Image Segmentation of X-Ray and Optical Images Using U-Net/U-Net++ Based Deep Learning Architecture

By

Tanmoyee Sharma

17121035

Zaharat Tabassum

16221014

Ritu Banik

16221003

S.M.Arifur Rahman

16221002

A thesis submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of
B.Sc. in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering
BRAC University
Spring 2021

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing a degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

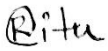
Student's Full Name & Signature:

Tanmoyee Sharma

17121035

Zaharat Tabassum

16221014



Ritu Banik

16221003

S.M.Arifur Rahman

16221002

Approval

The thesis/project titled “Image Segmentation of X-Ray and Optical Images Using U-Net/ U-Net++ Based Deep Learning Architecture” submitted by

1. Tanmoyee Sharma 17121035
2. Zaharat Tabassum 16221014
3. Ritu Banik 16221003
4. S. M. Arifur Rahman 16221002

of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Bachelors of Electrical and Electronic Engineering on 09 June, 2021.

Examining Committee:

Supervisor: (Member)	Abu S.M. Mohsin, PhD Assistant Professor Department of Electrical and Electronic Engineering, BRAC University _____
Program Coordinator: (Member)	Abu S.M. Mohsin, PhD Assistant Professor Department of Electrical and Electronic Engineering, BRAC University _____
Departmental Head: (Chair)	Md. Mosaddequr Rahman, PhD Professor and Chairperson Department of Electrical and Electronic Engineering, BRAC University _____

Abstract

Image segmentation is a fundamental section of the current healthcare system to segment and detect diseases such as disease of the lung (pneumothorax), cancer, diabetic retinopathy, dengue, malaria, heart disease, Alzheimer's disease, liver disease, rheumatoid arthritis, and so on. Lung segmentation, Cell segmentation, Brain segmentation, Liver segmentation are some of the popular medical segmentations.

In this study, we worked on two different types of images x-ray image and optical image for lung (Pneumothorax) and cell (nucleus) image segmentation. For both cases, we employed U-Net++ for image classification and segmentation to detect and identify Pneumothorax or cell nuclei. Additionally, we incorporated several image recognition models U-Net, ResNet34, Inception V3 within U-Net++ architecture and investigated which model provides better accuracy with minimum loss. The findings of our study will be not only beneficial for clinicians for accurate diagnosis but also will be helpful to lessen diagnostic limitations.

Dedication

This project is especially dedicated to the respected faculty members who assisted and directed us to successfully complete the research work.

We'd also like to dedicate our project to our parents, who have been tremendously supportive till the end of our research.

Acknowledgement

We would like to express the most profound and most sincere appreciation to our Thesis Supervisor Abu S. M. Mohsin, PhD, Assistant Professor of Department of Electrical & Electronic Engineering (EEE), BRAC University, for his pinpoint direction and support throughout this work. Without his guidance and consistent help this dissertation would not have been achieved. With proper guidance and timely response of our respected sir, we could come to this stage of completing our project in time.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation: Deep Learning in Image Segmentation

An effective deep learning research area is growing in the medical field to diagnose and section diseases and different cells. Deep learning has been widely used in Image Segmentation as it plays an essential role in disease detection and comparison considering diagnostic efficiency and accuracy. Image segmentation is a fundamental section of the current healthcare system to segment and detects diseases such as disease of the lung (pneumothorax), cancer, diabetic retinopathy, dengue, malaria, heart disease, Alzheimer's disease, liver disease, rheumatoid arthritis, and so on. Lung segmentation, Cell segmentation, Brain segmentation, Liver segmentation are some of the popular medical segmentations. Image Segmentation method is beneficial for clinicians for accurate diagnosis and also helps to lessen diagnostic limitations. Furthermore, it has several advantages such as easy data storing, faster processing, communication, multiple copy systems while maintaining quality, and flexible manipulation [1]. Therefore, we have decided to design pneumothorax and nucleus cell segmentation using U-NET/U-NET++.

1.2 Objectives and Goals

Segmentation for any disease is mainly focused on finding a proper treatment method and also to classify the disease. There are different genres of diseases and various methods are usually used to check if that particular disease falls into a specific genre of disease or it needs to be researched to find a proper cure and also sometimes proper vaccines for prevention. However, our main objective here is to compare "Pneumothorax", which is considered as a collapsed lung from x-ray

images, and “Nucleus Cell” from divergent optical images. The segmentation of this particular disease and cell using “U-NET” which is a process done with image segmentation has the primary objective of relieving the pressure from the patient’s lung and cells. We can also prevent recurrences if we can successfully compare pneumothorax and nucleus cells.

1.3 Background of Deep Learning in Image Segmentation

Image segmentation is a vital part of many visual understanding systems. Image segmentation is a key theme in image processing and computer vision with applications for instance medical image analysis, augmented reality and robotic insight. In terms of medical image analysis, with the help of Deep Learning (DL), we can segment various cells such as collapsed lung tissues, brain cells, breast tissues, nucleus cells, and so on. Moreover, we can even use multiple types of image formats such as JPG, PNG, DICOM, and so on. Those help us to use any format of image, we are comfortable to work with. Recently, numerous algorithms have been expanded for image segmentation. With the growing success of deep learning models, there has been a considerable number of works aimed at growing image segmentation approaches using deep learning. [2] Many semantic segmentation problems have been solved by using deep learning architectures (mostly CNN and FCN) over the past few years and this architecture outperforms other approaches in terms of accuracy and efficiency by a significant margin. [44] Image segmentation in deep learning has two types which are Semantic segmentation and Instance segmentation. To classify the objects belonging to the same class in the image, semantic segmentation is used to maintain a single label. On the other hand, instance segmentation is a combination of object detection and segmentation. However, Image segmentation is a part of CNN architecture where various methods we can use to evaluate a deep learning segmentation model. Pixel accuracy, mean pixel accuracy, Intersection

over Union (IoU) and Mean IoU, and also Dice coefficient and Dice loss. There are various pre-trained models. [Table 1.1]

Model	**Size**	**Top-1 Accuracy**	**Top-5 Accuracy**	**Parameters**	**Depth**
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	17 MB	0.704	0.895	4,253,864	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
Densenet169	57 MB	0.762	0.932	14,307,880	169
Densenet201	80 MB	0.773	0.936	20,242,984	201

Table 1.1: Description for Individual Models [42]

1.4 Thesis Overview

Here, we will talk about the chapters we arranged for discussing various topics related to our project.

In chapter 01, we introduce the use of Deep Learning in Image Segmentation and our goals of using it in our project. We discuss the detailed background of Deep Learning and how it contributes

to different Cell Segmentation with different Image and model formats. We also declare our ambition of using deep learning in pneumothorax's x-ray images and nucleus cell optical images for segmentation.

In chapter 02, we elaborately discuss the Architecture, Basic Components, and Training of CNN and FCN and Data Augmentation. First of all, we were informed about CNN and FCN, how it architecturally builds up, how it processes the images, the basic components in it, and how we train it with different matrices. Then we describe the data arguments, how we optimized the data with the different procedures and using different models and matrices.

In chapter 03, we declare the first part of our project, Cell Segmentation for X-ray of Pneumothorax using U-Net/U-Net++. Here, we establish our model with different encoders such as Xception, Resnet34, and Inception34, compare their different matrices with each other, and find the best output among them. After doing our all evaluation, we analyze that InceptionV3's accuracy matric for epoch 30 and batch size 32 gives us the best result.

In chapter 04 we announce the second part of our project, Cell Segmentation for Optical Images of Nucleus Cell using U-Net/U-Net++. At this time, we establish our model with different encoders such as Resnet34 and Inception34, compare their different matrices with each other and find the best output among them. Here After all the analyses, we find out our model's accuracy matric gives the best result for epoch 30 and batch size 32.

In chapter 05 we conclude our paper by mentioning the Future work that we want to work on and the challenges we can take in the further development of our project.

Chapter 2

Theory and Background Study

2.0 Introduction

This chapter is all about the detailed information about the process that we have been working with. CNN and FCN architecture, Image processing of a computer, and finally, activation of the functions. This will give us an overview of the used methods and also clarify the details.

2.1 Architecture of CNN and FCN

For computer vision tasks, convolutional neural networks (CNN) work really great. Taking help from different differential functions, each layer that CNN has, performs one volume of activation to another. However, CNN architecture can be made of different layers organized by API. The main 3 layers are –

1. Convolution layer
2. Fully connected layer
3. Polling layer

There are four layers having certain parameters and hyperparameters. These parameters act like a variable and are optimized automatically during the training period. On the other hand, Hyperparameters are those kinds of variables that need to be fixed or set before starting. FCN or Fully conventional Network is introduced here. For semantic segmentation, Fully Convolutional Networks or FCNs architecture is heavily used. In the FCN network, there are no “Dense” layers. But there is, 1x1 convolutions that complete the task of fully connected layers. FCN replaces the fully connected layer of the last layer of the general classic classification network model with convolution. The basic FCN structure is shown below in: [Figure 2.1]

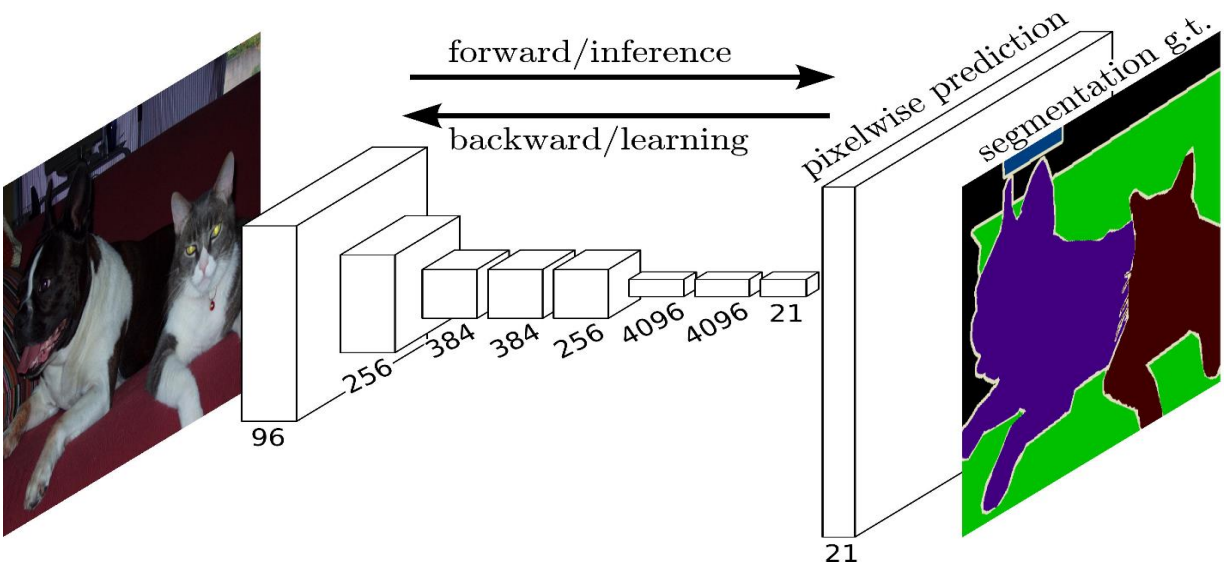


Figure 2.1: Fully Convolutional Neural Network Architecture for Semantic Segmentation [36]

The different parameters and hyperparameters of CNN are shown below: [Table:2.1]

Different Layers	Used Parameters	Used Hyperparameters
Convolution layer	Kernel.	Kernel size, kernel number, stride, padding, activation function.
Polling layer	None.	Pooling method, filter size, stride and padding.
Fully Connected (FC) layer	Weights.	Number of weights, Activation function.
Others layers	None.	Architecture of model, learning rate, optimizer, loss function, regulations, epochs, weight initialization.

Table 2.1: Different Parameters and Hyperparameters of CNN [3]

So, summing up the architecture in order:

1. Input

2. Convolution

3. Polling

4. Fully connected

Each of the layers has certain impacts and descriptions. The description is given below:

1. **Input:** Firstly, in the input segment, (B*A*3) is the found values for raw image pixels. Where B is known as the width, A is the height and the number 3 represents the three-color channels which are R, G, B.
2. **Convolution layer:** This layer does the job of lining up the feature and image patch which computes the output of neurons connected to the input regions. Then with the corresponding feature, each image pixel is multiplied and then added. Because of this, Numbers of filters are being used. Finally, divide the results by the total number of pixels that are provided in the feature.
3. **Pooling layer:** In this layer, downsampling operations and shrinking the image stack into a smaller size are performed. It chooses a window size and a stride. Then allows to walk the window across the filtered image and take maximum value from each window. This is how the work in this layer works.
4. **Fully Connected (FC):** Finally, in this layer, the actual classification activity happens and also computes the class score. Here the filtered and shrink images are taken and are put into a single value of list.

Picture of a Convolutional Neural Network (CNN) architecture overview and how it is trained is given below:: [Figure 2.2]

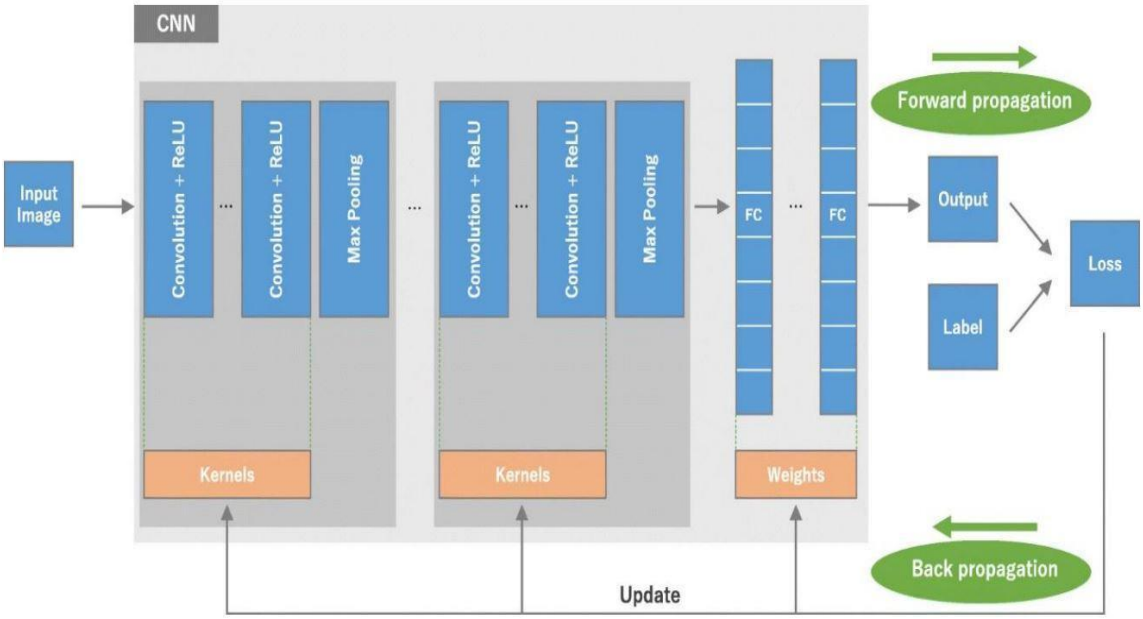


Figure 2.2: A Convolutional Neural Network architecture and Training Process [3]

2.2 Image Processing or Reading of a Computer

A typical computer reads an image in 0s and 1s. In an image, a pixel is considered as the smallest unit. When a digital image is considered, it is stored as a combination of pixels. Each pixel has a different number of channels. If a grayscale image is considered, it has only one pixel or one channel and also the computer reads the image as 2D matrices. So, black and white image size is R*C where R is the row and C is the column. To further explain the fact, a computer usually reads an image as a matrix of numbers between 0 to 255 [29]. If it is a colored image, it consists of three different channels: red, green and blue. Each channel of the matrix represents the intensity of the brightness of the pixel. For colored images, there is one 2D matrix for each channel so three matrices for red, green and blue channels are stacked onto each other which makes the matrix 3D. So, the image size for colored image is R*C*3 where, 3 is the channels and R and C is the row and column. There is an example of how a computer sees a grayscale image (on the right): [Figure 2.3]

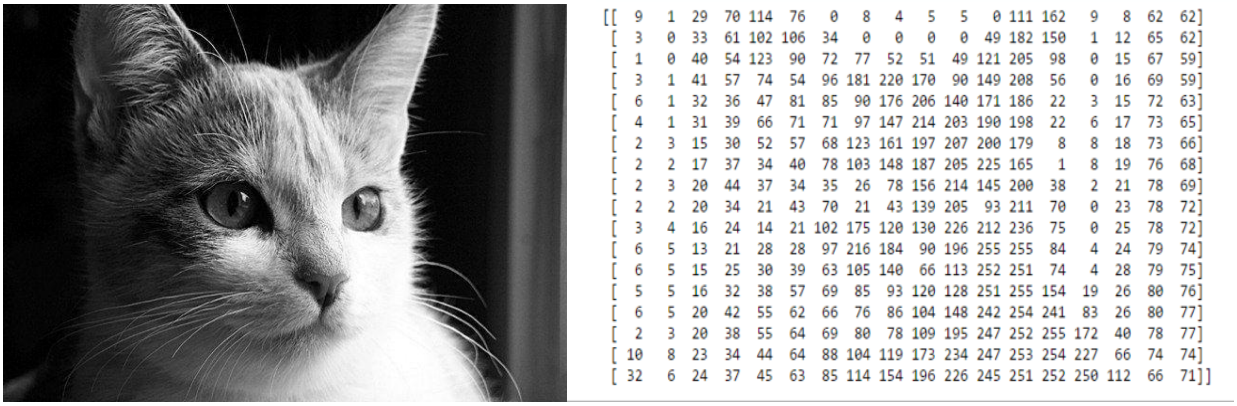


Figure 2.3: How Computer Reads Grayscale Image [32]

In the image given below, the color image consists of three channels red, green and blue: [Figure 2.4]

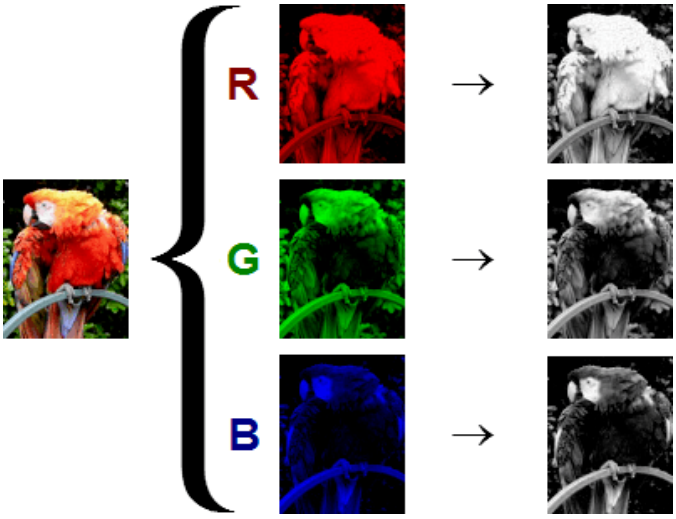


Figure 2.4: RGB Image [32]

2.3 Basic CNN and FCN Components

CNN consists of 3 alternate layers, firstly comes convolution and after that comes the pooling layer and finally there is one or more than one fully connected layer in it. The composition of CNN components plays a vital role in designing different regulatory units like batch organization and drop-out and thus enhances the performance. The layers or the components are explained properly in the following segments.

2.3.1 Convolution Layer

Parameters very much focus on kernels which are known as learnable kernels. Almost the entire entity, kernels spread along the input. But they are not very big in spatial dimensionality. While data passes through a convolutional layer, the particular layer rolls on to each filter which happens across the spatial dimensionality of the input and produces a 2D activation map [4]. If we go through research, for each value here, the scalar product is calculated in that kernel. Then they will find a particular feature at a spatial position of the input, active kernels will be found then which are known as activations. However, for every kernel there's a corresponding activation map, of which is linked along the depth dimension to make the full output volume using the convolutional layer. Convolutional layers can effectively reduce the complexity of the model through optimizing its output. These are optimized using three hyperparameters which are the stride, the depth, and setting zero-padding. A typical convolution layer is shown below in [Figure 2.5]

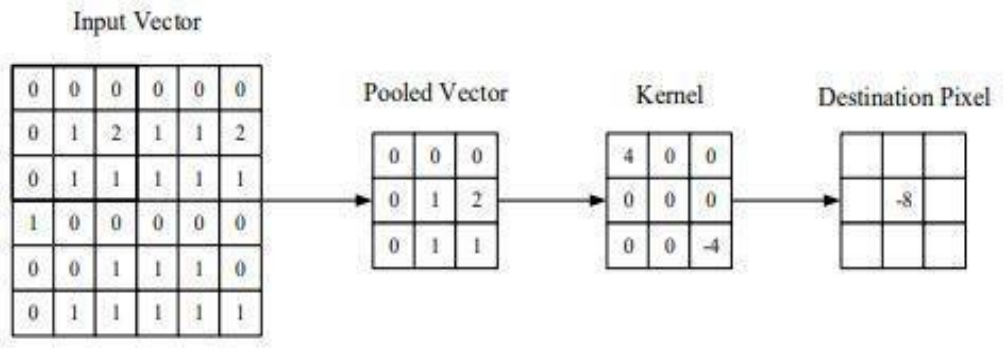


Figure 2.5: A Typical Convolution Layer [4]

2.3.2 Non-Linear Activation function

To compute the weighted sum of inputs and biases, activation functions are used in a neural network. It is used in turns to decide if a neuron can be activated or not [5]. There are mainly two types of activation functions. Various categories of Non-linear activation function are shown

below [Figure 2.6]

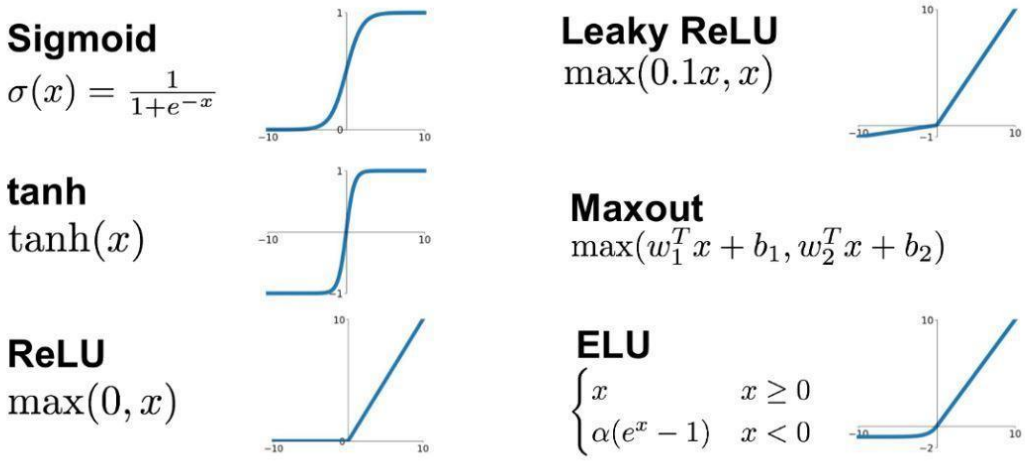


Figure 2.6: Different Activation Functions [6]

ReLU: The basic formula for ReLU is very simple: *max (0, z)*. Though its name is "Rectified Linear Units" it's not linear and gives the same benefits as Sigmoid but with a comparatively better performance. There are mainly two types of ReLU which are (i) leaky ReLU and another one is (ii) Parametric ReLU. [4]

Sigmoid: For logistic regression, sigmoid is heavily used. Sigmoid considers a real value as input and provides output as another value between 0 and 1. The sigmoid activation function translates the input range in (-∞, ∞) to the range in (0,1). X is known as a plus bias which is calculated by weights multiplied by input and this x is transferred to the activation function. Once it is transferred, this will transform the value between 0 and 1. That is what a sigmoid function does where 0.5 is the threshold. If above 0.5 it is taken as 1, and below that it is considered as 0. If the output is 1 explains that the neuron is activated and that results in the neuron transferring the signal.

tanh: It is another activation function that is usually used in between layers of neural networks. It provides a mapping of input values with the range of -1 to 1. The derivative tanh can be used as a function itself.

2.3.3 Pooling layer

The pooling layer activity consists of constructing a 2D filter over every channel of the feature map then concludes the options or features which are inside the region and concealed by the filter. Pooling layer is situated between convolution layers and performs down sampling operations. Pooling layers help to reduce the dimensions related to the feature maps. That is why, the parameter numbers to learn gets lesser. Also, the amount of computation happened in the network. The in-plane dimensionality of the feature map is reduced in such a way that signifies a translation invariance to small portions and distortions. Not only that, it also decreases the subsequent learning rate. The pooling layer concludes the options or features which are included in a region of the feature map, which is created by the convolution layer. So, the afterwards actions which are completed, concludes features in place of accurately positioned features which are results from the convolution layer. This gives the space for the model to be more open to variations in the position of options or features in the input image. Primarily there are two operations in the Pooling layer. One is Max-pooling and other is the Global average pooling.

- **Global average pooling:** Average pooling calculates the average of the elements consisting inside the region of the feature map concealed by the filter. Average pooling provides the average of the features consisting of a patch. This particular average pooling is used for extreme down sampling where feature map having measurement of height x width is down sampled into 1x1 array. Here is a picture example of the process given

below: [Figure 2.7]

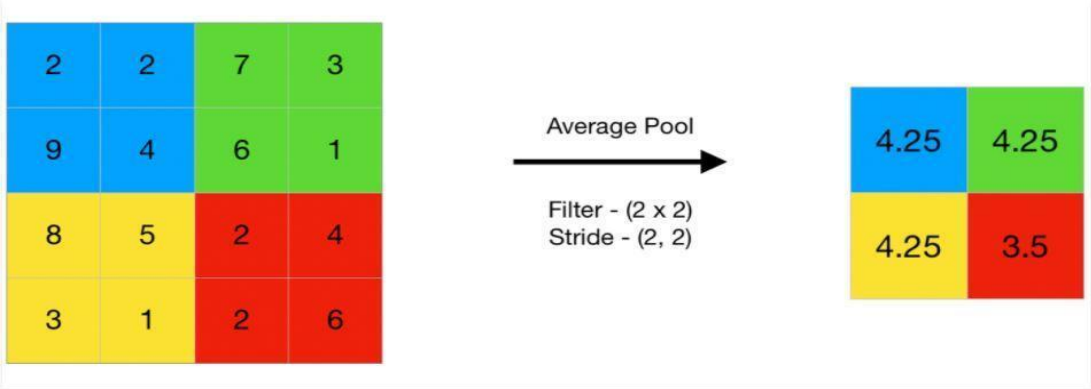


Figure 2.7: Global Average Pooling [30]

- **Max pooling 2D:** Max pooling is basically known as a pooling operation that rectifies the maximum element from the region of the feature map topped by the filter. However, it is a discretization process where sample values are calculated using a max filter to find the maximum value. In the pool to down sample an input, reducing its dimensionality and to make decisions about the features based on the down sampled subsamples. A picturized example is given below: [Figure 2.8]

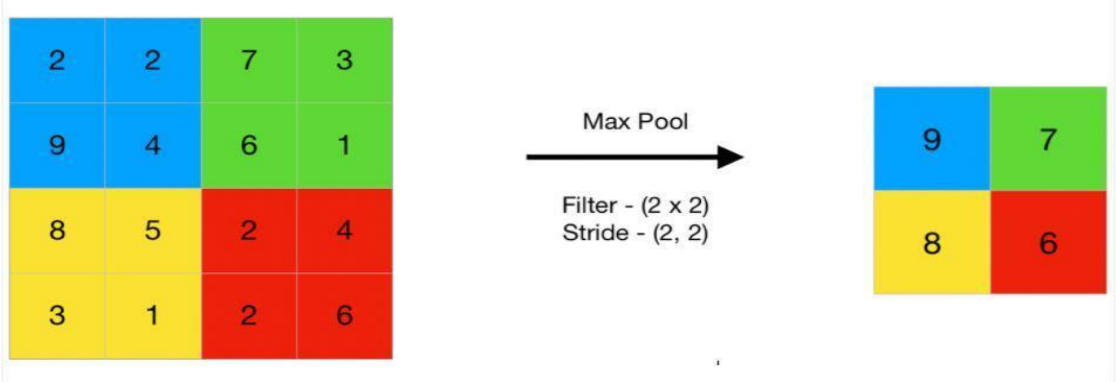


Figure 2.8: Max Pooling 2D [30]

2.3.4 Fully Connected Layer (FC Layer)

The architecture of CNN consists of a number of hidden layers among which, the last few layers are formed by a Fully Connected Layer. By following bias offset with matrix multiplication, all the activation can be measured.

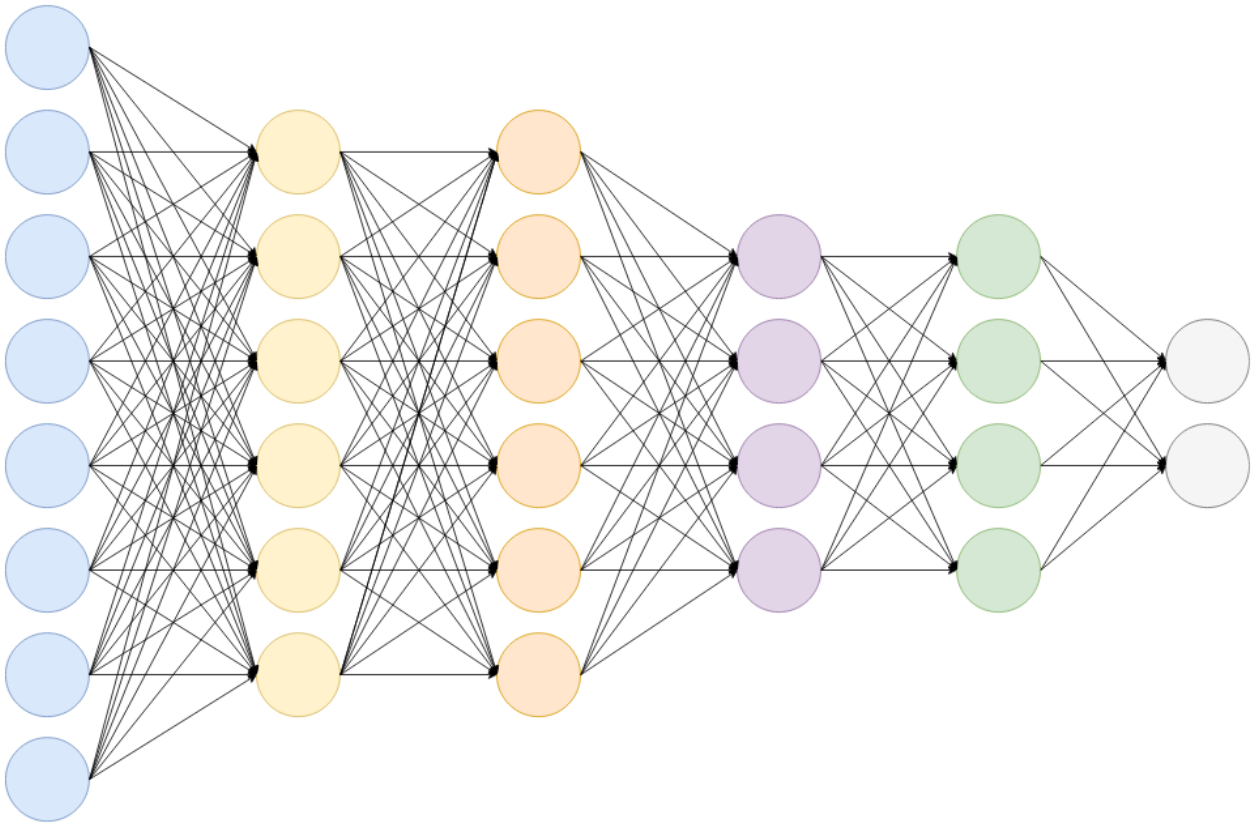


Figure2.9: Fully Connected Layer [37]

The input to this layer is the output received from the final Pooling or Convolutional Layer, which enters the fully connected layer after being flattened, that is, all the values are unrolled into a vector. The resultants are then connected to fully connected layers and perform a mathematical operation similar to the Artificial Neural Network calculation [37]. The equation is as follows:

g (Wx + b) …….. (1)

From the equation 1 [37],

x = input vector with dimension; b = bias vector with dimension;

W = weight matrix with dimension; g = activation function.

The fully connected layers are passed after repeating this calculation for each. [37]

Each of these layers are also connected by some non-linear activation functions. These activation functions are usually required to be selected according to each task. They perform to set output real values from the last fully connected layer to the target class probabilities. The value range must be between 0 and 1 including the sum of all values as 1. [3] Standard choice for the last layer activation functions depending on different working areas are shown below [3], [Tab. 2.2]

Task	**Activation Function**
Binary classification	Sigmoid
Multiclass Classification	Sigmoid
Multiclass Classification	Softmax
Regression to continuous values	Identity

Table 2.2: Last Layer Activation for Different Tasks

2.4 Training of CNN

Training of a network signifies step of CNN architecture where training dataset is provided with the finding's kernel in convolution layers and weights in fully connected layers. The weights are being made computer-readable through a training process for adjusting the filter values and it is called backpropagation. It is mostly used for training CNN.

2.4.1 Backpropagation

Firstly, Backpropagation contains four distinct sections which are:

- Forward pass,
- Loss function,
- Backward pass,
- Weight updating.

The training image which consists of a particular number of arrays passes through the whole network. At first, weights are generally in random initialization and the output doesn't give any preference to a reasonable conclusion. As the network fails to develop the classification of what is expected to be, it goes to the loss function backpropagation.

The loss function calculates the gradients used to minimize the error for each training example during the learning process. But, for a correct prediction, it is necessary to minimize the amount of loss by finding the weights which are greatly responsible for direct contribution to the loss of the network. Thus, backward pass is used to find the responsible weights. This can be done by using various weight initialization techniques, such as – uniform distribution, Xavier (or Glorot) initialization, He initialization, and so on.

- **Xavier Uniform (Glorot):**

$$W_{Xu} = \left[-\frac{\sqrt{6}}{f_{in}+f_{out}}, \frac{\sqrt{6}}{f_{in}+f_{out}} \right] \text{ ……..(2)}$$

- **He Uniform:**

$$W_{Hu} = \left[-\sqrt{\frac{6}{f_{in}}}, \sqrt{\frac{6}{f_{in}}} \right] \text{ ……...(3)}$$

The Xavier Uniform Equation [7], (eq 2) works best with Sigmoid AF, while the He Uniform equation [7], (eq 3) works well with the ReLu Activation function.

2.4.3 Learning Rate

The importance of the learning rate is that, whenever the model weights are updated, it measures the amount of change in response to the estimated error, so that the step size is neither too big nor small. Learning rate has to be dealt with very carefully as a small value for learning rate can result in a very high time for the training process. It is usually a very small number (e.g., 0.0001) that is multiplied with the gradients while scaling them, making sure the changes made to the weights are also very small.

2.4.4 Gradient Descent

This parameter is the first-order optimization algorithm, commonly used while training a machine learning model to decrease the loss of the network [40]. It provides us with the function's direction along with the rate of increase. The gradient descent is formulated as follows [3]:

$$\omega = \omega - \alpha * \frac{\partial L}{\partial \omega} \quad \text{.....(4)}$$

Here [3], (eq 4),

ω = weight (learnable parameter), α =learning rate, L=loss function.

The gradients working for the loss function responsible for the weights are generally measured using a subset of the data used for training images known as mini-batch. Thus, the method is named as mini-batch gradient descent, alternatively known as the stochastic gradient descent (SGD). A hyperparameter to this is the mini-batch size. Besides, various improved methods of the gradient descent algorithm have been suggested and are also used widely. Some of these methods are SGD with momentum, Adam and RMSprop. [3]

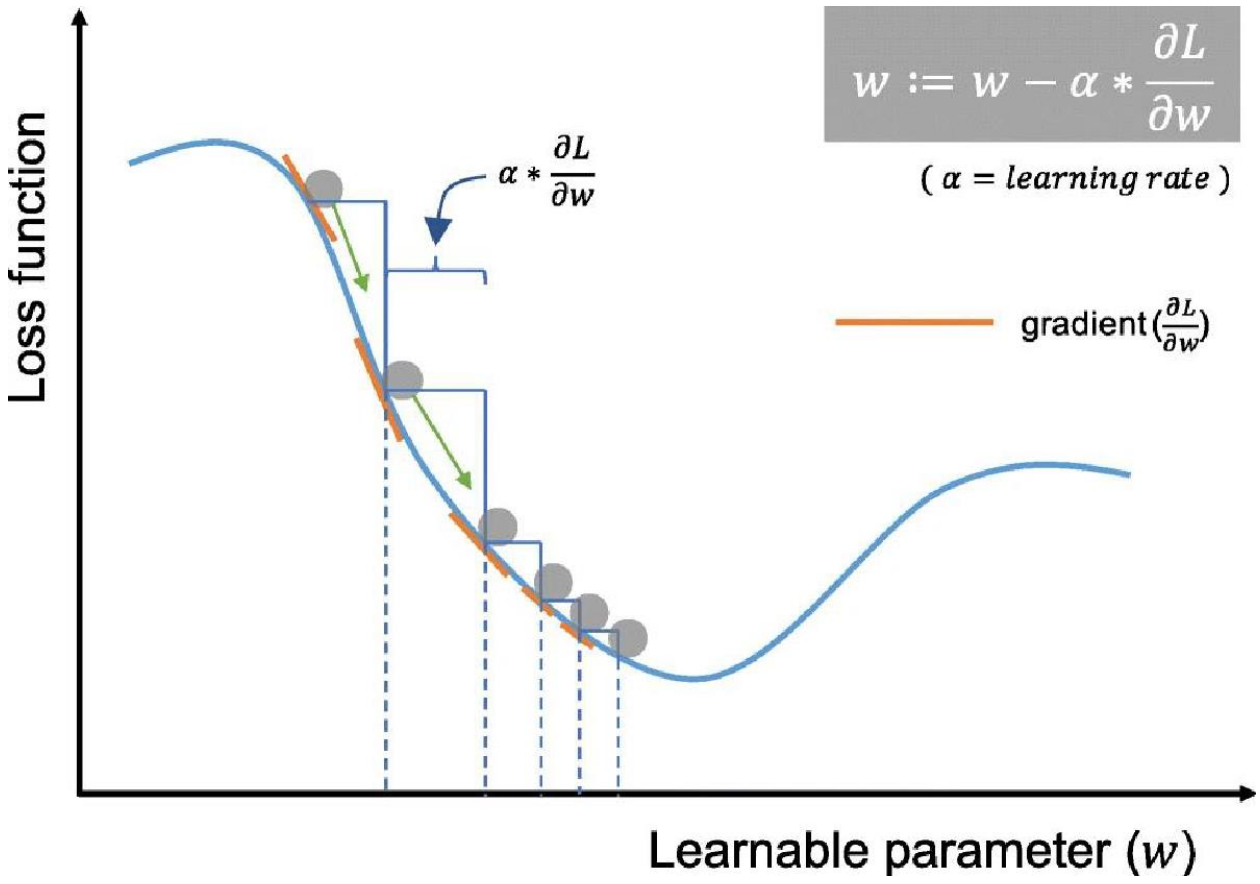


Figure 2.10: Gradient Descent with Optimization Algorithm [33]

2.5 Data Augmentation

The procedures to expand the diversity of data with the addition of marginally improved copies of current data along with synthesizing latest data from the existing data is declared as Data Augmentation. It is possibly utilized in order to label the aggregate of data and can be used to address the class variation complication in classification tasks in data analysis. It radically helps to reduce overfitting. To prevent overfitting in neural, DA is being used. Also, we use augmented data to train a machine learning model in order to enhance the model's performance. [8] For the performance of DL model encountering an enormous dataset is pivotal. However, DA can enhance the performance of the model of already existing data. Audio, Text, Images any other types of data can be augmented. Among these, image augmentations are widely used. For image augmentation

geometric transformations (flip, crop, rotate or translate images), changing RGB channels, intensity of colors, sharpen or blur an image, mixing image with one another techniques are being followed for word augmentation word shuffling, word replacement and syntax-tree manipulation can be done. For noise augmentation, noise injection, changing the noise of tape methods are being followed. [9], [10] We may follow as many methods we want to at one and this is how plenty of distinctive data can be achieved from previous data. To create an image dataset Keras is being used in Python. Keras has a plenty of preprocessing layers which is being used for Data Augmentation**.** The Keras DL neural network library presents the capability to fit models using image data augmentation with the Image Data Generator class. Keras Image Data Generator class is designed to provide real-time data augmentation which is the main benefit of using Keras. [11] Significant differences can be noted in model executions for different applications with or without data augmentation. As below the difference is shown –

Application	Performance without Augmentation	Performance with Augmentation	Augmentation method
Image classification	57%	78.6%	Simple Image based
Image classification	57%	85.7%	GAN based
NMT	11 BLEU	13.9 BLEU	Translation data augmentation
Text classification	79%	87%	Easy Data Augmentation

Table 2.3: Model Performance for Different Applications with & without Data Augmentation

2.6 Used Algorithm

2.6.1 Algorithm using U-NET++

The **U-Net** is a convolutional arrange design for quick and exact segmentation of images. For image segmentation of higher accuracy, U-Net++ is an effective U-Net Architecture for Medical Image Segmentation. The shape of U-Net looks like a typical 'U' shape. [35] The shape for U-Net is given in [Figure 2.11]. The architecture contains 3 numbers of sections and each of them contains numbers of convolution layers among themselves which are called:

1. **The Contraction** - There are two 3X3 convolution layers and after that a 2X2 max pooling.
2. **The Bottleneck** - Two 3X3 CNN layers are incorporated here and followed by a 2X2 up convolution layer.
3. **The Expansion Section** - two 3X3 CNN layers followed by a 2X2 Upsampling layer.

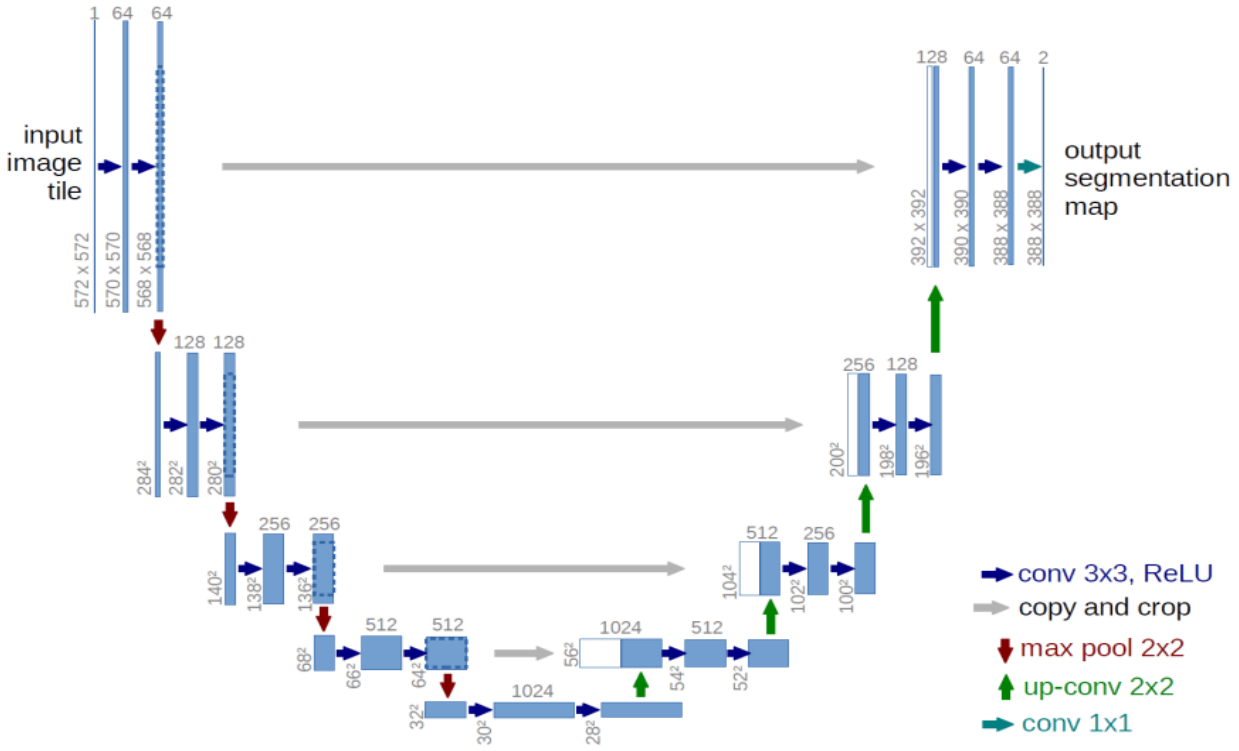


Figure 2.11: U-Net Architecture [12]

Furthermore, U-Net++ consists of U-Nets and it aims to overcome two important existing barriers of the U-Net [39]:

1) The depth of optimal architecture is not known.

2) The skip connection design is very restrictive.

U-Net++, which is an encoder and decoder architecture all together constructed based on the famous U-Net, has found antiquated results on many medical image segmentation tasks [13].

U-Net++ aims to improve segmentation accuracy by adding dense block and convolution layers between the encoder-decoder. Below is an illusion of U-Net++ architecture in [Figure 2.11].

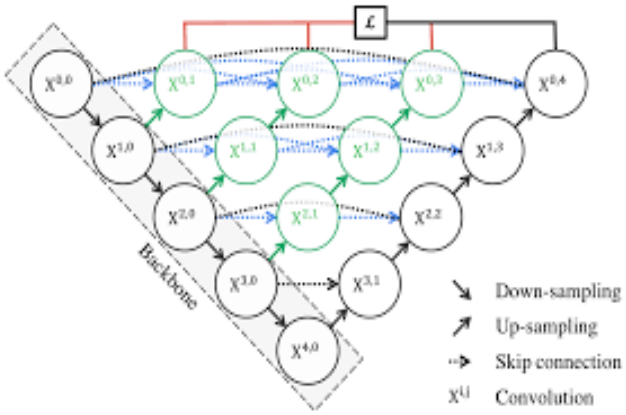


Figure 2.12: U-Net++ Architecture [14]

2.6.2 Algorithm using Pre-trained Encoders:

Encoders are the models that we use in the simulation and code. A pre-trained encoder helps to get high accuracy and also helps to get a high-performance rate compared to non-pre-trained encoders. There are several models which can be used as encoders such as DenseNet, Xception, Resnet, Inception, VGG19 Etc. However, in this case, DenseNet121, InceptionV3, Resnet34 and Xception are the featured models encoded within the U-Net.

1. **InceptionV3:** The "Inception" is a micro-architecture that was first brought into the world by 'Szegedy et al' [31]. The journal that it was published in is named 'Going Deeper with Convolutions'. The purpose of the inception model is to behave as a "multi-level feature extractor". Its main work is to compute 1×1, 3×3, and 5×5 convolutions within the same module of the network. Then the output of these filters is fed into the next layer in the network and after that connected with the channel dimension. A demonstration of the Inception model is given in [Figure 2.12]

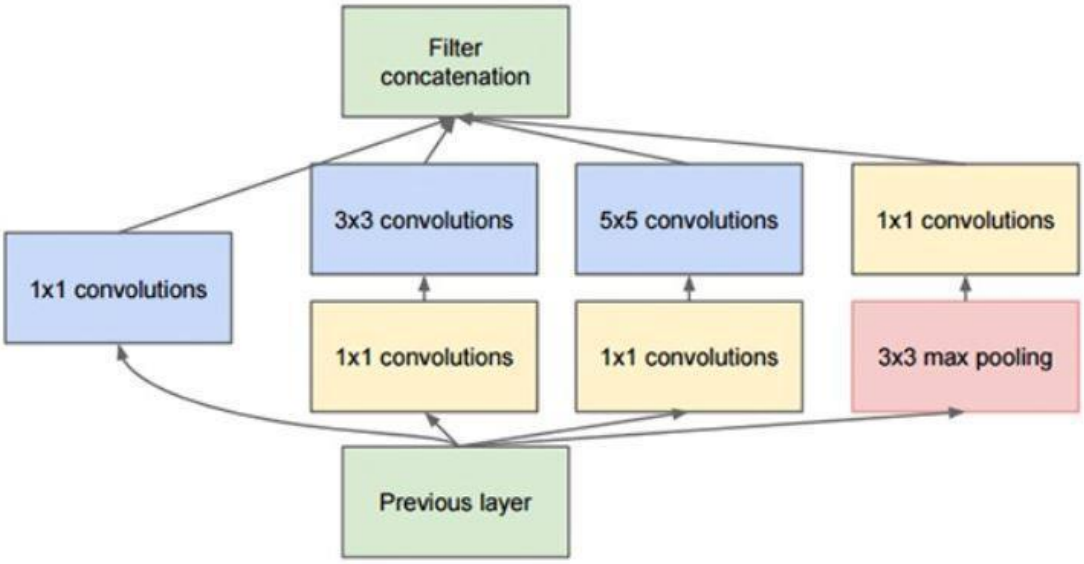


Figure 2.13: The original Inception Module Used in GoogleNet [26]

The V3 from the name of the model indicates the version number of Inception. In our case, the version that is used is the 3rd version.

2. **Xception:** This CNN architecture contains 71 layers deep and 23 million parameters. The model architecture was initially inspired by the architecture of Inception-v3. But, became the only CNN architecture completely dependent on depth-wise separable convolutional layers after substituting convolutional blocks with the layers [15], [Figure 2.13].

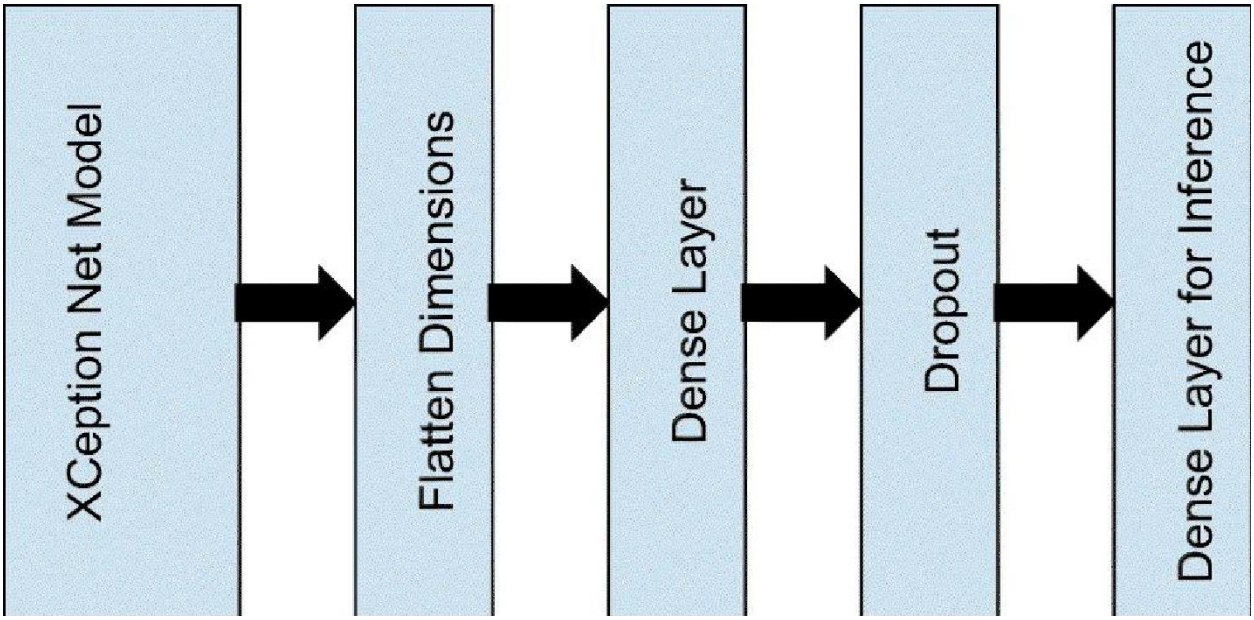


Figure 2.14: Xception Architecture [41]

3. **DenseNet121:** DenseNet uses only 0.8M parameters, which is fewer than other CNN architectures. The architecture of this model follows a typical approach of layers from modern transfer learning known as dense connections. This approach is formed of mainly dense blocks. Compared to residual connections, features by concatenation are combined using these dense connections. Inside every dense block, the layers are interconnected directly, to ensure maximum flow of information. This direct connection also helps to improve the gradient propagation while training, reducing the vanishing gradient problem. [45] [Figure 2.15]

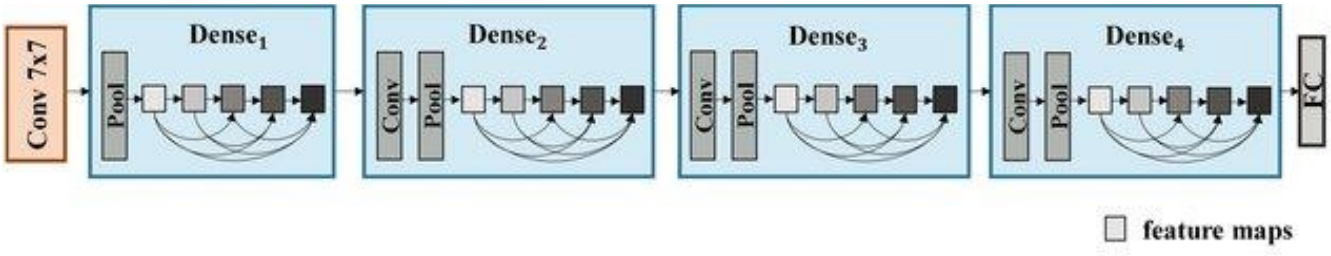


Figure 2.15: DenseNet121 Architecture [45]

4. **ResNet34:** The full form of ResNet is a Residual network. ResNet34 is a convolution neural network that contains 34 layers that can be used for image classification and segmentation. The basic architecture of Resnet34 along with its 34 layers are provided below in [Figure 2.16]

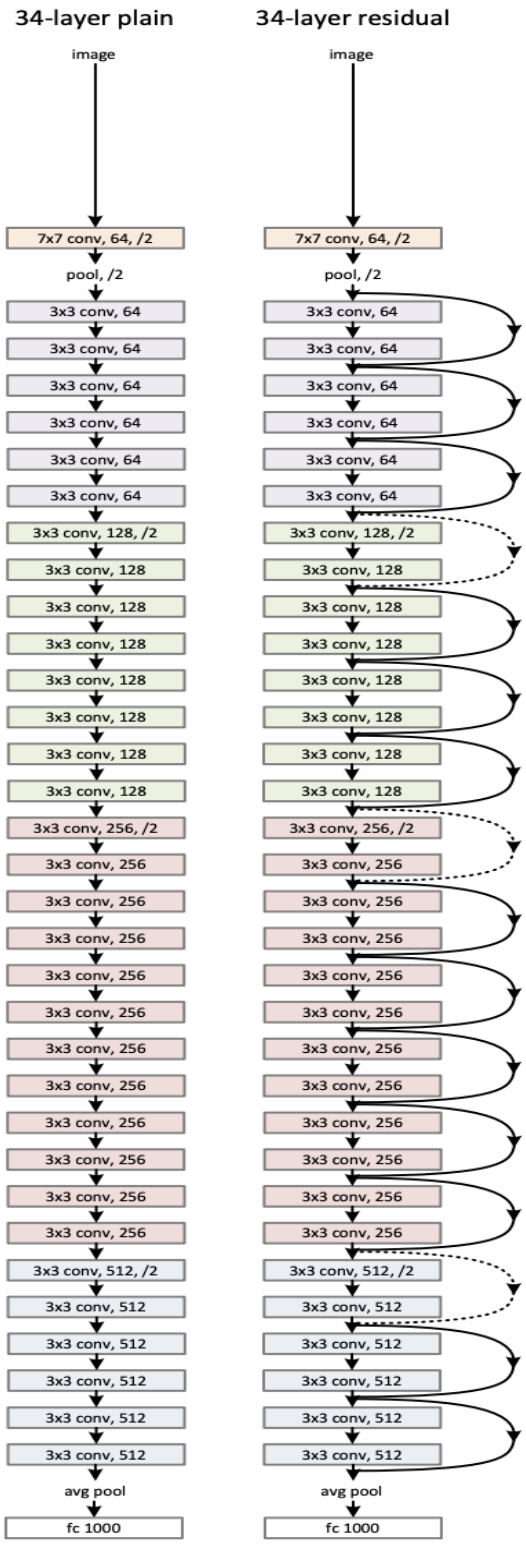


Figure 2.16: Architecture of ResNet34 [34]

2.7 Metrics for Performance Evaluation

2.7.1 IoU Metric

The Intersection over Union (IoU) metric is an approach frequently used to appraise the accuracy of an object detector on an individual dataset. IoU is also known as the Jaccard Index. This metric is correlated to the Dice coefficient that is implemented frequently as a loss function while training. [24] To begin with, the area of overlap between the predicted bounding box and the ground truth bounding box is being calculated in the numerator. Afterward, the area of union by both the predicted bounding box and the ground-truth bounding box is being calculated in the denominator. Finally, the IoU is dividing the area of overlap by the area of union yields. The formula of IoU is as shown in Equation (1) -

$$\textbf{IOU} = \frac{\textit{Area of Overlap}}{\textit{Area of Union}} \quad \textbf{.....(5)}$$

The formula to calculate IoU is given below in [Fig 2.17]

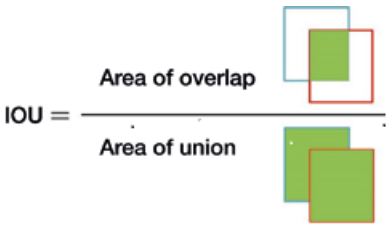


Figure 2.17: Formula of IoU [16]

In an available set of images, the IoU shows the resemblance between the predicted region and the ground-truth region for an object existing based on true positive, false positive and false negative ratios as shown in Equation (6) [17]-

$$\textbf{IoU} = \frac{TP}{FP+TP+FN} \quad \textbf{.....(6)}$$

Calculating Overlapping Region of IoU metric as follow-

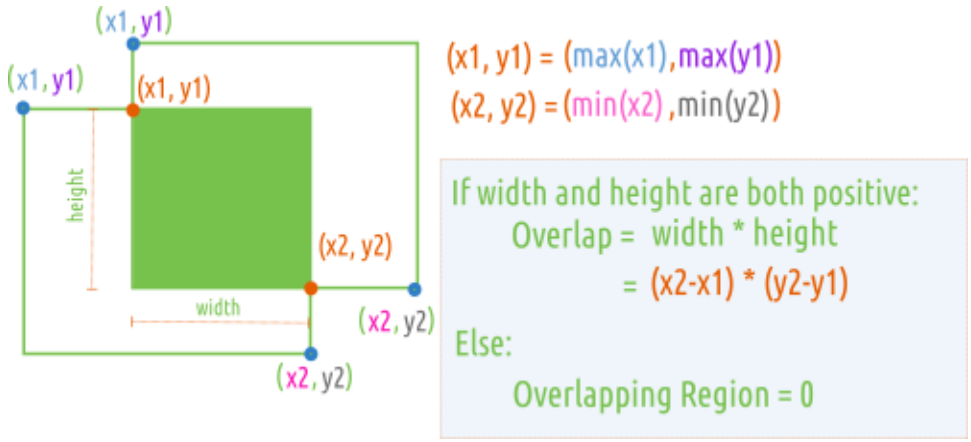


Figure 2.18: Overlapping Region of IoU [25]

If an IoU score is >0.5, then it is usually considered as a 'satisfactory' prediction. [12] Score of 1 indicates that the predicted bounding box matches with the ground truth bounding box precisely whilst a score of 0 specifies that the predicted and true bounding box is not overlapping to some extent. [25]

2.7.3 Accuracy Metric

Accuracy metric is the procedure to evaluate a model's outcome. A**ccuracy** is the fraction of predictions of our model's precision. AC represents how close a measurement is to its literal value. Substandard data processing can lead to inaccurate results that are not very close to the literal value, thus accuracy metrics are being used to evaluate the model's results.

2.8 Loss Function

In a neural network model, the parameter values are optimized using a loss function. Loss functions convert a set of network parameter values into a scalar value that shows how successfully those parameters perform the task for which the network was designed to execute. [29] This is referred to as a cost function, which employs a forward pass to achieve a balance between output and prediction. The cost function is used to alter the weights of a neural network during training for

building a finer-fitting machine learning model. Fundamentally, the neural network is run on training set data during forwarding transmission, and outcomes are obtained, which in the case of classification represent the likelihood or certainty in probable categories. The loss function calculates a penalty for each difference between the target label and the neural network's outputs when these likelihoods are matched to the target labels. The partial derivative of the loss function is determined for all adaptable values of the neural network during backpropagation. These partial derivatives are used to alter the values. Back propagation iteratively modifies the adaptable values of a neural network to build a model with less loss under normal circumstances. [18]

The following equation (7) is the conventional binary cross-entropy loss function:

$$\boldsymbol{J_{bce}=-\frac{1}{M}\sum_{m=1}^{M}\ [y_m \times log\ log\ \left(h_\theta(x_m)\right) + (1 - y_m) \times log\ log\ (1 - h_\theta(x_m))]\ \ldots.(7)}$$

In this Eq (7), M denotes by number of training examples, y_m is the target label for training example m, x_m denotes by input for training example m, h_θ denotes a model with neural network weights. [18]

Chapter 3

Algorithm in Pneumothorax Detection

3.0 Introduction

In this chapter, we have included how we have worked on several algorithms for detecting pneumothorax. Besides, in the following subchapters, we have described in detail how we compiled, trained our model, evaluated the model and compared the results with other relevant results.

3.1 About Pneumothorax

“Pneumothorax” is the medical term for a collapsed lung which occurs if there is the availability of air or gas in the potential space between the visceral and parietal pleura of the lung, that can pair up oxygenation and ventilation. [39] It can be a complete collapse of the lung or a small portion of lung collapse. Due to pneumothorax the lung cannot expand as much as it should despite inhaling. At times, a segment of the lung can collapse because of a blockage or a need for pressure inside the lung, rather than pressure from the exterior of the lung. This condition is called atelectasis. There are several sorts of Pneumothorax and one can have more than one at a time and the symptoms can range from mild to severe. Severe range of Pneumothorax can be fatal however, Pneumothorax which is not very serious can heal on its own. Besides, severe pneumothorax can lead to another disease.

3.2 Detecting Pneumothorax

Generally, pneumothorax is detected using a chest X-ray. However, Computerized Tomography (CT) scans can provide more detailed images. Ultrasound imaging is another way to identify pneumothorax to some extent. However, for the detection of a pneumothorax, ultrasound has comparatively high sensitivity than traditional upright anteroposterior chest radiography (CXR).

Furthermore, to find out symptoms of pneumothorax, sonographic signs, including 'lung sliding', 'B-lines' or 'comet tail artifacts', 'A-lines', and 'the lung point sign' can also work [38]. However, such clinical diagnosis methods are relatively expensive. X-rays can result in tissue effects like cataracts, skin reddening, and hair loss. The reason behind is the relatively high levels of radiation exposure which can also cause cancer in later life. Besides CT scan exposes the patient to a very high radiation dose which however cannot be performed on pregnant women. [12] Moreover, in villages and remote areas laboratories and skilled professionals are not so readily available which results in low accuracy in this diagnosis. Modern systems can use several algorithms for image analysis. To detect pneumothorax automatic pneumothorax cell detection which requires an image of the x-ray sample of the patient is a prominent solution.

In this study we trained and implemented multiple accurate and efficient CNN and FCN models for pneumothorax cell detection in cell images using available dataset which are public. These are collected from Kaggle.

3.3 Literature Review

We survey different papers of other publishers who work with similar cases to compare their results with ours.

For the first paper, authors Qingfeng Wang, Qiyu Liu, Guoting Luo, Zhiqin Liu, Jun Huang, Yuwei Zhou, Ying Zhou, Weiyun Xu and Jie-Zhi Cheng used a total of 11051 X-ray images among which 5566 images are pneumothorax affected and 5485 images are not affected. For training the datasets, they used Adam optimizer along with a learning rate of 1e-4 also including weight decay of 1e-4 at 200 epochs. The images were taken at 256 x 256 resolution. Finally at 200 epochs [19], the authors received the following results:

Serial	Model	Accuracy
1	MS_scSE_FC-DenseNet	0.86

Table 3.1: Metrics Result for Different Models [19]

From the second paper, author Smit Kumbhani used a dataset of 27299 images containing training dataset, mask images and test images. The images were batched into 32 batches and at 40 epochs, IoU was the evaluation metric. U-Net encoded with EfficientNetB4 was used as an optimizer for 40 epochs. [43]

Serial	Model	IoU
1	U-Net with EfficientnetB4	0.7196

Table 3.2: Metrics Result for Different Models [43]

Finally, for our third paper, author Hongyu Wang, Hong Gu, Pan Qin and Jia Wang used PyTorch and torchvision for model training. They took 4 images each batch and 512 sampled RoIs for each image at a ratio of 1:3 of positives to negatives. Six CheXLocNets were trained for the process by in three stages using learning rate values of 10e−3, 10e−4, and 10e−5 at 10 epochs.[20]

Serial	Model	IoU
1	U-NET	0.76
2	ChexLocalNet I	0.70
3	ChexLocalNet II	0.75
4	ChexLocalNet III	0.69
5	ChexLocalNet IV	0.74
6	ChexLocalNet V	0.77
7	ChexLocalNet VI	0.69

Table 3.3: Metrics Result for Different Models [20]

3.4 Used Dataset

The dataset is taken from Kaggle which is a website for different experimental dataset. There are in total 27,299 images, among which, 12047 images are training datasets, 12047 numbers the mask for the training datasets and 3205 are our test datasets. The training datasets contain a mixture of Pneumothorax affected and non-infected chest x-ray images. The mask images pass through a number of different convolutional layers and then it learns the features of images according to its corresponding masks. It gives the label to objects and then it learns the features of images passed in its training. It will match the object of the image with its corresponding mask to learn the object features only and not unnecessary objects features. The images finally represent the affected region through black and white format. The test images are provided to determine the Pneumothorax affected region using our segmentation model. The images are of 3-channels (RGB). Their size varies between 110 -150 pixels which we later re-sampled to 256 x 256 output dimension, a channel depth of 3 to suit the input requirements of different classification algorithms we used. Various pre-processing techniques are also applied. Example of 10 samples of mask images is shown below. [Figure 3.1] Secondly, 10 samples of train images are given in [Figure 3.2]. Finally, samples of test images are given [Figure 3.3].

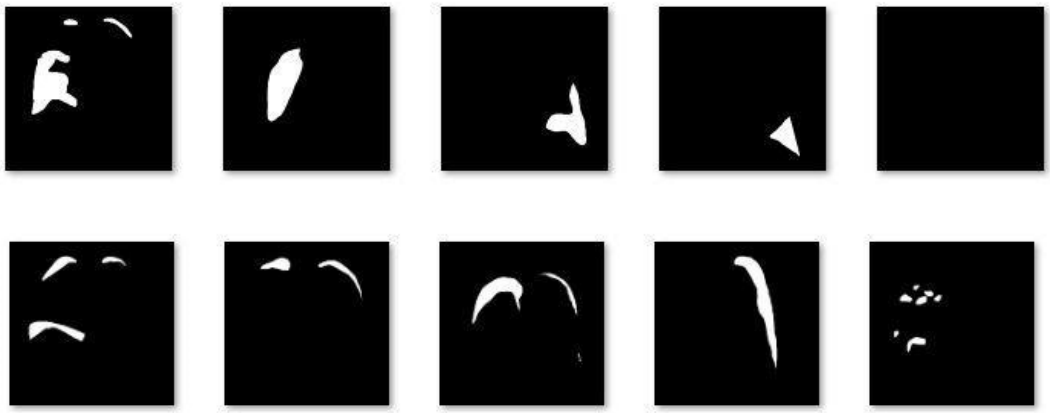


Figure 3.1: 10 Samples of Mask

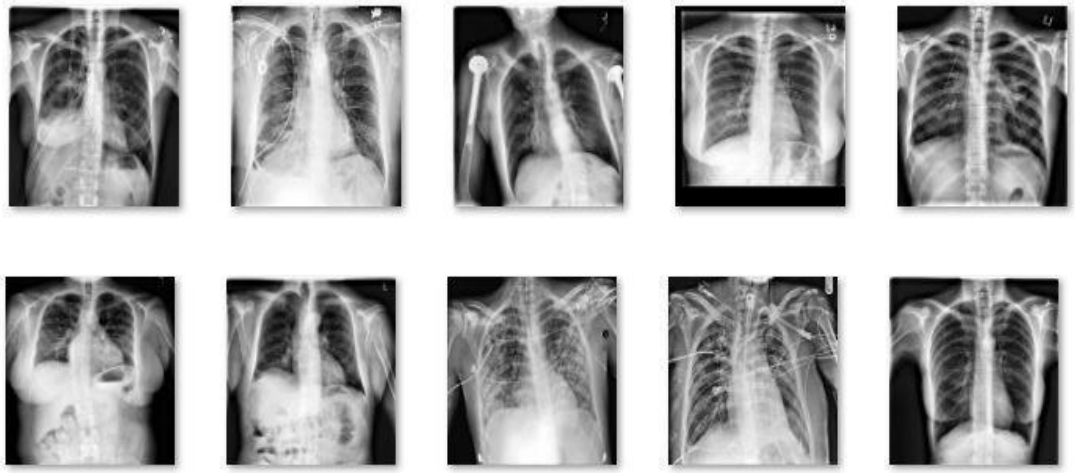


Figure 3.2: 10 Samples of Train

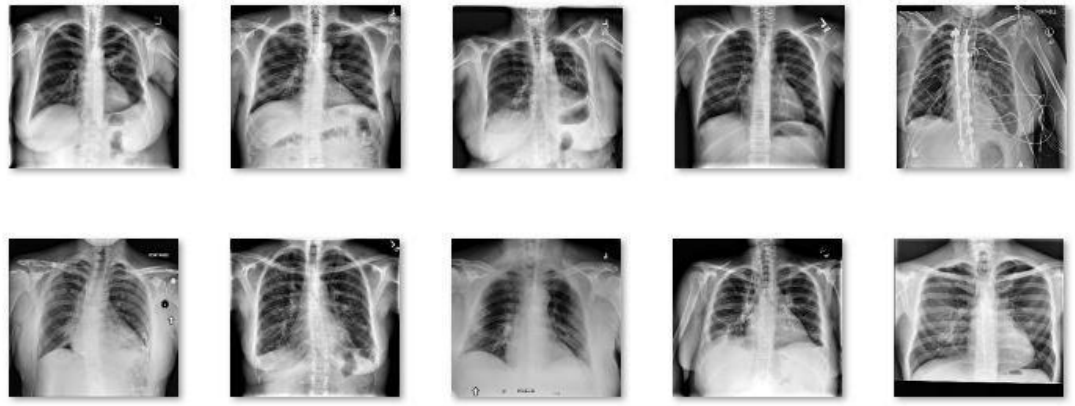


Figure 3.3: 10 Samples of Test

3.5 Basic Process which is followed

This section indicates the process of our project. It signifies the method that we have followed for the project. Here we have used U-Net as our model and pre-trained InceptionV3, ResNet34 and Xception as encoders.

3.5.1 Information about the ratio of Test and Training

The table below indicates the Epoch, Batch size and the details about the ratio of Test and Train.

Information	Details
Test and Train	20 percent and 80 percent is the ratio
Epoch	30
Batch size	32

Table 3.4: Epoch, Batch size & Ratio of Test and Train

3.5.2 Training the Model

We trained a U-Net model and also a U-Net model with three different pre-trained encoders separately. For the training process, we at first batched our data to 32 groups and took epochs of 30. We then changed certain values of data augmentations, that is, Horizontal Flip, Random Contrast, Random Brightness, Elastic Transform, Optical Distortion. At our final activation layer, we took Adam optimizer at a learning rate of 0.0001. We repeated the same process a few times for the different models.

3.5.3 Model building and model summary

For our project, using the same architecture we have used U-Net along with Xception, ResNet34 and InceptionV3 as pre-trained U-Net encoders for our segmentation model. The model summary for one of the models that we have used for the segmentation is shown below [Figure 3.4].

```
from segmentation_models import Unet

model = Unet('inceptionv3', input_shape=(256, 256, 3), encoder_weights=None)
model.summary()
```

```
Segmentation Models: using `keras` framework.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/ops/resource_variable_ops.py:1
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_backend.py:4070: The name tf

WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/keras/backend/tensorflow_backend.py:4074: The name tf

Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 256, 256, 3)  0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 128, 128, 32) 864         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 128, 128, 32) 96          conv2d_1[0][0]
__________________________________________________________________________________________________
decoder_stage4_upsampling (UpSa (None, 256, 256, 32) 0           decoder_stage3b_relu[0][0]
__________________________________________________________________________________________________
decoder_stage4a_conv (Conv2D)   (None, 256, 256, 16) 4608        decoder_stage4_upsampling[0][0]
__________________________________________________________________________________________________
decoder_stage4a_bn (BatchNormal (None, 256, 256, 16) 64          decoder_stage4a_conv[0][0]
__________________________________________________________________________________________________
decoder_stage4a_relu (Activatio (None, 256, 256, 16) 0           decoder_stage4a_bn[0][0]
__________________________________________________________________________________________________
decoder_stage4b_conv (Conv2D)   (None, 256, 256, 16) 2304        decoder_stage4a_relu[0][0]
__________________________________________________________________________________________________
decoder_stage4b_bn (BatchNormal (None, 256, 256, 16) 64          decoder_stage4b_conv[0][0]
__________________________________________________________________________________________________
decoder_stage4b_relu (Activatio (None, 256, 256, 16) 0           decoder_stage4b_bn[0][0]
__________________________________________________________________________________________________
final_conv (Conv2D)             (None, 256, 256, 1)  145         decoder_stage4b_relu[0][0]
__________________________________________________________________________________________________
sigmoid (Activation)            (None, 256, 256, 1)  0           final_conv[0][0]
==================================================================================================
Total params: 29,933,105
Trainable params: 29,896,689
Non-trainable params: 36,416
```

Figure 3.4: Segmentation Using Inception Model Summary

The encoder that we have used here in the U-Net architecture is InceptionV3. The shape for the input dataset is 256 x 256 x 3. The Inception architecture contains a total parameter of 29,933,105 where trainable parameters are 29,896,689 and 36, 416 are non-trainable parameters. The segmentation model epochs are provided below [Figure 3.5]

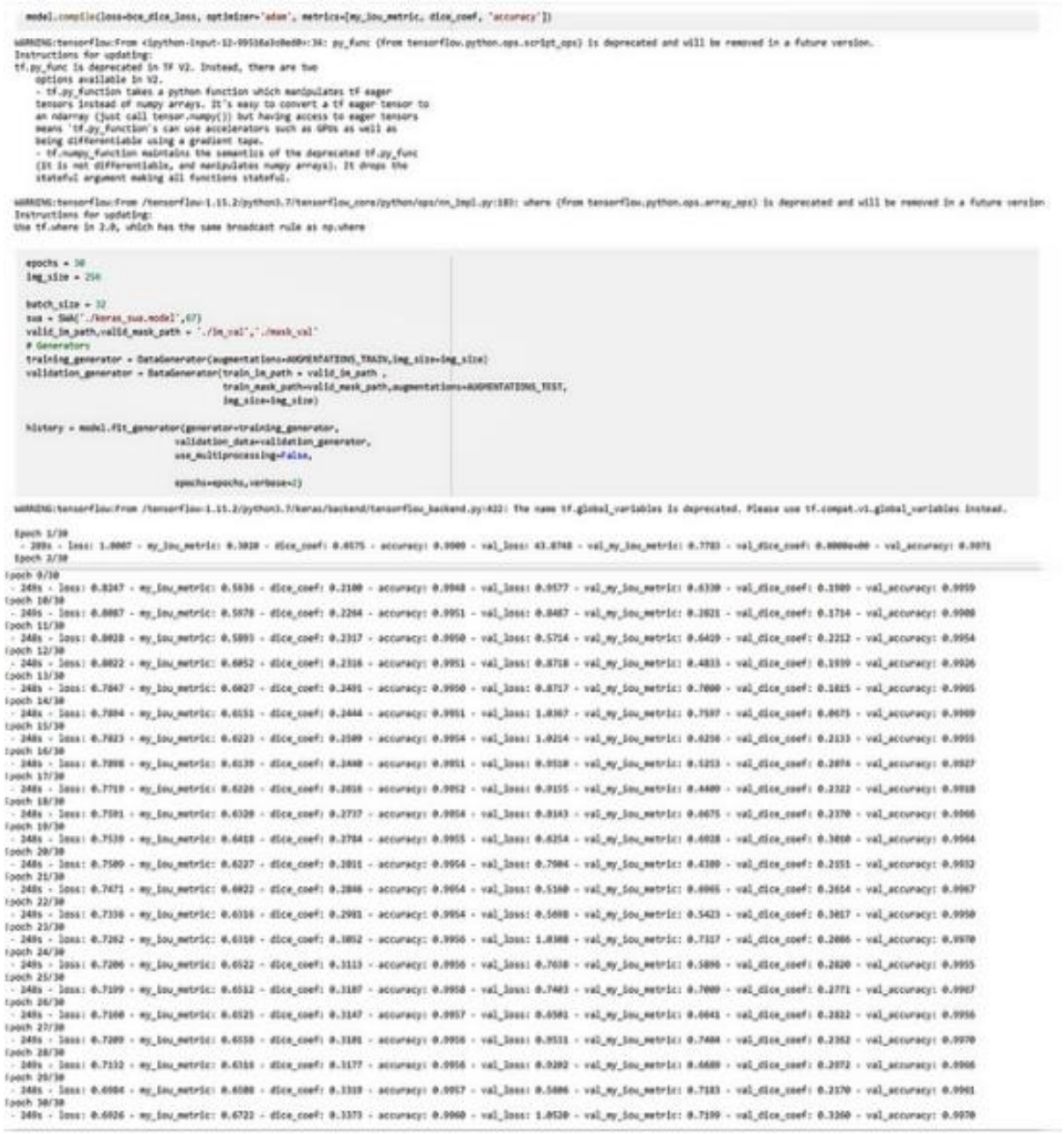


Figure 3.5: Segmentation Model Epochs

Here, we have taken Epoch 30 and batch size is provided 32. From here we can determine the value for different epochs, the value of IoU, and accuracy. The curves for the stated options are also generated from this process.

3.6 Outcomes and Discussion

In this section, the results or the outcomes of the project are discussed. In this section, the discussed topics are model metric and loss comparison. The reports will provide the decision of the segmentation and also detection process.

3.6.1 Model metrics

In this section, the IoU curve and accuracy curve are provided for 30 Epochs and 32 being the batch size. However, we have tested by changing the batch size and also multiple models were used to get the best result. So, the best model that we could find is U-Net encoded with InceptionV3. It provided us with the best result as detailed information: for the training dataset, IoU metric is 0.7685, accuracy is 0.9974. For validation datasets, IoU metric is 0.7633, accuracy is 0.9972. For the above stated epoch and batch size, IoU Curve [Figure 3.6], Accuracy [Figure 3.7] are as follows.

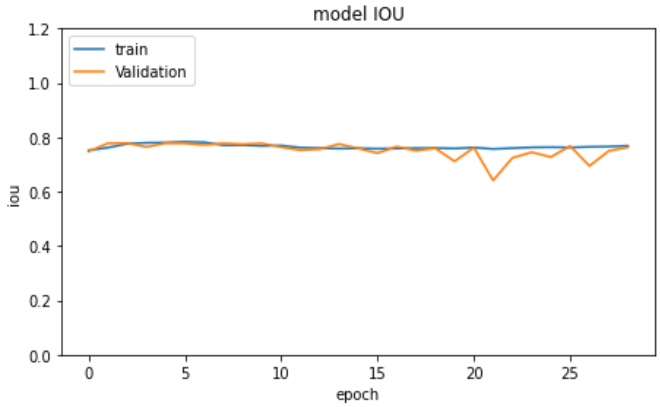


Figure 3.6: IoU Curve for U-Net encoded with Inception V3

Here, 30 epochs and a batch size of 32, IoU metric for training dataset and validation dataset are 0.7685 and 0.7633 respectively.

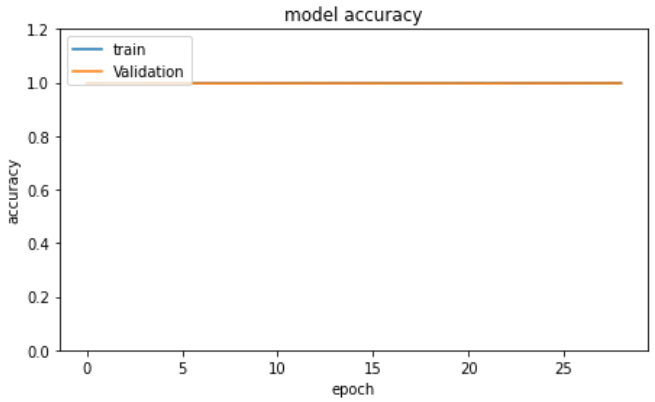


Figure 3.7: Accuracy Curve for U-Net encoded with Inception V3

The basic accuracy function formula is $\boldsymbol{A = (TP + TN) / (TP + TN + FP + FN)}$. Accuracy represents the percentage of accurate predictions that we have completed. In our project, we have different models but we kept the Epoch and batch size the same. The accuracy curve generated for InceptionV3 in the above graph shows, accuracy for training dataset 0.9974 and validation dataset 0.9972.

Similarly, at 30 Epoch and Batch Size of 32, graphical results for various models are shown below.

1. **U-Net:** The IoU Curve and also Accuracy Curve are given below in [Figure 3.8].

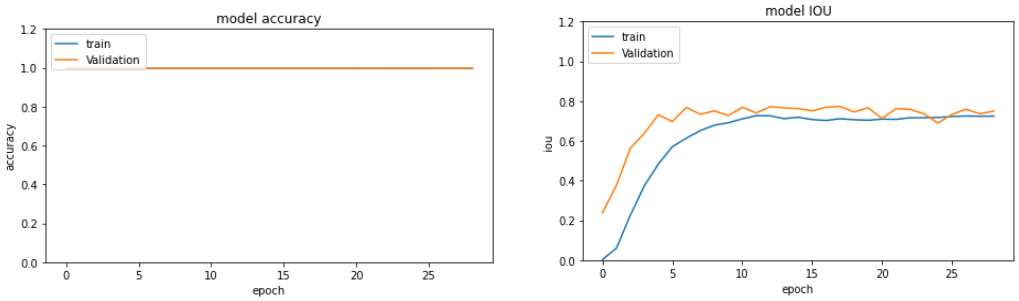


Figure 3.8: IoU Curve, Accuracy Curve for U-Net

From the above graph we get, for the training dataset, IoU metric of 0.7240 and accuracy of 0.9970. For the validation dataset, we get IoU metric of 0.7494, and accuracy of 0.9972.

2. **U-Net encoded with ResNet34:** The full form of ResNet is Residual network. ResNet34 is another model that we used to perform the segmentation. Using this model along with

the U-Net architecture, we received an IoU metric of 0.7688 and accuracy of 0.9972 for training datasets. And, IoU metric of 0.7634 and accuracy of 0.9971 for validation datasets [Figure 3.9].

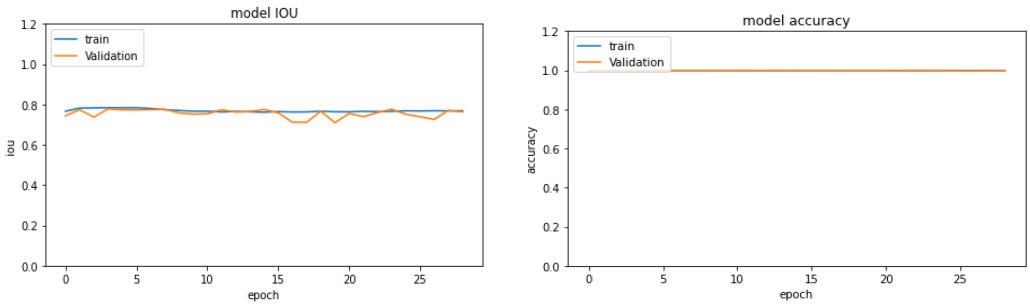


Figure 3.9: IoU Curve and Accuracy Curve for U-Net encoded with ResNet34

3. **U-Net encoded with Xception:** After using the Xception model as an encoder in the U-Net model, we received an IoU metric of 0.1561 and accuracy of 0.9984 for training datasets. And, IoU metric of 0.7654 and accuracy of 0.9976 for validation datasets [Figure 3.10].

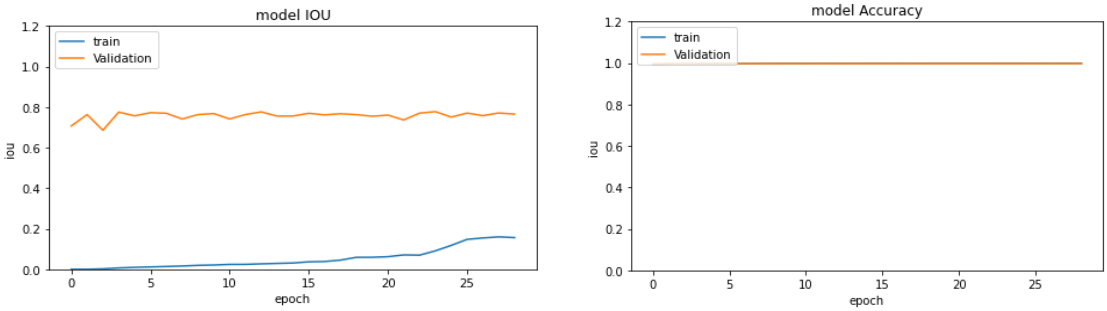


Figure 3.10: IoU Curve and Accuracy Curve for U-Net encoded with Xception

3.6.2 Model loss

Model loss represents the lack of prediction. The value of the loss is greater if the prediction is bad. If the accuracy of the model is greater in value, then the value for loss would be close to zero. We can get a report of validation and training data in the loss curve as well. The models that we used are U-Net and U-Net encoded with Inception V3, Resnet34 and Xception.

Loss using U-Net:

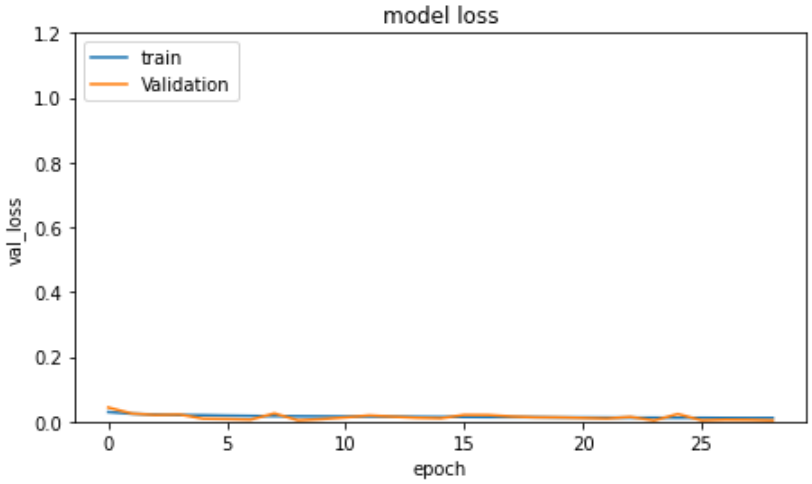


Figure 3.11: Loss Curve using U-Net

Loss using U-Net encoded with ResNet34:

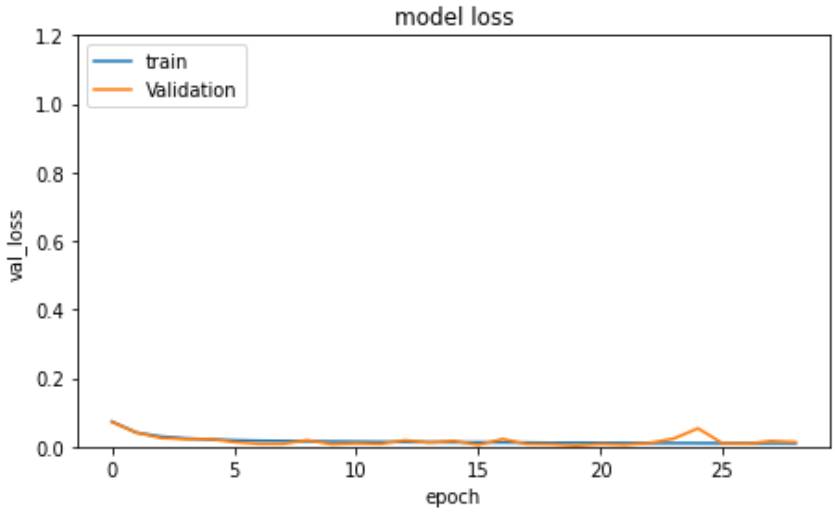


Figure 3.12: Loss Curve using U-Net encoded with ResNet34

Loss using U-Net encoded with InceptionV3:

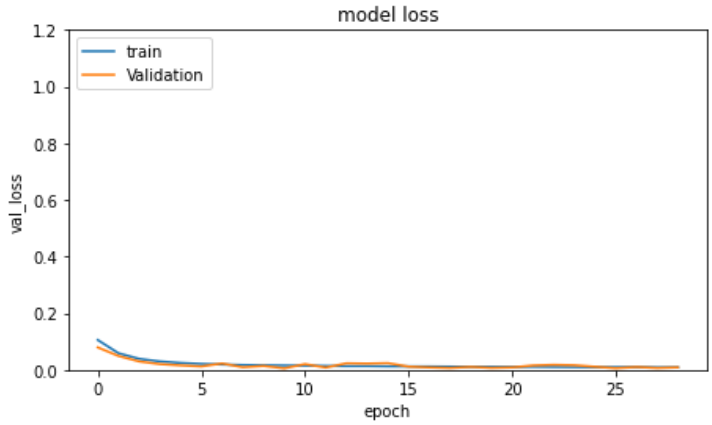


Figure 3.13: Loss Curve using U-Net encoded with InceptionV3

Loss using U-Net encoded with Xception:

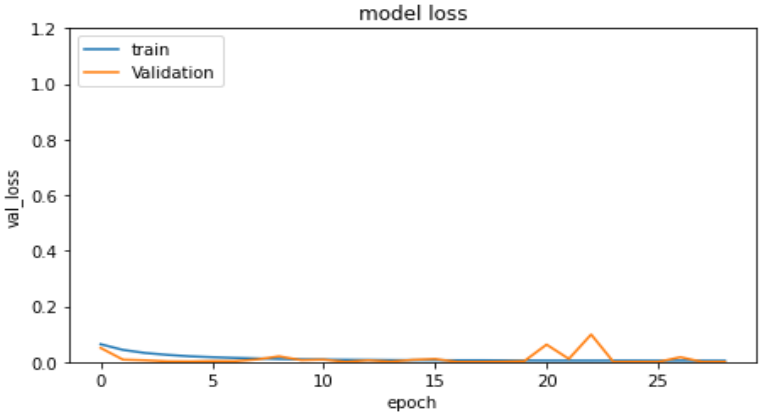


Figure 3.14: Loss Curve using U-Net encoded with Xception

3.6.3 Output Images

After doing Prediction over our models, we get the best Output Images with high accuracy with the U-Net encoded with InceptionV3 model. In [Figure 3.15] and [Figure 3.16] we can see the segmentation of pneumothorax specified with two different colors. The red color indicates the Real mask provided in Dataset and the green color indicates the generated mask that was generated in our project. If the red color and green color overlapped each other, it would mean our prediction for this image is correct. On the other hand, if these do not overlap each other, it will mean our prediction for that image is wrong. Again, the red colored area from our output test images [Figure 3.17] shows that, using this architecture can easily detect the affected areas on X-ray images. Thus, in our output images, we can see, most of the images are accurate as our model accuracy is higher than loss.

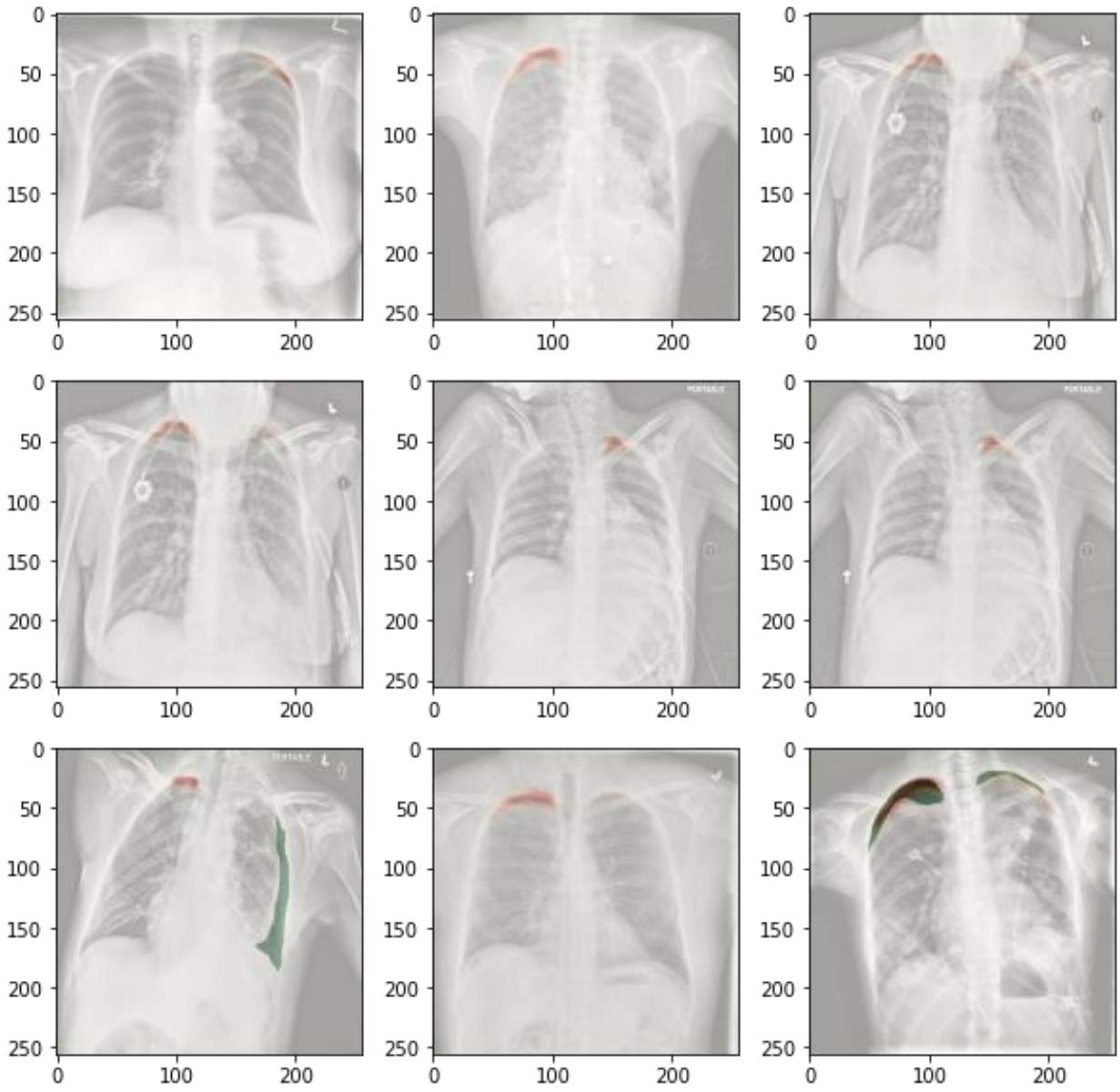


Figure 3.15: Output Image for U-Net encoded with InceptionV3 model

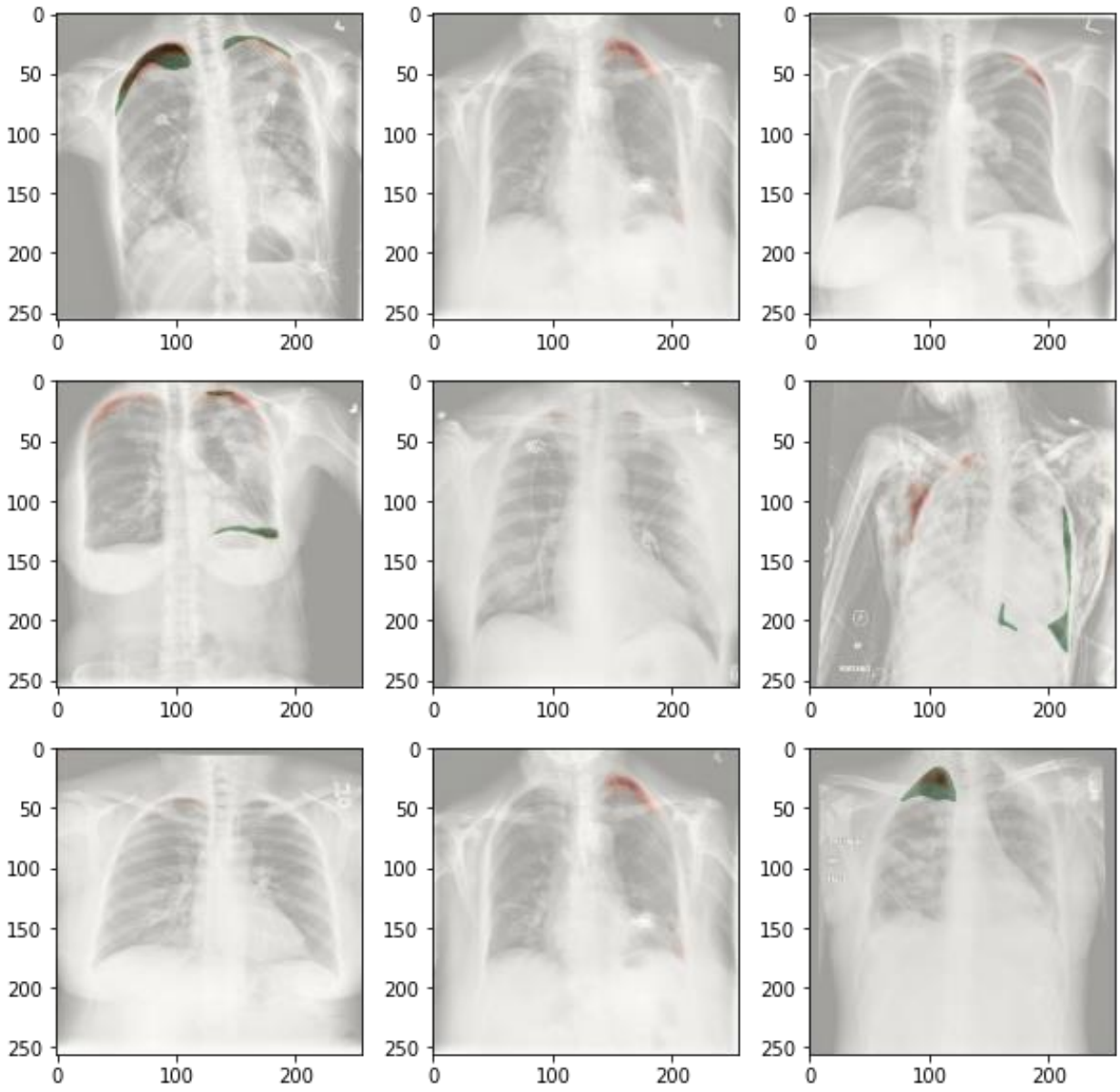


Figure 3.16: Output Image for U-Net encoded with InceptionV3 model

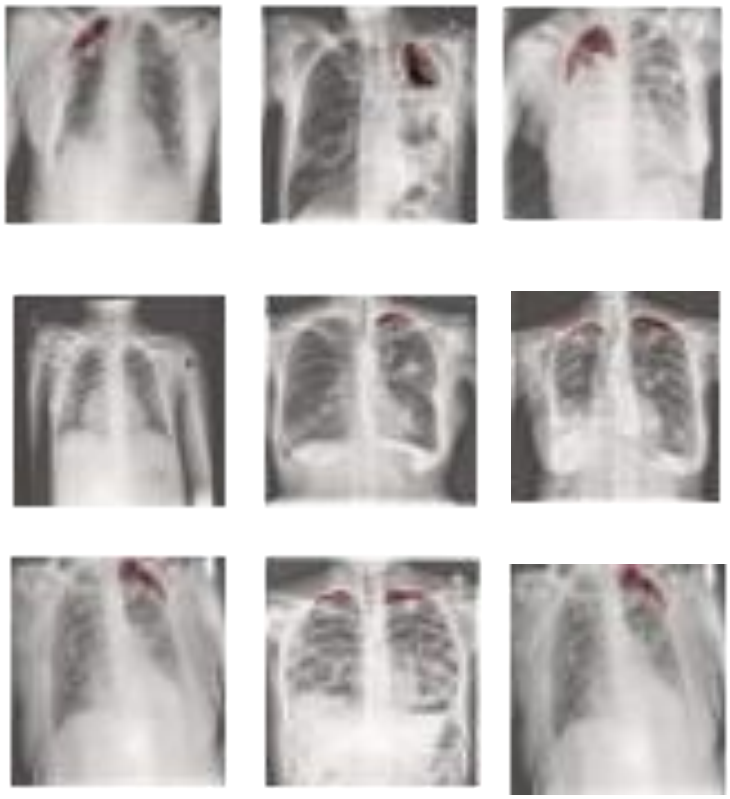


Figure 3.17: Output Test Images for U-Net encoded with InceptionV3 model

3.6.4 Comparison Between results

Comparison between different models gives us a clear overview of what model gives us the most accurate answer or what is the most effective model between the 3 models we have used U-Net and U-Net encoded with Inception V3, Resnet34 and Xception. Here is a table provided for the comparison between different parameters of the 3 models. The Epoch is 30 and the batch size is kept at 32 for the best possible prediction. [Table:3.2]

Model	Validation Data			Training Data		
	Accuracy	IoU	Loss	Accuracy	IoU	Loss
U-Net	0.9970	0.72	0.0035	0.9972	0.75	0.0110
U-Net with ResNet34	0.9971	0.76	0.0132	0.9972	0.77	0.098
U-Net with InceptionV3	0.9972	0.76	0.0088	0.9974	0.77	0.0087
U-Net with Xception	0.9976	0.76	2.2147e-04	0.9984	0.16	0.0043

Table 3.5: Results from all the Used Models

Combining and comparing with all the results we received from our literature review we get,

Serial	Model	Accuracy	IoU
1	MS_scSE_FC-DenseNet	0.86	-
2	U-Net with EfficientnetB4	-	0.7196
3	U-NET		0.76
4	ChexLocalNet I		0.70
5	ChexLocalNet II		0.75
6	ChexLocalNet III		0.69
7	ChexLocalNet IV		0.74
8	ChexLocalNet V		0.77
9	ChexLocalNet VI		0.69
12	***U-Net with Xception (ours)***	***0.9974***	***0.77***

Table 3.6: Result and Literature Comparison

Even though the difference between the models doesn't exceed a higher percentage, at a definite and limited GPU, our models showed smoother results compared to the other models. The literature gap previously with batch size 16 remained at 0.02 while with increasing the batch size it increased to 0.06 value. Besides, the loss function of one of the above-mentioned models didn't decrease for the validation batch, while our model was seen to eventually decrease starting from an extremely low value.

3.6.4 Chapter Conclusion

This chapter gives us information about the Datasets, prediction results along with image results using color image segmentation, IoU curves and accuracy curves for three different models, the comparison between the losses along with the curves of loss. To conclude, among the 4 models which are U-Net and U-Net encoded with InceptionV3, ResNet34 and Xception we get the best possible prediction from the InceptionV3 model as it gives us the highest accuracy value which is 0.9974. The above-given comparisons and graphs among different models also indicate it.

Chapter 4

Deep Learning Algorithm in Cell Nucleus Segmentation

4.0 Chapter Introduction

In this chapter, we will see how we have worked on several deep learning models for the segmentation of nucleus cells from optical images. Histopathology is the detection of diseases of the tissues and it requires examining tissues or cells under a microscope. Often, histopathology reports are known as pathology reports or also biopsy reports. In the following subchapters, we will explain in detail how we compiled, trained our model, evaluated the model and differentiated the results.

4.1 About Cell Nucleus

A cell is the basic structure of living organisms. A nucleus is a membrane-bound organelle that involves the cell's chromosomes. A cell contains many organelles. Among organelles, the cell nucleus is the most significant one. Organelles have two main functions: they store the cell's genetic material or DNA, and they control the cell's operations, which include development, intermediate metabolism, protein synthesis, and cell division. In disease detection cell size, number and shapes play a significant role. A cell's nucleus regulates and controls its operations, such as growth and metabolism, and contains genes and other structures that contain hereditary information.[27]

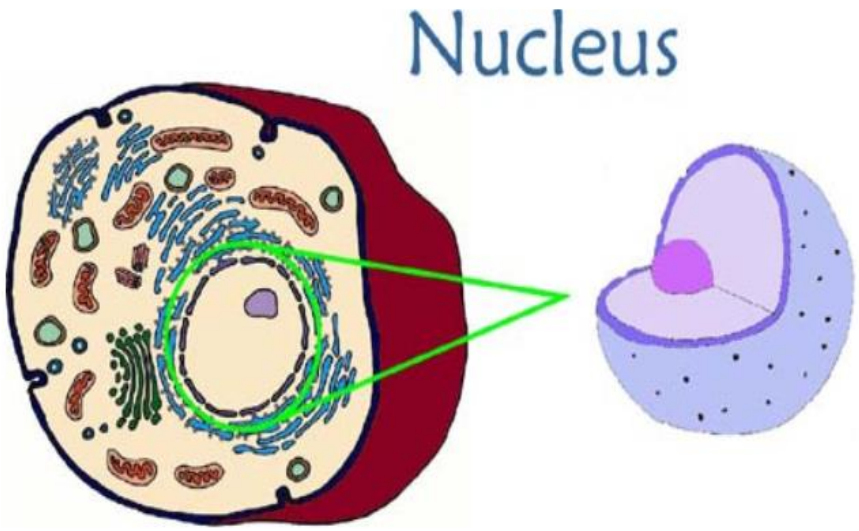


Figure 4.1: Cell Nucleus [28]

The nucleus consists of various components, including the nuclear envelope, nuclear lamina, nucleolus, chromosomes, and nucleoplasm. For the nucleus to perform all of its operations, all of these components must work together.

4.2 Cell Nucleus Segmentation from Divergent Image

Traditionally cell nucleus detection requires an Optical test for the medical diagnosis that is done by pathologists under the microscopes. Nuclei detection and segmentation following medical diagnosis are faced with multiple challenges. The main types of specimens are received by the pathology laboratory which is a costly and time-consuming process. Annually a significant number of people die due to the disease of cancer as it takes a long to detect disease and due to its invasive nature, it is very difficult to cure with the increasing stage. Therefore, analyzing the primary mutation in cells of the patient biopsy or histopathology is foremost to survive from this disease.

In the quantitative analysis of imaging data for biological and biomedical applications, segmenting the nuclei of cells in microscope images is generally the very primary step [21]. With the rapid

growth of deep learning over the past few years, DL-based methods are widely adopted in computational pathology and multiple applications are being applied to cell nucleus segmentation for faster detection. Deep learning has remarkable possibilities to interpret strenuous problems in cellular image segmentation [21]. However, these DL techniques are usually complex and require different algorithms, model ensemble, pre, and post-image processing and powerful computing resources, etc. In this research, we have trained and implemented U-Net/U-Net++ models for detection in cell nuclei using divergent images from a publicly available dataset in Kaggle.

4.3 Literature Review

We survey different papers of other publishers who work with a similar case to compare their results with ours.

In the first paper, Nuclei Segmentation in Optical Images Using Deep Neural Networks, Mr. Peter Naylor and his team practice with 3 well-known architectures in semantic segmentation, named PangNet, Fully Convolutional Net (FCN) and DeconvNet. They used accuracy, intersection over union (IU), F1 score and a performance score. In Table 4.1 we can see the results they got from these models. [22]

	PangNet	DeconvNet	FCN	Ensemble
Accuracy	0.924	0.954	0.944	0.944
IoU	0.722	0.814	0.782	0.804
Recall	0.655	0.773	0.752	0.900
Precision	0.814	0.864	0.823	0.741
F1	0.676	0.805	0.763	0.802
Performance	0.803	0.875	0.862	0.922

Table 4.1: Metrics Result for Different Models [22]

In the second paper, Nuclei Segmentation and Detection using Deep Convolutional Neural Networks, Mr. Rohan Pudipeddi and his team chose a modified CNN architecture, U-Net. The results obtained show that U-Net can be a promising approach to efficiently segment and identify the cell's nuclei. They used Evaluation Metrics for 200 epochs. The evaluation metrics used were Intersection over Union (Jaccard Distance) and F1 score also known as Dice Coefficient. In Table 4.2 we can see the IoU and F1 results for Various models where they get the best result by using U-Net.[17]

Model	IoU	F1
LinkNet	75.12	85.66
SegNet	75.40	85.83
ENet	74.69	85.35
PSPNet	75.92	86.16
ICNet	74.19	85.05
U-Net	**88.70**	**91.82**

Table 4.2: Metrics Result for Different Models [17]

In the third paper, Microscopy cell nuclei segmentation with enhanced U-Net, Mr. Feixiao Long reformed the encoding branch of U-Net. He changes the number of filters in the first layer, customized to 8 and 16 and the model depth was 5 for both cases. He got 17% of validation loss, 0.503±0.217 of average IoU and 57.1% of average precision for 25 epochs.[23]

	U-Net	U-Net++ *	U-Net+ (T.C.)	U-Net+ (U.S.)
Ave. IOU (m±s)	0.526±0.170	0.564±0.181	0.572±0.179	0.567±0.177
Ave. precision (%)	59.5	63.0	62.5	62.4
No. of weights (M)	0.486	0.566	0.484	0.435
GPU infer. time (ms/patch)	9.5	14.1	12.4	14.0

Table 4.3: Metrics Result for Different Models [23]

4.4 Used Datasets

The dataset we have used to feed our algorithm is a publicly used dataset from Kaggle [26]. A huge number of portioned nuclei images are included in this dataset. The images were obtained underneath a variety of conditions and change within the cell variety, exaggeration, and imaging methodology. To test an algorithm's capacity to extrapolate over these varieties, the dataset has been planned. Every picture is defined by a related ImageId. There is a folder with ImageId. That folder contains files that have a location for an image. There are two subfolders within this folder.

- The image file is stored in the images folder.
- Masks include the nucleus's segmented masks. Only the training set incorporated this folder. Each mask has a single nucleus. Overlapping masks are not permitted.

Images from concealed experimental conditions are stored in the second stage dataset. As they are human-annotated datasets, several types of errors can be identified. If errors in the training set are discovered, they can be manually corrected.

There are in total 670 numbers of train images with multiple masks and 65 test images in our dataset. The images are of 3-channels (RGB). Their size varies between 110 -150 pixels which we later re-sampled to 256 x 256 output dimension, a channel depth of 3 to outfit the input

requirements of different classes of algorithms that we used. Various pre-processing methods are enforced too. There are three samples used here which are mask train and test samples. The mask passes through several different convolutional layers and then it learns the features of images according to their corresponding masks. It gives the label to objects and then it learns the features of images passed in its training.

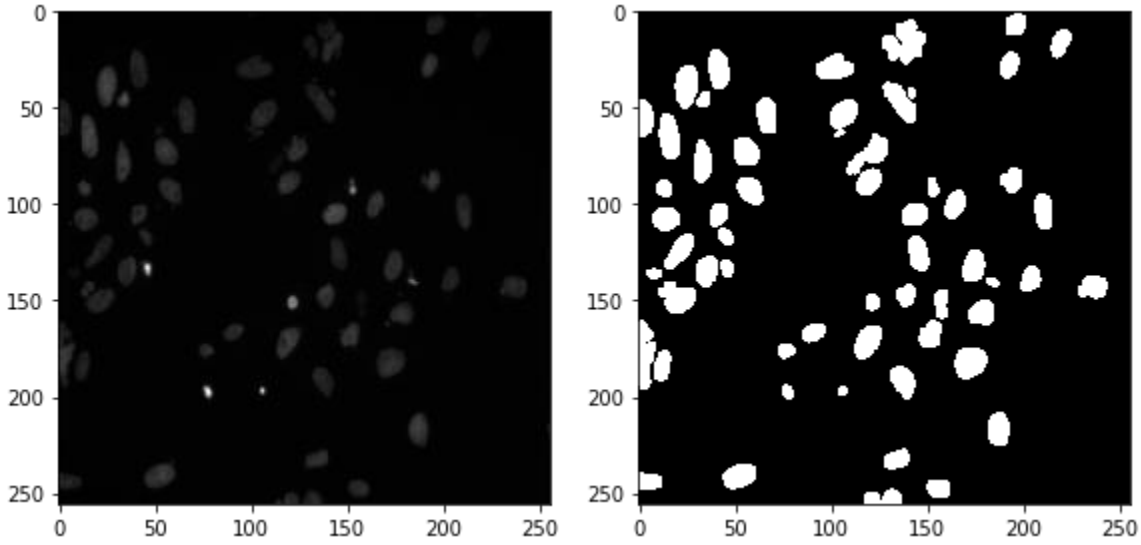


Figure 4.2: Input Dataset for U-Net

4.5 Basic Procedure

The following chapter consists of our second code process which is for the cell nucleus. The subsequent subsections under 4.5 illustrate the steps we have used to implement our models. We have used a similar process using U-Net, U-Net encoded with InceptionV3.

4.5.1 Information about the ratio of Test and Training datasets

The table below indicates the Epoch, Batch size and the details about the ratio of Test and Train datasets.

Information	Details
Test and Train	20 percent and 80 percent is the ratio
Epoch	30
Batch size	32

Table 4.4: Epoch, Batch size & Ratio of Test and Train

4.5.2 Model building and Model Summary

For our project, using the same architecture we have used U-Net and U-Net encoded with DenseNet121, ResNet34 and InceptionV3 as segmentation models. Now for the code that we used for the segmentation is [Figure 4.3].

```
inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = Lambda(lambda x: x ) (inputs)
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (s)
c1 = Dropout(0.1) (c1)
c1 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c1)
p1 = MaxPooling2D((2, 2)) (c1)
c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (p1)
c2 = Dropout(0.1) (c2)
c2 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c2)
p2 = MaxPooling2D((2, 2)) (c2)

c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (p2)
c3 = Dropout(0.2) (c3)
c3 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c3)
p3 = MaxPooling2D((2, 2)) (c3)

c4 = Conv2D(128, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (p3)
c4 = Dropout(0.2) (c4)
c4 = Conv2D(128, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c4)
p4 = MaxPooling2D(pool_size=(2, 2)) (c4)

c5 = Conv2D(256, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (p4)
c5 = Dropout(0.3) (c5)
c5 = Conv2D(256, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c5)

u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (c5)
u6 = concatenate([u6, c4])
c6 = Conv2D(128, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (u6)
c6 = Dropout(0.2) (c6)
c6 = Conv2D(128, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c6)

u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c6)
u7 = concatenate([u7, c3])
c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (u7)
c7 = Dropout(0.2) (c7)
c7 = Conv2D(64, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c7)

u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (u8)
c8 = Dropout(0.1) (c8)
c8 = Conv2D(32, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c8)

u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (u9)
c9 = Dropout(0.1) (c9)
c9 = Conv2D(16, (3, 3), activation='elu', kernel_initializer='he_normal', padding='same') (c9)

outputs = Conv2D(1, (1, 1), activation='sigmoid') (c9)
```

Figure 4.3 Segmentation using U-Net

The summary of our model:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape         Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 256, 256, 3) 0
_________________________________________________________________________________________________
lambda (Lambda)              (None, 256, 256, 3)  0           input_1[0][0]
_________________________________________________________________________________________________
conv2d (Conv2D)              (None, 256, 256, 16) 448         lambda[0][0]
_________________________________________________________________________________________________
dropout (Dropout)            (None, 256, 256, 16) 0           conv2d[0][0]
_________________________________________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 16) 2320        dropout[0][0]
_________________________________________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 16) 0           conv2d_1[0][0]
_________________________________________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 32) 4640        max_pooling2d[0][0]
_________________________________________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128, 32) 0           conv2d_2[0][0]
_________________________________________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 32) 9248        dropout_1[0][0]
_________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 64, 64, 32)   0           conv2d_3[0][0]
_________________________________________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 64)   18496       max_pooling2d_1[0][0]
_________________________________________________________________________________________________
dropout_2 (Dropout)          (None, 64, 64, 64)   0           conv2d_4[0][0]
_________________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 64)   36928       dropout_2[0][0]
_________________________________________________________________________________________________

anspose_2 (Conv2DTrans (None, 128, 128, 32) 8224        conv2d_13[0][0]
_________________________________________________________________________________________________
te_2 (Concatenate)     (None, 128, 128, 64) 0           conv2d_transpose_2[0][0]
                                                        conv2d_3[0][0]
_________________________________________________________________________________________________
 (Conv2D)              (None, 128, 128, 32) 18464       concatenate_2[0][0]
_________________________________________________________________________________________________
 (Dropout)             (None, 128, 128, 32) 0           conv2d_14[0][0]
_________________________________________________________________________________________________
 (Conv2D)              (None, 128, 128, 32) 9248        dropout_7[0][0]
_________________________________________________________________________________________________
anspose_3 (Conv2DTrans (None, 256, 256, 16) 2064        conv2d_15[0][0]
_________________________________________________________________________________________________
te_3 (Concatenate)     (None, 256, 256, 32) 0           conv2d_transpose_3[0][0]
                                                        conv2d_1[0][0]
_________________________________________________________________________________________________
 (Conv2D)              (None, 256, 256, 16) 4624        concatenate_3[0][0]
_________________________________________________________________________________________________
 (Dropout)             (None, 256, 256, 16) 0           conv2d_16[0][0]
_________________________________________________________________________________________________
 (Conv2D)              (None, 256, 256, 16) 2320        dropout_8[0][0]
_________________________________________________________________________________________________
 (Conv2D)              (None, 256, 256, 1)  17          conv2d_17[0][0]
=================================================================================================
ams: 1,941,105
 params: 1,941,105
able params: 0
_________________________________________________________________________________________________
```

Figure 4.4: Segmentation Model Summary

Epoch for segmentation as follows:

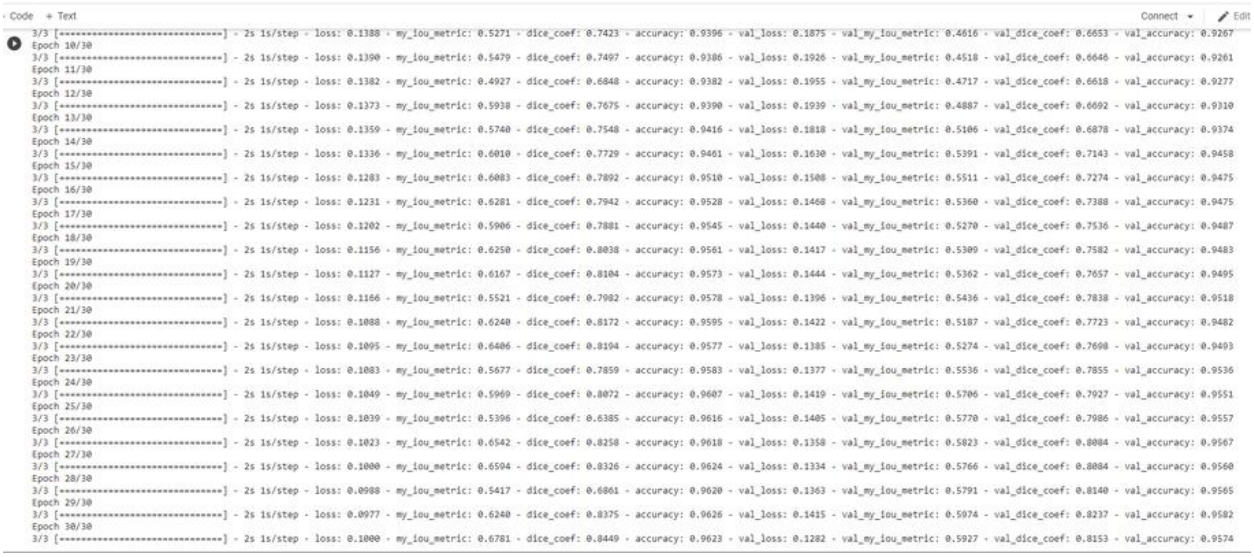


Figure 4.5: Epoch List for Segmentation

4.5.3 Test and Training the Model

While training the U-Net model, we have used 30 epochs and a batch size of 32. We received accuracy on the training dataset for IoU is 0.678 and accuracy is 0.9623 and on the validation set IoU is 0.5927, and Accuracy is 0.9574.

4.6 Outcomes and Discussions

In this part of the following chapter, the results or the discussion of the project are mentioned in Chapter 3.6. In this section, we will review model metrics, loss comparison. The reports will provide the decision of the segmentation and detection process.

4.6.1 Model Metrics

In this section, the IoU curve and accuracy curve are provided for 30 Epochs and 32 batch sizes. However, we have tested by changing the batch size and also multiple models were used to get the best result. The best model that we could find is U-Net as it provided us with the best result. For the training datasets, we received, IoU of 0.6781 and accuracy of 0.9623 and for the validation

datasets, we received, IoU of 0.5927 and accuracy of 0.9574. For above-stated epoch and batch size, IoU Curve [Figure 4.6] and Accuracy [Figure 4.8] graphs are as follows.

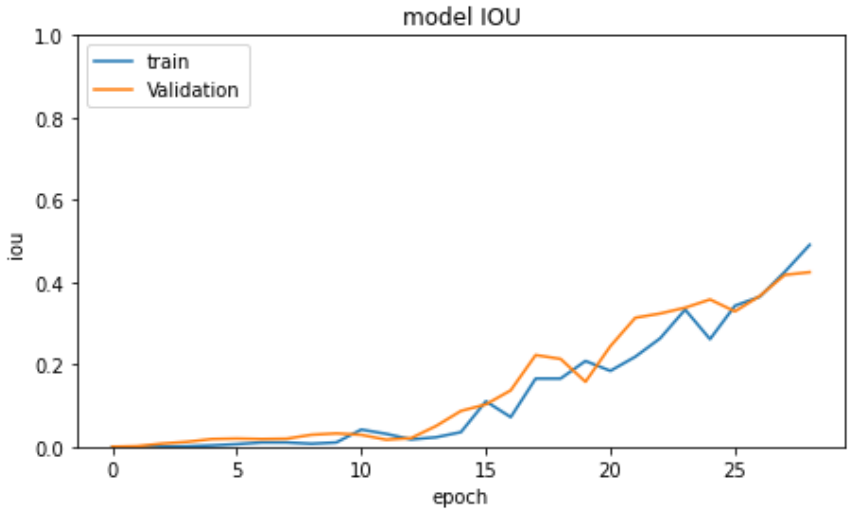


Figure 4.6: IoU Curve for U-Net

Here, IoU metric is 0.6781 for training datasets and 0.5927 for validation dataset at 30 epochs and batch size of 32.

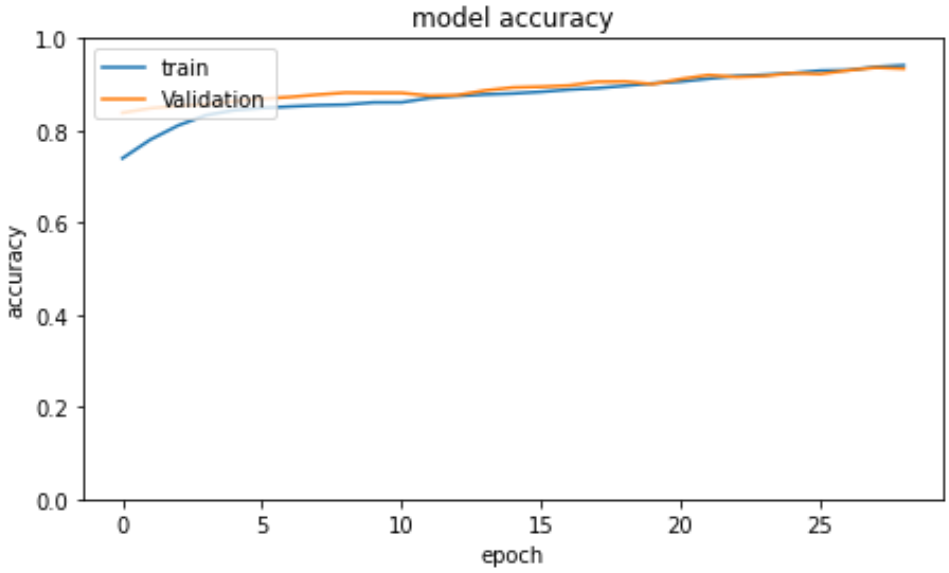


Figure 4.7: Accuracy curve for U-Net

Here in the above graph, accuracy for training dataset is 0.9623 and for validation dataset is 0.9574 at 30 epochs and batch size of 32.

Now similarly, For Same Epoch and Batch size which is 30 epochs and Batch Size of 32, graphical results for different U-Net encoded with different pre-trained models are provided below:

1. **U-Net encoded with InceptionV3:** Here, for training datasets, IoU is 0.6917 and accuracy is 0.9706 and for validation datasets, IoU is 0.4146 and accuracy is 0.9256.

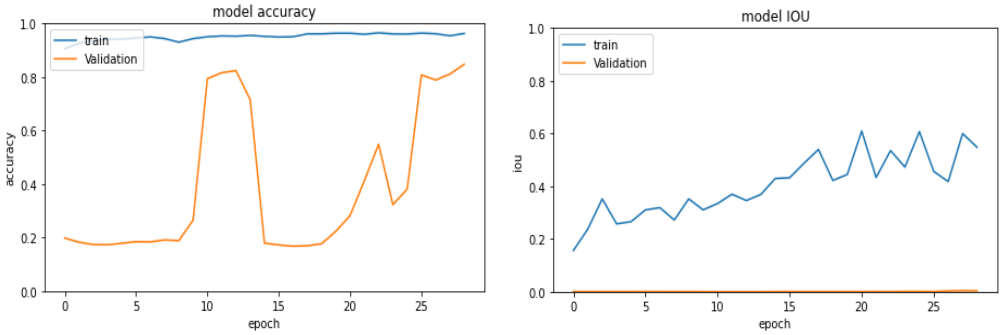


Figure 4.8: Accuracy and IoU Curve for U-Net encoded with InceptionV3

2. **U-Net encoded with ResNet34:** Here, for training datasets, IoU is 0.6066 and accuracy is 0.9667 and for validation datasets, IoU is 0.1145 and accuracy is 0.8751.

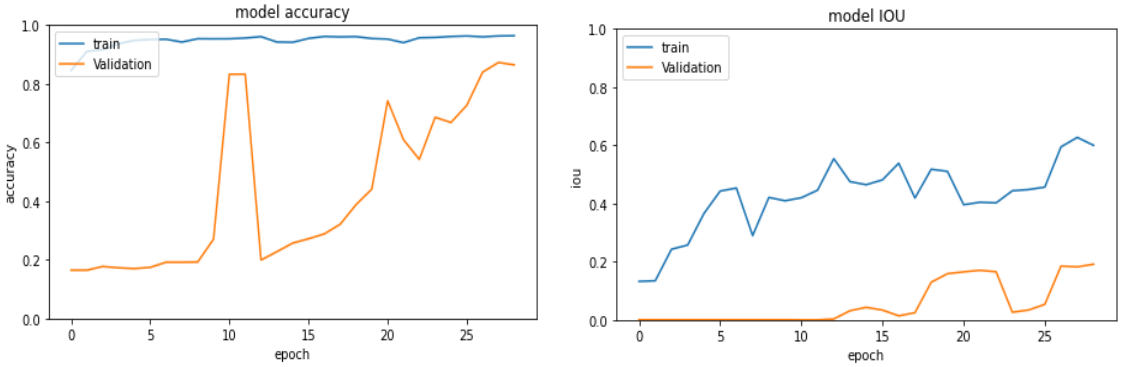


Figure 4.9: Accuracy and IoU Curve for U-Net encoded with ResNet34

3. **U-Net encoded with DenseNet121:** Here, for training datasets, IoU is 0.5197 and accuracy is 0.9564 and for validation datasets, IoU is 0.0302 and accuracy is 0.8682.

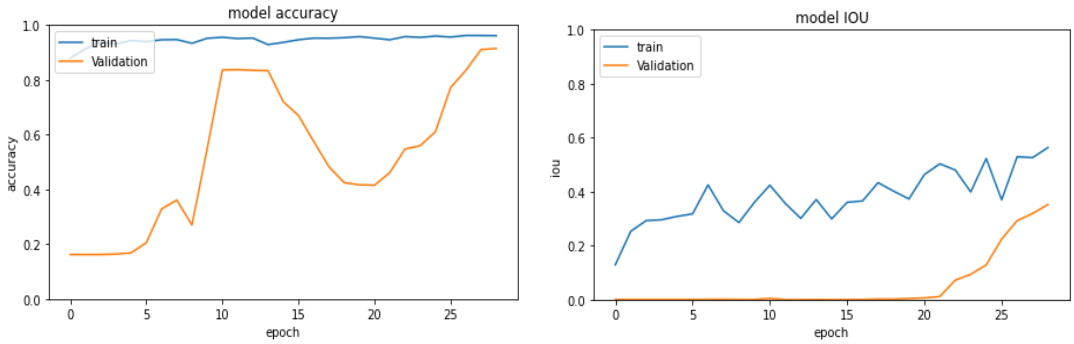


Figure 4.10: Accuracy and IoU Curve for U-Net encoded with DenseNet121

4.6.2 Model Loss

Loss using U-Net

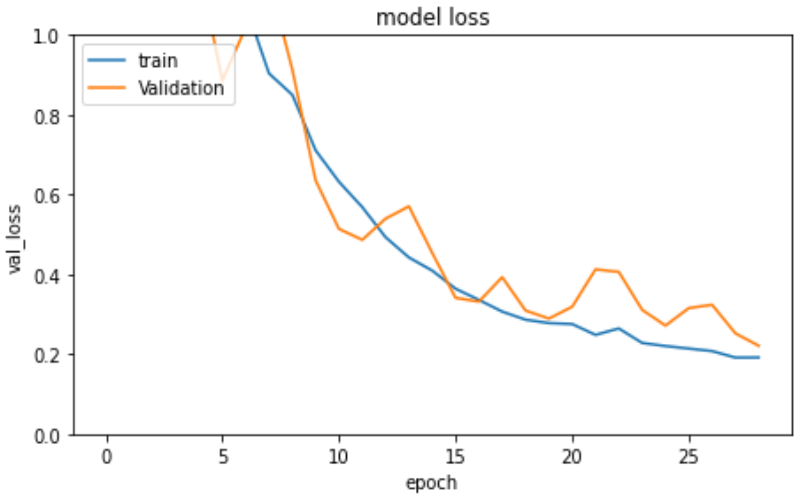


Figure 4.11: Loss Curve for U-Net

Loss using U-Net encoded with InceptionV3

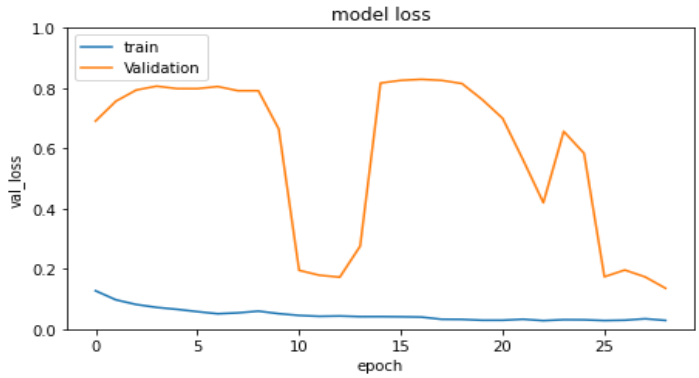


Figure 4.12: Loss Curve for U-Net encoded with InceptionV3

Loss using U-Net encoded with ResNet34

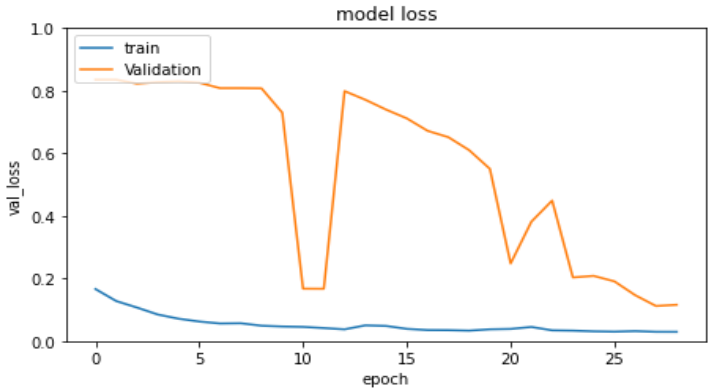


Figure 4.13: Loss Curve for U-Net encoded with ResNet34

Loss using U-Net encoded with DenseNet121

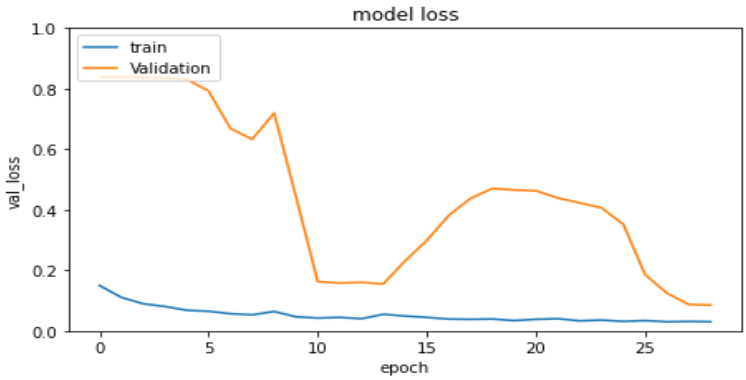


Figure 4.14: Loss Curve for U-Net encoded with DenseNet121

4.6.3 Output Image

After doing Prediction over our models, we get the best Output Images with high accuracy with the U-Net model.

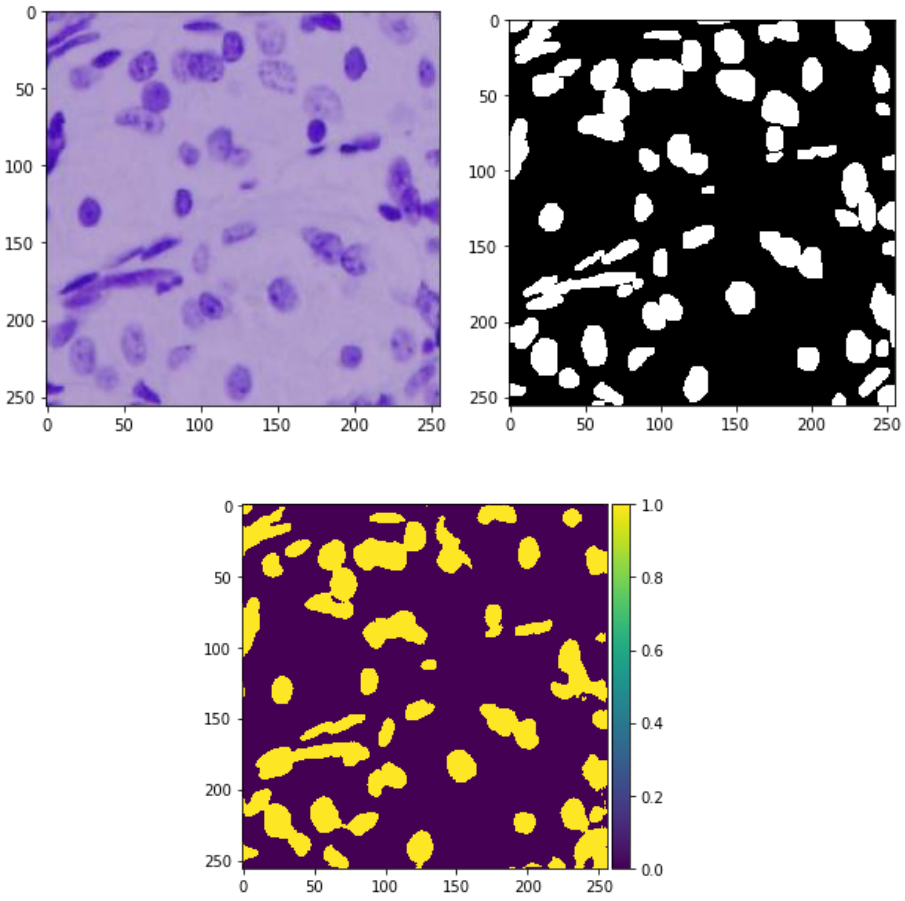


Figure 4.15: Output Image for U-Net

In [Figure - 4.12] we can see the segmentation of the cell nucleus. In the first output image, we have taken a random picture from train datasets. In the second output image, we found a merged image of all mask images of the first random picture and in our third output image, we have implemented color image segmentation. Yellow-colored masks denote nucleus cells on the other hand purple-colored masks identify different cells.

4.6.4 Comparison between Results

Now in Table 4.5 and Table 4.6, we summarized our results with different models and compared our outputs with others.

Models	Validation Data			Training Data		
	Accuracy	IoU	Loss	Accuracy	IoU	Loss
U-Net	**0.9574**	**0.5927**	**0.1282**	0.9623	**0.6781**	0.1000
U-Net encoded with InceptionV3	0.9256	0.4146	0.1887	**0.9706**	0.6917	**0.0740**
U-Net encoded with ResNet34	0.8545	0.1689	0.1369	0.9621	0.5792	0.0288
U-Net encoded with DenseNet121	0.8032	0.2938	0.1920	0.9590	0.5145	0.0330

Table 4.5: Our Metrics and Loss Function Results for Different Model

Model	**Accuracy**	**IoU**	**Loss**
1. Fully Convolutional Net (FCN)[22]	0.944	0.782	-
2. U-Net[17]	-	0.887	Not mentioned
3. U-Net[23]		0.526±0.170	0.170
U-Net (Ours)	**0.9574**	**0.5927**	**0.1282**

Table 4.6: Compare Metrics and Loss Function Results.

If we compare the paper between theirs and ours, we can see all of them use U-Net model structure only, where we use different encoders and parameters such as Inception, ResNet34 and Desnet121. Because of using these encoders and parameters, we get a better result in every metric except IoU in models 1&2 which is shown in Table 4.5 as our activation function was sigmoid that is not suitable for IoU. Moreover, they get the result in a high range of epochs such as in the second paper Mr. Rohan Pudipeddi and his team used 200 epochs. On the other hand, we get our result by taking 30 epochs and batch size 32. Lastly, we can say, we get better output with a less complicated method than others.

4.6.4 Conclusion

This chapter provides us with information about the datasets, prediction results as well as image results using color image segmentation, IoU curves and accuracy curves for different models, the comparison between the losses along with the curves of loss. To conclude, using InceptionV3, U-Net models we get the best possible prediction from

U-Net model as it gives us the most accurate value which is 0.9574. The above given comparisons and graphs among different models also indicate it.

Chapter 5

Conclusion and Future work

5.1 Thesis Conclusion

Here, Chapter wise, we will overview the whole Thesis project.

In chapter 01, we talk about Deep Learning in Image Segmentation and Overall Thesis Structure in detail.

In chapter 02, we include a decorative discussion about CNN and FCN and describe our Data Arguments.

In chapter 03, we show the Datasets, prediction results along with image results and compare between matrices where we find out the U-Net encoded with InceptionV3 model give sus the best result.

In chapter 04, we show the Datasets, prediction results along with image results and compare between matrices where we get the best result by using the U-Net model.

5.2 Future Work and Challenges

Over the last few years, image segmentation in pneumothorax and nucleus cells using deep learning algorithms has developed considerably. Higher accuracy and lower loss rate of our models make us hopeful about our motive of unerring segmentation. Therefore, we are planning to modify this project further in the future. First of all, by using different models, we want to increase the results of IoU and Accuracy matrices. Besides, we desire to use shape detection for detecting the disease more accurately. Additionally, for the betterment of disease detection, we want to include a 3-dimensional image dataset alongside using more biomedical information. We are hoping this will bring a comprehensive change for medical technology as well. Moreover, in the case of a nucleus cell, we want to specify the normal and cancerous cells of different types.

However, during this project we face some challenges that give us a hard time to come over. First of all, we face problems when processing the dataset. As the amount of data was huge, it took a long time to train all the images. Moreover, we need to find the dataset that contains valid mask images as we need to confirm that the mask was done by the expert. But it is hard to find out certain information over the internet.

Lastly, we aspire that our research will help others who want to work with similar models in future in order to choose the best model for image segmentation.

Reference

[1] Y. Mohamed Y. Abdallah and T. Alqahtani, “Research in Medical Imaging Using Image Processing Techniques,” *Med. Imaging - Princ. Appl. [Working Title]*, 2019, doi: 10.5772/intechopen.84360.

[2] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, “Image Segmentation Using Deep Learning: A Survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–22, 2021, doi: 10.1109/TPAMI.2021.3059968.

[3] A. Patil and M. Rane, “Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition,” *Smart Innov. Syst. Technol.*, vol. 195, pp. 21–30, 2021, doi: 10.1007/978-981-15-7078-0_3.

[4] K. O’Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” no. December, 2015, [Online]. Available: http://arxiv.org/abs/1511.08458.

[5] M. A. T. Hs, “Activation Functions and their Derivatives – A Quick & Complete Guide,” 2021, [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/.

[6] C. Norman, “AI in Pursuit of Happiness, Finding Only Sadness: Multi-Modal Facial Emotion Recognition Challenge,” no. September, 2019, [Online]. Available: http://arxiv.org/abs/1911.05187.

[7] S. Ruder, “An overview of gradient descent optimization algorithms,” pp. 1–14, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[8] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding Data Augmentation for Classification: When to Warp?,” *2016 Int. Conf. Digit. Image Comput. Tech. Appl. DICTA 2016*, 2016, doi: 10.1109/DICTA.2016.7797091.

[9] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0197-0.

[10] Y. Wen *et al.*, “Combining Ensembles and Data Augmentation can Harm your Calibration,” no. 2017, pp. 1–21, 2021.

[11] L. I. Augmentation and K. Imagedatagenerator, “Image Augmentation on the fly using Keras ImageDataGenerator !”

[12] M. S. ul Islam, “Using deep learning based methods to classify salt bodies in seismic images,” *J. Appl. Geophys.*, vol. 178, p. 104054, 2020, doi: 10.1016/j.jappgeo.2020.104054.

[13] U. Baid *et al.*, “A Novel Approach for Fully Automatic Intra-Tumor Segmentation With 3D U-Net Architecture for Gliomas,” *Front. Comput. Neurosci.*, vol. 14, 2020, doi: 10.3389/fncom.2020.00010.

[14] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, “UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation,” *IEEE Trans. Med. Imaging*, vol. 39, no. 6, pp. 1856–1867, 2020, doi: 10.1109/TMI.2019.2959609.

[15] R. Jain, M. Gupta, S. Taneja, and D. J. Hemanth, “Deep learning based detection and analysis of COVID-19 on chest X-ray images,” *Appl. Intell.*, vol. 51, no. 3, pp. 1690–1700, 2021, doi: 10.1007/s10489-020-01902-1.

[16] K. Gan *et al.*, “Artificial intelligence detection of distal radius fractures: a comparison between the convolutional neural network and professional assessments,” *Acta Orthop.*, vol. 90, no. 4, pp. 394–400, 2019, doi: 10.1080/17453674.2019.1600125.

[17] R. Pudipeddi, P. Phukan, and A. Gunda, “Nuclei Segmentation and Detection using Deep Convolutional Neural Networks,” *2020 11th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2020*, pp. 1–5, 2020, doi: 10.1109/ICCCNT49239.2020.9225478.

[18] Y. Ho and S. Wookey, “The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling,” *IEEE Access*, vol. 8, pp. 4806–4813, 2020, doi: 10.1109/ACCESS.2019.2962617.

[19] Q. Wang et al., "Automated segmentation and diagnosis of pneumothorax on chest X-rays with fully convolutional multi-scale ScSE-DenseNet: a retrospective study", *BMC Med Inform Decis Mak* 20, 317, 2020. https://doi.org/10.1186/s12911-020-01325-5.

[20] H. Wang, H. Gu, P. Qin, and J. Wang, “CheXLocNet: Automatic localization of pneumothorax in chest radiographs using deep convolutional neural networks,” *PLoS One*, vol. 15, no. 11 November, pp. 1–15, 2020, doi: 10.1371/journal.pone.0242013.

[21] J. C. Caicedo *et al.*, “Nucleus segmentation across imaging experiments: the 2018 Data Science Bowl,” *Nat. Methods*, vol. 16, no. 12, pp. 1247–1253, 2019, doi: 10.1038/s41592-019-0612-7.

[22] P. Naylor, M. Lae, F. Reyal, and T. Walter, “Nuclei segmentation in histopathology images using deep neural networks,” *Proc. - Int. Symp. Biomed. Imaging*, pp. 933–936, 2017, doi: 10.1109/ISBI.2017.7950669.

[23] Y. Cui, G. Zhang, Z. Liu, Z. Xiong, and J. Hu, “A deep learning algorithm for one-step contour aware nuclei segmentation of histopathology images,” *Med. Biol. Eng. Comput.*, vol. 57, no. 9, pp. 2027–2043, 2019, doi: 10.1007/s11517-019-02008-8.

[24] J. Jordan, "Evaluating image segmentation models.", Jeremy Jordan, 2018

[25] "Ronny Restrepo", *Ronny.rest*, 2021. [Online]. Available: http://ronny.rest/tutorials/module/localization_001/iou/. [Accessed: 30- Jun- 2021].

[26] “2018 Data Science Bowl | Kaggle”, Kaggle.com. 2018, [online] Available: https://www.kaggle.com/c/data-science-bowl-2018 [Accessed 6 June 2021].

[27] M. Davidson, "Molecular Expressions Cell Biology: The Cell Nucleus", *Micro.magnet.fsu.edu*, 2000. [Online]. Available: https://micro.magnet.fsu.edu/cells/nucleus/nucleus.html. [Accessed: 30- Jun- 2021].

[28] "Nucleus - Animal and Plant Cells", *Sites.google.com*. [Online]. Available: https://sites.google.com/site/animalandplantcells/Home/nucleus. [Accessed: 30- Jun- 2021].

[29] L. Johnson, "Loss Functions — Theanets 0.7.3 documentation", *Theanets.readthedocs.io*, 2015. [Online]. Available: https://theanets.readthedocs.io/en/stable/api/losses.html. [Accessed: 30- Jun- 2021].

[30] Khosla, S., "CNN | Introduction to Pooling Layer"GeeksforGeeks., 2020 [online] Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer [Accessed 6 June 2021].

[31] A. Rosebrock, "ImageNet: VGGNet, ResNet, Inception, and Xception with Keras - PyImageSearch", PyImageSearch, 2017. [Online]. Available: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras. [Accessed: 06- Jun- 2021].

[32] "How do computers see an image? | CommonLounge", Commonlounge.com. [Online]. Available: https://www.commonlounge.com/discussion/244616b76d3d40f88e8f12103a22743d. [Accessed: 06- Jun- 2021].

[33] D. Upadhyay, "How a computer looks at pictures: Image classification," Data Driven Investor, 17-Feb-2019. [Online]. Available: https://medium.com/datadriveninvestor/howa-computer-looks-at-pictures-image-classification-a4992a83f46b.

[34] "Resnet34 Fast.ai v2 Classification Model", Roboflow. [Online]. Available: https://models.roboflow.com/classification/resnet34. [Accessed: 06- Jun- 2021].

[35] S. R. Vanimereddy, "Brain Tumour Image Segmentation using U-Net Architecture.", DEV Community, 2020. [Online]. Available: https://dev.to/sairajesh/brain-tumour-image-segmentation-using-u-net-architecture-1ho2. [Accessed: 06- Jun- 2021].

[36] G. Adusumilli, "U-Net: Convolutional Networks for Biomedical Image Segmentation-Summarized", *Computer Vision and Machine Learning*, 2021.

[37] Arc, "Convolutional Neural Network", *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05. [Accessed: 30- Jun- 2021].

[38] R. Parmar, "Common Loss functions in machine learning", *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23. [Accessed: 30- Jun- 2021].

[39] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks", 2015. [Online]. Available: https://arxiv.org/abs/1511.08458. [Accessed: 30- Jun- 2021].

[40] Ö. Çiçek, A. Abdulkadir, S. Lienkamp, T. Brox and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation", *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, pp. 424-432, 2016. Available: 10.1007/978-3-319-46723-8_49 [Accessed 30 June 2021].

[41] “CNN Architectures over a timeline (1998-2019),” Aismartz.com, 24-Oct-2019.

[Online]. Available: https://www.aismartz.com/blog/cnn-architectures/.

[42] "Keras documentation: Keras Applications", *Keras.io*. [Online]. Available: https://keras.io/api/applications/. [Accessed: 30- Jun- 2021].

[43] S. Kumbhani, "Semantic Segmentation for Pneumothorax Detection & Segmentation", *Medium*, 2020. [Online]. Available: https://becominghuman.ai/semantic-segmentation-for-pneumothorax-detection-segmentation. [Accessed: 30- Jun- 2021].

[44] J. Le, "How to do Semantic Segmentation using Deep learning", *AI & Machine Learning Blog*, 2021. [Online]. Available: https://nanonets.com/blog/how-to-do-semantic-segmentation-using-deep-learning/. [Accessed: 30- Jun- 2021]

[45] B. Cui, X. Chen and Y. Lu, "Semantic Segmentation of Remote Sensing Images Using Transfer Learning and Deep Convolutional Neural Network With Dense Connection," in IEEE Access, vol. 8, pp. 116744-116755, 2020, doi: 10.1109/ACCESS.2020.3003914.