Diagnosis of Breast Cancer from Histopathological Image: An Image Segmentation Approach using Deep Learning

Submitted By:

NAME OF STUDENT: EMROZE ISLAM ROLL NO.xxxxxxx

Submitted To:

MD. SHOHEL PARVEZ
ASSISTANT PROFESSOR

A thesis submitted to the Department of Industrial Engineering and Management, Khulna University of Engineering & Technology, Khulna in partial fulfillment of the requirements for the degree of ***"Bachelor of Science in Industrial and Production Engineering"***

11 April, 2022

DECLARATION

This is to certify that the thesis work entitled " **Diagnosis of Breast Cancer from Histopathological Image: An Image Segmentation Approach using Deep Learning** " has been carried out by Emroze Islam in the Department of Industrial Engineering and Management, Khulna University of Engineering & Technology, Khulna, Bangladesh. The above thesis work or any part of this work has not been submitted anywhere for the award of any degree or diploma.

SUBMITTED BY

Emroze Islam
Roll: xxxxxxx

SUPERVISED BY

Md. Shohel Parvez
Assistant Professor
Department of Industrial Engineering and Management
Khulna University of Engineering & Technology

COUNTERSIGNED

Dr. Mihir Ranjan Halder
Head,
Department of Industrial Engineering and Management
Khulna University of Engineering & Technology

ACKNOWLEDGEMENT

The author wishes to convey his heartfelt appreciation to Md. Shohel Parvez, Assistant Professor, Department of Industrial Engineering and Management, for his continuous supervision and invaluable guidance throughout the thesis. The thesis' progress and completion needed a great deal of guidance and assistance from a large number of people, and the author is quite fortunate to have received it all. All that the author has accomplished is only as a result of such supervision and help, and the author would like to express his gratitude to them.

The author would then like to express his gratitude to his Department Head, Dr. Mihir Ranjan Halder (Professor and Head of the Department of Industrial Engineering and Management), for his motivation and cooperative spirit. The author is grateful and lucky to have received consistent encouragement, support, and advice from all of the faculty members at the Department of Industrial Engineering and Management, KUET, who assisted him in successfully completing this thesis work. Additionally, we are grateful to Dr. Quazi Sazzad Hossain, Professor, and Vice-Chancellor of KUET, Khulna, for promoting an excellent educational and research environment.

10 April, 2022 Emroze Islam

ABSTRACT

Breast cancer is one of the most feared diseases due to its high mortality rate. The number of breast cancer patients is increasing at an alarming rate not only in Bangladesh but also all around the world. Early diagnosis is crucial for increasing the survival rate of patients. In this thesis, an image segmentation-based diagnosis method is proposed that can reduce the diagnosis time of breast histopathological images as well as make the overall diagnosis process simpler for pathologists and less error-prone. In this work, a special type of Convolutional Neural Network called UNet will be used. Inception-V3, ResNet-34, and EfficientNet_B2 will be used as backbone. After comparison, the best model will be used for prediction. The segmentation model will predict the region of nuclei responsible for various breast cancer. This segmentation model will result in a quick diagnosis of breast cancer as well as a cheaper solution as we don't have to import the technology from a foreign country. With proper support, this can be deployed in any hospital at a relatively low cost.

Keywords:

CNN, DNN, NuCLS, UNet, Deep Learning, Machine Learning.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Nomenclature

CNN	Convolutional Neural Network
DNN	Deep Neural Network
VGG	Visual Geometry Group
WSI	Whole Slide Image
ROI	Region of Interest
NuCLS	Nucleus Classification, Localization and Segmentation
TCGA	The Cancer Genome Atlas Program
ANN	Artificial Neural Network

Chapter I

Introduction

Breast cancer is one of the most feared diseases. It has a very high mortality rate. In 2020, worldwide total female breast cancer cases were 2261419 and total breast cancer-related deaths were 684996 and the total number of new breast cancer cases in Bangladesh was 13028 [All female patients], and the total number of deaths because of breast cancer was 6783(Sung et al., 2021).

In Bangladesh, 22.5 female out of 100,000 suffers from breast cancer and 19.3 of them are aged between 15 to 44 years old(Begum et al., 2019). So, breast cancer is extremely dangerous for young and middle-aged women. Also, in Bangladesh lack of proper equipment and trained specialist makes this already demanding situation a lot harder to address.

Early identification and diagnosis of breast cancer are critical for patients' survival. With the help of modern computer technology, various steps of breast cancer diagnosis can be done quickly.

Enabling computers to learn on their own is the issue that machine learning aims to answer and as a branch of computer science and statistics, it's at the heart of artificial intelligence and data science and is growing at a breakneck pace and with the rise of modern computer hardware both novel learning algorithms and theory, as well as the ever-increasing amount of available data and low-cost processing, have been driving forces in recent machine

learning advancements(Jordan and Mitchell, 2015). We are seeing increased applications of data-intensive machine-learning approaches in a variety of fields, from health care to manufacturing to education to financial modeling to law enforcement to marketing.

Different algorithms are being used in Machine Learning to solve problems as algorithms are not one-size-fits-all, and data scientists frequently point out this fact and depending upon the nature and requirements of the problem one is trying to answer, the number of variables involved, and the best path to use, the type of algorithm used will vary(Ray, 2019).

A new field of research in machine learning, known as deep learning or hierarchical learning, has evolved since 2006(Bengio, 2009).

Through a variety of signal and information processing applications, both traditional and new, with widened scopes, the techniques developed through deep learning research have already had a significant impact over the past several years and these applications include key aspects of machine learning and artificial intelligence as well as a wide range of signal and information processing work(Arel et al., 2010).

Starting from neural nets to the most recent generative adversarial networks, deep learning architectures have evolved far. Deep learning approaches are outperforming conventional machine learning techniques as a result (Mathew Amitha and Amudha, 2021).

Deep learning has outperformed a variety of classical computer vision algorithms used for image classification, object detection, face recognition, and image segmentation, among other tasks(Voulodimos et al., 2018).

Deep learning algorithms for medical image diagnosis have been proposed in recent years, with impressive outcomes in a wide range of applications such as registration and segmentation. There has been significant interest in using DNNs, especially CNNs, to tackle issues involved in medical image classification and segmentation, such as segmentation of lungs tissue, biological cells, and membranes, tumors, bone tissue as well as classification of various diseases from images, and automatic labeling of images i.e. extracting the contents of an image for various purposes(Lee June-Goo Jun Sanghoon, 2017).

Researchers have been working hard to create more and more data to develop various machine learning and deep learning model. One key issue in medical histopathological data is that a lot of data is available but unfortunately these data are unlabelled and are of no use to anyone. One approach for labeling large unlabelled data is crowdsourcing where a group of individual persons labels the data and then specialists verify the labeled data and applying this concept, Amgad et al. labeled breast histopathological images with the assistance of non-pathologists and then verified the labeled data with the assistance of pathologist(Amgad et al., 2021).

In this work, using the concept of transfer learning and special CNN architecture called UNET, multiple models were developed. The performance of these models was then evaluated and compared with each other.

The rest of the report is arranged as followed. Chapter II represents the literature review, chapter III contains the problem statement and objectives, chapter IV includes methodology, and chapter V represents the result and discussion while chapter VI represents the conclusion and recommendation.

Chapter II

Literature Review

Breast cancer is a prominent reason of mortality in postmenopausal women, responsible for 23 percent of all cancer fatalities, and breast cancer awareness campaigns can play a critical role in the early diagnosis of breast cancers, hence enhancing the chance of survival of breast cancer patients(Akram et al., 2017). Even though the specific cause of breast cancer is unknown, various risk factors for the disease have been identified and they include old age, family medical history of breast cancer, particular changes in the breast(s), genetic alterations, insufficient physical movement, alcohol intake, overweight, malnutrition, ethnicity, and radiation treatment to the chest(Ataollahi et al., 2015).

For the last few years, scientists have been using deep learning in medical image diagnosis. Wang et al. (at the International Symposium on Biomedical Imaging) proposed an award-winning deep learning system with the aim of classifying whole-slide images and detecting cancer metastases in breast sentinel lymph node images(Wang et al., 2016). Another study suggested a CNN-based method to classify 4 types of breast histopathology images where CNN extracted features were used to train a support vector machine classifier that achieves a 77.8% accuracy while using relatively small training examples(Araújo et al., 2017). Another study conducted by Litjens et al. demonstrated that deep learning enhances the efficacy of prostate and breast cancer diagnosis (Litjens et al., 2016). In multiple studies, CNN was utilized to segment tumors and other brain regions. Pereira et al. segmented tumors in brain MRI images using CNN (Pereira et al., 2016). In another research, Moeskops et al. provided a method for automatically segmenting MR brain pictures into a variety of tissue categories using CNN (Moeskops et al., 2016). Kumar et al. used an

autoencoder to extract deep features from CT images and classified lung nodules as benign or malignant using those features while achieving a classification accuracy of 75.01 percent (Kumar et al., 2015). Wu et al. used a deep learning network for the purpose of improving radiologists' cancer screening performance where they trained their model on over 1000000 images and achieves an AUC of 0.895 (Wu et al., 2019). Rakhlin et al. developed CNN-based classifiers where they achieve an accuracy of 87.2% when classifying four classes and 93.8% when classifying two classes (Rakhlin et al., 2018). Another method using two CNNs was proposed by Nazeri et al. where one CNN extracts feature from patches while the other one performs whole-slide classification (Nazeri et al., 2018).

One key challenge in medical image analysis is the lack of properly labeled data. Researchers around the world have been working hard to publish properly labeled data so that other people can develop various deep learning models easily. Created by Spanhol et al. the BreakHis dataset contains 2,480 benign and 5,429 malignant images of breast tumor tissue taken from a total of 82 patients(Spanhol et al., 2016). Recently another dataset named DiagSet was published by Koziarski et al. which contains over two and a half million histopathological image patches of prostate cancer extracted from 430 fully annotated scans (Koziarski et al., 2021). Another recent dataset, BRACS, has 4539 ROIs collected from 547 WSI, each of which has been annotated by three certified pathologists (Brancati et al., 2021). Another breast histopathological dataset was developed by Amgad et al. using crowdsource approach for the segmentation purpose that contains over twenty thousand segmentation annotations(Amgad et al., 2019). This study makes use of the NuCLS dataset, which comprises over 220,000 annotated nuclei from TCGA breast cancer images (Amgad et al., 2021).

Chapter III

Problem Statement and Objectives

3.1 Problem Statement

A histopathological whole slide image is shown in figure 3.1 which is generated by TCGA Research Network: https://www.cancer.gov/tcga.

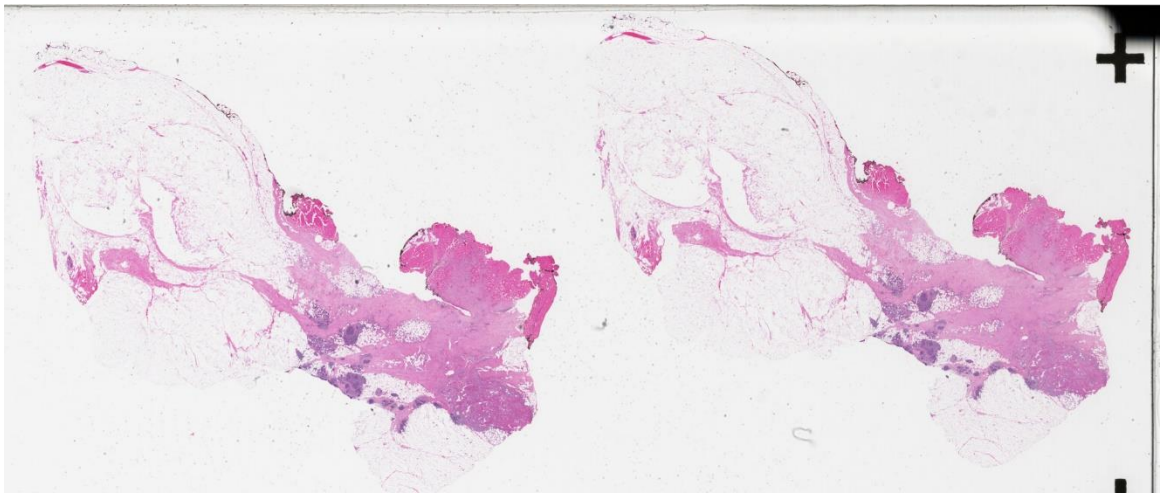


Fig.3.1 A histopathological WSI from TCGA

After zooming in on this image 40 times figure 3.2 is generated.

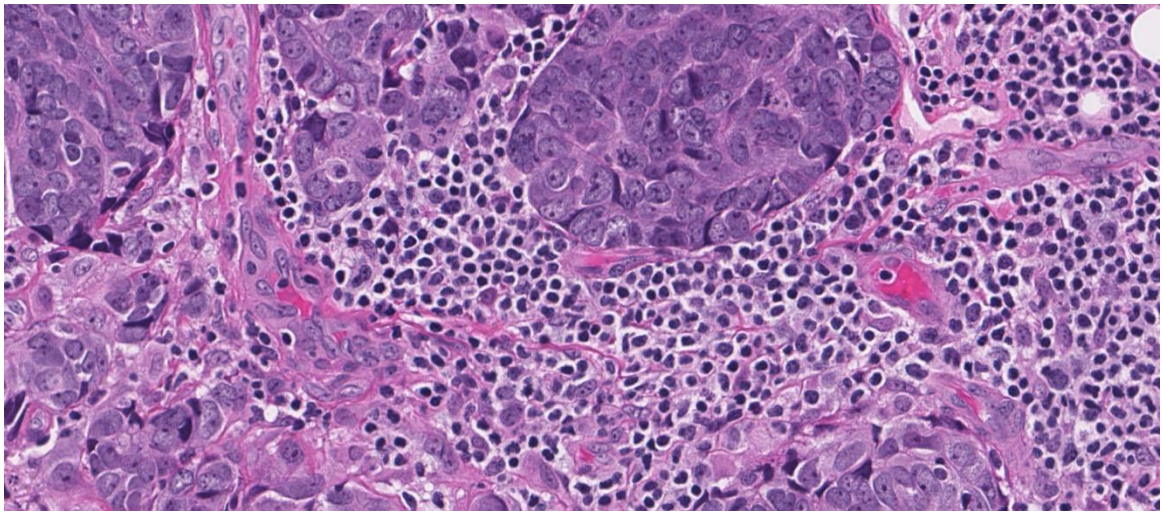


Fig.3.2 After zooming in on the WSI 40 times.

If the original image is zoomed in 80 times figure 3.3 will be generated.

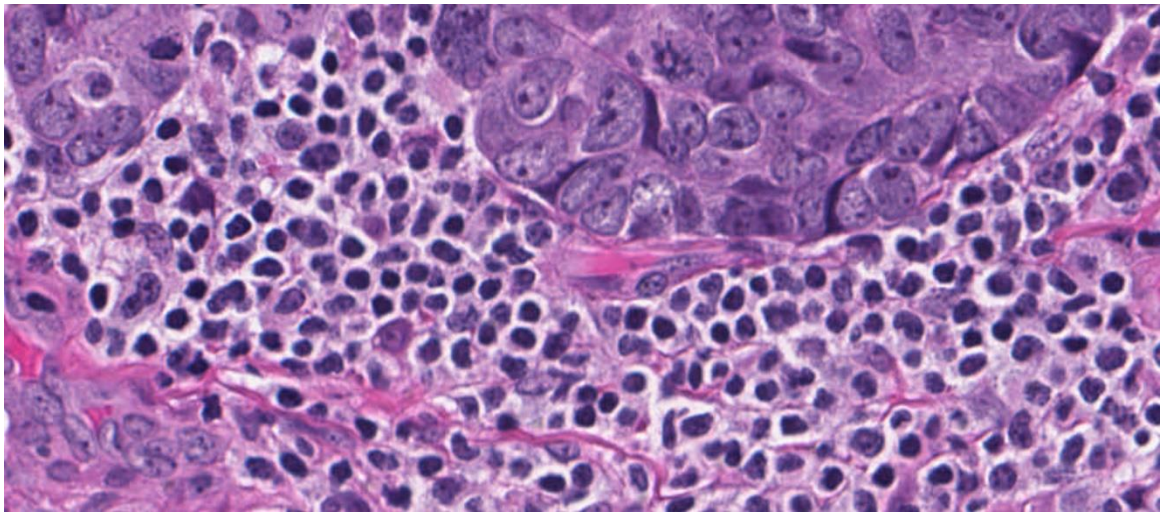


Fig.3.3 WSI after zooming in 80 times

Now after 80times zooming, the nuclei are clearly visible. At this stage, Pathologists have to carefully observe various nuclei, their regions, and their arrangement pattern and identify

anomalies for diagnosing histopathology slides. This is very difficult and tiresome work and prone to human error, especially when observing the complete slide from one end to another end. If a computer-based system can be developed that can identify the region of nuclei automatically as shown in figure 3.4 then the diagnosing task will become easier for pathologists.

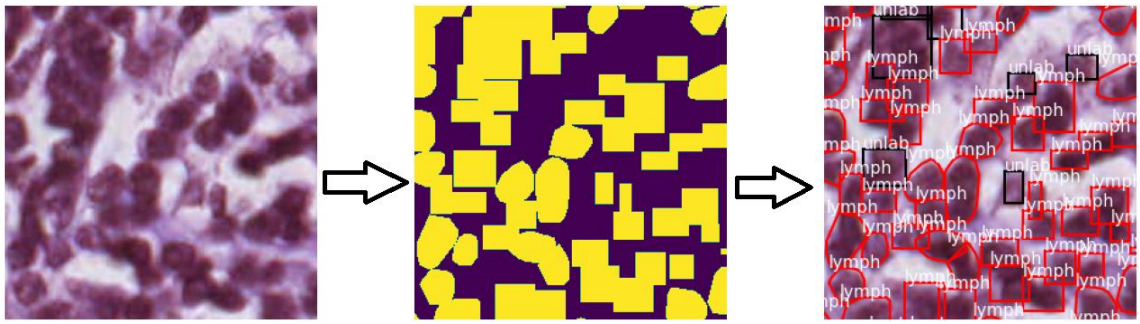


Fig 3.4 Original image, prediction using computer and identification by pathologist.(From left to right)

3.2 Objectives

The objectives of this experiment are given below.

1. To collect, analyze and preprocess histopathological data(Image).
2. To develop multiple segmentation models using deep learning for localizing nuclei in the given breast histopathological images.
3. To evaluate the performance of each model.
4. To compare all these models with each other.

Chapter IV

Methodology

The steps followed in this work are shown in the flowchart shown in figure 4.0.

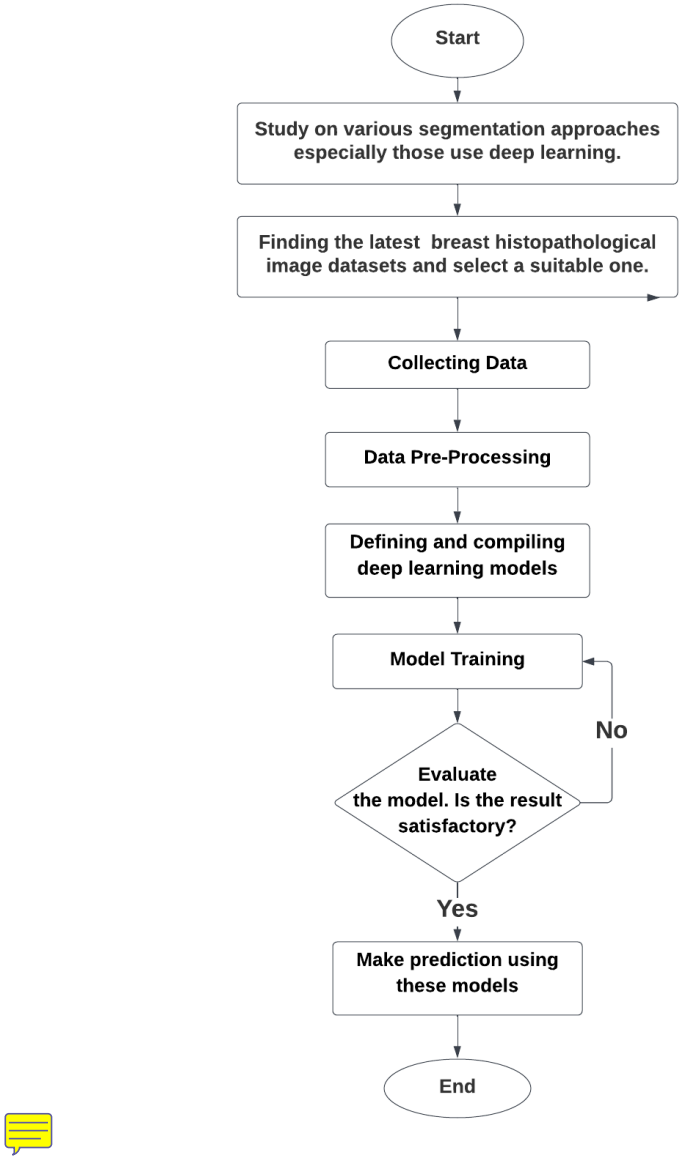


Fig 4.0 Flowchart containing various steps followed in this work

4.1 Dataset

In this work, the NuCLS dataset was used and the actual histopathological images in this dataset were collected from TCGA, and then the images were labeled with the help of non-pathologists as well as algorithms and then reviewed by pathologists(Amgad et al., 2021).

The dataset actually contains two datasets. One is called single-rater and the other one is multi-rater. The single rater dataset was used here as it was created by non-pathologist and corrected by specialists. The single rater dataset contains 1744 images. There are 59,485 nuclei.

The dataset folder contains four folders. These four folders contain main images, masks, annotation coordinates, and visualization images. The coordinates are given in a .csv file while the other data are in .png files. The images are taken in such a way that each pixel represents 0.2 microns. The original masks are modified for binary segmentation. In the modified mask files, the value of any pixel that belongs to any nuclei is 1 while the background is set to zero.

The sample images from the dataset are shown in figure 4.1.

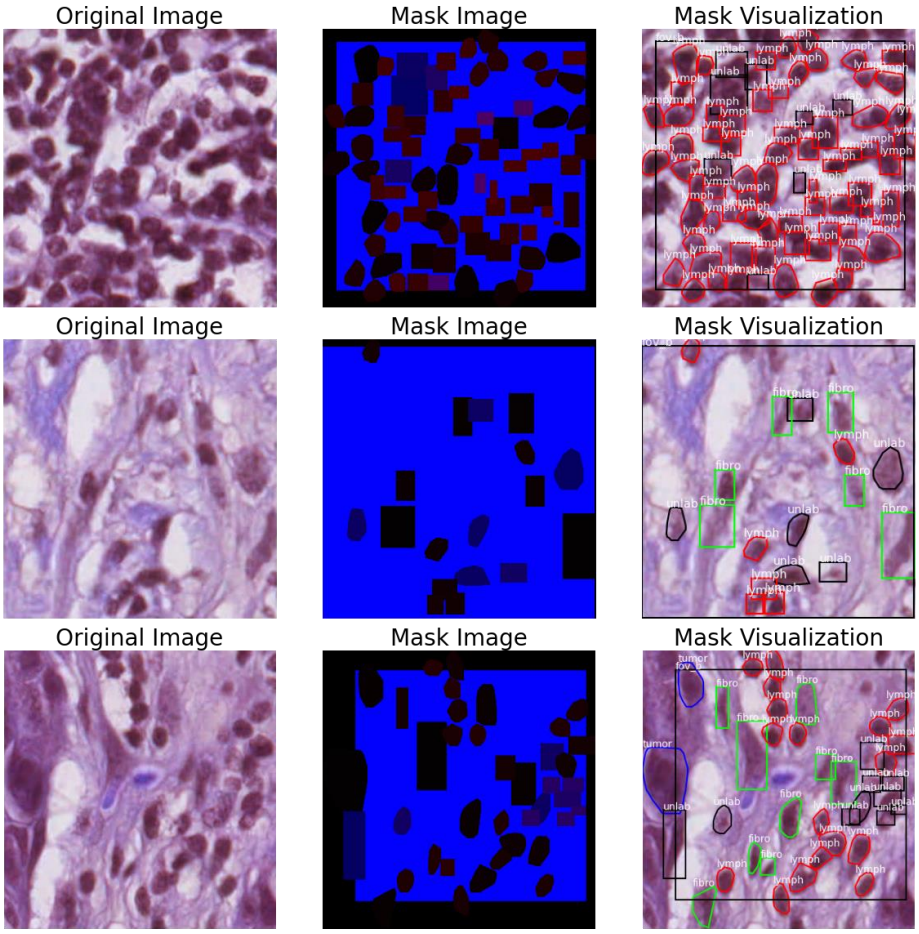


Fig.4.1 Sample images from the dataset

The dataset contains histopathological images as well as corresponding masks. Both images and masks are in .png format. All the images do not have an identical resolution. All these images are RGB images. In the mask images, the class labels are encoded in the first channel. The unique nucleus instances are encoded in the second and third channels.

The dataset contains 13 raw nucleus classes. These 13 classes belong to 7 main classes. These main classes belong to four superclasses.

4.2 Data Preprocessing

Before using any data, it must be changed and modified to the desired structure so that it can be used in the desired system. Various preprocessing steps are used for preprocessing purposes. They are explained below.

4.2.1 Removing corrupted files

Sometimes image files in a dataset can be corrupted. Especially when opening using the OpenCV library, the missing pixel values are filled by other numbers. This results in improper image files. This can lead to false predictions in the case of machine learning or deep learning models. In order to avoid this, scikit-image library can be used to figure out which files are corrupted. When opening an image using scikit-image, it returns an error message if the file is corrupted. Then the corrupted files can be removed from the dataset.

4.2.2 Data normalization

The data should be normalized before using them. Especially in the case of neural networks. A neural network performs better when the data is normalized. If the input images are in RGB format and the color depth is 8bit, then for each pixel there are 3 values for 3 color channels. The value of each channel for each pixel ranges between 0 to 255. To normalize the images, the range 0-255 must be converted to a range of 0 to 1. If we divide each color channel value of each pixel by 255, then the images will be normalized. In the case of 8bit grayscale images there will be only 1 color channel, so dividing each pixel value by 255 will normalize the images. An example of an image normalization of a 3x3 grayscale image is demonstrated in figure 4.2.

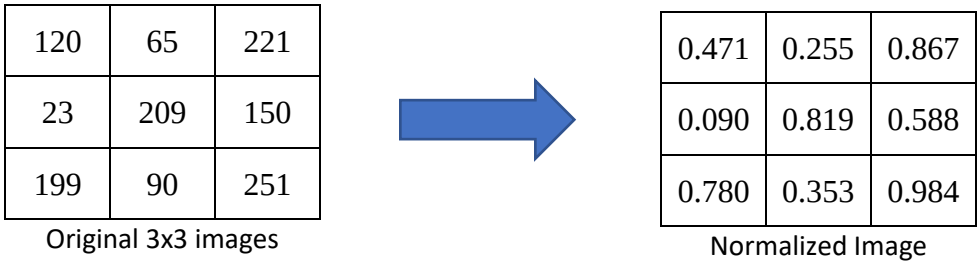


Fig. 4.2 Example of 3x3 8bit grayscale image normalization

4.2.3 Image Augmentation

Augmentation is the process of modifying existing data for the purpose of creating more data. Image augmentation effectively increases the size of the dataset by generating altered versions of existing images. There are different kinds of image augmentation processes, such as horizontal flipping, vertical flipping, image rotation, etc. They are explained below.

Image rotation is a common image augmentation process. In this process, Images are rotated to create new images. An example of image rotation is given in figure 4.3

Fig.4.3 Example of 90º Image Rotation

Another important technique is flipping. Flipping can be said an extension of rotation. Horizontal and vertical flipping is represented in figure 4.4.

Fig.4.4 An example of flipping (From the left original image, horizontally flipped image, and vertically flipped image)

4.2.4 Center Cropping

Images in a dataset may not have an identical resolution. Usually without any processing images comes in different shape. To achieve one single resolution for each image various techniques can be applied depending on the task and image in the dataset. One such method is applying center crop. Cropping means cutting a portion of a given image. The Center crop indicates cutting an image from the center. For applying center crop one such resolution has to be selected so that the selected resolution doesn't exceed the dimension of any images in the dataset. An example of cropping a 2x2 image from a 4x4 image using the center crop method is demonstrated in figure 4.5.

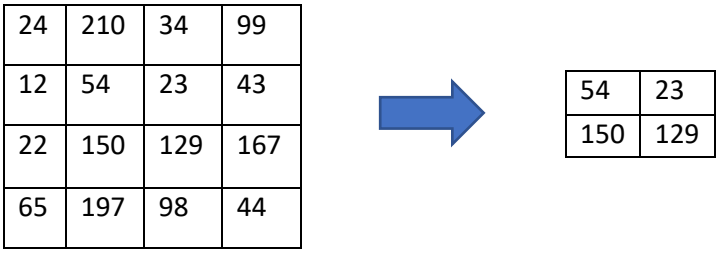


Fig.4.5 Example of center cropping a 2x2 image from a 4x4 image

4.3 Convolution Neural Network:

The working principle of the artificial neural network is inspired by the biological neurons in the animal brain. The convolution neural network is a special type of artificial neural network with convolution layers. It is primarily used in computer vision. The biological process in the animal visual cortex inspired the CNN and the structure of neurons in CNN represents the structure of the visual cortex("Convolutional Neural Network," 2022).

A CNN could gradually extract higher and higher levels of image contents. Without using any preprocessing that extracts features, a convolutional neural network can extract features

directly from images. Generally, CNN accepts an image as a 3-dimensional matrix, where 2 dimensions represent height and width and the other dimension represents the color channel.

CNN is comprised of several types of layers such as the convolution layer, the pooling layer, and the dense layer.

4.3.1 Convolution Layer:

The convolution operation is the heart of the convolution layer. As the name suggests, convolution operations are done here. From a mathematical perspective convolution between 2 functions generate a new function that expressed how the shape of one function is affected by the other one. The convolution of an image is actually a simple linear operation.

In convolution first, a small matrix is used in convolution called " kernel" which is also known as "filter". This kernel then slides over the image(image is represented as a matrix) and an element-wise multiplication is carried out with the part of the image the kernel is currently on and then summing up the result into a single value. This final value is the pixel value. As the kernel slides over, the process is repeated thus converting an image matrix into another matrix. An example of convolution is shown in figure 4.6.

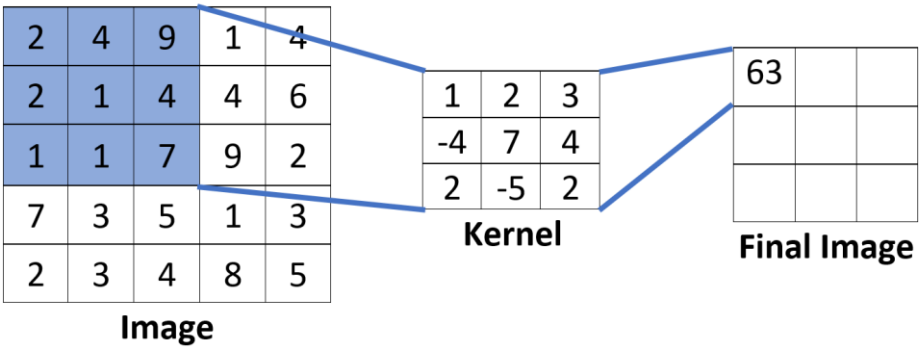


Fig.4.6 Example of Convolution operation using a 3x3 kernel and stride 2.

The features extracted in convolution depend on the kernel value. An image can be focused or blurred, or different edges can be extracted using various kernels.

Usually, there are some important hyperparameters for CNN. They are filters, kernel size, strides, and padding.

A kernel often called a mask, is a small matrix used in image processing for blurring, sharpening, embossing, and edge detection, among other things. This is accomplished by convolution operation between the kernel and an image. The kerne size represents the size of the kernel matrix.

During convolution operation, the kernel matrix slides over the input image matrix for a number of pixels. This number of pixels is known as “stride”. Usually, the filter slides one pixel at a time. But if required it can slide over multiple pixels. An example of stride 1 is demonstrated in figure 4.7.

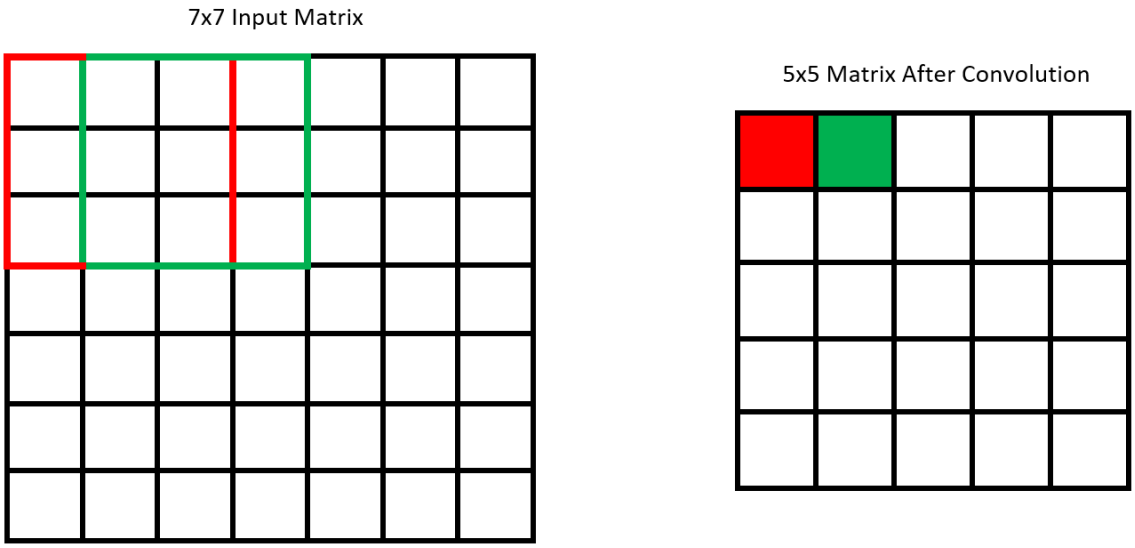


Fig.4.7 Example of stride 1

The matrix edges are trimmed off during convolution operation. A matrix with a 7x7 dimension gets reduced to a 5x5 dimension matrix. As the kernel does not have any way to extend beyond the edge, the edge pixel will never be at the center of the kernel which means they will be trimmed off. But in many operations, we need to retain the original dimension. One common and clever solution is to pad the edges with extra fake pixels of zero value.

34	3	209
55	105	5
23	22	43

0	0	0	0	0
0	34	3	209	0
0	55	105	5	0
0	23	22	43	0
0	0	0	0	0

Fig.4.8 An example of padding.

This is why it's called zero padding. The output dimension becomes identical to the input dimension due to zero padding. An Example of zero padding is given in figure 4.8.

Another important hyperparameter is the number of learnable filters. They are small spatially in terms of width and height but extend to the entire depth of input volume. In the convolution layer, each filter generates a different 2D activation map. The output volume will be created by stacking these activation maps along the depth dimension.

4.3.2 Activation Layer

This layer employs the activation function that contributes to the determination of whether or not a neuron will fire. This function non-linearly transforms the input signal. Some of the popular activation functions are ReLU, leaky ReLU, and sigmoid.

The sigmoid activation function is demonstrated in figure 4.9.

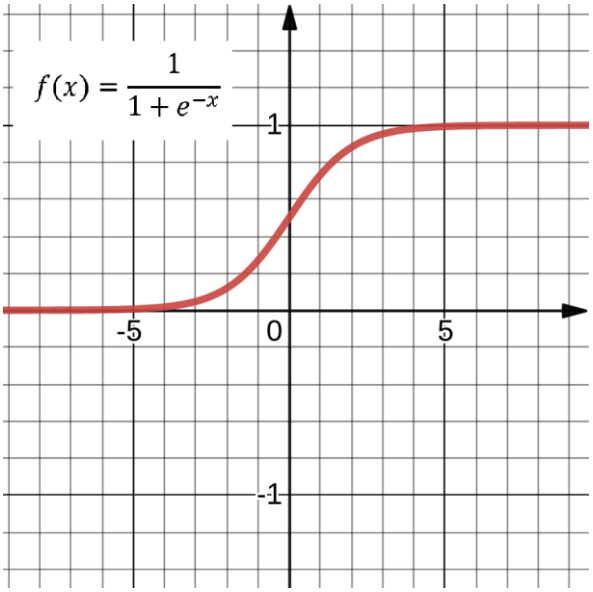


Fig.4.9 Sigmoid Activation Function

The sigmoid function is employed because it exists between zero and one. As a result, it is particularly useful for models in which the output is expected to be a probability. Given that probability ranges exclusively between 0 and 1, the sigmoid function is the appropriate choice. But the sigmoid function is not recommended to use in the hidden layer as they make a machine learning or deep learning model susceptible to various problems during training due to the vanishing gradient problem. Occasionally, this function caused the model to become stuck during training.

ReLU stands for “Rectified Linear Unit”. The ReLU is currently the most frequently utilized activation function. ReLU is incorporated into nearly all convolutional neural networks. The ReLU function is demonstrated in figure 4.10.

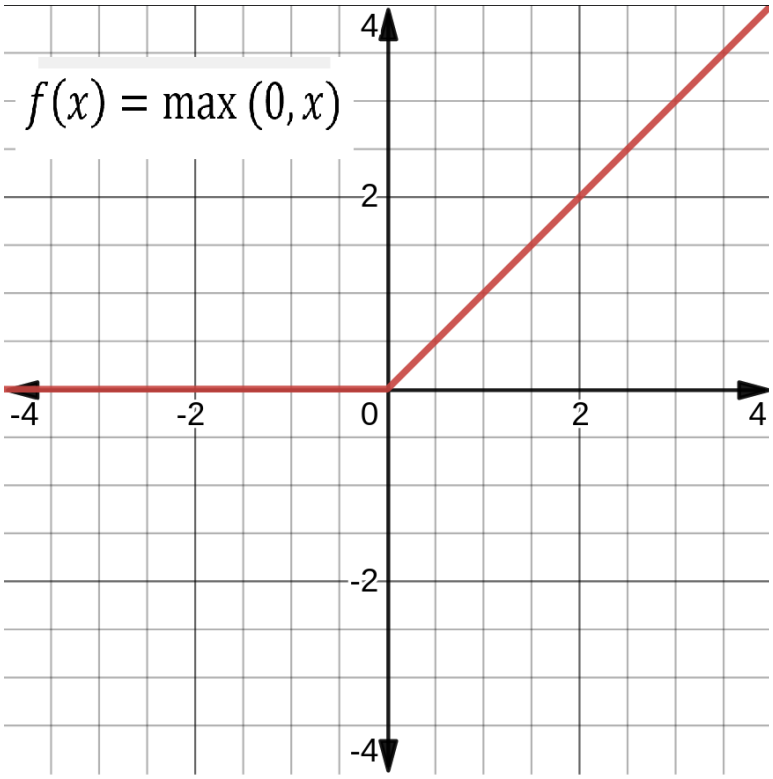


Fig 4.10. ReLU function

If the input is less than or equal to zero then f(x) is zero. But if the input is x then f(x) is x. The ReLU doesn't map negative values properly as all negative values turn into zero immediately. But compared to the sigmoid function its computational complexity is significantly reduced meaning it requires less time and resources to compute the function which in turn reduces the training time for a deep learning model. ReLU also addresses the vanishing gradient problem caused by the sigmoid function.

4.3.3 Pooling Layer

The pooling layer is a critical component of CNN. It successfully decreases the image's size while preserving its features. As a result, the parameter and size of the networks are reduced. Two types of pooling are used in CNN. They are average pooling and max pooling. Max pooling is shown in figure 4.11.

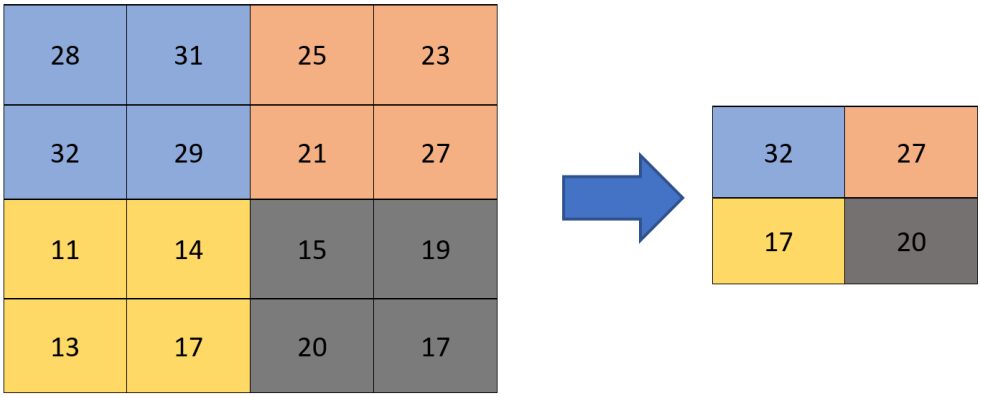


Fig.4.11 Example of Max Pooling for 2x2 filter and stride 2

In max-pooling, the filter slides over the input image for a given stride value. If the stride is 2 then the filter will slide 2 pixels at a time and each time it will pick the maximum pixel value from the region the filter is currently on.

The average pooling is similar to max pooling but instead of picking the largest pixel value, it calculates the average pixel value from the filter region. An example of average pooling is shown in figure 4.12.

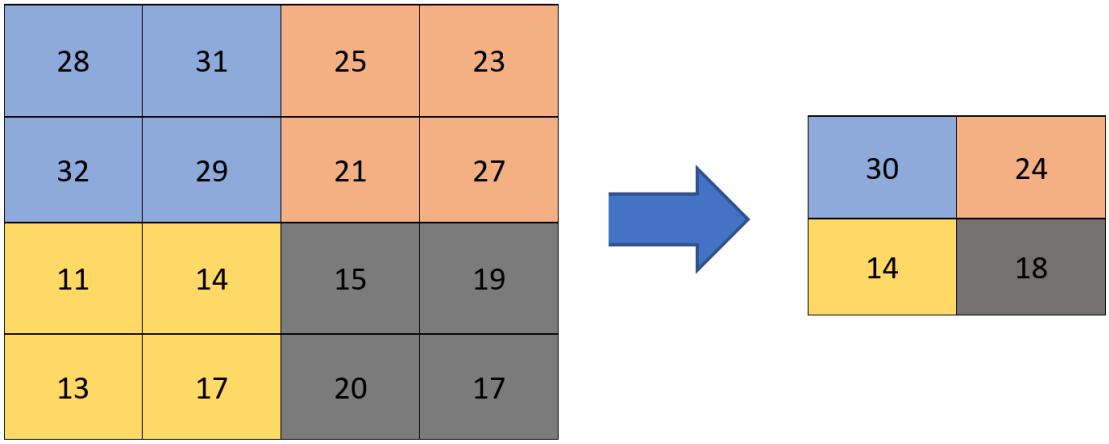


Fig.4.12 An example of Average Pooling for 2x2 filter and stride 2

4.4 Transpose Convolution

In CNN, convolution layers and pooling layers reduce the input's spatial dimension generally width and height for images. This way the input image is downsampled. But another type of CNN, called transpose convolution can be used to increase the reduced spatial dimension of input which is also known as fractionally-strided convolution (Dumoulin and Visin, 2016). To put it simply, transpose convolution restores the resolution that was lost in the prior layer i.e. upsamples the input.

A demonstration of basic transpose convolution is given in figure 4.13.

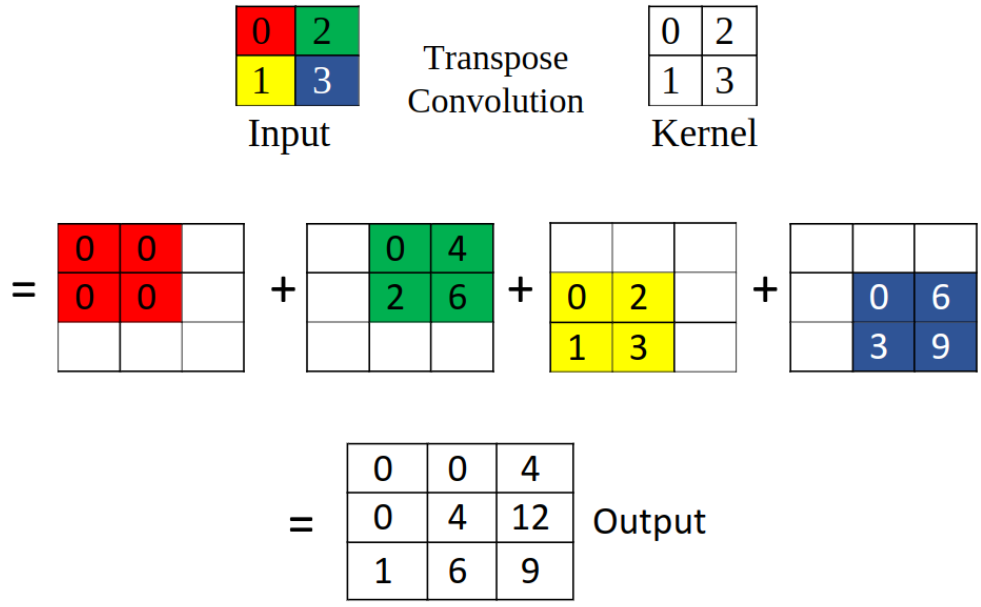


Fig.4.13 Basic Transpose Convolution

Let's consider, that a 2x2 input picture matrix, as well as a 2x2 kernel, were provided. Sliding the kernel matrix, over the input image matrix twice in each row and twice in each column with a stride of 1 produces a total number of four intermediate matrices. Each intermediate matrix is a (2*2-1)x(2*2-1) matrix which is initialized with zero. Each element or pixel in the input image matrix is multiplied by the kernel for the purpose of computing each intermediate matrix, and the resulting 2x2 matrix substitutes a portion of each intermediate matrix. It's worth noting that the location of the pixel in the input image matrix used for the computation corresponds to the position of the replacement component in each intermediate matrix. Finally, the output is calculated by adding up all of the intermediate matrices.

Similar to general CNN, transpose convolution's main parameters are stride, padding, kernel size, and the total number of filters.

4.5 Transfer Learning

Transfer learning is applying knowledge gained while solving one problem to another related problem. The concept of transfer learning for the neural network was first given by Stevo Bozinovski and Ante Fulgosi in 1976 (Bozinovski, 2020).

In general, in machine learning, the transfer learning concept is to train an ML model on one dataset and then retrain the model on another dataset hooping that the model will perform better on the second dataset as the model already knew what features to extract. The method will work if the features are common for both cases, which means that the features are suitable for both the basic and the target tasks, rather than the base task-specific (Yosinski et al., 2014).

In recent years, multiple CNNs were published by researchers all around the world. These CNNs are trained on thousands and sometimes millions of images. For example, many CNN models are trained on the ImageNet dataset which contains more than a million images (Deng et al., 2009). Using these architectures, along with their pre-trained weight to train on a new dataset will result in shorter train time as well as better performance. These models can extract a variety of features that would be nearly impossible to do if we had to train a new network from scratch. Training a neural network is also a time-consuming and computationally intensive task. Ordinary computers are incapable of quickly and effectively training massive networks.

In this work, UNet architecture and various pre-trained models were used as the backbone for UNet.

4.6 UNet

While a convolutional neural network, in general, is ideal for classification tasks, in which the input is an image and the output is a single label, in biomedical applications, one must not only determine if there is a disease but also localize the region of anomaly. UNet, which originated from convolutional neural networks, was developed and utilized for the very first time in 2015 for the purpose of analyzing biomedical images(Ronneberger et al., 2015).

UNet is capable of localizing and distinguishing boundaries since UNet performs classification on each pixel, ensuring that the input and output have identical height and width. For example, if the input image has a resolution of 4x4 then the output image will also have a 4x4 resolution.

4.6.1 UNet architecture

The original UNet architecture is given in figure 4.14.

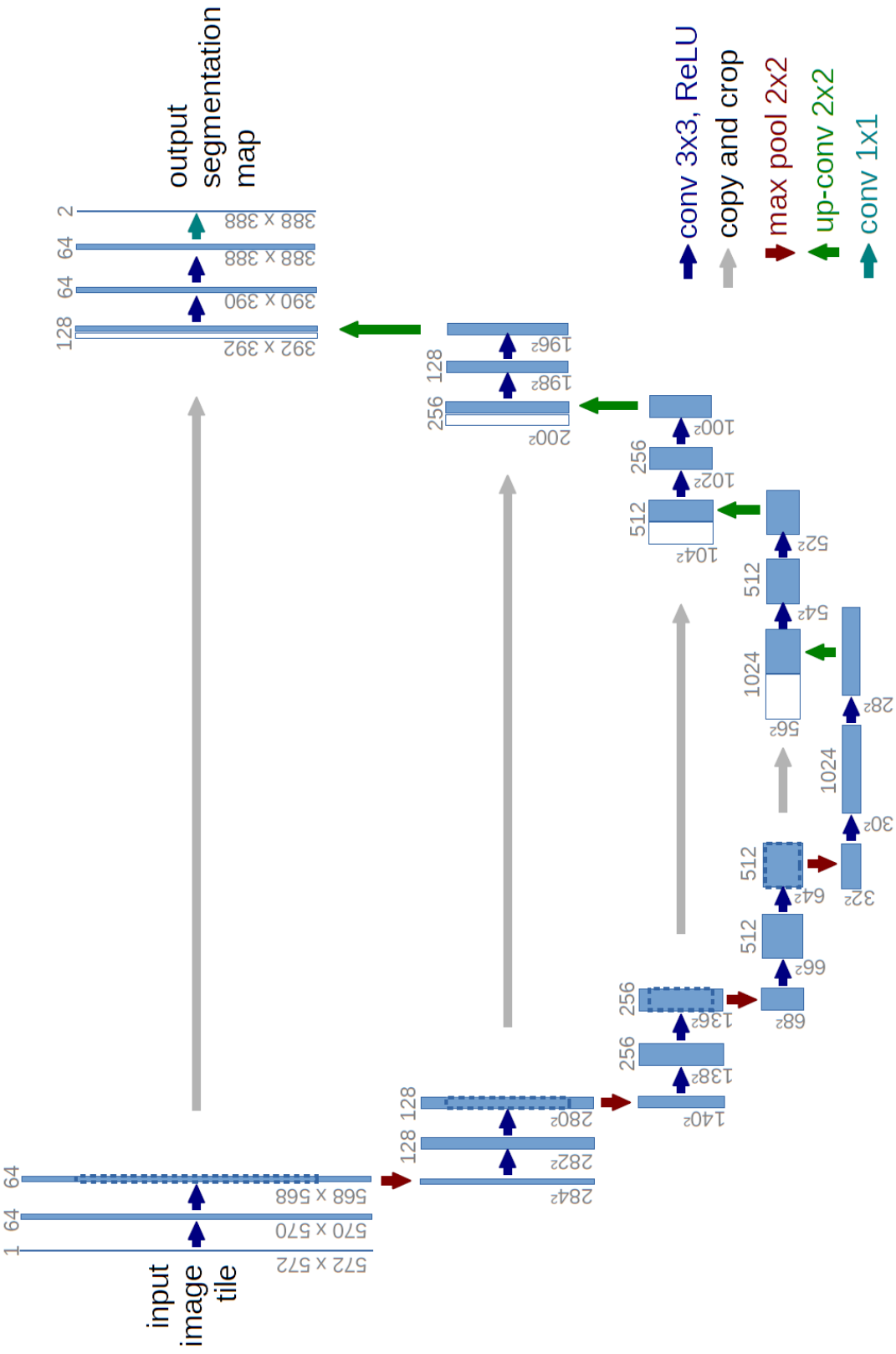


Fig.4.14 Original UNet architecture (Ronneberger et al., 2015)

Right after looking at the UNet architecture, it can be understood why it is called UNet. The architecture looks like the English letter “U”. This is a symmetrical structure that is made up of two major components. The part on the left is known as the contracting path and the part on the right is known as the expansive path. The contracting and expansive paths are also known as encoder and decoder paths, respectively. The general convolution process is used as a building block in the contracting path. The expanding path is comprised of transpose 2D convolution layers also known as up convolution layers.

After careful observation, it can be observed that both the contraction and expansive path can be divided into multiple smaller blocks. In the contraction path, there are two convolution layers of kernel size 3x3 accompanied by a pooling layer of filter size 2x2 and stride 2. So the formula is conv_layer_1 -> conv_layer2 -> Pooling Layer -> dropout_layer(Optional). And this formula is repeated another three times. Then there are 2 more convolution layers without any pooling layer.

After this, the expansive path or decoder starts. In this decoder part, the image will be restored to its original height and width. This decoder part starts with an up-convolution layer. After concatenation with the skip connection as shown in figure 4.14, there are 2 more convolution layers of kernel size 3x3. Then up-convolution, concatenation, and 2 convolution layers will be repeated 3 more times. Finally, a convolution layer of 1x1 kernel size and filter size equal to the total number of classes is used. At this point, the output is the desired segmentation map.

One important thing about UNet is that it does not have any dense layers.

It must be noted that the skip connections from the encoder part that are concatenated with the layers in the decoder part as shown in figure 4.14 are especially important. Because this

connection enables the UNet to use details learned from the encoder part to be used in the decoder part and construct a final image.

In UNet, the arrangement of layers in the encoder part determines the arrangement of layers in the decoder part. The layer arrangement is called the backbone. In this work, the original backbone of UNet was replaced by other pre-trained backbones.

4.6.2 Backbones for UNet

The architectural structure of the encoder part of UNet defines how various layers in the encoder part are arranged which in turn determines the layer arrangement in the decoder part is called the backbone. We replaced this backbone with other pre-trained networks. We used three backbones which resulted in three final segmentation models. These backbones are InceptionV3, ResNet-34 and EfficientNetB2.

4.6.2.1 VGG19

Developed by the Visual Geometry Group from oxford university, VGG-19 is a 19 layer deep convolutional neural network that is trained on the ImageNet dataset and capable of classifying 1000 different objects (Simonyan and Zisserman, 2014). A visual demonstration of VGG16 is given in the following figure 4.15.

Input Image (224x224x3)
Conv3x3 (64)
Conv3x3 (64)
MaxPool
Conv3x3 (128)
Conv3x3 (128)
MaxPool
Conv3x3 (256)
Conv3x3 (256)
Conv3x3 (256)
Conv3x3 (256)
MaxPool
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
MaxPool
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
Conv3x3 (512)
MaxPool
Fully Connected (4096)
Fully Connected (4096)
Fully Connected (1000)
SoftMax

Fig.4.15 VGG19 CNN model

VGG-19 start with an RGB image of resolution 224x224. Then there are two convolution layers of kernel size 3x3 and filter size 64 then followed by the max pool layer. Then there are 2 more convolution layers of filter size 128 then followed by a max pool layer. Then again four more convolution layers(Filter size 256) and one max pool layer. Then there are four more convolution layers(filter size 512) and one max pool layer and these 5 layers were repeated again. Now there are 2 fully connected layers of 4096 neurons and in the end, there is a fully connected layer of one thousand neurons. The softmax activation

function is used for the last fully connected layer. So in total 16 convolution layers and 3 fully connected layers are used in this model.

The model was trained on the ImageNet dataset that contains over a million images.

4.6.2.2 EfficientNet

EfficientNet achieves state-of-the-art performance on both ImageNet and other classification tasks while being incredibly efficient because it requires the least flops for inference (Tan and Le, 2019). The EfficientNet has 8 variants from B0 to B7. The main difference between these variants is the size of the input image. The visual representation of the model is shown in figure 4.16.

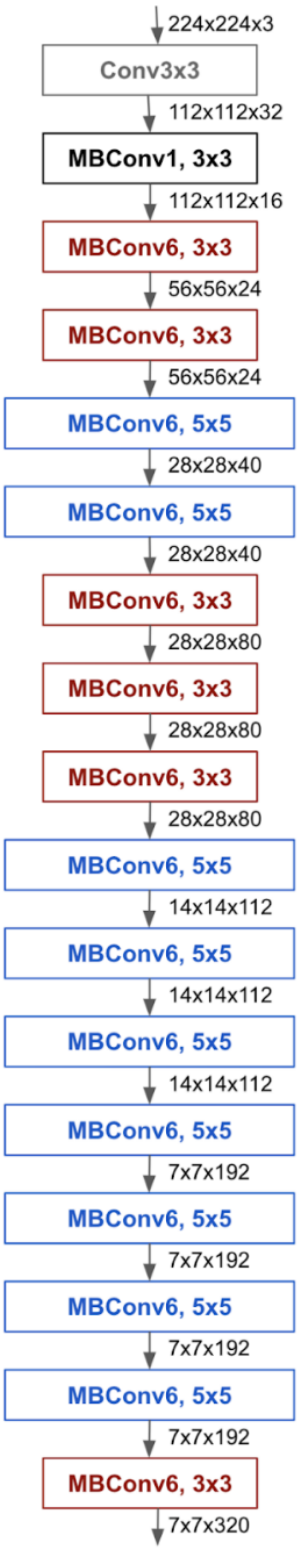


Fig.4.16 Baseline Network EfficientNet-B0 ("EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling," 2019)

The EfficientNet models outperform existing CNNs in terms of accuracy and efficiency while reducing the total number of parameters and FLOPS by such a great order of magnitude. On ImageNet, EfficientNet-B7 achieves 97.1 % top-5 / 84.4 % top-1 accuracy while being 6.1 times faster on CPU inference and 8.4 times smaller than Gpipe. When compared to the commonly employed ResNet-50, the EfficientNet-B4 uses identical FLOPS while increasing top-level accuracy from 76.3% to 82.6%. A total 6.3% improvement over ResNet-50.

4.6.2.3 ResNet-34

He et al. From Microsoft Research developed ResNet which addressed one of the famous problems seen in various previous neural networks known as “Vanishing Gradient” (He et al., 2016). It was observed that the deeper network the better the performance. This is understandable as a network with more layers tends to have more capabilities. But in reality, it was noticed that instead of increasing, the performance degrades after some depth. It was one of the major drawbacks of the VGG series of networks. After multiple applications of the chain rule, the gradients from which the loss function is computed quickly drop to zero. As a result, the weights do not ever update their values, and hence no learning takes place.

The ResNet-34 model is shown in figure 4.17.

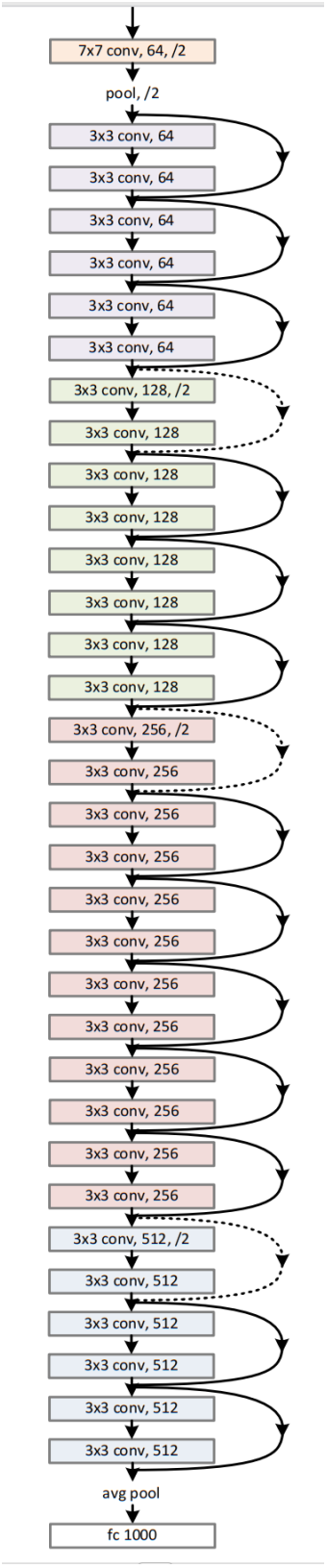


Fig 4.17 ResNet-34 Architecture (He et al., 2016)

ResNet architecture fixes the vanishing gradient problem with the help of skip connections. Gradients can flow backward from later layers to early filters via skip connections. All the convolution layer in this network has a kernel size of 3x3. The only difference among all the convolution layers is the number of filters. 64, 128, 256, and 512 filter sizes are used in various parts of this network. There are 34 convolution layers and one fully connected layer.

4.6.2.4 Inception-V3

As an improvement over Inception-V1 and V2, Inception-V3 was introduced where one of the major progress was utilizing factorized 7x7 convolution(Szegedy et al., 2016). The architecture of Inception-V3 is given in figure 4.18.

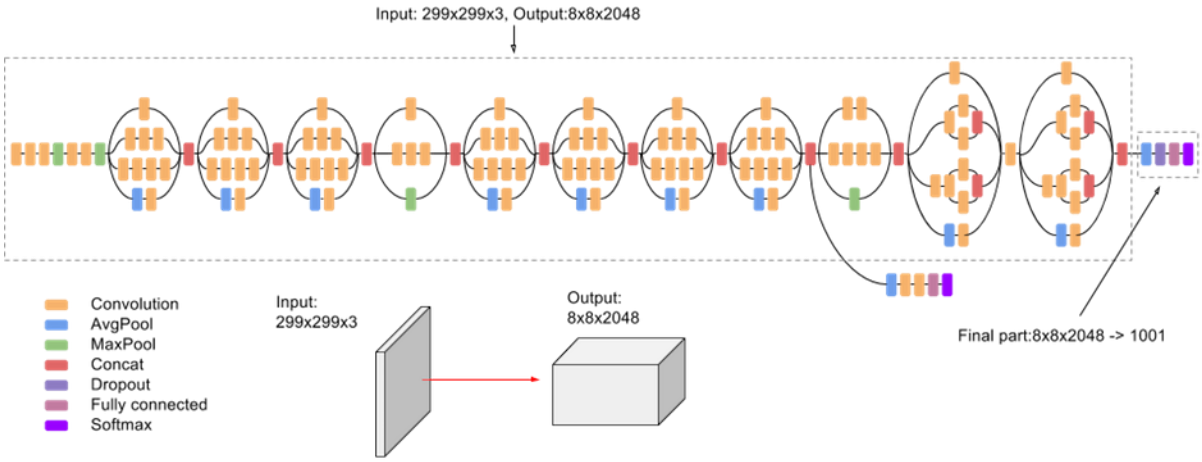


Fig 4.18 Inception-V3 Architecture (“Advanced guide to inception V3 | cloud TPU | google cloud,” n.d.)

Inception V3 has a more extensive network than V1 and V2 but from a computation perspective, it is less expensive. And also the model didn't compromise speed. The model has 42 layers and about 24 million parameters. The efficiency of Inception V3 is higher than that of VGGNet. It has a top-5 error rate of 5.6% and a top-1 error rate of 21.2%.

4.7 Metrics for evaluation

Depending on the task, different metrics are used for evaluating a neural network's performance. For the segmentation task, primarily the IOU score and F score are used.

4.7.1 IOU Score or Jaccard Coefficient

First developed by Paul Jaccard in 1912, the Jaccard Index or the Jaccard coefficient is used for estimating the sample sets' similarity and diversity (Jaccard, 1912). The Jaccard coefficient is also known as Intersection Over Union and is widely used for measuring the performance of segmentation tasks. The predicted mask and ground truth mask for a hypothetical segmentation task are demonstrated in figure 4.19.

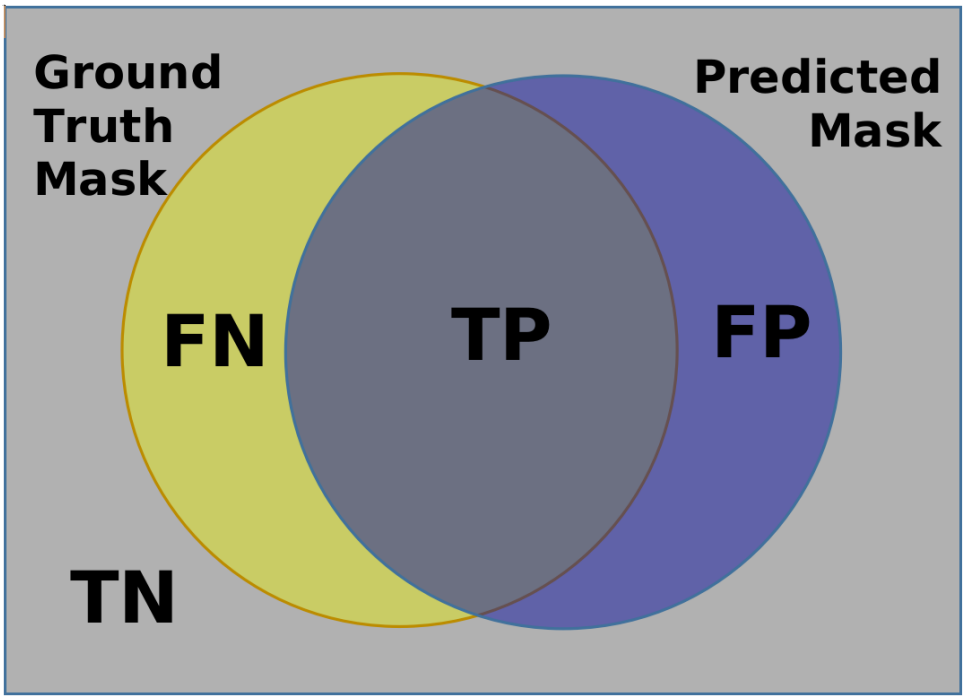


Fig 4.19 Predicted mask and ground truth mask for a hypothetical segmentation task

The yellow region is False Negative or FN, the blue region is false positive or FP, the gray common area of both circles is true positive or TP and the area outside of both circles is true negative or TN. Now the IOU Score can be calculated by using the equation 4.1

$$Jaccard's\ Co-efficient\ = \frac{Inter\,sec\,tion}{Union} = \frac{TP}{TP+FN+FP} \text{.....(4.1)}$$

Here,

TP = True positive area

FN = False Negative area

FP = False Positive area

4.7.2 Dice Co-efficient

For figure 3.16, the dice Coefficient can be calculated using equation 4.2.

$$Dice\ Co-efficient\ =\ \frac{2\times Inter\,sec\,tion}{Union+Inter\,sec\,tion}=\frac{2\times TP}{2\times TP+FN+FP}\text{.....(4.2)}$$

Here,

TP = True positive area

FN = False Negative area

FP = False Positive area

4.8 Development Environment

In this work, everything was done using programming. Python programming language and various library that supports python language such as TensorFlow, NumPy, Matplotlib, OpenCV, segmentation-models, etc were used in this work. Various Linux commands such as cp, unzip and mkdir were used.

4.8.1 Python 3

Python3 i.e., version 3 was used in this work (van Rossum and Drake, 2009). Python is one of the most popular languages around the world right now. It is relatively easy to understand. It is an open-source programming language i.e. its source code is open for all.

Also, python can be easily integrated into any programming-based project. It is used in web development, software development, machine learning, data science, game development, etc.

4.8.2 TensorFlow

Originally developed by researchers from Google, TensorFlow is an open-source deep learning framework that is widely used in the machine learning and deep learning research field and is one of the most popular deep learning frameworks (Abadi et al., 2016). It is a cross-platform system, that can be used with CPU, GPU, embedded system, and other various hardware. It is primarily written in c++ and provides an interface for other programming languages such as python, javascript, etc to access the library. Detailed information about TensorFlow code can be found in their GitHub repository(Tensorflow, n.d.).

4.8.3 NumPy

NumPy is an acronym that stands for Numerical Python. First developed by Travis Oliphant, NumPy is a Python library for dealing with arrays and is comprised of multidimensional array objects and a set of algorithms for processing them(Harris et al., 2020). It is incredibly fast and more compact compared to the python list. It also consumes less memory when it comes to storing data and provides a mechanism for specifying the type of data. NumPy is primarily implemented in C programming languages which is one of the reasons that make it very fast.

4.8.4 Matplotlib

Initially developed by John D. Hunter, Matplotlib is a library, developed for the purpose of visualizing and plotting data for python programming language and its numerical computation library NumPy with the idea of being able to make basic plots using just a few instructions (Hunter, 2007). It is now an open-source project. It originated as a tool to mimic the graphics commands of MATLAB, but it is now completely free of MATLAB. While Matplotlib is mostly implemented in pure Python, it makes extensive utilization of python numeric library NumPy and other extension code for the purpose of ensuring that it performs well even with arrays of large size.

4.8.5 OpenCV

Initially created by Intel, OpenCV is a cross-platform, open-source library for computer vision that is written in C/C++ and now has over two and a half thousand optimized algorithms including classical computer vision as well as cutting edge computer vision and various machine learning algorithms (Bradski, 2000). Various image processing operations such as reading, writing, resizing, rotating, convolution as well as various machine learning algorithms such as object detection, face detection, object recognition, etc. can be performed with the help of OpenCV.

4.8.6 Split-Folder

This library is completely written in python and published under MIT license and can be used to divide image datasets into train, test, and validation datasets according to a given ratio (“Split-folders,” 2018).

4.8.7 Segmentation Models

Segmentation Models is a library that was developed for creating various neural networks for image segmentation using just a few lines of code (Yakubovskiy, 2019). Various loss functions and metrics were implemented in this library which makes it easier to develop segmentation models. The library supports 4 model architectures and 25 backbones with pre-trained weights for each type of architecture were also provided.

4.8.8 Keras

Keras is a free and open-source software package that functions as a high-level API for the TensorFlow library and enables scientists and engineers to fully exploit TensorFlow 2's scalability and cross-platform features (Chollet and others, 2015).

4.8.8 Linux Commands

Various Linux command was used in this work. The command “cp” is used for copying a file from one directory to another. The command “mkdir” is used when a folder is needed to be created and “unzip” is used for uncompressing “.zip” files. Also “Tree” is used for finding out various directories.

4.9 Experimental Procedures

The various procedures for conducting this experiment are given below.

4.9.1 Data Collection

Amgad et al. published the NuCLS dataset and the data was collected according to the method mentioned by the authors (Amgad et al., 2021).

4.9.2 Data Preprocessing

After collecting the dataset, it was carefully observed. The dataset was scanned for corrupted files, but none was found. After analyzing the dataset, it was noted that all the images in the dataset do not have an identical resolution. Various images as well as their features and resolution were carefully observed. Then it was concluded that a center crop of 256x256 pixels will result in retaining most features of each image without exceeding the minimum possible dimension along with height or width.

Also, each mask has three channels. The first channel of each mask contains the class value of each pixel belonging to 13 nucleus types. To simplify our task, the values of all the pixels that represent any type of nucleus were changed to 1, and the rest of the pixel values were changed to 0. Examples of center cropped image and their corresponding modified mask as well as their visualization was given in figure 4.20.

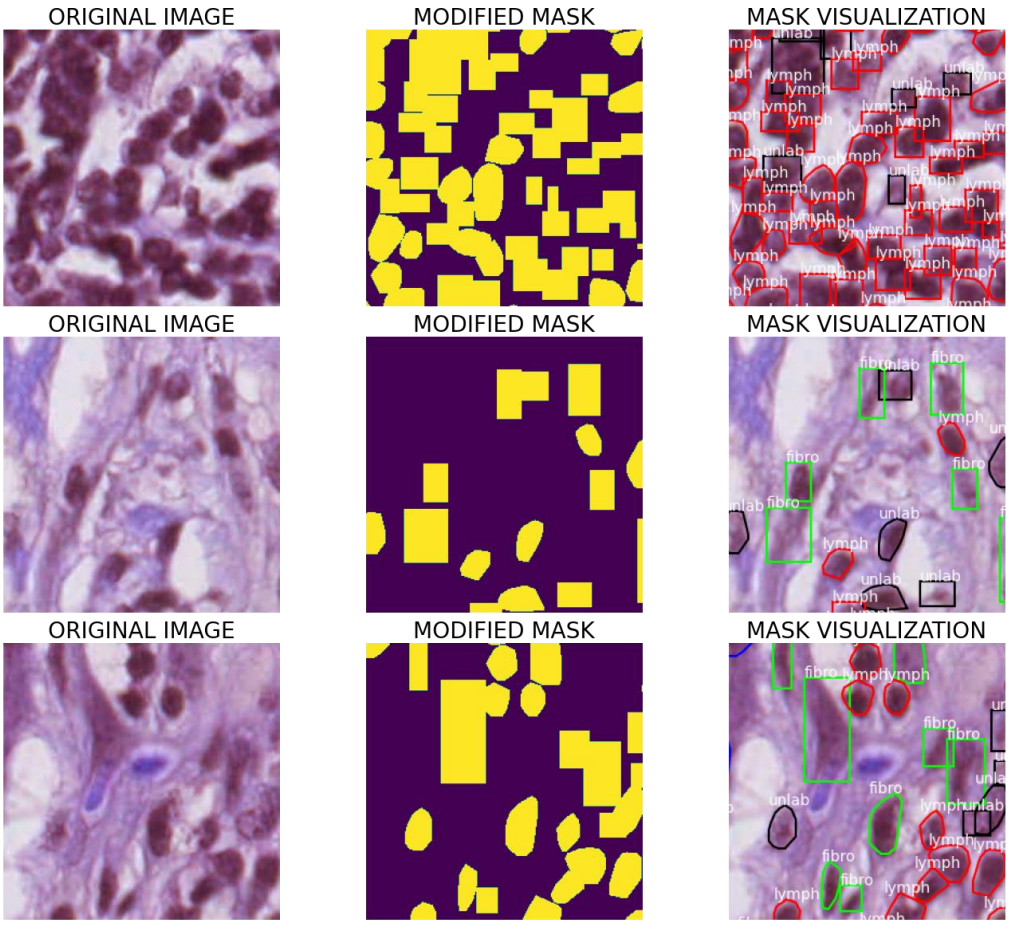


Fig.4.20 Example of original images, modified mask, and mask visualization after 256x256 center cropping

After center cropping and mask modifying, the dataset was divided into test and train datasets using split-folders library. 90% of the dataset was used for training while for the testing purpose 10% of the dataset was used. Then the images of the test and training dataset were copied to various folders according to the reference of the keras image data generator.

The images were then loaded using the Keras image data generator. Image augmentation such as horizontal flip and vertical flip were enabled inside the image data generator. Also, fill mode was set to “reflect”.

Also, for each backbone of UNET, the image must be processed according to the backbone. The function “get_preprocessing” provided by the “Segmentation Model” library, was used for this purpose and was implemented along with the image data generator function.

4.9.3 Defining deep learning models and compiling them

UNET architecture was implemented using the “Segmentation Models” library. Three backbones were used. They are “Inception-V3”, “ResNet-34”, and “Efficient Net B2”. Each backbone was initialized using ImageNet weights for the purpose of transfer learning.

Other hyperparameters and their values are given in table 4.1.

4.1 Hyperparameters and their corresponding values.

Hyper-parameters	Value
Optimizer	Adam
Learning Rate	0.00005
Loss Function	Combination of Focal and Jaccard loss
Batch Size	8
Steps per epoch for training	196
Steps per epoch for Validation	21
Epoch	80

2 metrics were used for evaluation. They are the IOU score and F-score which were also known as Jaccard Coefficient and Dice Co-efficient respectively.

4.9.4 Model Training

After defining the models, each model was trained on the training dataset. At the same time after training the performance was validated on the test dataset.

4.9.5 Evaluation and Retraining

After training the training curve and validation curve for the loss function and metrics were observed. If it was seen that a model isn't performing well after training, then the hyperparameters were tuned and the model was retrained again. This tuning and retraining process continues until the model achieves satisfactory performance.

4.9.6 Performance Comparison and Prediction

After achieving satisfactory results, all the models were compared with each other. Then the models were used for prediction.

Chapter V

Result and Discussion

Three backbones were used. They are Inception-V3, EfficientNet B2, and ResNet-34. Their training and validation accuracy, loss, and F-score after training for 80 epochs are given in table 5.1.

Table 5.1 Comparison of IOU Score, Focal Jaccard loss, and F-Score of all the models

UNet Backbone		**IOU Score**	**Focal Jaccard Loss**	**F-Score**
Inception V3	Training	0.8494	0.1743	0.9178
	Validation	0.7020	0.3767	0.8220
EfficientNet B2	Training	0.8317	0.1950	0.9072
	Validation	0.7060	0.3684	0.8253
ResNet-34	Training	0.8539	0.1694	0.9205
	Validation	0.7019	0.3798	0.8223

The training and validation curves for the IOU score for the Inception-V3 backbone are given in figure 5.1.

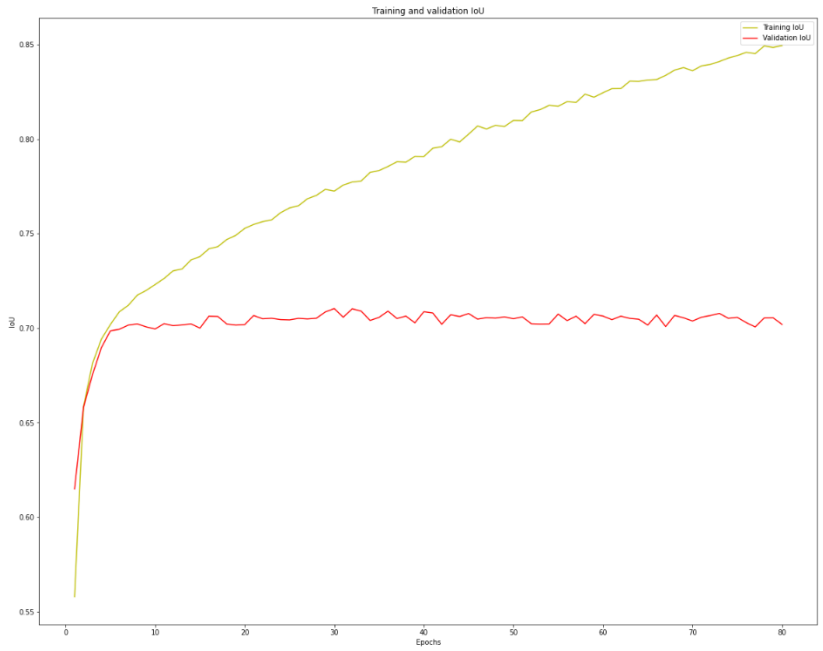


Fig.5.1 The training and validation curves for the IOU score for the Inception-V3

The training and validation curves for the loss function for the Inception-V3 backbone are given in figure 5.2.

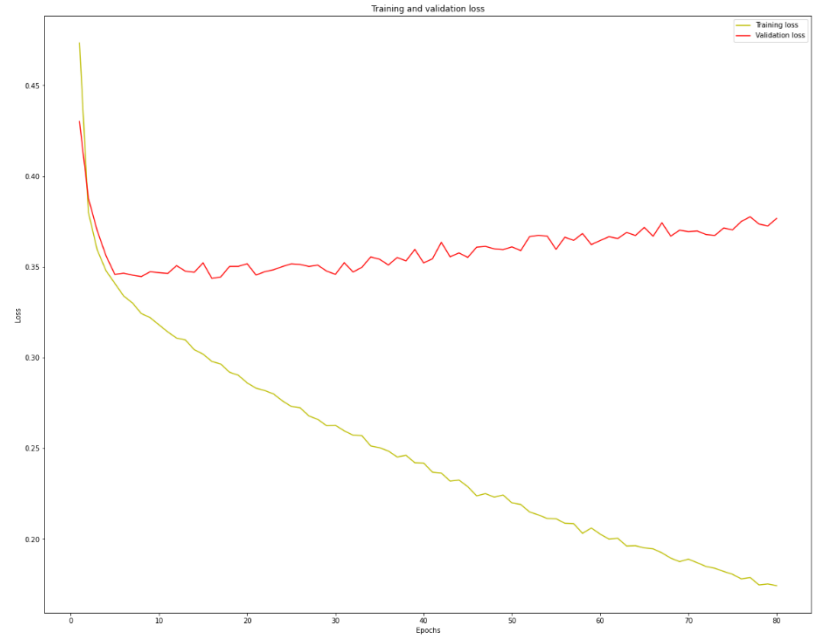


Fig.5.2 The training and validation curves for the loss function for the Inception-V3

The training and validation curves for the F-score for the Inception-V3 backbone are given in figure 5.3.

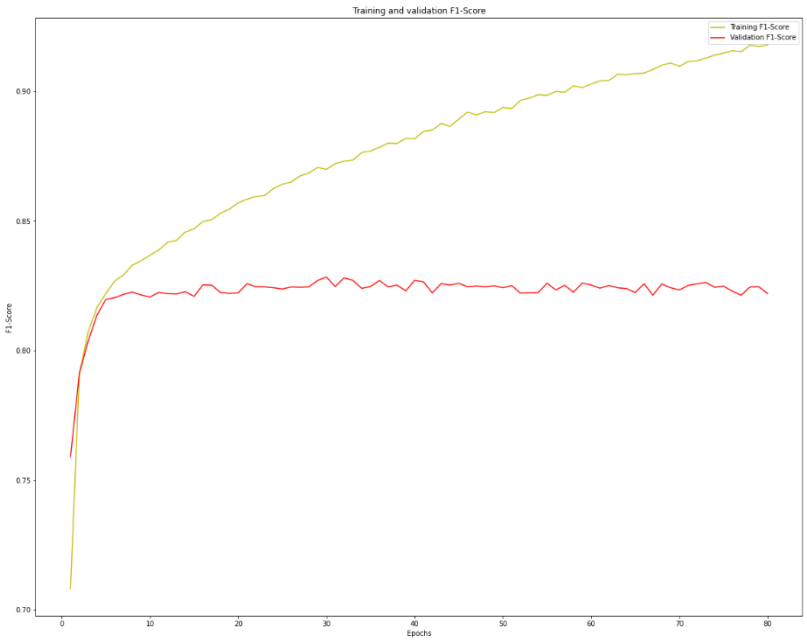


Fig.5.3 The training and validation curves for the F-score for the Inception-V3

The training and validation curves for the IOU score for the EfficientNet B2 backbone are given in figure 5.4.

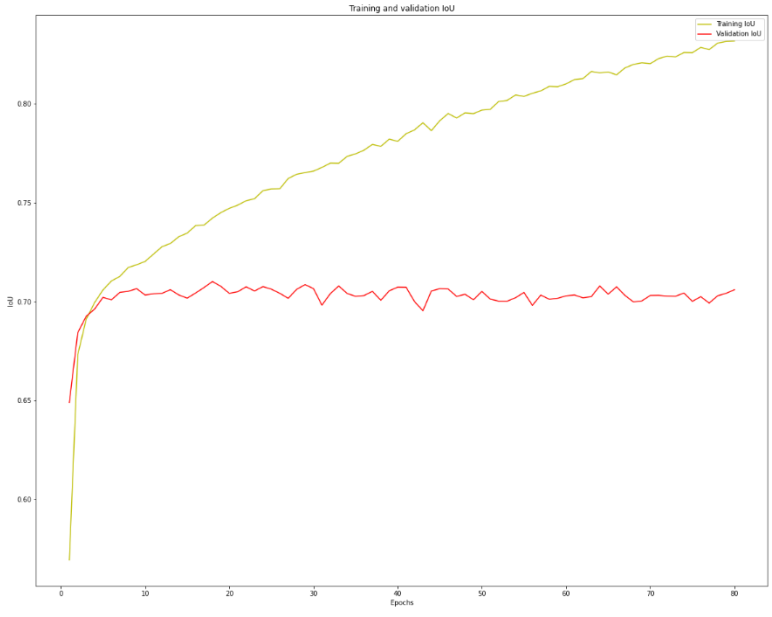


Fig.5.4 The training and validation curves for the IOU score for the EfficientNet B2

The training and validation curves for the loss function for the EfficientNet B2 backbone are given in figure 5.5.

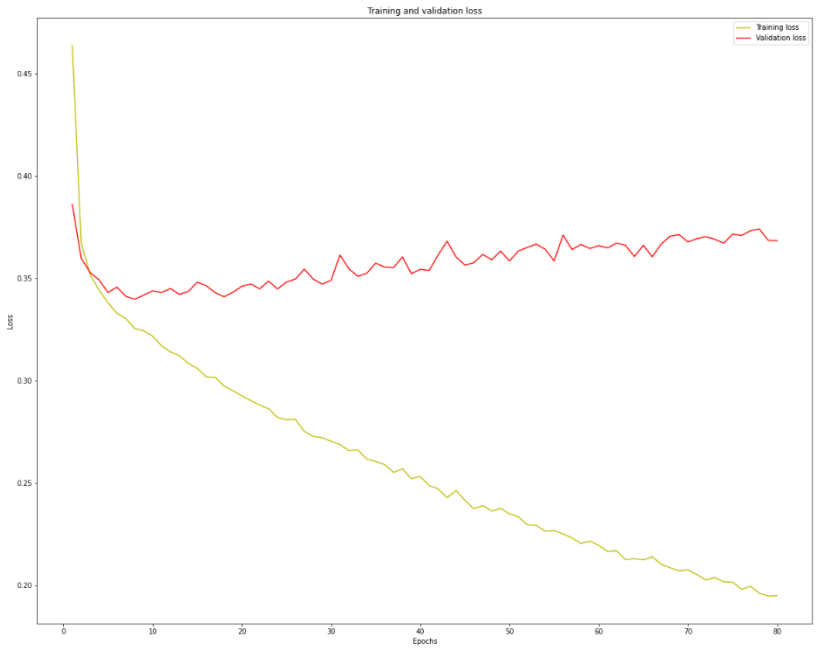


Fig.5.5 The training and validation curves for the loss function for the EfficientNet B2

The training and validation curves for the F-score for the EfficientNet B2 backbone are given in figure 5.6.

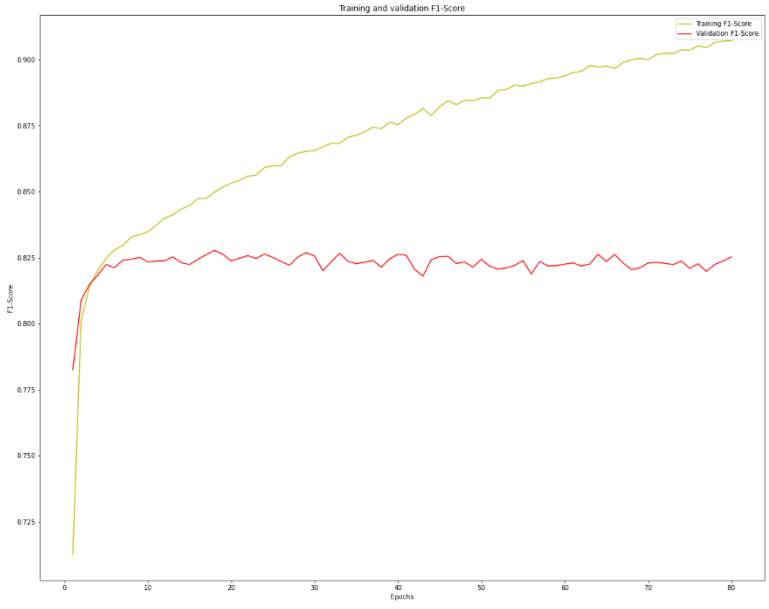


Fig.5.6 The training and validation curves for the F-score for the EfficientNet B2

The training and validation curves for the IOU score for the ResNet-34 backbone are given in figure 5.7.

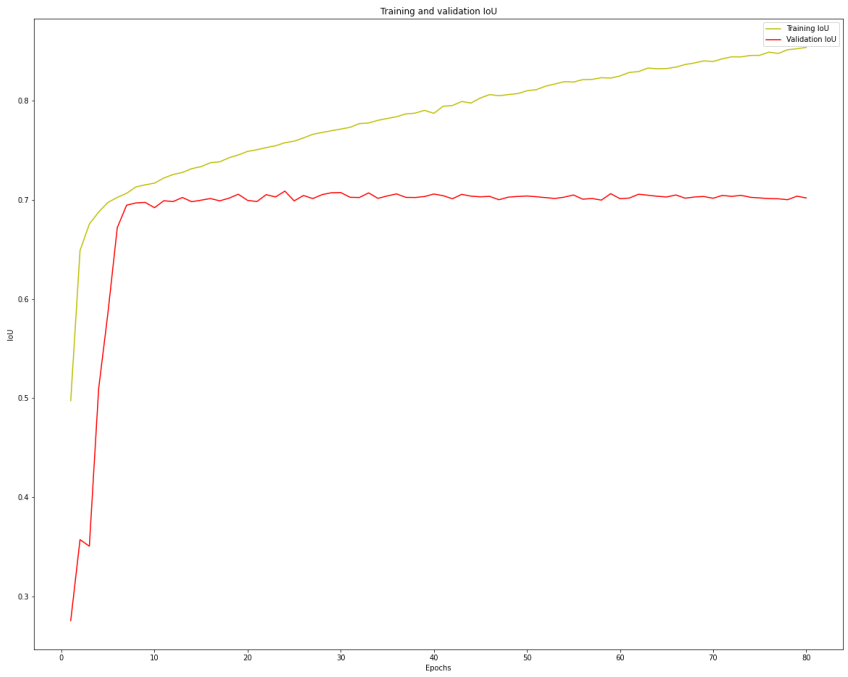


Fig.5.7 The training and validation curves for the IOU score for the ResNet-34

The training and validation curves for the loss function for the ResNet-34 backbone are given in figure 5.8.

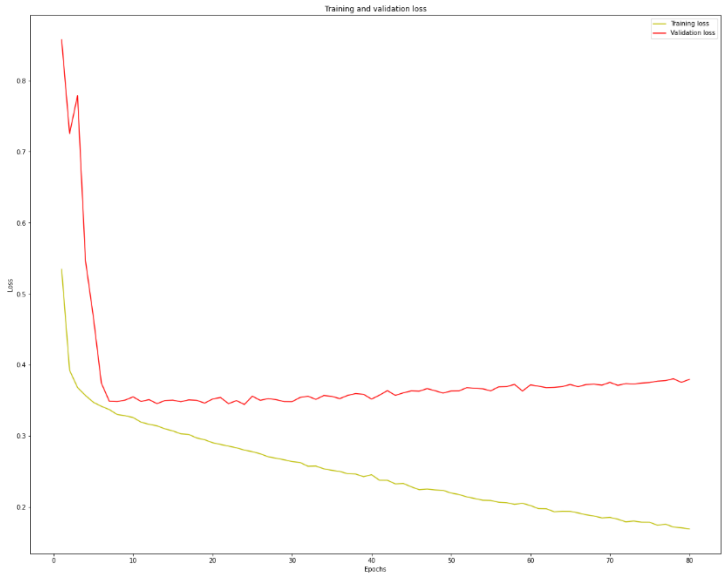


Fig.5.8 The training and validation curves for the loss function for the ResNet-34

The training and validation curves for the F-Score for the ResNet-34 backbone are given in figure 5.9.

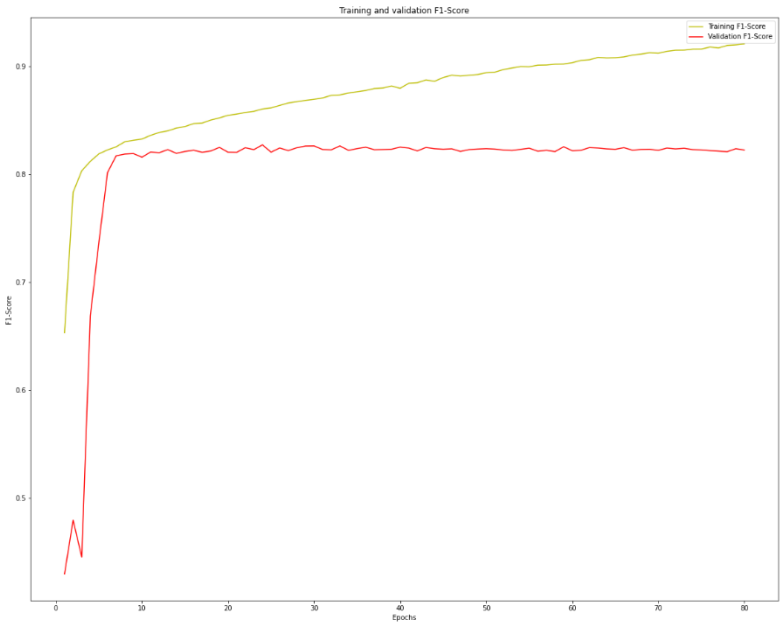


Fig.5.9 The training and validation curves for the F-Score for the ResNet-34

Among all the segmentation models, RedNet-34 achieves a maximum training IOU score of 0.8539. But the highest validation IOU score 70.60% is achieved by EfficientNet B2. In terms of F-Score or dice co-efficient, the ResNet-34 performs best during training which achieved 92.05% but EfficienmtNet B2 performs better during validation where it achieves 82.53%. Inception-V3 achieves the lowest loss value of 17.43% but again EfficientNet B2 performs better during validation and achieved the lowest loss of 36.84%. So, EffientNet B2 performs best during validation and ResNet-34 performs best during training. So, EfficientNet B2 is more generalized than other models.

Prediction using Inception-V3 is shown in figure 5.10.

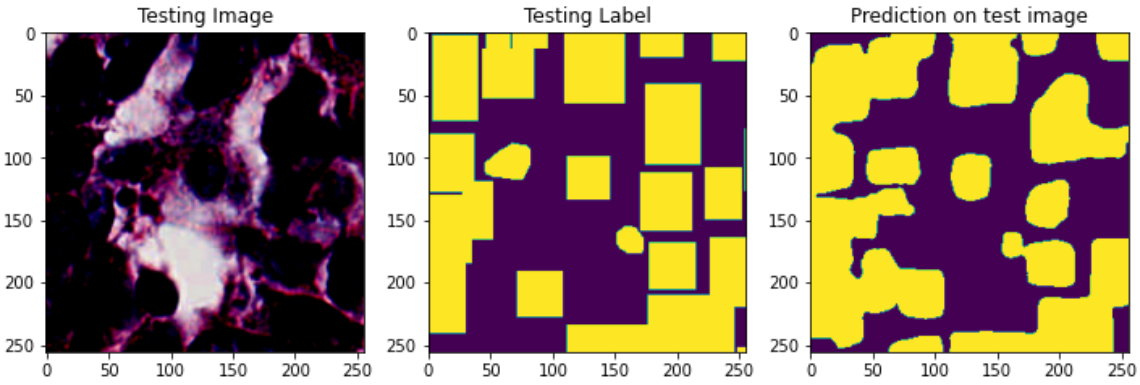


Fig.5.10 Prediction using Inception-V3. (From the left, actual image, actual mask, predicted mask)

Prediction using EfficientNet-B2 is given in figure 5.11.

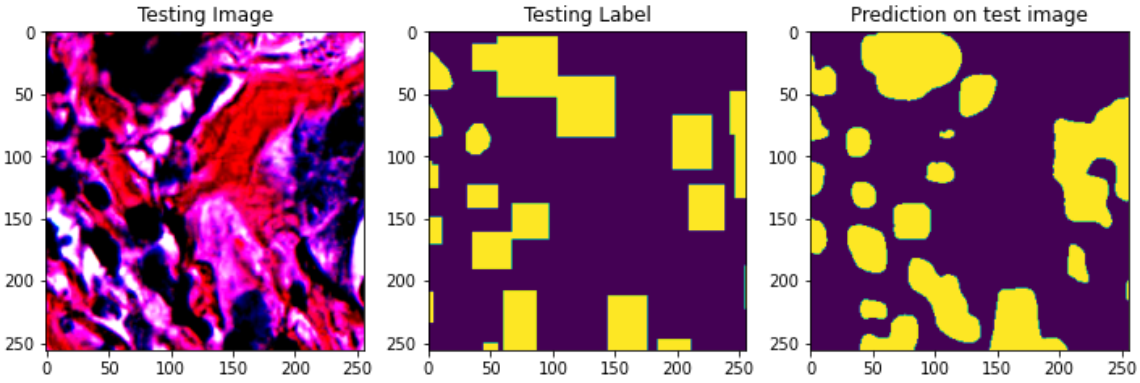


Fig.5.11 Prediction using EfficientNet-B2. (From the left, actual image, actual mask, predicted mask)

Prediction using ResNet-34 is given in figure 5.12.

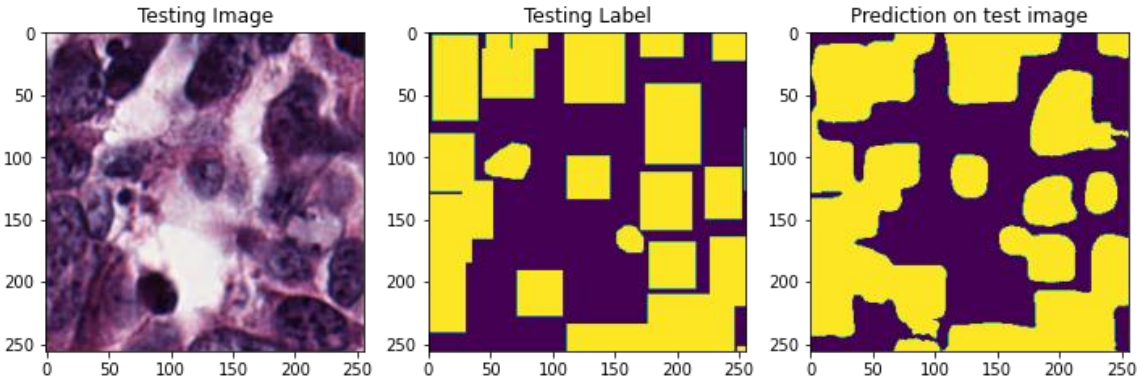


Fig.5.12 Prediction using ResNet-34. (From the left, actual image, actual mask, predicted mask)

Chapter VI

Conclusion and Recommendation

In this work, we have collected data, pre-process them, and then trained 3 segmentation models based on UNet which is a special type of CNN. We have used 3 backbones initialized with pre-trained ImageNet weights. Among all the models, ResNet-34 performs best but when it comes to validation EfficientNet B2 performs best. Which indicates that it is a more generalized model and gave the best performance. But this is a binary segmentation, so we can only get the region of nuclei. We were not able to train more advanced neural networks due to hardware and other limitation. But more advanced networks can be used for multiclass semantic segmentation, where not only the nuclei region but also the type of each nucleus can be predicted.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., others, 2016. Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283.

Advanced guide to inception V3 | cloud TPU | google cloud, n.d. . Google.

Akram, M., Iqbal, M., Daniyal, M., Khan, A.U., 2017. Awareness and current knowledge of breast cancer. Biol Res 50, 33. https://doi.org/10.1186/s40659-017-0140-9

Amgad, M., Atteya, L.A., Hussein, H., Mohammed, K.H., Hafiz, E., Elsebaie, Maha A T, Alhusseiny, A.M., AlMoslemany, M.A., Elmatboly, A.M., Pappalardo, P.A., Sakr, R.A., Mobadersany, P., Rachid, A., Saad, A.M., Alkashash, A.M., Ruhban, I.A., Alrefai, A., Elgazar, N.M., Abdulkarim, A., Farag, A.-A., Etman, A., Elsaeed, A.G., Alagha, Y., Amer, Y.A., Raslan, A.M., Nadim, M.K., Elsebaie, Mai A T, Ayad, A., Hanna, L.E., Gadallah, A.M., Elkady, M., Drumheller, B., Jaye, D., Manthey, D., Gutman, D.A., Elfandy, H., Cooper, L.A.D., 2021. NuCLS: A scalable crowdsourcing, deep learning approach and dataset for nucleus classification, localization and segmentation. CoRR abs/2102.09099.

Amgad, M., Elfandy, H., Hussein, H., Atteya, L.A., Elsebaie, Mai A T, Abo Elnasr, L.S., Sakr, R.A., Salem, H.S.E., Ismail, A.F., Saad, A.M., Ahmed, J., Elsebaie, Maha A T, Rahman, M., Ruhban, I.A., Elgazar, N.M., Alagha, Y., Osman, M.H., Alhusseiny, A.M., Khalaf, M.M., Younes, A.-A.F., Abdulkarim, A., Younes, D.M., Gadallah, A.M., Elkashash, A.M., Fala, S.Y., Zaki, B.M., Beezley, J., Chittajallu, D.R., Manthey, D., Gutman, D.A., Cooper, L.A.D., 2019. Structured crowdsourcing enables convolutional segmentation of histology images. Bioinformatics 35, 3461–3467. https://doi.org/10.1093/bioinformatics/btz083

Araújo, T., Aresta, G., Castro, E., Rouco, J., Aguiar, P., Eloy, C., Polónia, A., Campilho, A., 2017. Classification of breast cancer histology images using Convolutional Neural Networks. PLOS ONE 12, e0177544. https://doi.org/10.1371/journal.pone.0177544

Arel, I., Rose, D.C., Karnowski, T.P., 2010. Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]. IEEE Computational Intelligence Magazine 5, 13–18. https://doi.org/10.1109/MCI.2010.938364

Ataollahi, M.R., Sharifi, J., Paknahad, M.R., Paknahad, A., 2015. Breast cancer and associated factors: a review. J Med Life 8, 6–11.

Begum, S.A., Mahmud, T., Rahman, T., Zannat, J., Khatun, F., Nahar, K., Towhida, M., Joarder, M., Harun, A., Sharmin, F., 2019. Knowledge, Attitude and Practice of Bangladeshi Women towards Breast Cancer: A Cross Sectional Study. Mymensingh Med J 28, 96–104.

Bengio, Y., 2009. Learning Deep Architectures for AI. Foundations and Trends® in Machine Learning 2, 1–127. https://doi.org/10.1561/2200000006

Bozinovski, S., 2020. Reminder of the first paper on transfer learning in neural networks, 1976. Informatica 44.

Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Brancati, N., Anniciello, A.M., Pati, P., Riccio, D., Scognamiglio, G., Jaume, G., de Pietro, G., di Bonito, M., Foncubierta, A., Botti, G., others, 2021. BRACS: A Dataset for BReAst Carcinoma Subtyping in H&E Histology Images. arXiv preprint arXiv:2111.04740.

Chollet, F., others, 2015. Keras.

Convolutional Neural Network, 2022. . Wikipedia.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. ImageNet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition. pp. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

Dumoulin, V., Visin, F., 2016. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285.

EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling [WWW Document], 2019. . Google.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778.

Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. Comput Sci Eng 9, 90–95.

Jaccard, P., 1912. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. New Phytologist 11, 37–50. https://doi.org/10.1111/j.1469-8137.1912.tb05611.x

Jordan, M.I., Mitchell, T.M., 2015. Machine learning: Trends, perspectives, and prospects. Science (1979) 349, 255–260. https://doi.org/10.1126/science.aaa8415

Koziarski, M., Cyganek, B., Olborski, B., Antosz, Z., Żydak, M., Kwolek, B., Wasowicz, P., Bukała, A., Swadźba, J., Sitkowski, P., 2021. DiagSet: a dataset for prostate cancer histopathological image classification. arXiv preprint arXiv:2105.04014.

Kumar, D., Wong, A., Clausi, D.A., 2015. Lung Nodule Classification Using Deep Features in CT Images, in: 2015 12th Conference on Computer and Robot Vision. pp. 133–138. https://doi.org/10.1109/CRV.2015.25

Lee June-Goo Jun Sanghoon, C.Y.-W.L.H.K.G.B.S.J.B.K.N., 2017. Deep Learning in Medical Imaging: General Overview. kjr 18, 570–584. https://doi.org/10.3348/kjr.2017.18.4.570

Litjens, G., Sánchez, C.I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., Hulsbergen - van de Kaa, C., Bult, P., van Ginneken, B., van der Laak, J., 2016. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. Scientific Reports 6, 26286. https://doi.org/10.1038/srep26286

Mathew Amitha and Amudha, P. and S.S., 2021. Deep Learning Techniques: An Overview, in: Hassanien Aboul Ella and Bhatnagar, R. and D.A. (Ed.), Advanced Machine Learning Technologies and Applications. Springer Singapore, Singapore, pp. 599–608.

Moeskops, P., Viergever, M.A., Mendrik, A.M., de Vries, L.S., Benders, M.J.N.L., Išgum, I., 2016. Automatic segmentation of MR brain images with a convolutional neural network. IEEE Trans Med Imaging 35, 1252–1261.

Nazeri, K., Aminpour, A., Ebrahimi, M., 2018. Two-stage convolutional neural network for breast cancer histology image classification, in: International Conference Image Analysis and Recognition. pp. 717–726.

Pereira, S., Pinto, A., Alves, V., Silva, C.A., 2016. Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images. IEEE Transactions on Medical Imaging 35, 1240–1251. https://doi.org/10.1109/TMI.2016.2538465

Rakhlin, A., Shvets, A., Iglovikov, V., Kalinin, A.A., 2018. Deep convolutional neural networks for breast cancer histology image analysis, in: International Conference Image Analysis and Recognition. pp. 737–744.

Ray, S., 2019. A Quick Review of Machine Learning Algorithms, in: 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). IEEE, pp. 35–39. https://doi.org/10.1109/COMITCon.2019.8862451

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 234–241.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Spanhol, F.A., Oliveira, L.S., Petitjean, C., Heutte, L., 2016. A Dataset for Breast Cancer Histopathological Image Classification. IEEE Transactions on Biomedical Engineering 63, 1455–1462. https://doi.org/10.1109/TBME.2015.2496264

Split-folders, 2018. . PyPI.

Sung, H., Ferlay, J., Siegel, R.L., Laversanne, M., Soerjomataram, I., Jemal, A., Bray, F., 2021. Global Cancer Statistics 2020: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries. CA: A Cancer Journal for Clinicians 71, 209–249. https://doi.org/10.3322/caac.21660

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2818–2826.

Tan, M., Le, Q., 2019. Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning. pp. 6105–6114.

Tensorflow, n.d. Tensorflow/tensorflow: An open source machine learning framework for everyone. GitHub.

van Rossum, G., Drake, F.L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., 2018. Deep Learning for Computer Vision: A Brief Review. Computational Intelligence and Neuroscience 2018, 7068349. https://doi.org/10.1155/2018/7068349

Wang, D., Khosla, A., Gargeya, R., Irshad, H., Beck, A.H., 2016. Deep Learning for Identifying Metastatic Breast Cancer. arXiv preprint arXiv:1606.05718. https://doi.org/10.48550/ARXIV.1606.05718

Wu, N., Phang, J., Park, J., Shen, Y., Huang, Z., Zorin, M., Jastrzebski, S., Févry, T., Katsnelson, J., Kim, E., others, 2019. Deep neural networks improve radiologists' performance in breast cancer screening. IEEE Trans Med Imaging 39, 1184–1194.

Yakubovskiy, P., 2019. Segmentation Models. GitHub repository.

Yosinski, J., Clune, J., Bengio, Y., Lipson, H., 2014. How transferable are features in deep neural networks? Adv Neural Inf Process Syst 27.

Apendices

Preprocessing Code

In [2]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
from tqdm import tqdm
import collections as c
```

In [3]:
```
!mkdir final
!mkdir final/images/
!mkdir final/masks/
!mkdir final/vis/

img_path = 'images/'
mask_path = "masks/"
vis_path ='visualization/'

des_img = "final/images/"
des_mask ="final/masks/"
des_vis = "final/vis/"
file_list = os.listdir(mask_path)

print(file_list[0])
```

```
TCGA-BH-A0B3-DX1_id-5ea40a41ddda5f839898cda9_left-86611_top-43407_bottom-437
45_right-86902.png
```

In [4]:
```
def center_crop(dir_,new_h,new_w):
    img = cv2.imread(dir_)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    height = img.shape[0]
    width = img.shape[1]
    left = int((width - new_w)/2)
    top = int((height - new_h)/2)
    right = int((width + new_w)/2)
    bottom = int((height + new_h)/2)
    cropped_image = img[top:bottom, left:right]
    return cropped_image
new_h = 256
new_w = 256
```

In [ ]:
```
for i in range(len(file_list)):
    if file_list[i].endswith(".png"):
        print(i)
        img = center_crop(img_path + file_list[i],new_h,new_w)
        mask = center_crop(mask_path + file_list[i],new_h,new_w)
        mask = mask[:,:,0]
        for k in range(len(mask)):
            for j in range(len(mask[k])):
                if(mask[k][j]==253):
                    mask[k][j]=0
                #elif(mask[k][j]==6 or mask[k][j]==1):#labelling mitotic f #          mask[k][j]=1

                #elif(mask[k][j]==2 or mask[k][j]==7 or mask[k][j]==5):#lab #        mask[k][j] = 2

                #elif(mask[k][j]==3 or mask[k][j]==4): # mask[k][j] = 3

                #elif(mask[k][j]==10 or mask[k][j]==12 or mask[k][j]==8 or #         mask[k][j] = 4

                else:
                    mask[k][j]=1

        vis = center_crop(vis_path + file_list[i],new_h,new_w)

        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        vis = cv2.cvtColor(vis, cv2.COLOR_RGB2BGR)

        cv2.imwrite(des_img+file_list[i],img)
        cv2.imwrite(des_mask+file_list[i],mask)
        cv2.imwrite(des_vis+file_list[i],vis)
```

In [6]:
```
!pip install split-folders

import splitfolders

input_dir = "final/"
output_dir = "main_dataset/"

splitfolders.ratio(input_dir, output=output_dir,seed=42, ratio=(.9, .1), gro
```

```
Requirement already satisfied: split-folders in /home/emroze/vir/lib/python
3.10/site-packages (0.5.1)
WARNING: You are using pip version 22.0.3; however, version 22.0.4 is availa
ble.
You should consider upgrading via the '/home/emroze/vir/bin/python -m pip in
stall --upgrade pip' command.
Copying files: 5232 files [00:00, 6694.35 files/s]
```

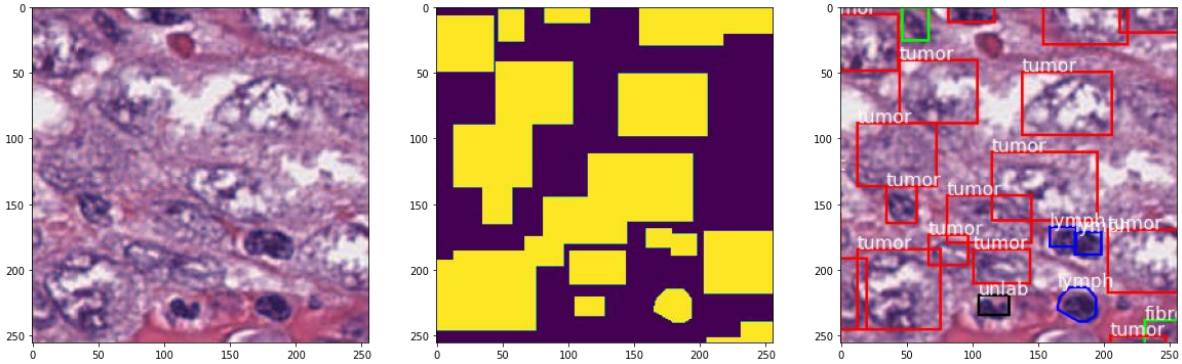
In [7]:
```
!mkdir -p Final_dataset/train_img/train
!mkdir -p Final_dataset/train_mask/train
!mkdir -p Final_dataset/train_vis/train

!mkdir -p Final_dataset/val_img/val
!mkdir -p Final_dataset/val_mask/val
!mkdir -p Final_dataset/val_vis/val

!cp  main_dataset/train/images/*.png  Final_dataset/train_img/train/
!cp  main_dataset/train/masks/*.png  Final_dataset/train_mask/train/
!cp main_dataset/train/vis/*.png Final_dataset/train_vis/train

!cp  main_dataset/val/images/*.png  Final_dataset/val_img/val
!cp  main_dataset/val/masks/*.png  Final_dataset/val_mask/val
!cp main_dataset/val/vis/*.png Final_dataset/val_vis/val
```

In [8]:
```
list_ = os.listdir("Final_dataset/train_img/train/")
print(len(list_))
list_ = os.listdir("Final_dataset/train_mask/train/")
print(len(list_))
list_ = os.listdir("Final_dataset/train_vis/train/")
print(len(list_))

list_ = os.listdir("Final_dataset/val_img/val/")
print(len(list_))
list_ = os.listdir("Final_dataset/val_mask/val/")
print(len(list_))
list_ = os.listdir("Final_dataset/val_vis/val/")
print(len(list_))
```

```
1569
1569
1569
175
175
175
```

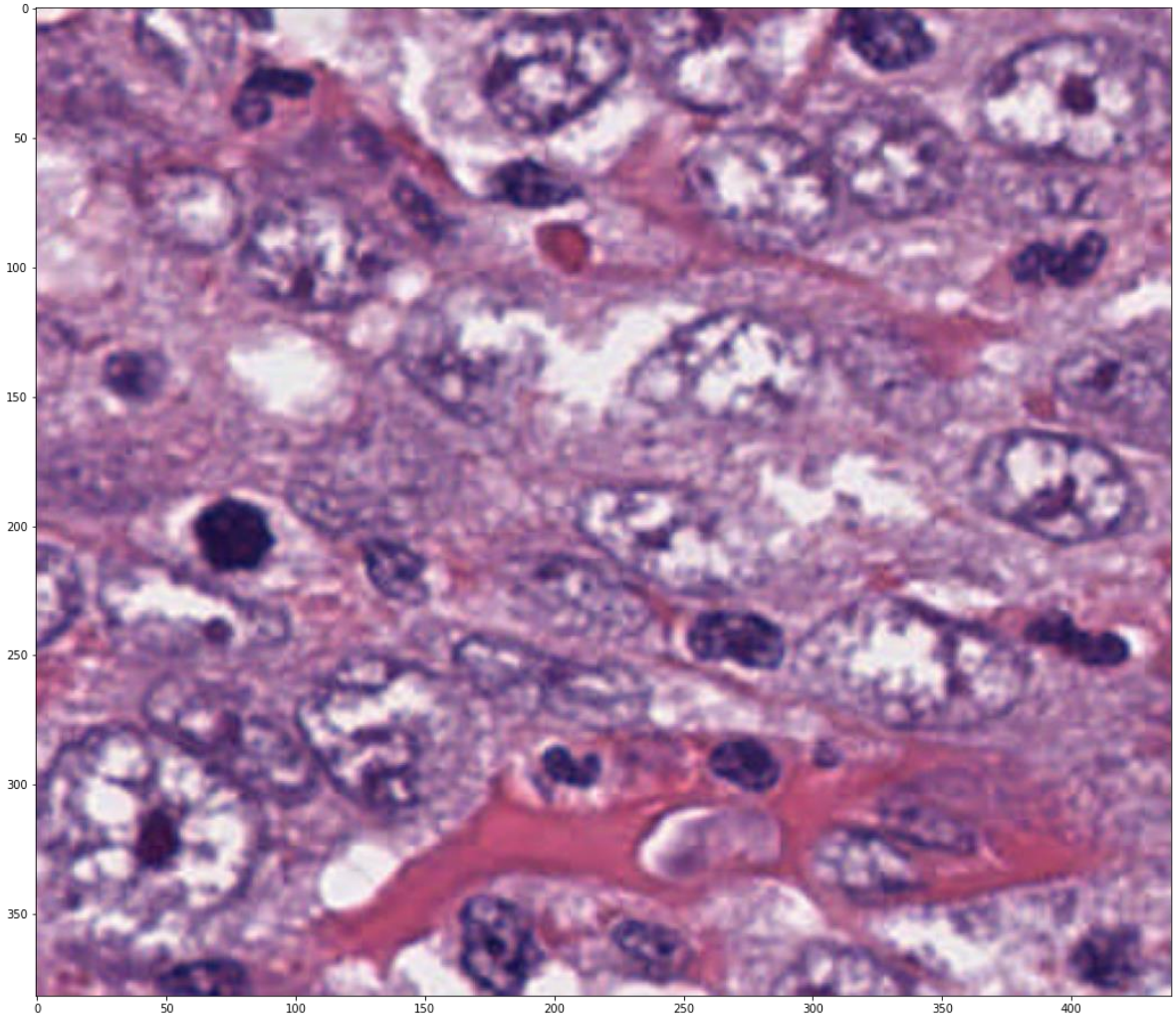
In [9]:
```
trn_img_dir = "Final_dataset/train_img/train/"
trn_mask_dir = "Final_dataset/train_mask/train/"
trn_vis_dir = "Final_dataset/train_vis/train/"

img = os.listdir(trn_img_dir)
mask =os.listdir(trn_mask_dir)
vis = os.listdir(trn_vis_dir)

img_num = 298

print(len(img))
print(len(mask))
print(len(vis))
print(trn_img_dir+img[img_num])
im = cv2.imread(trn_img_dir+img[img_num])
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
mask = cv2.imread(trn_mask_dir+img[img_num],0)
print(np.unique(mask,return_counts=True))
vis = cv2.imread(trn_vis_dir+img[img_num])
vis = cv2.cvtColor(vis, cv2.COLOR_BGR2RGB)

import matplotlib as mpl
mpl.rcParams["figure.figsize"]=(20,16)
f, axarr = plt.subplots(1,3)
axarr[0].imshow(im)
axarr[1].imshow(mask)
axarr[2].imshow(vis)
plt.show()
```

```
1569
1569
1569
Final_dataset/train_img/train/TCGA-EW-A6SB-DX1_id-5ea40a8eddda5f839899118b_l
eft-36108_top-58095_bottom-58398_right-36456.png
(array([0, 1], dtype=uint8), array([33656, 31880]))
```



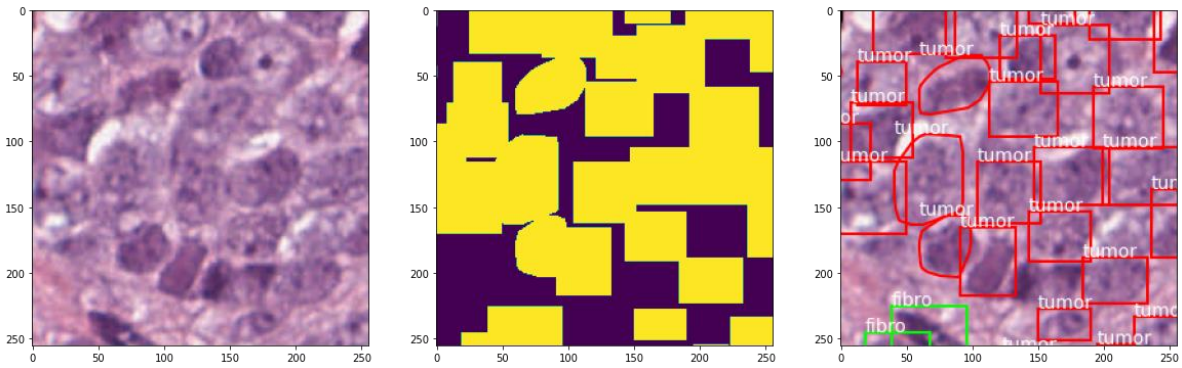
In [10]:
```
im = cv2.imread("images/TCGA-EW-A6SB-DX1_id-5ea40a8eddda5f839899118b_left-36
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
plt.imshow(im)


im = center_crop("masks/TCGA-EW-A6SB-DX1_id-5ea40a8eddda5f839899118b_left-36
print(np.unique(im[:,:,0],return_counts=True))
```

```
(array([  1,   2,   3,  99, 253], dtype=uint8), array([29401,   900,  1234,
345, 33656]))
```


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

In [11]:
```
val_img_dir = "Final_dataset/val_img/val/"
val_mask_dir = "Final_dataset/val_mask/val/"
val_vis_dir = "Final_dataset/val_vis/val/"

img = os.listdir(val_img_dir)
mask =os.listdir(val_mask_dir)
vis = os.listdir(val_vis_dir)

img_num = 149

print(len(img))
print(len(mask))
print(len(vis))

im = cv2.imread(val_img_dir+img[img_num])
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
mask = cv2.imread(val_mask_dir+img[img_num],0)
print(np.unique(mask,return_counts=True))
print(mask.shape)
vis = cv2.imread(val_vis_dir+img[img_num])
vis = cv2.cvtColor(vis, cv2.COLOR_BGR2RGB)

import matplotlib as mpl
mpl.rcParams["figure.figsize"]=(20,16)
f, axarr = plt.subplots(1,3)
axarr[0].imshow(im)
axarr[1].imshow(mask)
axarr[2].imshow(vis)
plt.show()
```

```
175
175
175
(array([0, 1], dtype=uint8), array([22682, 42854]))
(256, 256)
```



Main code for training and prediction:

In [ ]:
```
from google.colab import drive
#drive.flush_and_unmount()
drive.mount('/content/gdrive',  force_remount=True)
```

```
Mounted at /content/gdrive
```

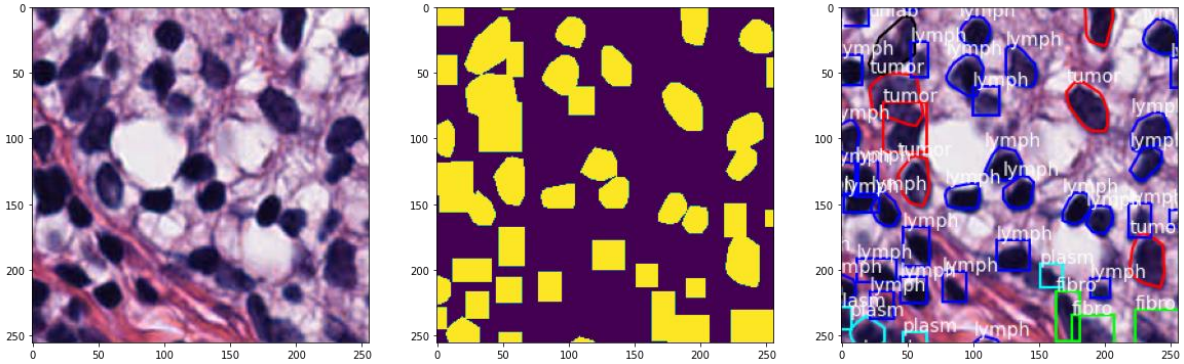
In [ ]:
```
!cp -p /content/gdrive/MyDrive/Colab\ Notebooks/Thesis/Final_dataset.zip /tm
```

In [ ]:
```
!unzip /tmp/Dataset.zip -d /
```

In [ ]:
```
!pip install segmentation_models
!pip uninstall -y h5py

!pip install h5py==2.10.0
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
import segmentation_models as sm
import tensorflow as tf
from tensorflow.keras.metrics import MeanIoU
import random
```

In [ ]:
```
train_img_dir = "/Final_dataset/train_img/train/"
train_mask_dir = "/Final_dataset/train_mask/train/"
train_vis_dir = "/Final_dataset/train_vis/train/"

img_list = os.listdir(train_img_dir)
msk_list = os.listdir(train_mask_dir)

num_images = len(os.listdir(train_img_dir))


img_num = random.randint(0, num_images-1)

img_for_plot = cv2.imread(train_img_dir+img_list[img_num], 1)
print(train_img_dir+img_list[img_num])
img_for_plot = cv2.cvtColor(img_for_plot, cv2.COLOR_BGR2RGB)

mask_for_plot =cv2.imread(train_mask_dir+msk_list[img_num], 0)

vis_for_plot = cv2.imread(train_vis_dir+img_list[img_num], 1)
vis_for_plot = cv2.cvtColor(vis_for_plot, cv2.COLOR_BGR2RGB)

labels, count = np.unique(mask_for_plot, return_counts=True)
print("Mask labels are: ", labels, " and the counts are: ", count)

import matplotlib as mpl
mpl.rcParams["figure.figsize"]=(20,16)
f, axarr = plt.subplots(1,3)
axarr[0].imshow(img_for_plot)
axarr[1].imshow(mask_for_plot)
axarr[2].imshow(vis_for_plot)
plt.show()
```

```
/Final_dataset/train_img/train/TCGA-AN-A0XU-DX1_id-5ea40b25ddda5f83989991fd_
left-26615_top-21755_bottom-22032_right-26896.png
Mask labels are:  [0 1]  and the counts are:  [43955 21581]
```



In [ ]:

```
seed=24
batch_size= 8
n_classes=2

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
from tensorflow.keras.utils import to_categorical

#Use this to preprocess input for transferlearning

BACKBONE = 'inceptionv3'
preprocess_input = sm.get_preprocessing(BACKBONE)

#Define a function to perform additional preprocessing after datagen. #For example, scale images,
convert masks to categorical, etc.

def preprocess_data(img, mask, num_class):
    #Scale images

    #img=scaler.fit_transform(img.reshape(-1,img.shape[-1])).reshape(img. #print(img)

    img = preprocess_input(img)          #Preprocess based on the pretrained backbon #Convert mask to
    one-hot

    mask = to_categorical(mask, num_class)

    return (img,mask)
```

In [ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
def trainGenerator(img_path, mask_path, num_class):
    img_data_gen_args =  dict(horizontal_flip=True,vertical_flip=True,fill_mo

    image_datagen = ImageDataGenerator(**img_data_gen_args)
    mask_datagen = ImageDataGenerator(**img_data_gen_args)

    image_generator = image_datagen.flow_from_directory(
        img_path,
        class_mode = None,
        batch_size = batch_size,
        seed = seed)

    mask_generator = mask_datagen.flow_from_directory(
        mask_path,
        class_mode = None,
        color_mode = 'grayscale',
        batch_size = batch_size,
        seed = seed)

    train_generator = zip(image_generator, mask_generator)

    for (img, mask) in train_generator:
        img, mask = preprocess_data(img, mask, num_class)
        yield (img, mask)



val_img_path = "/Final_dataset/val_img/"
val_mask_path = "/Final_dataset/val_mask/"
val_img_gen = trainGenerator(val_img_path, val_mask_path, num_class=n_classe

train_img_path = "/Final_dataset/train_img/"
train_mask_path = "/Final_dataset/train_mask/"
train_img_gen = trainGenerator(train_img_path, train_mask_path, num_class=n_


!ls /Final_dataset/val_img/val/ | wc -l
!ls /Final_dataset/val_mask/val/ | wc -l
!ls /Final_dataset/train_img/train/ | wc -l
!ls /Final_dataset/train_mask/train/ | wc -l
```

In [ ]:
```
x, y = train_img_gen._next__()
for i in range(0,3):
    image = x[i]
    print("x",i,"= ", x[i])
    mask = np.argmax(y[i], axis=2)
    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.subplot(1,2,2)
    plt.imshow(mask)
    plt.show()

x_val, y_val = val_img_gen._next__()

for i in range(0,3):
    image = x_val[i]
    mask = np.argmax(y_val[i], axis=2)
    plt.subplot(1,2,1)
    plt.imshow(image)
    plt.subplot(1,2,2)
    plt.imshow(mask)
    plt.show()
```

In [ ]:
```
val_image_path = "/Final_dataset/val_img/val/"
train_image_path = "/Final_dataset/train_img/train/"
num_train_imgs = len(os.listdir(train_image_path))
num_val_images = len(os.listdir(val_image_path))
steps_per_epoch = num_train_imgs//batch_size
val_steps_per_epoch = num_val_images//batch_size


IMG_HEIGHT = x.shape[1]
IMG_WIDTH  = x.shape[2]
IMG_CHANNELS = x.shape[3]

n_classes=n_classes

print(num_train_imgs)
print(num_val_images)
print("steps_per_epoch for training: "+str(steps_per_epoch))
print("steps_per_epoch for validation: "+str(val_steps_per_epoch))
```

In [ ]:
```
# define model sm.set_framework('tf.keras')
sm.framework()


metric1 =sm.metrics.IOUScore()
metric2 = sm.metrics.FScore()


model = sm.Unet(BACKBONE, encoder_weights="imagenet",
                input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
                classes=n_classes,
                activation='softmax')
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00005),  los
```

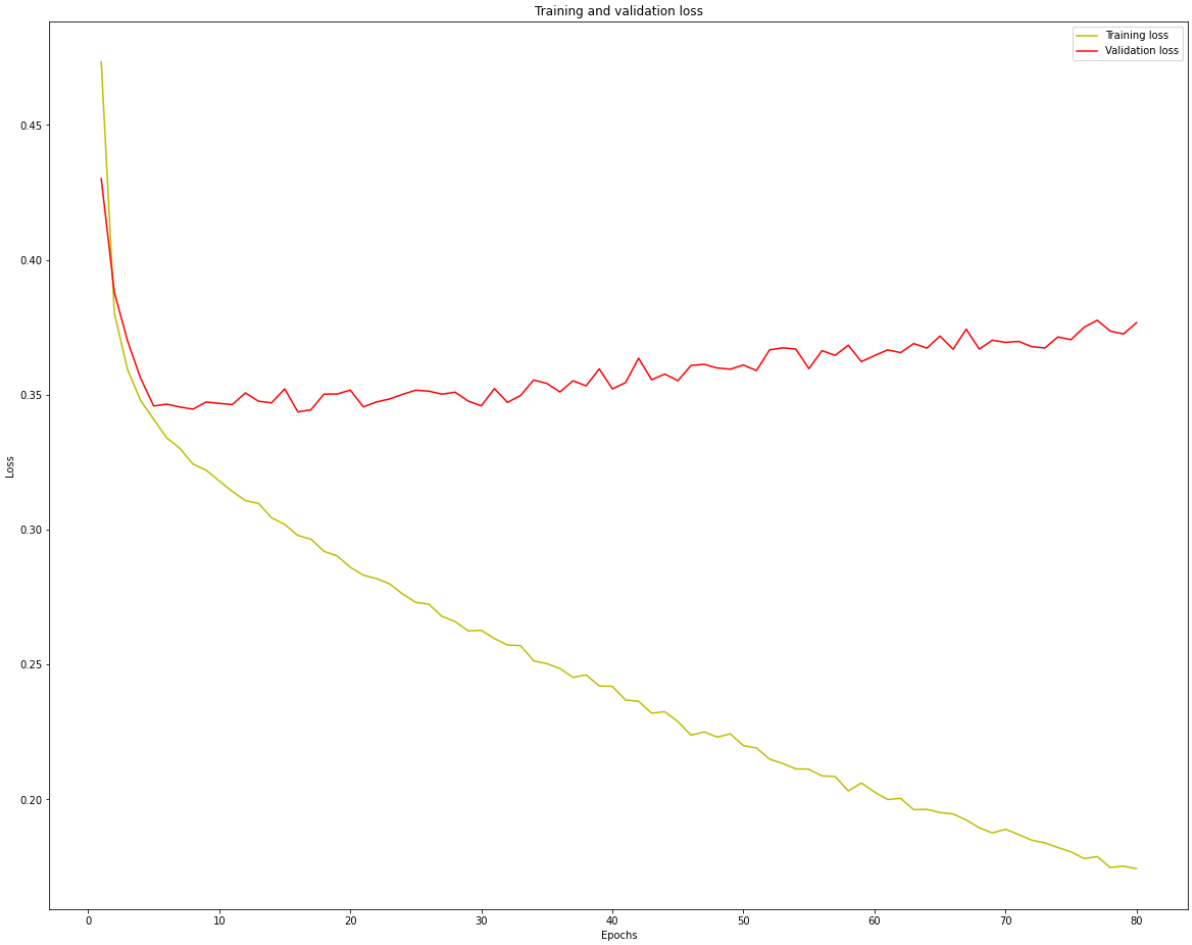
```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 27s 0us/step
87924736/87910968 [==============================] - 27s 0us/step
```

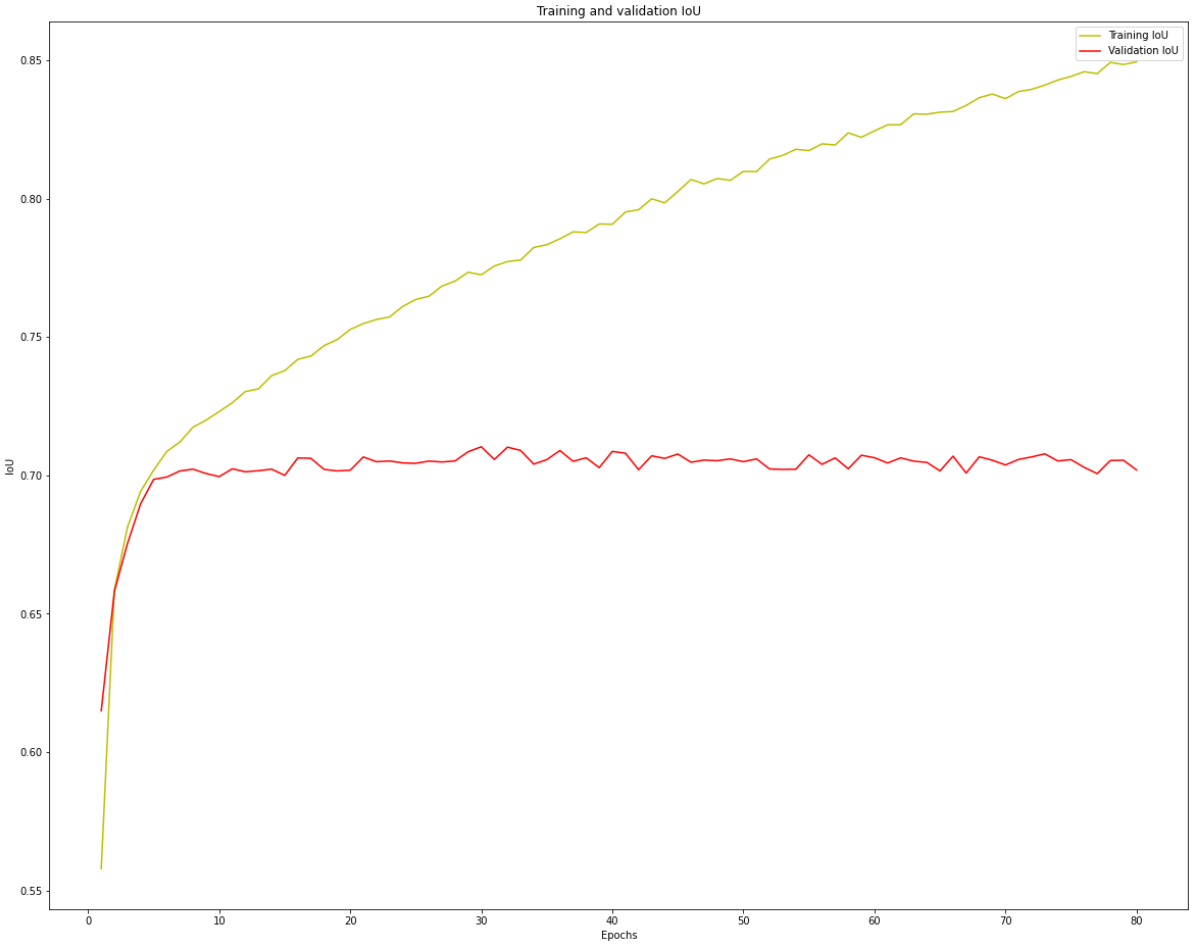
In [ ]:
```
print(model.summary())
print("Input Shape",model.input_shape)
print("Output  Shape",model.output_shape)
```

In [ ]:
```
history=model.fit(train_img_gen,
          steps_per_epoch=steps_per_epoch,
          epochs=80,
          verbose=1,
          validation_data=val_img_gen,
          validation_steps=val_steps_per_epoch
          )
```

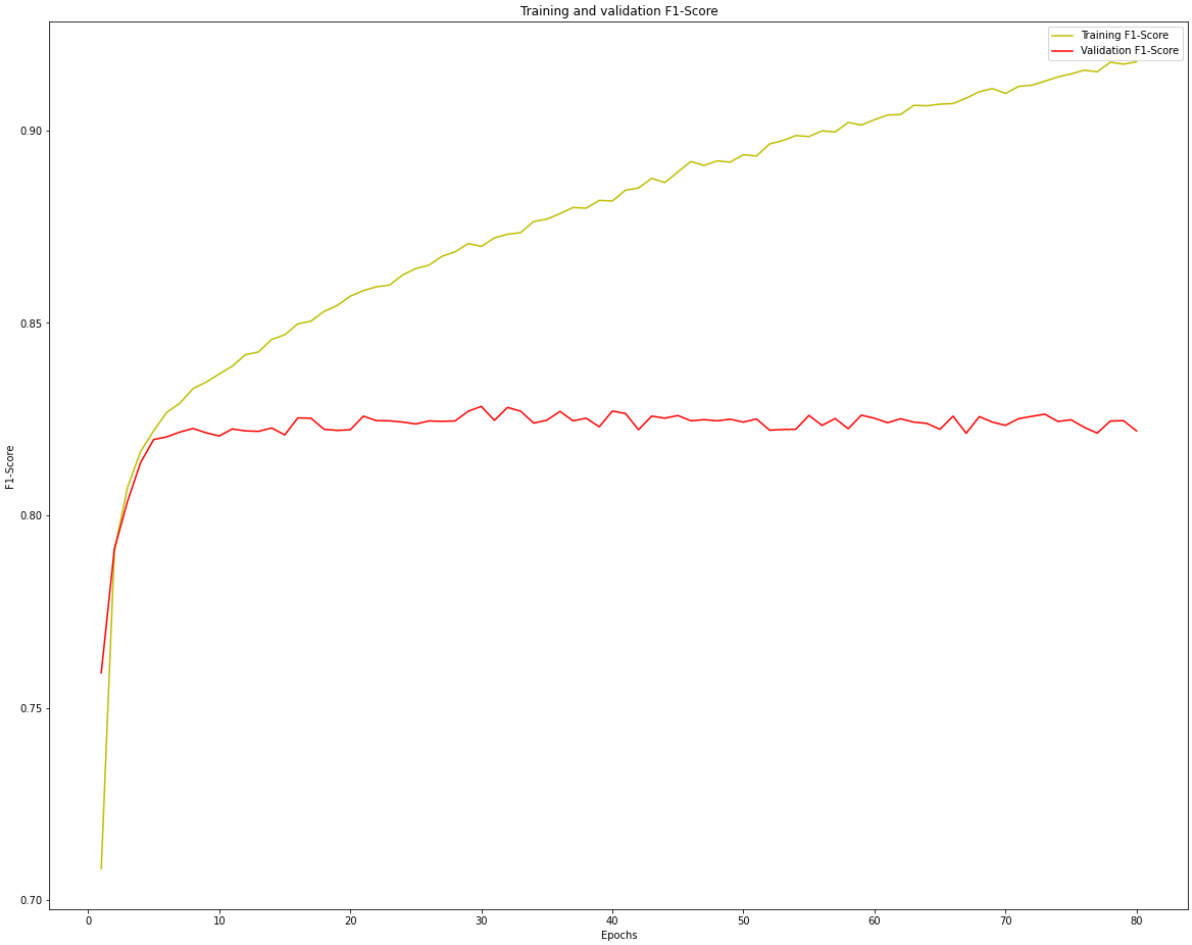
In [ ]:
```
model.save('/content/gdrive/MyDrive/Colab  Notebooks/Thesis/imagenet_Inceptio
```

In [ ]:
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['iou_score']
val_acc = history.history['val_iou_score']

plt.plot(epochs, acc, 'y', label='Training IoU')
plt.plot(epochs, val_acc, 'r', label='Validation IoU')
plt.title('Training and validation IoU')
plt.xlabel('Epochs')
plt.ylabel('IoU')
plt.legend()
plt.show()
```


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.45
0.40
0.35
0.30
0.25
0.20
0
10
20
30
40
50
60
70
80



In [ ]:
```
f1 = history.history['f1-score']
val_f1 = history.history['val_f1-score']

plt.plot(epochs, f1, 'y', label='Training F1-Score')
plt.plot(epochs, val_f1, 'r', label='Validation F1-Score')
plt.title('Training and validation F1-Score')
plt.xlabel('Epochs')
plt.ylabel('F1-Score')
plt.legend()
plt.show()
```



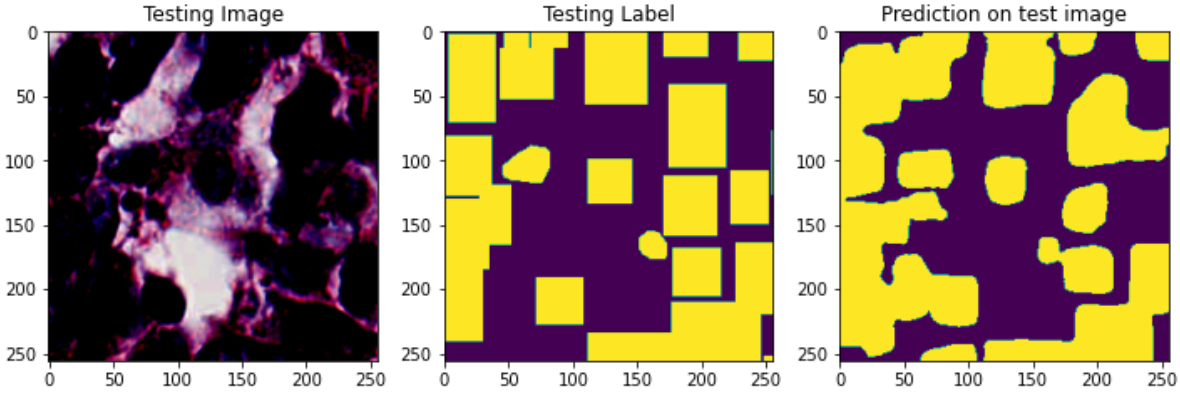
In [ ]:

```
from keras.models import load_model

model = load_model('/content/gdrive/MyDrive/Colab Notebooks/imagenet_Incepti

#batch_size=32 #Check IoU for a batch of images #Test generator

using validationdata.


test_image_batch, test_mask_batch = val_img_gen._next__()

#Convert categorical to integer for visualization and IoU calculation test_mask_batch_argmax =
np.argmax(test_mask_batch, axis=3) test_pred_batch = model.predict(test_image_batch)
test_pred_batch_argmax = np.argmax(test_pred_batch, axis=3)


n_classes = 2
IOU_keras = MeanIoU(num_classes=n_classes)
IOU_keras.update_state(test_pred_batch_argmax, test_mask_batch_argmax)
```

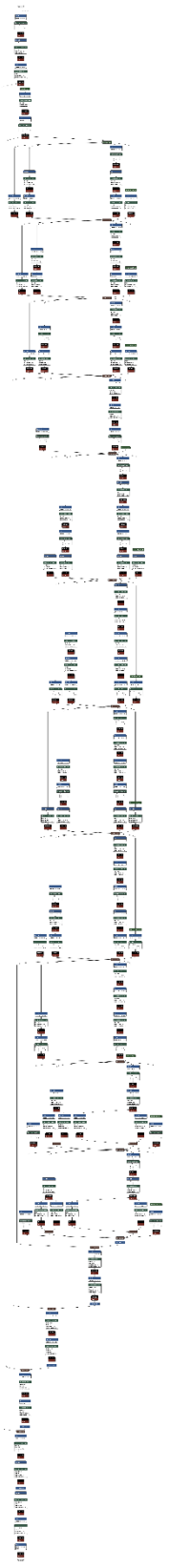
```
Mean IoU = 0.7101134
```

```
In [ ]: img_num = random.randint(0, test_image_batch.shape[0]-1)

plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(test_image_batch[img_num])
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(test_mask_batch_argmax[img_num])
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(test_pred_batch_argmax[img_num])
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB
data ([0..1] for floats or [0..255] for integers).
```



UNet with Inception-V3 Backbone



Unet with ResNet34 backbone

UNet with EfficientNet B2 backbone