Classification Using Deep Learning
Brain Tumor Classification Using Deep Learning
Choose File
pred7.jpg

Result: Yes Brain Tumor

Figure 26 Detection model result 2

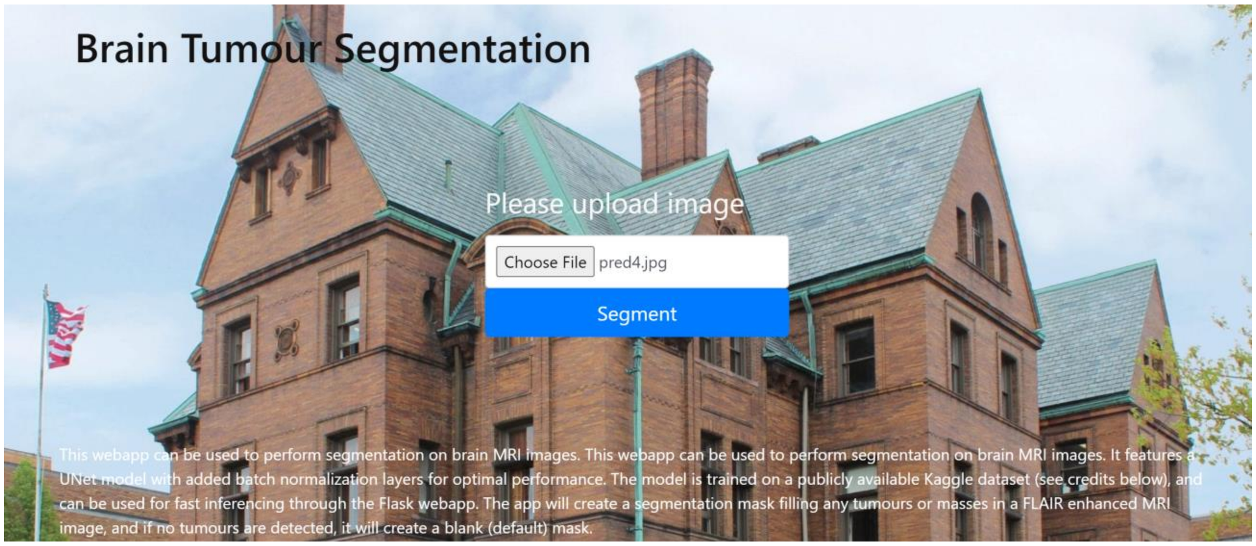
This Html file pops up asking for the MRI image as input. Uploading MRI image gives the classification of the Brain Tumor.

5.3.2 Accuracy and Success Criteria

The accuracy and success criteria of the model is the accuracy which is 0.9875 equals to 98.75 accuracy for prediction.

5.3.3 Segmentation of Brain Tumor

This Html file pops up asking for the MRI image as input. Uploading MRI image gives the Segmentation of the Brain Tumor.



This webapp can be used to perform segmentation on brain MRI images. This webapp can be used to perform segmentation on brain MRI images. It features a UNet model with added batch normalization layers for optimal performance. The model is trained on a publicly available Kaggle dataset (see credits below), and can be used for fast inferencing through the Flask webapp. The app will create a segmentation mask filling any tumours or masses in a FLAIR enhanced MRI image, and if no tumours are detected, it will create a blank (default) mask.

Figure 27 Segmentation model upload

The result looks like below

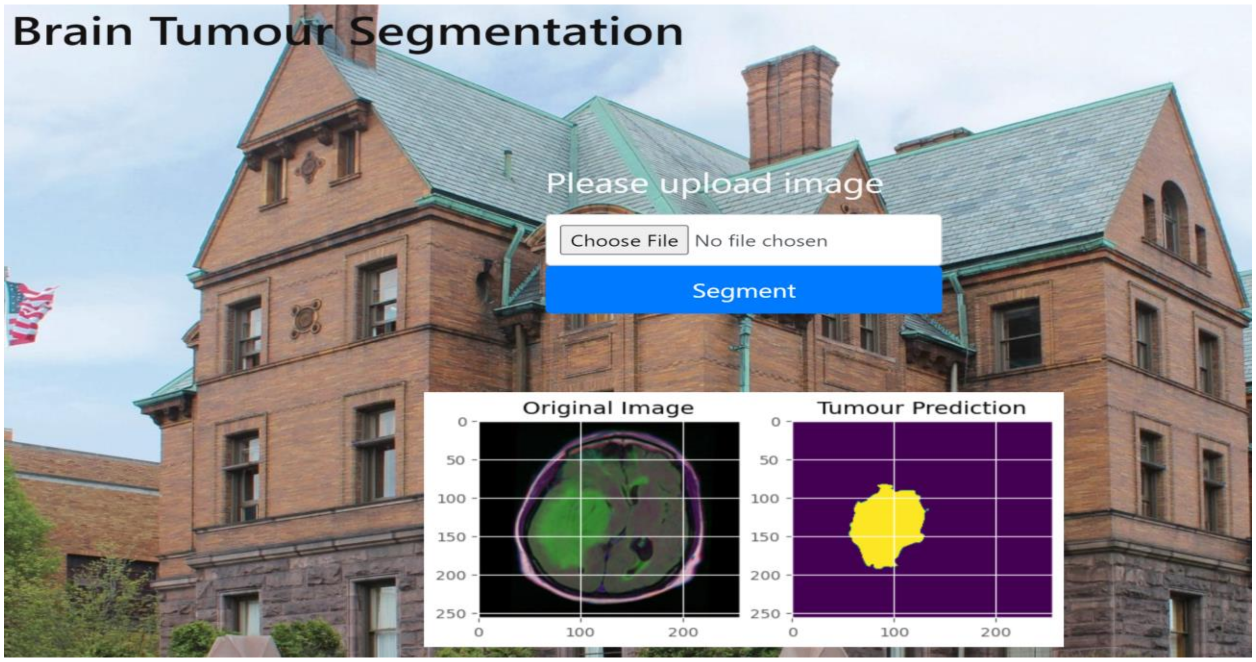


Figure 28 segmentation model result

5.3.4 Accuracy and Success Criteria

The accuracy and success criteria of the model is the accuracy which is 0.87872 equals to 87.87 accuracy for Segmentation.

Chapter 6

Conclusion

Although, there are many classification tools they lack using machine learning algorithms. This solves the problem of the classifies the tumor to medical specialist. This gives an idea of whether tumor is present or not. It helps prevent the manual delineation. Time sensitive. Finally, this study opens the opportunity applying the approach created in real-time models, enabling its use and integration with real navigational applications. Additionally, navigation is now made easier by digital content. The usage of object recognition and picture enhancement algorithms will facilitate user completion of this assignment. Hopefully what I present and proposed to do that as project, I did the classification and segmentation based on the MRI image. Expectation of accuracy met as calculated, according to the machine learning accuracy as the parameter. This project contributes by combining navigation and image processing from a cartographic standpoint. This reduces the effort required of users to concentrate on the parts that need to be perceived by creating an abstraction of the real world through the design and customization of instances that represent tangible objects.

6.1 Challenges & Future Scope

The challenges I faced in segment the tumor location, which is the biggest task to segment the tumor region on given data set. With the help of Torch, we can segment the tumor. However resolved them and provided a solution to similar people who are facing the same problem over GitHub and stack overflow. The future scope of this project is more hyper-tunning needs to be done for more accuracy. Further development for this project will be survival predication based on the deep learning algorithm. However, this varies from the doctors list. We can keep a database that tell about the patient survival history dates from when they got diagnosed by a doctor severity of the patient may be completely cure from the disease or fatality. False positive can be handled by better tuning. Ture positive need to be handle with care and try increase accuracy with better model tune up activity.

Chapter 7 Glossary

MRI – Magnetic Resonance Image

CUDA- Compute Unified Device Architecture

CNN- Convolutional neural network

Chapter 8 References

References

[1] "National Brain Tumor Society," National Brain Tumor Society, 2022. [Online]. Available: https://braintumor.org/brain-tumors/about-brain-tumors/brain-tumor-facts/. [Accessed 2022].

[2] J. JORDAN, "jeremyjordan.me," 21 May 2018. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/.

[3] L. Chato, E. Chow and S. Latifi, "Wavelet Transform to Improve Accuracy of a Prediction Model for Overall Survival Time of Brain Tumor Patients Based On MRI Images," in *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, New York, NY, USA, 2018.

[4] H. T. Zaw, N. Maneerat and K. Y. Win, "Brain tumor detection based on Naïve Bayes Classification," in *2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, Luang Prabang, Laos, 2019.

[5] A. Tiwari, S. Srivastava and M. Pant, "Brain tumor segmentation and classification from magnetic resonance images: Review of selected methods from 2014 to 2019," *ScienceDirect,* vol. 131, no. ISSN 0167-8655, pp. 244-260, 2020.

[6] J. Amin, M. Sharif, M. Raza, T. Saba and M. Almas Anjum, "Brain tumor detection using statistical and machine learning method," *Computer Methods and Programs in Biomedicine,* vol. 177, no. 0169-2607, pp. 69-79, 2019.

[7] G. Hemanth, M. Janardhan and L. Sujihelen, "Design and Implementing Brain Tumor Detection Using Machine Learning Approach," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Tirunelveli, India, 2019.

[8] E. Tabares and G. Fernando, "Machine Learning Image Segmentation to Improve Object Recognition in Mixed Reality," cartographymaster.eu, München, 2020.