REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : SIOD11/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Système d'Information Optimisation et Décision (SIOD)**

Une approche basée deep-learning pour la segmentation d'images médicales

---

Par :

KISRANE MOHAMED MESSAOUD

Soutenu le …/07/2021 devant le jury composé de :

	grade	Président
Tigane samir	MCB	Rapporteur
	grade	Examinateur

Année universitaire 2020-2021

Abstract

The COVID-19 pandemic is an infectious disease that affects many people all over the world, and has caused thousands of deaths since the beginning of it's spread. Reverse transcription polymerase chain reaction (RT-PCR) it is one of the standard traditional diagnostic methods for detecting COVID-19 in infected persons. However, recent reports have stated that the sensitivity of RT-PCR may not be high enough to detect COVID-19, and it takes a long time to detect those infected with COVID-19.

One of the most successful automated methods that achieve state-of-the-art technology is deep learning, which is a convolutional neural network ($CNNs$).

Diagnosing COVID-19 based on deep learning ($CNNs$) applied to lung computed tomography (CT) scans captured from old COVID-19 patients could be a feasible solutions for detecting and labeling infected tissues on CT lung images of new patients.

In this work, we built one of the most popular CNNs-based architectures that is U-Net and applied it to CT lung scans, for automatic detection and segmentation of Covid-19 affected regions of the lung, then evaluated this architecture's performance using different metrics.

Acknowledgment

First, i would first like to thank my supervisor, **Dr Samir Tigane**, whose expertise was invaluable in guide me to complete this work. your insightful feedback pushed my thinking to the limit and my abilities to work in high level.

I would like to thank my colleagues as well as those working in the Department of Computer Science whom I met during my academic years. I would like to give a special thanks to **Professor Abdelhamid Djeffal**, for his valuable guidance throughout my studies in first semester.

Finally, tremendous thanks the friends (bachir, abdelali, ayoub, and my uncle youcef el-rayeh) and family for their continuous support and belief throughout my entire academic career.

Contents

List of Figures

Listings

List of Tables

General Introduction

Medical image segmentation has an essential role in computer-aided diagnosis systems in different applications. The vast investment and development of medical imaging modalities such as X-ray, computed tomography (CT), microscopy, and magnetic resonance imaging (MRI), attract researchers to implement new medical image-processing algorithms. Image segmentation is considered the most essential medical imaging process as it extracts the region of interest through a semi-automatic or automatic process. It divides an image into areas based on a specified description, such as segmenting body organs/tissues in the medical applications for border detection, tumor detection/segmentation, and mass detection [**1**].

Computed Tomography (CT) is used to detect tumors in pneumonia, lungs, tuberculosis, emphysema, or other pleura (the membrane covering the lungs) diseases. Lung CT image segmentation is a necessary initial step for lung image analysis. The main challenges of segmentation algorithms are exaggerated due to intensity in-homogeneity, presence of artifacts, and closeness in the gray level of different soft tissue [**2**].

COVID-19 is a disease causing thousands of deaths daily. Early diagnosis of this disease proved to be one of the most effective methods for infection tree pruning. The large number of COVID-19 patients is rendering health care systems in many countries. A trusted automated technique for identifying and quantifying the infected lung regions would be advantageous. Currently, there is an urgent need for efficient tools to assess the diagnosis of COVID-19 patients. U-Net is one of the most common deep learning techniques for diagnosis of COVID-19 infected people. U-Net is presenting a feasible solution for semantically segmenting infected tissue regions in CT lung images [**3**].

Next two chapters, we will introduce and implement this technique for segmenting infected tissues on CT lung images of such patients. Then we will try to evaluate the performance of this technique in segmentation of infected CT-lung tissues using different images evaluation metrics.

Chapter 1

Introduction to CT Segmentation Methods

1.1 Introduction

An image is a way of transferring information, and the image contains lots of useful information. Understanding the image and extracting information from the image to accomplish some work is an important area of application in digital image technology, and the first step in understanding the image is the image segmentation. In practice, it is often not interested in all parts of the image, but only for some certain areas which have the same characteristics. Image segmentation is one of the hotspots in image processing and computer vision. It is also an important basis for image recognition [**4**]. From the medical side, image segmentation has become a necessity because it divides an image into areas based on a specified description, such as segmenting body organs/tissues or detecting tumors, etc [**1**]. In this chapter, we introduce some of the important image segmentation techniques, and we try to focus on deep learning techniques for medical images segmentation.

1.2 Computed-Tomography (CT)

1.2.1 CT Concepts

Computed tomography (CT) is a medical imaging procedure that uses a combination of X-rays to build cross-sectional images or scans inside the body of the patient. Each cross-sectional image represents a ”slice” of the patient being imaged. These cross-sectional images are used for a variety of diagnostic and therapeutic purposes (e.g. Fig. 1.1) [**5**].

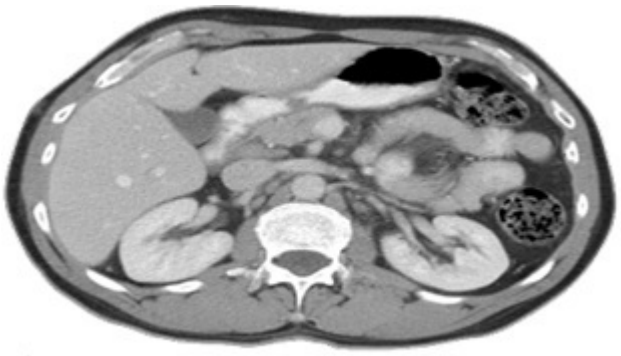


Figure 1.1: Cross-sectional Image of Abdomen.

1.2.2 CT Principle

CT is based on the fundamental principle that the density of the tissue passed by the x-ray beam can be measured from the calculation of the **attenuation coefficient**. Using this principle, CT allows the reconstruction of the density of the body, by two dimensional sections Perpendicular to the axis of the acquisition system [**5**].

The CT x-ray tube (typically with energy levels between 20 and 150 keV), emits N photons per unit of time. The emitted x-rays form a beam which passes through the layer of **biological material** of thickness Δx. A detector placed at the exit of the sample, measures $n + \Delta n$ photons, Δn smaller than 0. Attenuation values of the x-ray beam are recorded and data used to build a 3D representation of the scanned object/tissue [**5**].

An electron volt (eV) is defined as the energy required to accelerate a single electron at rest through an electron potential difference of 1 volt in a vacuum. Electronvolts are a more mathematically convenient method to describe the miniscule quantity of energy associated with photons in imaging physics. The energy required in diagnostic x-rays falls within the range of kiloelectronvolts (keV), whereas the energy required in radiotherapy is often in terms of megaelectronvolts (MeV) [**6**].

Biological materials are natural biocompatible materials that comprise a whole or a part of a living structure or biomedical device that performs, augments, or replaces a natural function. Biological materials are most often engineered for medical, biotechnology and pharmaceutical applications [**7**].

The attenuation coefficient is a measure of how easily a material can be penetrated by an incident energy beam (e.g. ultrasound or x-rays). It quantifies how much the beam is weakened by the material it is passing through [**5**].

1.3 Image Segmentation

Image segmentation is an essential but critical component in low–level vision, image analysis, pattern recognition, and now in robotic systems [**8**]. In this section, we present a concept with some techniques used for image segmentation.

1.3.1 Concepts of Image Segmentation

Intuitively, image segmentation is the process of dividing an image into different regions such that each region is homogeneous **[9]**. In another meaning, the union of any two adjacent regions is empty. An additional requirement would be that these regions had a correspondence to real homogeneous regions belonging to objects in the scene **[9]**. The classical formal definition of image segmentation is as follows:
If $P(R)$ is a homogeneity predicate defined on groups of connected pixels, then the segmentation is a partition of the image M into connected components or regions $\{R_1, \cdots, R_n\}$ such that **[9]**:

1. **Complete**: $M = \bigcup_{i=1}^{n} R_i$.
2. **Disjoint subsets**: $R_i \cap R_j = \emptyset, \forall i \neq j$.
3. **Uniform regions**: $P(R_i) =$**True**, for all regions R_i.
4. **Maximal regions**: $P(R_i \cup R_j) =$**False**, when $i \neq j$ and both sets R_i and R_j are neighbors.

1.3.2 Concepts of Image Segmentation

The image segmentation approaches can be categorized into two types based on properties of image:

1. **Discontinuity detection based approach**

   This is the approach in which an image is segmented into regions based on discontinuity. The edge detection based segmentation falls in this category in which edges formed due to intensity discontinuity are detected and linked to form boundaries of regions **[10]**.

2. **Similarity detection based approach**

   This is the approach in which an image is segmented into regions based on similarity. The techniques that fall under this approach are: thresholding techniques, region growing techniques and region splitting and merging. All of these techniques divide

the image into regions having a similar set of pixels. The clustering techniques also use this methodology. These techniques divide the image into a set of clusters having similar features based on some predefined criteria **[10]**.

There are several existing techniques used for image segmentation. These techniques have their own importance. These all techniques can be approached from two basic approaches of segmentation i.e. region based or edge based approaches **[10]**. Fig. 1.2 represents some of this techniques.

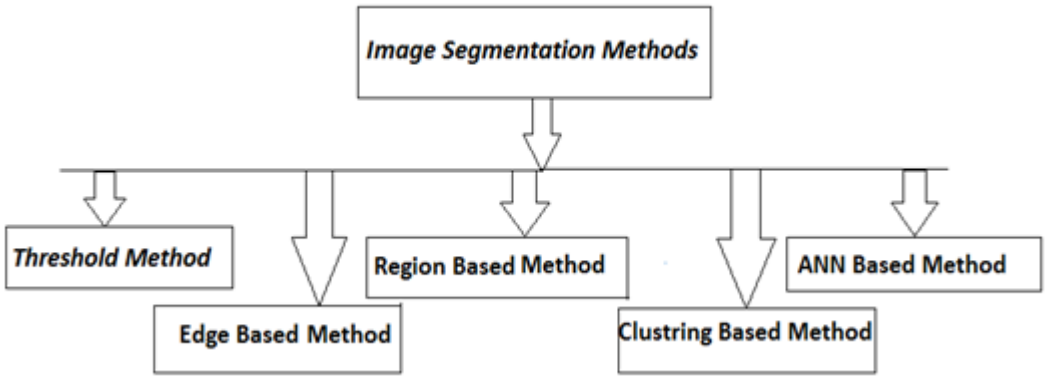


Figure 1.2: Some of the Image segmentation techniques.

1.3.3 Image Segmentation Algorithms

1. **Thresholding**

   Thresholding is the simplest segmentation method. The pixels are partitioned depending on their intensity value. Global thresholding, using an appropriate threshold T **[11]**:

$$g(x,y) = \begin{cases} 1, if\ f(x,y) > T \\ 0, if\ f(x,y) \leqslant T \end{cases}$$

   Where f(x,y) is the original image pixels and g(x,y) is the thresholded image. Since g contains only two values (1 for foreground pixels and 0 for background pixels), it is called a binary image. It can serve as a mask because each location (x,y) the original image has a value of 1 in the mask if this is a feature pixel (e.g. Fig. 1.3).

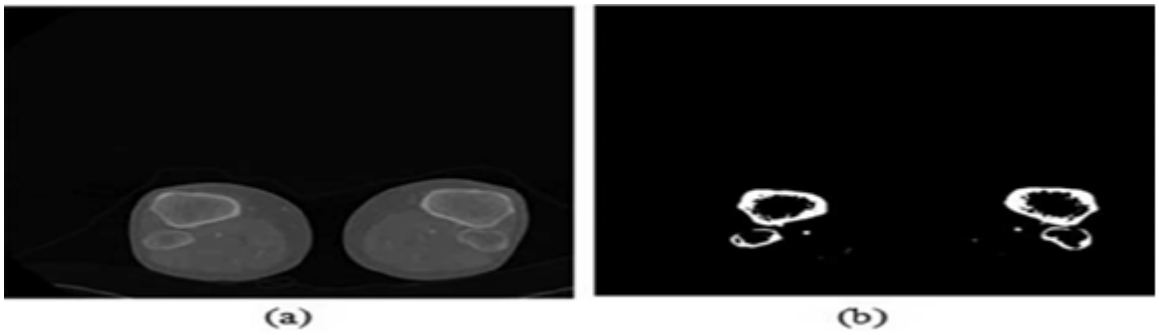


Figure 1.3: Applying thresholding on CT-image, (a) original image, (b) extracted image using thresholding segmentation [**12**].

2. **Edge Based Segmentation**

   The edge based segmentation methods are based on the rapid change of intensity value in an image because a single intensity value does not provide good information about edges (e.g. Fig 1.5). Two steps are needed [**13**]:

   (a) **Edge detection (to identify "edgels" - edge pixels)**

   Operators detecting discontinuities (to detect the edges one of the basic edge detection techniques like sobel operator, canny operator and Robert's operator, etc. Can be used) [**13**].

   (b) **Edge linking – linking adjacent "edgels" into edges**

   Edges in a predefined neighborhood are linked if both magnitude and direction criteria are satisfied [**13**].

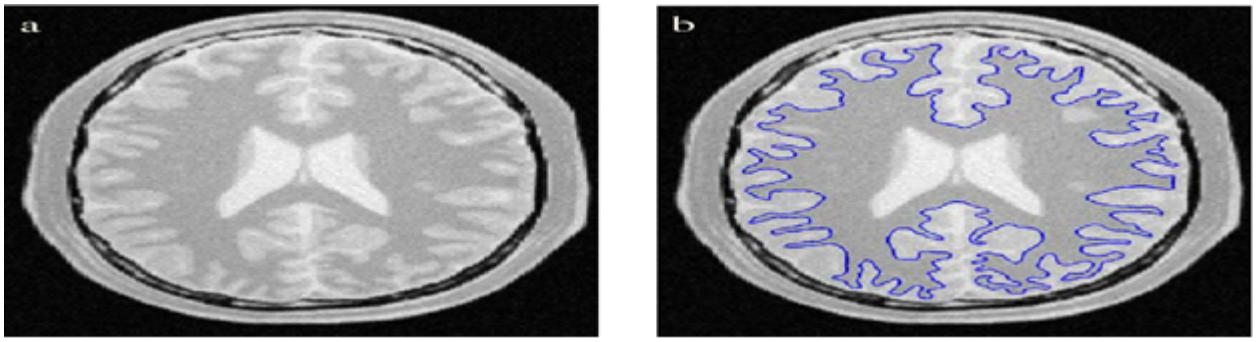


Figure 1.4: An Example of Edge Based Segmentation, a: original image, Right: Image with tissue boundary [**14**].

The basic two edge based segmentation methods are: **Gray histograms** and **Gradient based methods**. Result of these methods is basically a binary image. These

methods are the structural techniques based on discontinuity detection [**10**].

3. **Region Based Segmentation**

   The region-based segmentation method looks for similarities between adjacent pixels. That is, pixels that possess similar attributes are grouped into unique regions. As with all segmentation techniques, using gray-level intensity is the most common means of assigning similarity, but other possibilities exist, such as variance, color, and multi-spectral features.

   Region growing algorithm is a method of segmentation based on the approach region, the principle of this algorithm is as follows [**15**]:

   (a) An initial set of small parts of pixels are iteratively merged according to parallel restrictions [**15**].

   (b) Start by selecting an arbitrary seed pixel and compare it with neighboring pixels [**15**].

   (c) Region is grown from the seed pixels by adding the neighboring pixels that are parallel, growing the size of the region [**15**].

   (d) When the progress of one region stops, we simply select another seed pixel which does not belong to any region and start again [**15**].

   (e) This process is continuous until all pixels fit to some region.

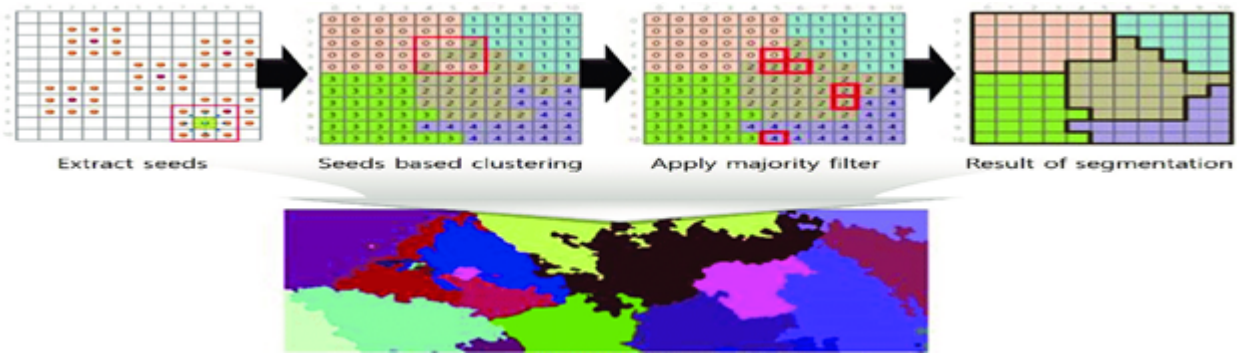


Figure 1.5: Result of performing the seeded region growing algorithm and correcting the abnormal pixel [**16**].

4. **Clustering Based Segmentation**

   Clustering method is one of the most popular algorithms in the image segmentation domain. It is an approach of classifying patterns or data into categories, which is on the basis of the samples in the same group having the higher similarity than the ones in different groups (particularly, k-means clustering algorithm, and fuzzy c-means clustering algorithm are the two popular examples of the state-of-the-art clustering methods) [**17**]. The clustering procedure on an image (X) of size (m × n), defined over dimensions, generates K clusters $\{C_1, C_2, , C_k\}$ subject to the following conditions [**18**]:

   (a) $C_i \neq \emptyset$, **for i =1,2,...,k.**

   (b) $C_i \cap C_j \neq \emptyset$, **for i and j = 1,2,...,k and** $i \neq j$**.**

   (c) $\cup_{i=1}^{k} C_i = X$.

5. **ANN Based Segmentation**

   Since 1990, artificial neural networks ($ANNs$) have come to be used as a different approach for image segmentation. Their properties, such as graceful degradation in the presence of noise, their ability to be used in real-time applications and the ease of implementing them with VLSI processors, led to a booming of ANN-based methods for segmentation. The ANN-based image segmentation techniques reported in the literature can mainly be divided into two categories: supervised and unsupervised methods. Supervised methods require expert human input for segmentation. Usually this means that human experts are carefully selecting the training data that is then used to segment the images. Unsupervised methods or clustering processes are semi or fully automatic. User intervention might be necessary at some point in the process to improve performance of the methods, but the results should be more or less human independent [**19**].

1.4 Deep Learning Approach

1.4.1 Neural Networks

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, sound, text or time series, must be translated **[20]** (see Fig 1.6).

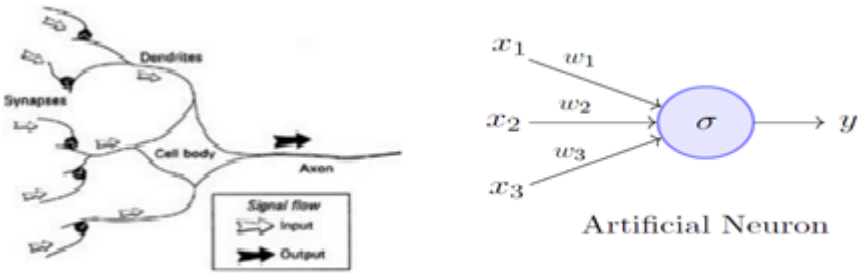


Figure 1.6: Left: Essential component of a human neuron, Right: An Artificial Neuron.

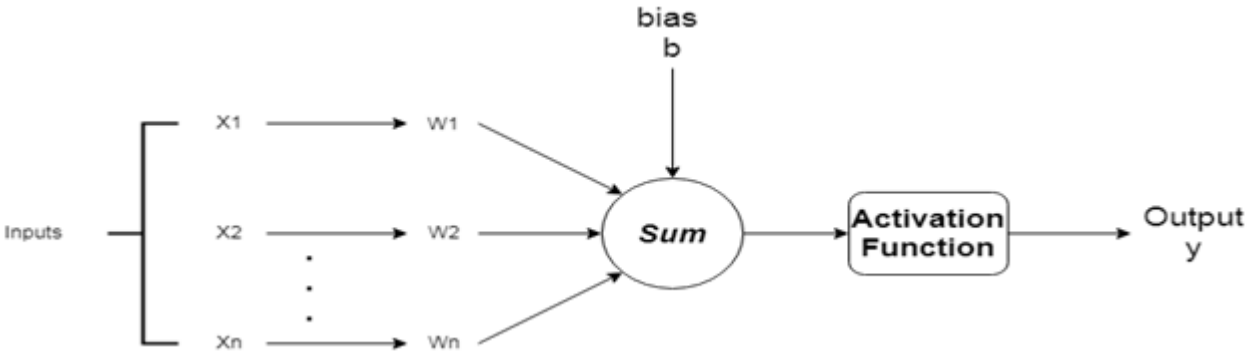


Figure 1.7: example of a neuron, input $(X_1.., X_n)$, their corresponding weights $(W_1.., W_n)$, and a bias (b), activation function applied to the weighted sum of the input and the Output **Y**.

The block diagram of Fig 1.7 shows the model of a neuron, which forms the basis for designing a large family of neural networks. Here, we identify basic elements of the neural model:

1. $X_1 ... X_n$ Are the inputs to the neuron.
2. $W_1 ... W_n$ Are the weights.
3. Product of weight and input gives the strength of the signal.
4. Sum = $\sum_{i=1}^{i=n} X_i W_i$. This Sum is then activated using the activation function F.
5. The output of this function is a single number denoted by y.

Name	Equation	Domain	Range
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$(-\infty, \infty+)$	$(0, 1)$
Tanh	$f(x) = \frac{2}{1+e^{-x}} - 1$	$(-\infty, \infty+)$	$(-1, 1)$
LRelu	$f(x, y) = \begin{cases} 0, if\ x \leqslant 0 \\ x, if\ x > 0 \end{cases}$	$(-\infty, \infty+)$	$[0, \infty+)$

Table 1.1: Some of the Activation functions used by ANN.

1.4.2 What Is Deep Learning?

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning from the unstructured or unlabeled data. Also known as deep neural networks **[21]**.

How Deep Learning Works

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. The term "deep" usually refers to the number of hidden layers in the neural network **[22]**.

Traditional neural networks only contain 2-3 hidden layers, while deep networks can have more than 3 hidden layers. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction **[22]** (see. Fig 1.8).

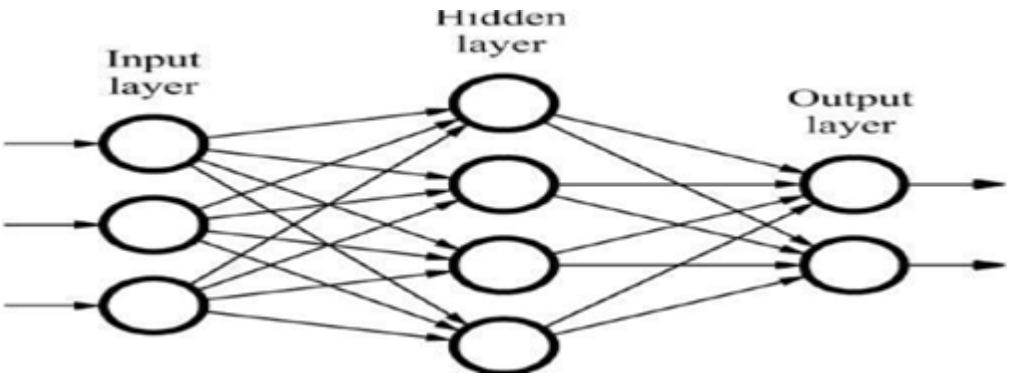


Figure 1.8: A Neural network organized in layers [**22**].

1.5 Training Procedure

In deep learning, predictive models use various underlying algorithms to infer mathematical relationships from training data. There are mainly three types of learning methods, namely:

1.5.1 Supervised Learning

In supervised learning, the model is fed a training dataset containing both observations (inputs) as well as their corresponding outcomes (outputs). The model then infers the mathematical mapping from inputs to outputs which it can use to classify future input test data [**23**].

1.5.2 Unsupervised Learning

In unsupervised learning, the model is fed with unclassified training data (only inputs). Then, the model categorizes test data points into different classes by finding commonalities between them [**23**].

The goal in such unsupervised learning problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine how the data is distributed in the space, known as density estimation [**23**].

1.5.3 Semi-Supervised Learning

Semi-supervised learning is halfway between supervised and unsupervised learning. A semi-supervised data set primarily contains unclassified training data points along with small amounts of classified data. Semi-supervised models feature two important advantages. One, they are more accurate than unsupervised models with the additional few classified data points. Two, they are significantly less laborious and time-intensive compared to supervised learning [**23**].

Semi-supervised learning may refer to either transductive learning or inductive learning. The goal of transductive learning is to infer the correct labels for the given unlabeled data. Self-training is commonly used for this method. The goal of inductive learning is used to deduce the mapping between input and output [**23**].

1.6 Deep Learning Categories

In this section below, we will present multiple deep learning architectures and explain their underlying algorithms. A general overview will be presented for each of the three main categories of neural networks, Convolutional Neural Networks, Pre-trained Unsupervised Networks, and Recurrent Neural Networks [**23**].

1.6.1 Convolutional Neural Networks (CNNs)

A convolutional neural network (CNNs) is a type of deep learning model most commonly applied to analyze and process data that has a grid pattern, such as images. CNNs are inspired by biological processes and are designed to emulate the neural connectivity found in the brain's visual cortex [**23**]. CNNs have a large range of applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing (NLP).

1. **CNN Structure**

   CNN is a mathematical construct that is typically composed of three types of layers (building blocks): convolution, pooling, and fully connected layers [**25**].

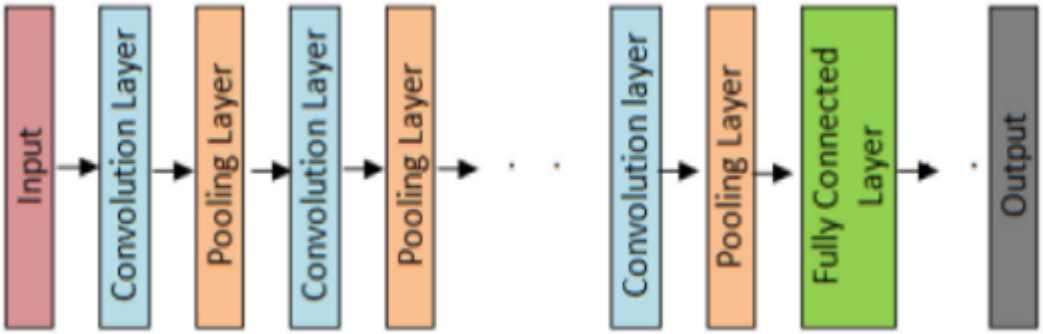


Figure 1.9: A Conceptual model of CNN [**24**].

The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification [**25**] (i.e. Fig 1.10).

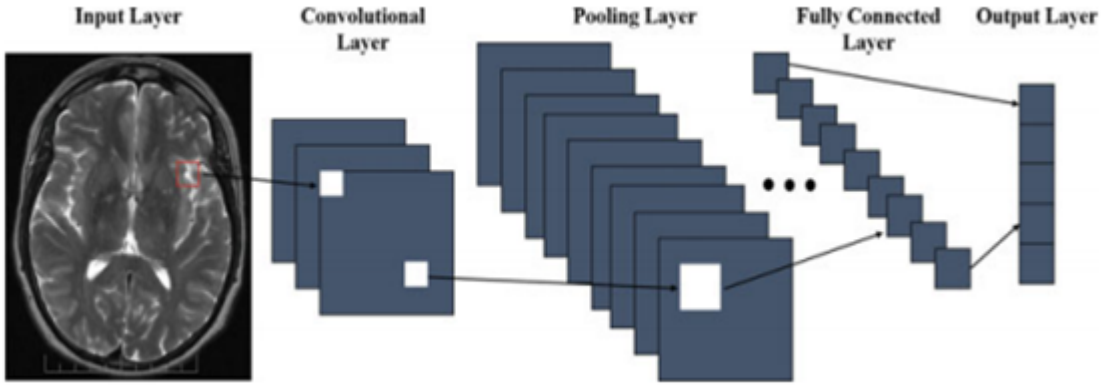


Figure 1.10: A Condensed Representation of Convolutional Neural Networks (CNN) [**23**].

2. **CNN Layers**

As we mentioned earlier, a CNN is composed of multiple building blocks (layers). In this subsection, we describe some of these building blocks with their role in the CNN architecture.

(a) **Convolutional Layer**

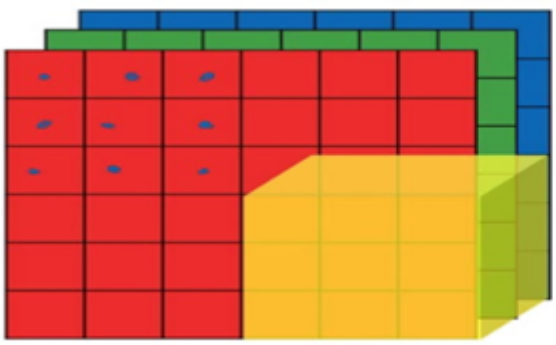
A convolution layer is a fundamental component of the CNN architecture that performs feature extraction. It contains a set of convolutional kernels (filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map [**25**].

(b) **What is a kernel?**

A kernel can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel. During the starting of the training process of a CNN model, all the weights of a kernel are assigned with random numbers. Then, with each training epoch, the weights are tuned and the kernel learns to extract meaningful features **[26]**.

(c) **Convolve Operation**

Let us first understand the input format to CNN. Unlike other classical neural networks where the input is a vector format, in CNN the input is a multi-channeled image (for RGB image as in left Fig 1.11, it is triple channeled. For Gray-Scale image as in right Fig 1.11, it is single channeled).



1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	100	76
34	66	134	223	201	3
255	123	89	55	32	2

Figure 1.11: left: An Example of a RGB image (6*6*3), right: A 6× 6 Gray-Scale image.

Now, in convolution operation, we take the filter (kernel) and slide it over all the complete image horizontally as well as vertically and along the way we take the dot product between kernel and input image by multiplying the corresponding values of them and sum up all values to generate one scalar value in the output feature map. This process continues until the kernel can no longer slide further **[26]** (see. Fig 1.12).

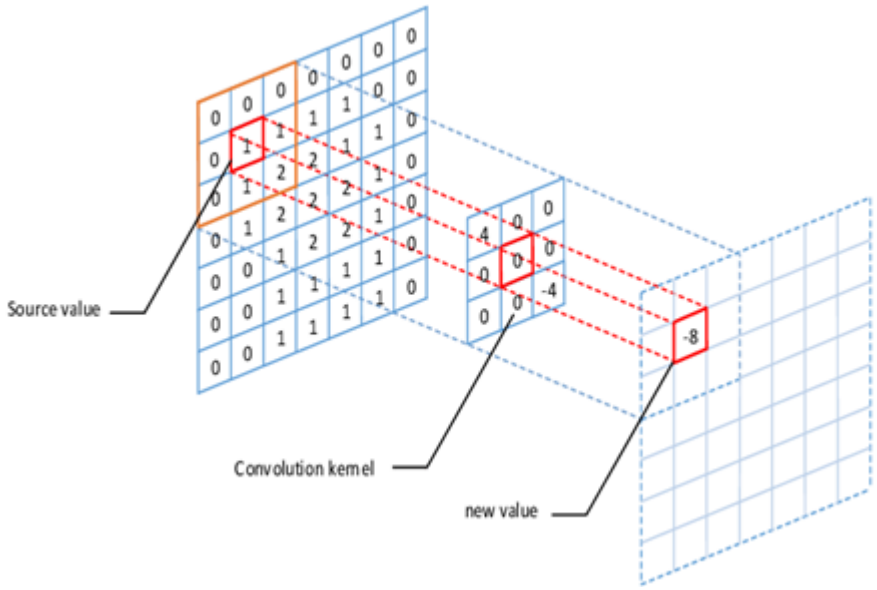


Figure 1.12: A Convolution Operation [**27**].

Formula to find the output feature map size after convolution operation as below [**26**]:

$$h' = [\frac{h-f+p}{s} + 1]$$
$$w' = [\frac{w-f+p}{s} + 1]$$

Note That:

i. h', w': Height and width of the output feature map respectively [**26**].

ii. h, w: Height and width of the input image respectively [**26**].

iii. f: The filter size [**26**].

iv. p: The padding of convolution operation [**26**].

v. s: The stride of convolution operation (the taken step size along the horizontal or vertical position) [**26**].

(d) **Pooling Layer**

A pooling layer provides a typical down-sampling operation which reduces the plane dimensionality of the feature maps and shrinks them to lower sized feature maps. While shrinking the feature maps it always preserves the most dominant features (or information) in each pool step. Filter size, stride, and padding are hyper-parameters in pooling operations, similar to convolution operations [**25**]. There are different types of pooling techniques used in different

pooling layers such as max pooling, min pooling, average pooling [**26**]. Max Pooling is the most popular and used pooling technique [**26**].

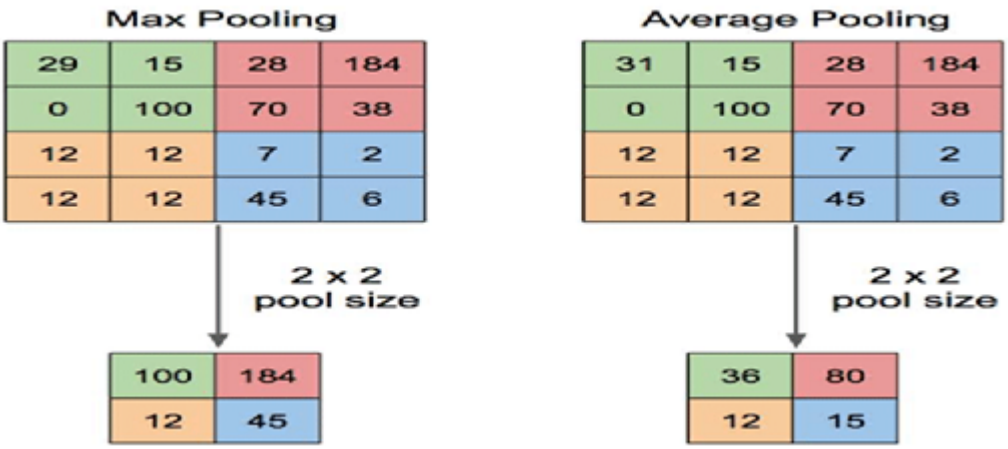


Figure 1.13: Illustration of Max Pooling and Average Pooling [**28**].

Formula to find the output feature map size after convolution operation as below [**26**]:

$$h' = [\frac{h-f}{s}]$$
$$w' = [\frac{w-f}{s}]$$

Note That:

i. h', w': Height and width of the output feature map respectively [**26**].

ii. h, w: Height and width of the input feature map respectively [**26**].

iii. f: Pooling region size [**26**].

iv. s: The stride of pooling operation [**26**].

(e) **Activation Functions**

The main task of any activation function in any neural network based model is to map the input to the output, where the input value is obtained by calculating the weighted sum of the neuron's input and further adding bias with it (if there is a bias). In other words, the activation function decides whether a neuron will fire or not for a given input by producing the corresponding output. In CNN architecture, after each learnable layer (convolutional and fully connected layers) non-linear activation layers are used. The non-linearity behavior of those layers enables the CNN model to learn more complex things

and manage to map the inputs to outputs nonlinearly. The important feature of an activation function is that it should be differentiable in order to enable error back-propagation to train the model. The most commonly used activation functions (see Table 1) in deep neural networks (including CNN) are **Sigmoid, Tanh, ReLU, LeakyReLU,** etc [**26**].

(f) **Fully Connected Layer**

Usually the last part (or layers) of every CNN architecture consists of fully-connected layers, where each neuron inside a layer is connected with each neuron from its previous layer. The last layer of Fully-Connected layers is used as the output layer (classifier) of the CNN architecture [**26**].

The Fully-Connected Layers are a type of feed-forward artificial neural network, and it follows the principle of traditional multi-layer perceptron neural network (MLP). The FC layers take input from the final convolutional or pooling layer, which is in the form of a set of metrics (feature maps) and those metrics are flattened to create a vector and this vector is then fed into the FC layer to generate the final output of CNN as shown in Fig 1.14 [**26**].

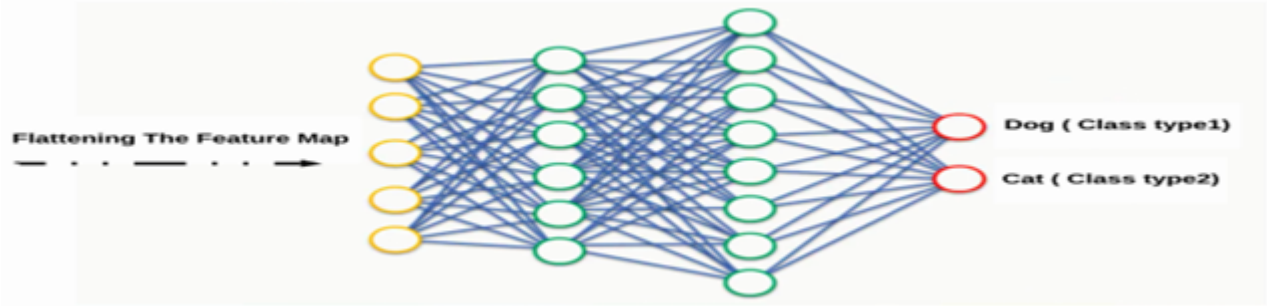


Figure 1.14: Architecture of Fully Connected Layers [**23**].

1.6.2 Pre-trained Unsupervised Networks

Data generation and feature extraction are very important applications in deep learning as we usually have limited training data. There are different techniques used to augment the initial dataset to provide a larger dataset. Using some of the most advanced deep learning architectures like generative adversarial networks (GANs) and auto-encoders. We could

generate synthetic data based off of the original dataset to improve model learning, both architectures belong to a family called Pre-trained Unsupervised Network (PUN) [**23**].

PUN is a deep learning model that uses unsupervised learning to train each of the hidden layers in a neural network to achieve a more accurate fitting of the dataset. An unsupervised learning algorithm is used to train each layer one at a time, independently, while using the previously trained layer as the input. After the pre-training is done on each layer, a fine-tuning step is performed on the whole network using supervised learning. Types of PUNs include Auto-encoders (i.e. Fig 1.15), and Generative Adversarial Networks (GAN), etc [**23**].

1. **Auto-Encoders**

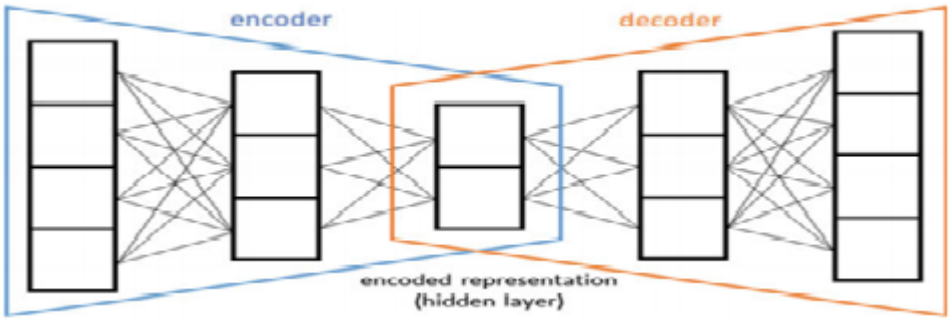


Figure 1.15: A representation of an auto-encoder [**23**].

Auto-encoders use unsupervised learning to learn a representation for dimensionality reduction where the input is the same as the output (i.e. Fig 1.15). The three parts of an auto-encoder include the input, output, and the hidden layer. The data is compressed into a smaller representation in the hidden layer then uncompressed to form an output that is similar to the input. This happens through two main steps of encoding and decoding of the auto-encoder algorithm. For example, the following is used to represent the mapping function between the input layer and hidden layer [**23**]:

$$Y = f_\theta(\widehat{X}) = s(W\widehat{X} + b).$$

Where the input $\widehat{x}$ is mapped to the hidden layer y, θ is the coding parameter, and W is the weighted matrix. Therefore the decoding function would be the following [**23**]:

$$Z = g_\theta(Y) = x(W'Y + b').$$

Z Would be the reconstruction of input X [**23**].

2. **Generative Adversarial Networks (GAN)**

   GANs are an example of a network that uses unsupervised learning to train two models in parallel. A key aspect of GANs (generative models in general) is how they use a parameter count that is significantly smaller than normal with respect to the amount of data on which the network is trained. A GAN network is made up of a discriminator D, and a generator G, which operate in parallel.

   The generator's goal is to be able to create a fake output that resembles a real output, with the generator training through its interactions with the discriminator and not from any actual content. The objective of the generator is to produce an output that is so close to real that. This output confuses the discriminator in being able to differentiate the fake data from the real data [**23**](see Fig 1.16).

   There are three steps in GANs which are [**23**]:

   (a) First, the generator takes in random numbers and returns an image. This generated image is fed into the discriminator alongside a stream of images taken from the actual dataset.

   (b) Second, the discriminator takes in both real and fake images and returns probabilities; an output close to 0 being the data from the generator is fake and an output close to 1 being that the data is real.

   (c) Third, the discriminator network provides feedback to the generator in order to train it and improve its output.

   GAN has the potential to be used in many applications and has been used in improving the resolution of images. Another useful application using GAN. Gan has the ability to create photos based on a detailed caption description [**23**].

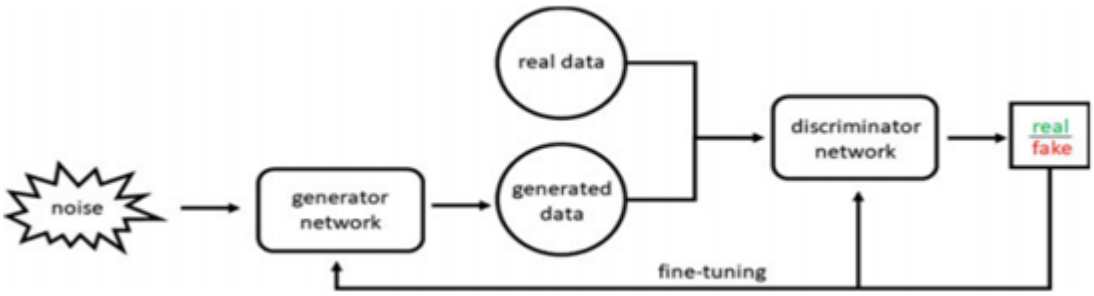


Figure 1.16: A representation of a Generative Adversarial Network (GAN) [**23**].

1.6.3 Recurrent Neural Networks

A strict feed-forward (e.g. CNNs) architecture does not maintain a short-term memory. Any memory effects are due to the way past inputs are represented to the network. The fundamental feature of a Recurrent Neural Network (RNN) is that the network contains at least one feed-back connection, so the activations can flow round in a loop. That enables the networks to do temporal processing and learn sequences. For example, perform sequence recognition/reproduction or temporal association /prediction [**29**].

RNNs are heralded for their ability to process and obtain insights from sequential data. Therefore, video analysis, image captioning, natural language processing (NLP), and music analysis all depend on the capabilities of recurrent neural networks. Unlike standard neural networks that assume independence among data points, RNNs actively capture sequential and time dependencies between data (i.e. Fig 1.17) [**23**].

RNNs generally augment the conventional multilayer network architecture with the addition of cycles that connect adjacent nodes or time steps. These cycles constitute the internal memory of the network which is used to evaluate the properties of the current data point at hand with respect to data points from the immediate past [**23**]. It is also important to note that most conventional feed-forward neural networks are limited to one to one mapping for input and output. RNNs, however, can perform one to many, many to many (e.g. translating speech), and many to one (e.g. identifying voice) mappings [**23**].

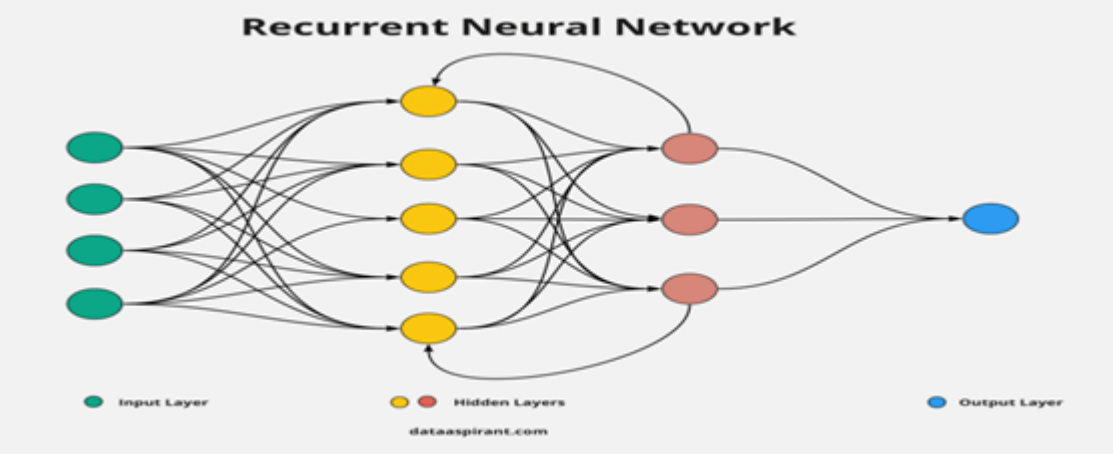


Figure 1.17: A condensed representation of Recurrent Neural Network RNNs [**30**].

1.7 Medical Image Segmentation Based on Deep Learning

CNN has been used in medical image segmentation in recent years. It has achieved great success in the field and auxiliary diagnosis [**31**]. In this Section, we will explain one of the most popular deep learning architectures for medical image segmentation: **U-Net**.

1.7.1 U-Net Architecture

The U-Net (see fig 1.18) was developed by Olaf Ronneberger. For Bio Medical Image Segmentation. The architecture contains two paths. First path is the contraction path (encoder) which is used to capture the context in the image. This path is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (decoder) which is used to enable precise localization using transposed convolutions. Thus, U-Net is an end-to-end fully convolutional network (FCN), it contains Convolutional layers and does not contain any dense layer, which makes it accept images of any size as input [**32**].

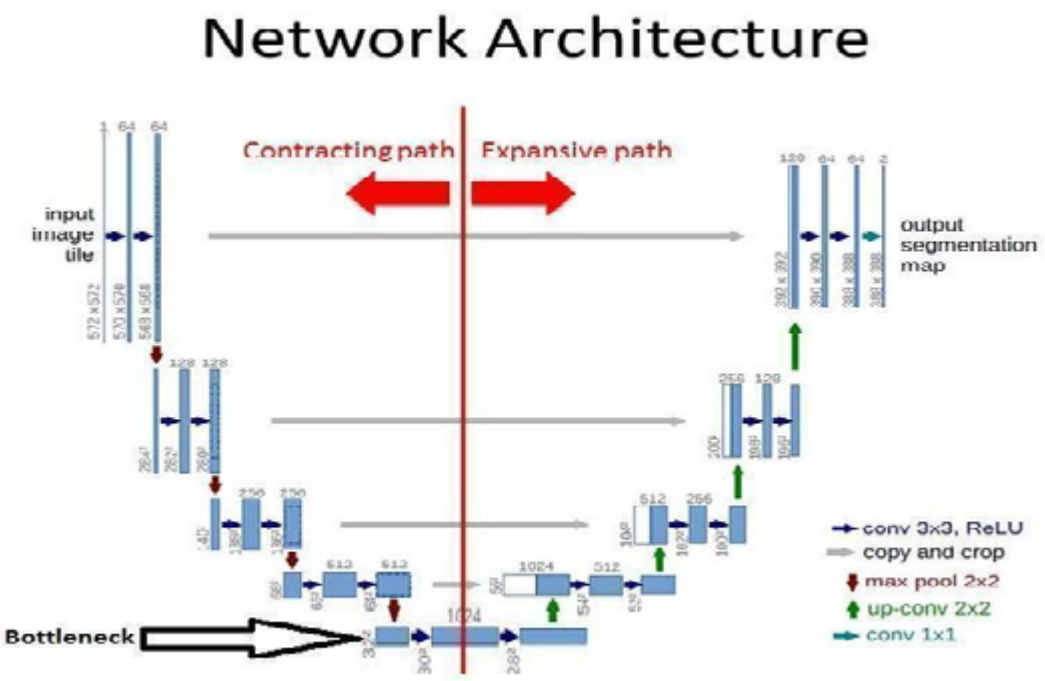


Figure 1.18: U-Net Network architecture.

1.7.2 U-Net layers

1. **Input layer**

   As we mentioned before, Unit networks can accept images of any size.

2. **Down-sampling path (contraction)**

   The contracting path is composed of **4 blocks**. Each block is composed of the repeated application of **two 3x3 convolutions layers (convolution operation with kernel size of 3x3)**. Each convolution operation is followed by a rectified linear unit (**ReLU** activation function) and a **2x2 max pooling** operation with **stride 2** for down-sampling [**33**].

3. **Bottleneck**

   This part of the network is between the contracting and expanding paths. The bottleneck is built from simply **two 3x3 convolutional layers** [**33**].

4. **Up-sampling path (expanding)**

   The expanding path is also composed of **4 blocks**. Each of these blocks is composed of a **2x2 Up-convolution layer with stride 2**.Then a repeated application of **two 3x3 convolutions layers (convolution operation with kernel size of 3x3)**. Each convolution operation is followed by a rectified linear unit (**ReLU** activation function). Result of this operation **concatenate** with the corresponding cropped feature map from the **contracting path** [**33**].

   Up-convolution

   Up-convolution is a convolution operation (just like normal convolution operation) while training the U-Net model. Using up convolution (Conv-Transpose) will also up-sample its input of feature maps (i.e1.19). Each up-convolution operation concatenates with the corresponding cropped feature map from the contracting path [**33**].

   Skip Connections

   U-Net is symmetric, the skip connection between the down-sampling path and the up-sampling path applies a concatenation operator. These skip connections intend to provide local information from down-sampling to the global information while up-sampling. Because of its symmetry, the network has a large number of feature maps in the up-sampling path, which allows it to transfer information [**33**].

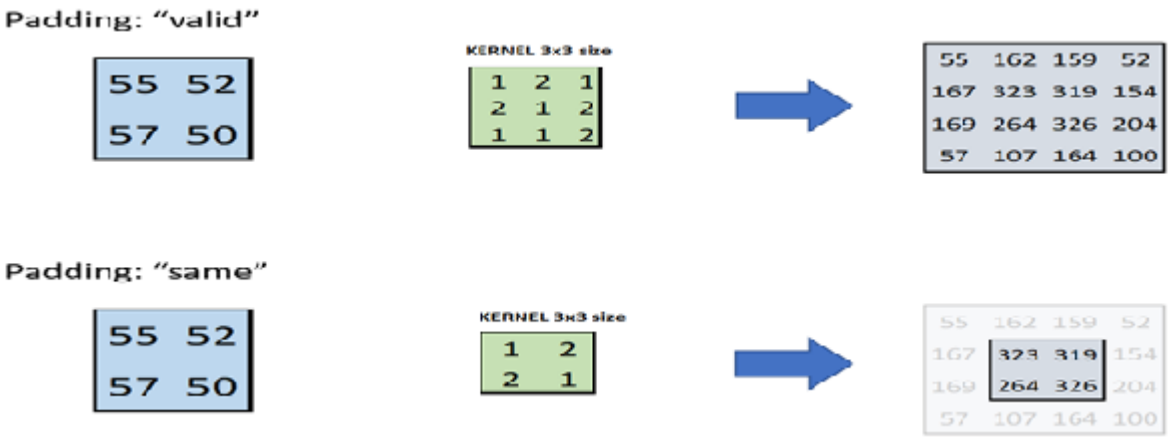


Figure 1.19: Up-convolution operation, above: padding='valid', at the bottom: padding='same' [**34**].

5. **Output Layer**

   This layer is composed of one block. This block is composed of one convolution layer (one convolution operation with kernel size of 1x1). This convolution operation is followed by a non-linear unit (sigmoid activation function).Final result of this layer is to obtain a segmentation map (output).

   The sigmoid activation function is defined by the following equation:

$$f(x) = \frac{1}{1+e^{-x}}$$

   The sigmoid function takes a **feature map generated from the last convolution operation** input and produces values in range of 0 to 1. The larger the input (more positive feature map values), the closer the output value will be to 1, whereas the smaller the input (more negative feature map values), the closer the output will be to 0. This whole operation produces a **segmentation map (output segmentation map)**.

   Note that the number of feature maps doubles at each pooling, starting with 64 feature maps for the first block, 128 for the second, and so on. The purpose of this contracting path is to capture the context of the input image in order to be able to do segmentation. This coarse contextual information will then be transferred to the up-sampling path by means of skip connections **[33]** (see. Fig. 1.18: Copy and Crop).

1.7.3 U-Net Advantages and Disadvantages

1. **U-Net Advantages**

   The U-Net combines the location information from the down-sampling path with the contextual information in the up-sampling path to finally obtain a general information combining localization and context, which is necessary to predict a good segmentation map **[35]**.

   No dense layer, so images of different sizes can be used as input (since the only parameters to learn on convolution layers are the kernel, and the size of the kernel is independent from input image's size) **[35]**.

2. **U-Net Disadvantages**

   Because many layers take a significant amount of time to train. Relatively high GPU memory footprint for larger images [**35**]. That means:

   (a) 640x959 image ⇒ you can fit 4-8 images in one batch with 6GB GPU.

   (b) 640x959 image ⇒ you can fit 8-16 images in one batch with 12GB GPU.

   (c) 1280*1918 ⇒ you can fit 1-2 images in one batch with 12GB GPU.

Chapter 2

CNN for Covid-19 Segmentation and Implementation

2.1 Introduction

Diagnosis and staging of COVID-19 are crucial for optimal management of the disease. To this end, novel image analysis methods need to be developed to assist radiologists with the detection and quantification of the COVID-19-related lung infections [**36**]. In this chapter, we implement various U-Net based segmentation network evaluation metrics $(accuracy, dice\_coef, IoU)$, and evaluate their performance in segmenting COVID-19 lesions on CT scans.

2.2 General Architecture

Our system uses deep learning for detecting and segmenting lung infected areas by Covid-19. The system takes data from a set of images of raw coronavirus infected lung as input, then performs some operations on these images (image resizing, image normalization), these operations called data preprocess. After data preprocess, the system divides these images into a training set and a test set. The system feeds the training model with the training set to produce a trained model capable of identifying and segmenting areas of infection with the Covid-19 at the lung level. The test set used to provide an unbiased evaluation of a final model fit on the training dataset.

The fig 2.1 represents the general architecture of U-Net for covid-19 CT-scans segmentation.

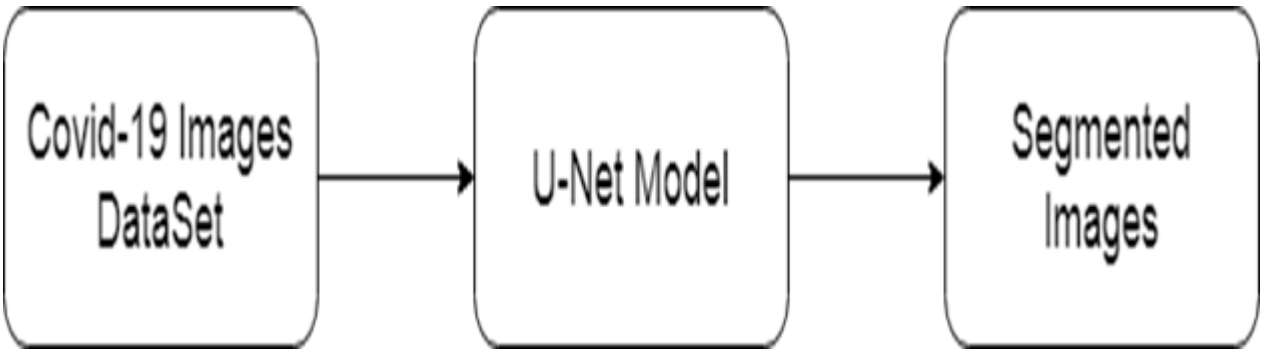


Figure 2.1: U-Net for covid-19 segmentation.

2.3 Detailed Architecture

We have used the COVID-19 CT segmentation dataset, which contains two versions (images of the two versions are in **NifTi format (*.nii)**.

The first version of this dataset contains 10 labeled COVID-19 CT scans, this dataset has three types of ground truth masks, which are called Ground Glass, Consolidation and Pleural Effusion. The original CT- scans and all ground truth masks have a size of 512 x 512 pi. These CT-scans crawled from coronacases.org [**37**].

The second version of the dataset was expanded to 829 images in which 373 of those are labeled as COVID-19 and the rest as normal. All ground truth masks as COVID-19 have Ground Glass masks but the majority of them are missing the latter two. The size of images and masks in the second version of this dataset is 630 x 630 pi. These CT scans crawled from radiopaedia.org [**37**].

2.3.1 Data Pre-process

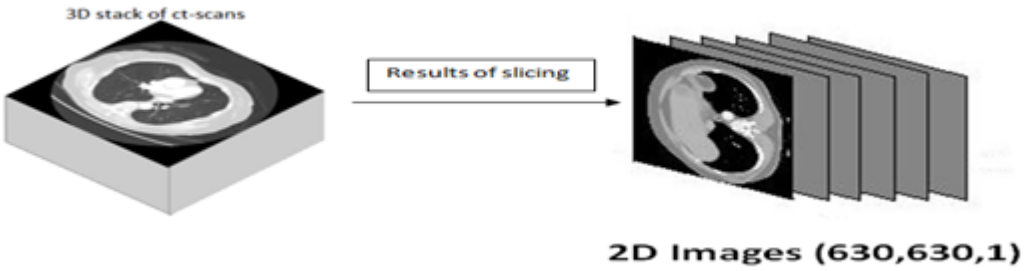


Figure 2.2: Representative example of images batch extraction method from CT-scans.

We have already sliced and used 661 images from the second version of covid-19 CT scans with their Ground Glass masks to build our U-Net model (e.g. Fig. 2.2). There are a few operations before building our model such as:

1. **Data Reshaping**

   Our images input shapes are **(630, 630, 1)**, the first two 630 represent the height and the length of each image, and the one represents the depth. Depth indicates the number of channel colors of each image. In our case we have one ct-scans type

in **grey scale (depth=1)** so in total we get one. To reduce model training time, we suggest reshaping our input images into **(256, 256, 1)**.

2. **Data Reshaping**

   After data reshaping, we have applied standard data normalization techniques. This technique is obtained by subtracting the **mean pixel** values from their individual pixel values and then dividing them by the **standard deviation** of the values of pixels. Data standardization means your data is internally consistent and each of your data sources has the same format and labels. The next formula represents data standardization for your input data [**38**]:

$$X_{stand} = \frac{X-\mu}{\sigma}$$

   Note That:

   (a) X: Individual pixel values.

   (b) μ: The mean pixel values.

   (c) σ: Standard deviation of pixel values.

2.3.2 Train U-Net Model

After data pre-process, and without going any further. We have already splitted our data into 3 parts as follows:

1. Model training data, which represents 70 % of the total data.
2. Model validation data, which represents 20 % of the total data.
3. Model testing data, which represents 10 % of the total data.

1. **Training process**

   As we mentioned in the previous chapter that the U-Net model comprises two main phases (contracting and expanding paths), this section explains the training process.

(a) **Down-sampling path**

After data preprocess, we feed the input layer of the U-Net model with this data. In this case, the input layer receives data with size of (256x256x1) as inputs. Then we pass the input layer into the first block of the contracting path.

Each block in the contracting path, is composed of the repeated application of two convolutions operations with Nnumber of kernels size of 3x3. N Starts from 32 and multiplies after each block end (e.g. 32, 64, 128 and so on), and becomes 512 in the bottleneck layer. Each convolution operation in this path is followed by a rectified linear unit (**ReLU** activation function) and a **2x2 max pooling** operation. The whole process of the contracting path has the same padding as the input.

(b) **Up-sampling path**

The expanding path is also composed of 4 blocks. Each of these blocks is composed of a 2x2 Up-convolution layer with stride 2. A repeated application of two convolutions operations with M number of kernels size of 3x3. M Starts from 256 and shrinks twice after each block end (e.g. 256, 128, 64 and so on), and becomes 32 in the second convolutions operations of the last block of the expanding path. Each Up-convolution operation in this path is concatenated with the corresponding cropped feature map from the contracting path (see. Fig. 1.18 : Copy and Crop Process). Each convolution operation in this path is followed by a rectified linear unit (reLU activation function).

(c) **Output Layer**

The output layer is the last layer in the U-NET model. This layer is composed of one **convolution operation** with **kernel size of 1x1** followed by a non-linear unit (**sigmoid** activation function).This operation produces a **segmentation map (output predicted mask)**.

2.3.3 Model Compilation process

Before we explain the model compilation process, we would like to announce that we create 3 U-Net models with the same previous architecture. Each of the three models has a

different compilation process. To check the overall performance of our trained models, we used one method call called compile. The compile method requires several parameters. These parameters are:

1. **The metrics parameter**

   Metric parameters enable quantification of the quality of an image interpretation result. This metric takes into account the a priori knowledge used by the interpretation algorithm and the ground truth associated with the original image **[39]**. These metrics are used to evaluate and compare between results of images segmentation obtained by our U-Net model and ground truth of the images (Ground Glass). Among the functions that we used:

   (a) **Pixel Accuracy**

   We are essentially evaluating a binary mask, a true positive represents a pixel that is correctly predicted to belong to the given mask (ground truth mask) whereas a true negative represents a pixel that is correctly identified as not belonging to the given mask **[40]**.

   $$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

   Note that:

   i. **TP (True Positive):** Represents a pixel that is correctly predicted to belong to the given mask.

   ii. **TN (True Negative):** Represents a pixel that is correctly predicted as not belonging to the given mask.

   iii. **FP (False Positive):** Represents a pixel that is incorrectly predicted as not belonging to the given mask, but this pixel is belonging to the given mask.

   iv. **FN (False Negative):** Represents a pixel that is incorrectly predicted to belong to the given mask, but this pixel is not belonging to the given mask.

   (b) **Intersection over Union (IoU, Jaccard Index)**

   The IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks **[40]**.

$$IoU = \frac{A \cap B}{A \cup B}$$

The intersection $A \cap B$ is comprised of the pixels found in both the prediction mask and the **ground truth mask**, whereas the union $A \cup B$ is simply composed of all pixels found in either the **prediction or target mask [40]**.

(c) **Dice Coefficient (F1 Score)**

Dice coefficient, which is essentially a measure of overlap between two samples. The Dice coefficient was originally developed for binary data, and can be calculated as **[41]**:

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

For the case of evaluating a Dice coefficient on predicted segmentation masks, we can approximate $|A \cap B|$ as the pixel-wise multiplication between the prediction and target mask, and then sum the resulting matrix. This sum divided by simple sum of two matrix $|A|$ and $|B|$ **[41]**.

2. **Loss functions**

The loss function is function that computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how your algorithm models the data **[42]**. Among the loss functions that we used:

(a) **Binary Cross-Entropy (Log loss)**

This loss function is used for binary problems. **We used this function with both pixel accuracy and IoU metrics** (Accuracy model, and IoU model). The formula of this loss function is given as:

$$h_p(Q) = -\frac{1}{N} \sum_{i=1}^{i=N} y_i.log(p(y_i)) + (1 - y_i).log(1 - p(y_i))$$

Where y is the true mask (**1 for white pixels and 0 for black pixels**) and $p(y)$ is the predicted probability of the point being white for all N pixels.

(b) **Soft Dice Loss (Log loss)**

We used this function with the **Dice Coefficient metric (Dice Coefficient model)**. The formula of this loss function is given as:

$$Dice_{loss} = 1 - Dice.$$

3. **The optimizer**

   Optimizer's algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible [**43**]. In our case, we choose **Adam optimizer** for each model that we created (Accuracy, IoU, and Dice Coefficient models).

Fig 2.3 represents the previously mentioned stages of building our U-Net model.

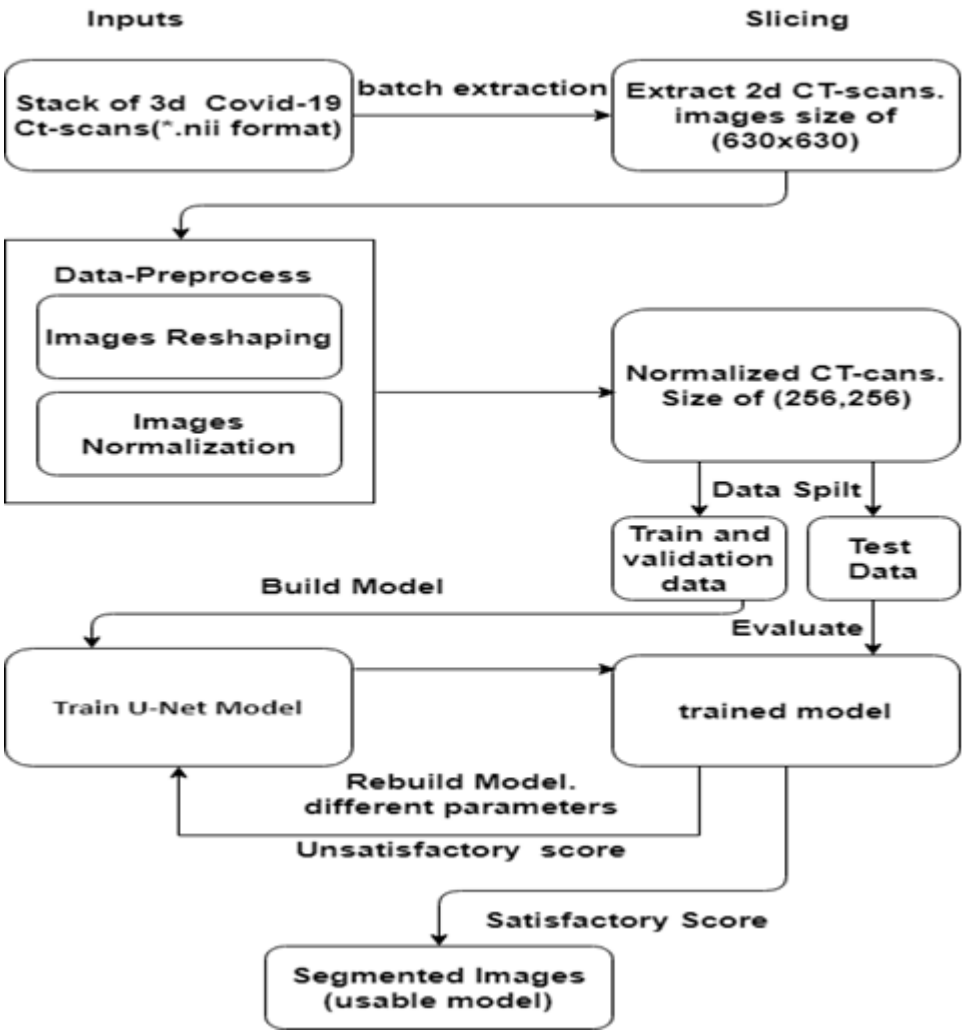


Figure 2.3: Detailed architecture of our system for covid-19 detection and segmentation.

2.4 Implantation Details

2.4.1 Data Source

In this work, we present a deep learning based framework for automatic segmentation of pathologic COVID-19- associated tissue areas from clinical CT images available from publicly available COVID-19 segmentation datasets [**37**]. The dataset contains 20 labeled COVID-19 CT scans as **NifTi format (*.nii)**. Left lung, right lung, and infections are labeled by two radiologists and verified by an experienced radiologist [**44**].

Dataset Available at:`https://zenodo.org/record/3757476#.YMYwHNUzbIX`

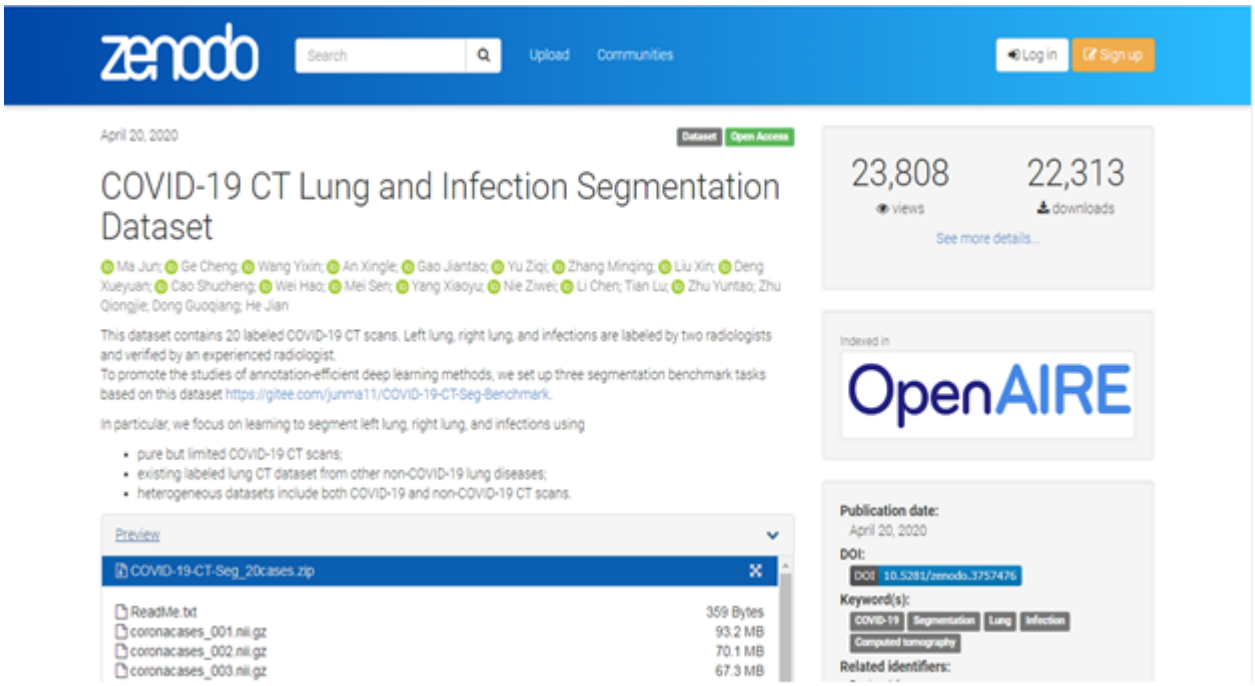


Figure 2.4: Covid-19 CT Lung and Infection Segmentation Dataset website.

2.4.2 Data Source

In this section, we introduce the tools used to release our general model. These tools are:

1. **Development Environment**

   As we know that deep learning requires a lot of computational power to run on. That means Deep Learning requires a lot of hardware. A simple deep learning

model train may take a lot of hours, maybe days on simple laptops (typically without GPUs). Which leads to an impression that Deep Learning requires big systems to run [**45**]. In our case we used **Google Colab** for the reason that Google Colab **supports deep learning projects** by providing **GPU backend servers and a free 12 of memory**.

2. **Programing Language**

   We used **python version 3.8** for developing this model, as we know that python provides many libraries of image manipulation and processing like numpy, pandas, etc. Python language is number one in the domain of developing deep learning models.

3. **Libraries and Frameworks**

   In this part, we present the most important libraries used to develop our deep learning pro, which are:

   (a) **Nibabel Library**

   The main objective of this library in our project is to load 3d stack of images saved as nifti format (*.nii) with their ground truth masks, and return 2d image, mask data from each stacked image, mask (**slicing 3d stack of images and masks on the axis = z**).

   Listing 2.1: Process of loading and slicing a 3d stack of images.

```
for Masks, Images in zip(training_masks, training_img):
   #we load 3D training image,training mask
   training_mask =
      nibabel.load((os.path.join('/content/drive/MyDrive/code
   /dataset/infection_mask/',Masks)))
   training_image =
      nibabel.load((os.path.join('/content/drive/MyDrive/code
   /dataset/ct_scans/',Images)))

   #slicing 2d training images,training masks using get_data()
      function where axis k = z;
```

```
    for k in range(training_image.shape[2]):
        image_2d = np.array(training_image.get_data()[::2, ::2,
            k])
        imgs_train.append(image_2d)
    for k in range(training_mask.shape[2]):
        mask_2d = np.array(training_mask.get_data()[::2, ::2, k])
        masks_train.append(mask_2d)
```

(b) **Numpy and sickit-learn librabies**

The main objective of the numpy library in our project is to treat images as an array of pixels. Another main objective of this function is to normalize all images data by using images standardization technique. We used sickit-learn library to reshape our data **(images with their masks)** first, then we used numpy library to normalize our data **(only images)**.

Listing 2.2: Data Preprocess (images reshaping then normalization).

```
#import numpy library
import numpy as np
def Images_resizing(imgs,img_cols,img_rows):
    imgs_p = np.ndarray((imgs.shape[0], img_rows, img_cols),
        dtype=np.uint8)
    #image reshaping
    for i in range(imgs.shape[0]):
        imgs_p[i] = resize(imgs[i], (img_cols, img_rows),
            preserve_range=True)
    imgs_p = imgs_p[..., np.newaxis]
    imgs_p = imgs_p.astype('float32')
    # images standardization process #
    mean = np.mean(imgs_p)
    std = np.std(imgs_p)
    imgs_p -= mean
    imgs_p /= std
    return imgs_p
```

(c) **Keras Library**

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras is the most used deep learning framework. We used this library to import, build, compile, and evaluate our U-Net model.

Listing 2.3: Representation of U-Net construction path (first block).

```
def get_unet():
    #Input Layer
    inputs = Input((256, 256, 1))
    #First Block of U-Net construction path
    # 3x3 double convolution operations
    conv1 = Conv2D(32, (3, 3), activation='relu',
        padding='same')(inputs)
    conv1 = Conv2D(32, (3, 3), activation='relu',
        padding='same')(conv1)
    #max 2x2 pooling Layer
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
```

2.5 Results

As we mentioned before, we splitted our data into 3 parts. We train the three models (Accuracy Model, IoU Model, and Dice Model) on 70 % of the data and validate it on 20 % of the remaining training data, then testing performance of each model using test data, this test data represents 10 % of the total data. The summary of these three models consist of the following steps:

1. These models were trained on 467 samples of images, and were validated on 119 samples, then tested on 61 samples.

2. Number of epochs are 100, 100, 250 for accuracy, IoU, and dice models respectively.

3. The training and the predicting time of the Accuracy model is 18 seconds between each epoch, and the first epoch takes 101 seconds. In total for 100 epochs we have 1883 seconds approximate to 31 minutes.

4. The training and the predicting time of the IoU model is 17 seconds between each epoch, and the first epoch takes 97 seconds. In total for 100 epochs we have 1780 seconds approximate to 30 minutes.

5. The training and the predicting time of the Dice model is 17 seconds between each epoch, and the first epoch takes 100 seconds. In total for 250 epochs we have 4333 seconds approximate to 72 minutes.

6. Best model weights are saved as ***.h5** file for each model.

2.5.1 Models Scores and Losses

The following figures represent Models score and loss during training and validation for each model:

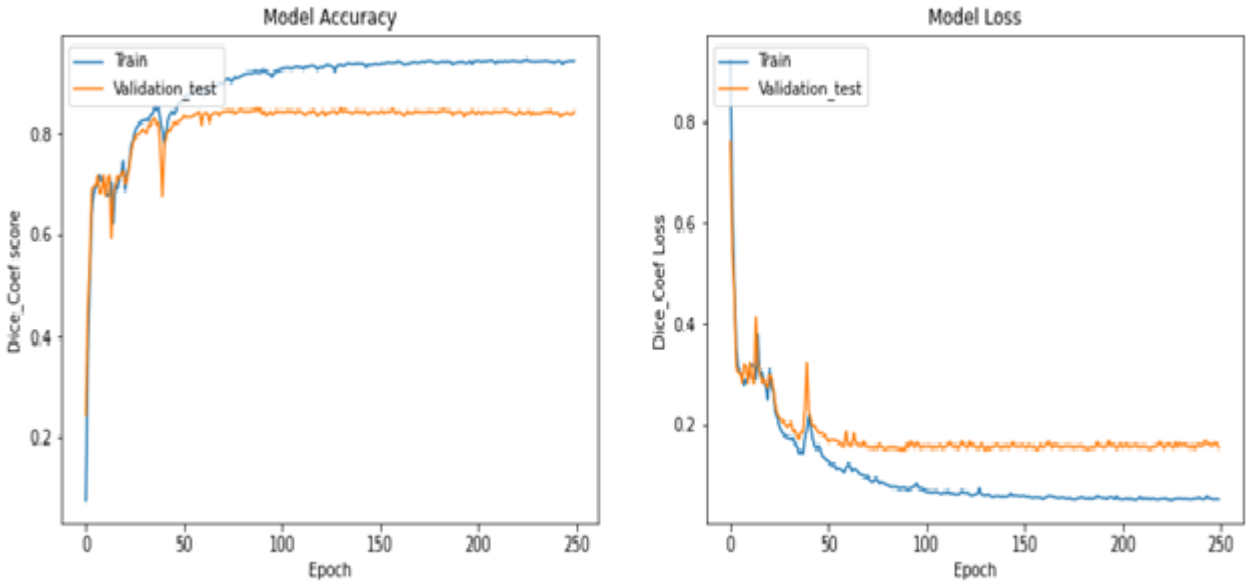


Figure 2.5: Dice_coef score history during training and validation, right: Dice_coef loss history during training and validation.

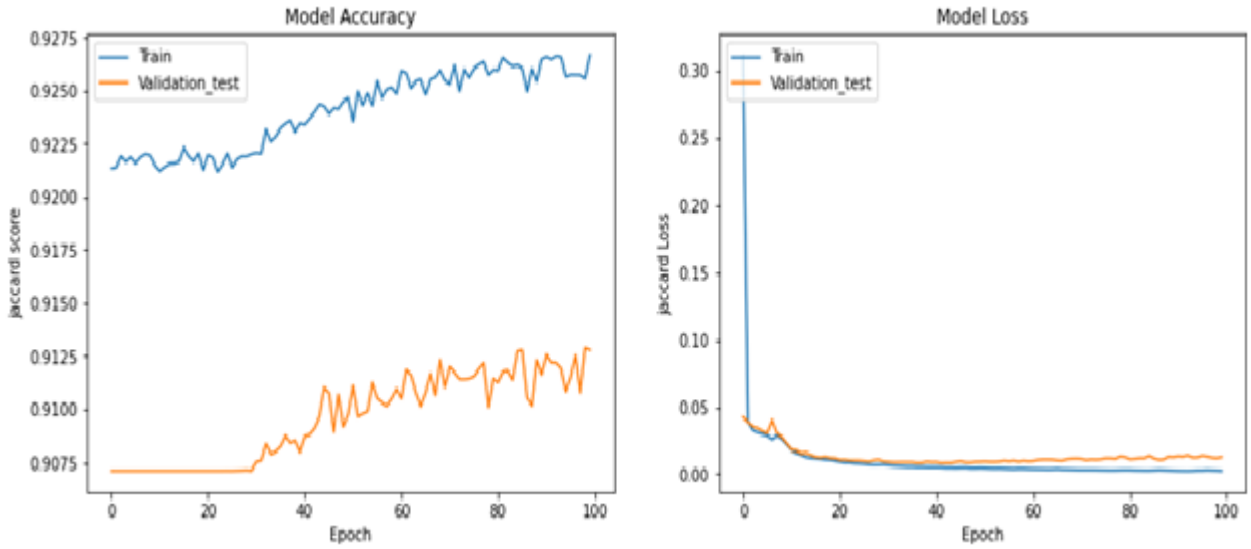


Figure 2.6: IoU score history during training and validation, right: IoU loss history during training and validation.

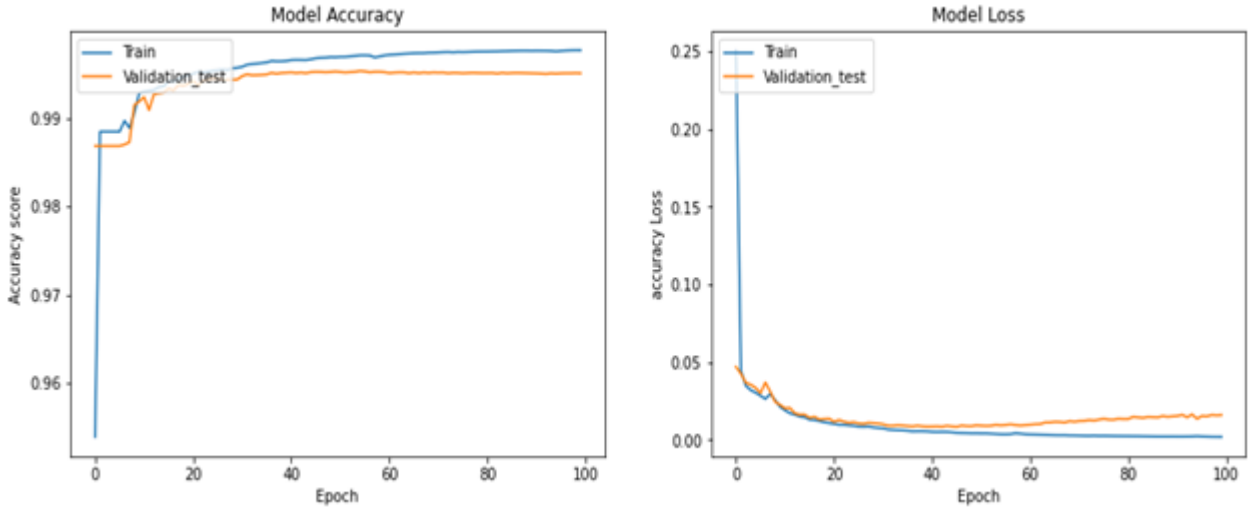


Figure 2.7: Accuracy score history during training and validation, right: Accuracy loss history during training and validation.

2.5.2 Segmentation Results

The following figures represent some output maps **(predicted masks)** after segmentation applied on test data:

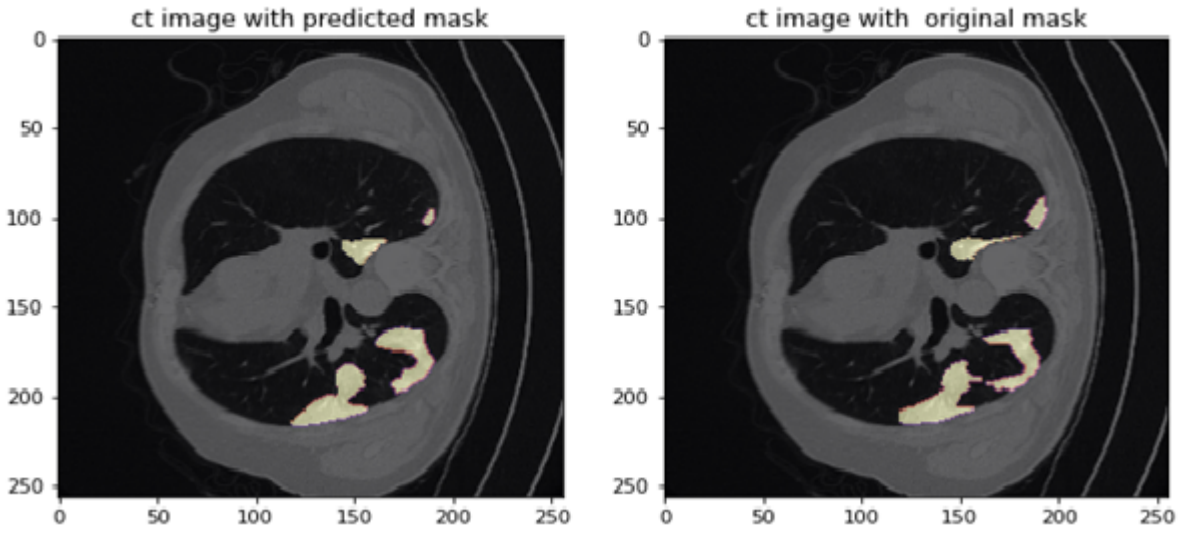


Figure 2.8: Results of Dice_coef segmentation metric, left: CT-scan with predicted covid-19 infection mask, right: CT-scan with true covid-19 infection mask.

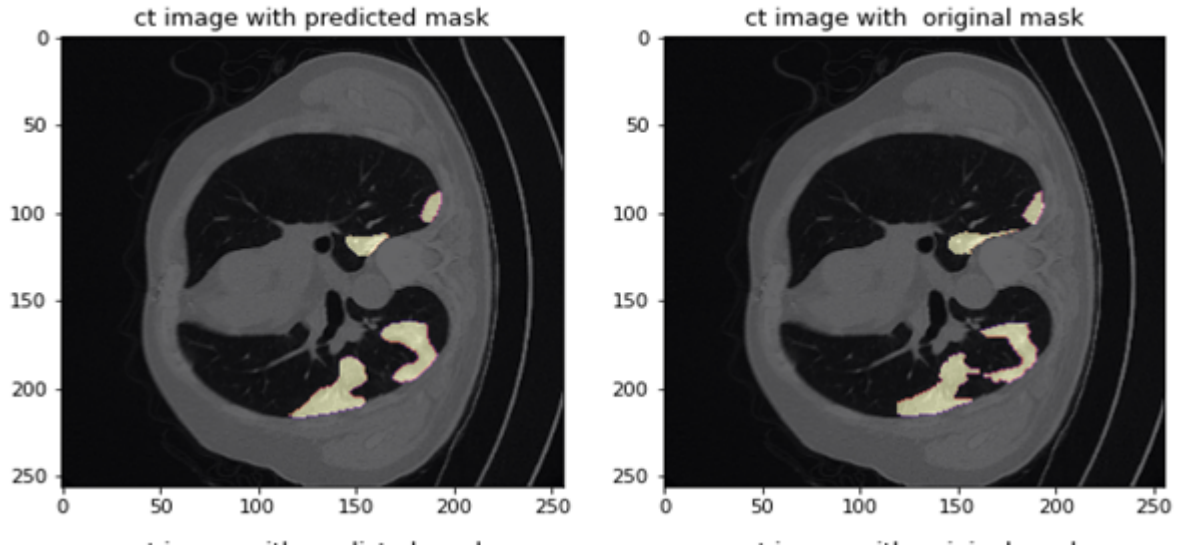


Figure 2.9: Results of IoU segmentation metric, left: CT-scan with predicted covid-19 infection mask, right: CT-scan with true covid-19 infection mask.

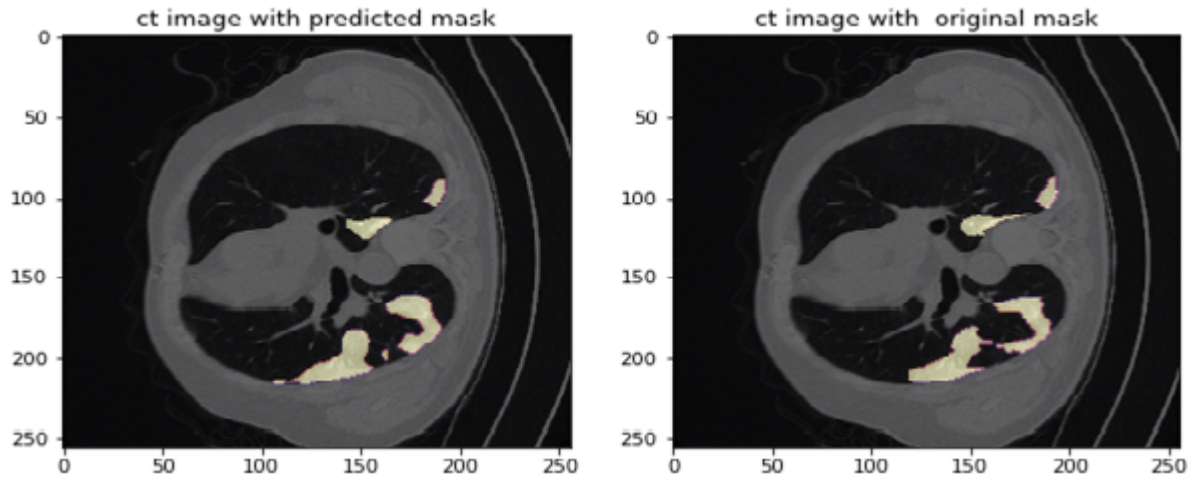


Figure 2.10: Results of Accuracy segmentation metric, left: CT-scan with predicted covid-19 infection mask, right: CT-scan with true covid-19 infection mask.

2.6 Discussion

The segmentation performances of the different compilation metrics of the U-Net model are presented in Table 1.1. The results indicate that the accuracy model performs better than the two other models, but the results are very close with insignificant differences specialy between IoU and Dice_coef models.

Metrics	Training time (for 100 epochs)	train+validation score	Test score
Accuracy	31 minutes	99.71%	99.48%
IoU	30 minutes	92.33%	87.42%
Dice_Coef	29 minutes	91.98%	86.03%

Table 2.1: A Comparison between various segmentation metrics results (Model scores).

2.7 conclusion

A novel deep learning framework for COVID-19 segmentation from CT images was reported [**37**]. In this work, we implemented state-of-the-art U-Net deep neural networks to automatically segment COVID-19 lesions on CT. This model receives a CT slice as input and gives a binary mask as output that specifies which areas of the image are involved with the infection. The results demonstrate that the U-Net model performs very well, achieving a dice coefficient of 0.86 and an IoU of 0.87.

We showed that our trained model achieved a high accuracy rate for detecting pathologic Covid-19 regions. We report the model performance under various hyper-parameter settings, which can be helpful for future research by the community to know the impact of different parameters on the final results.

General Conclusion

In this work, we used deep neural networks, specifically convolutional neural networks (CNNs), to automatically segment the infected tissues inside the lung caused due to the COVID-19 from CT images.

On the theoretical side, we introduced the deep learning technique in the field of image segmentation in general. We explained the components of convolutional neural networks from layers, kernels, etc. The role of each one of these components in the image segmentation process, then we talked in detail about one of the most popular and widely used deep learning architectures for segmenting medical images (U-Net architecture).

On the experimental side, we implemented this architecture using various image segmentation evaluation metrics to automatically detect and segment the areas of the lung infected with the COVID-19.

We noticed that the amount of data used to build the structure and some parameters played an important role in changing the performance of the decision model for automatic segmentation.

Since the data used in this work is widely available, this made our built model achieve satisfactory results using various image segmentation evaluation metrics.

A comparison between various image segmentation metrics of areas of COVID-19 infection shows that all of these metrics perform very well. The results obtained in this work represent promising prospects for the possibility of using deep learning to assist in an objective diagnosis of COVID-19 disease through CT images of the lung [**3**].

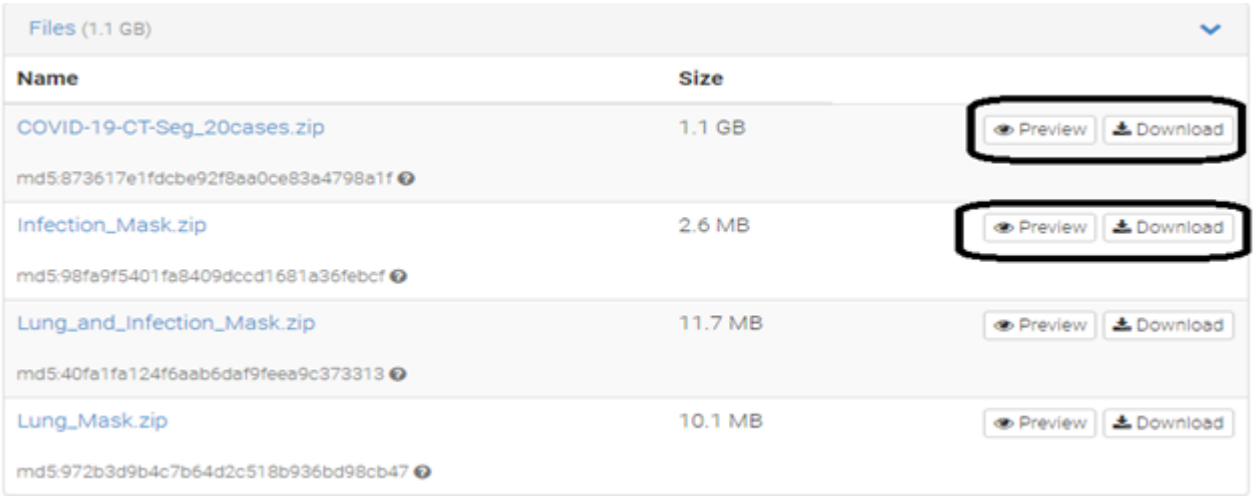
Appendices

3.1 Download and Upload Covid-19 Dataset

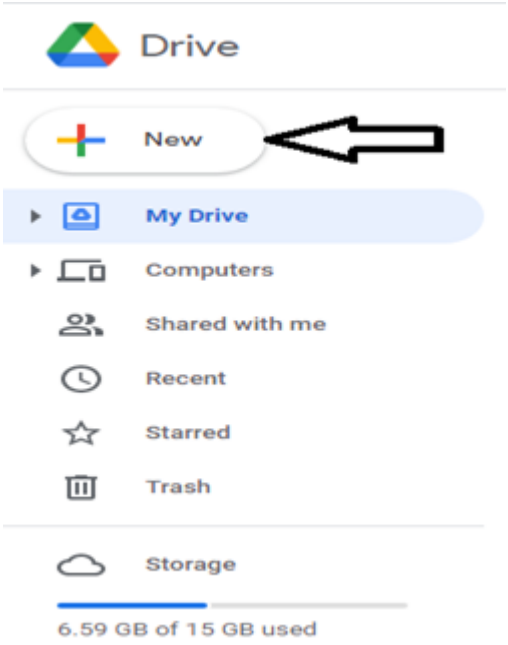
Dataset Available here: `https://zenodo.org/record/3757476#.YNwnS9UzbIX` Google Drive link: `https://drive.google.com/drive/my-drive`

After we access to Covid-19 dataset website, we go to the following section and download the following two zip files (as we selected in picture):

1. Download COVID-19-CT-SG_20cases.zip.
2. Download Infection_Mask.zip.



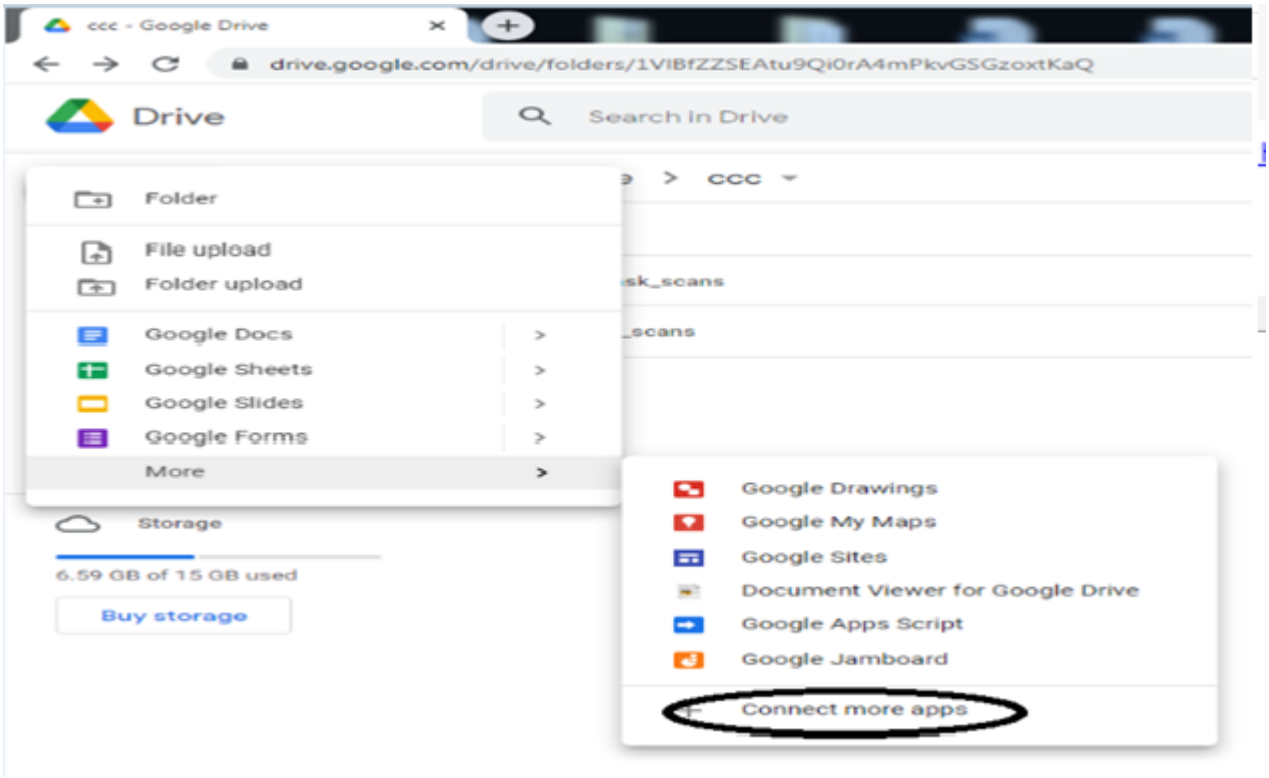
After downloading and extracting previous zip files, we create google drive account to upload these extracted files to it. All we need just Gmail to create one. After creating google drive, we create new Folder by clicking in new button in the left upper of the screen then choosing Folder and name it. Now after that create another two new folders inside this folder and name them CT_scans, Mask_scans respectively. Then upload Your Covid-19 CT scans into CT_scans folder by clicking New then File upload. The same thing for the Covid-19 infection masks. Upload it to Mask_scans Folder.

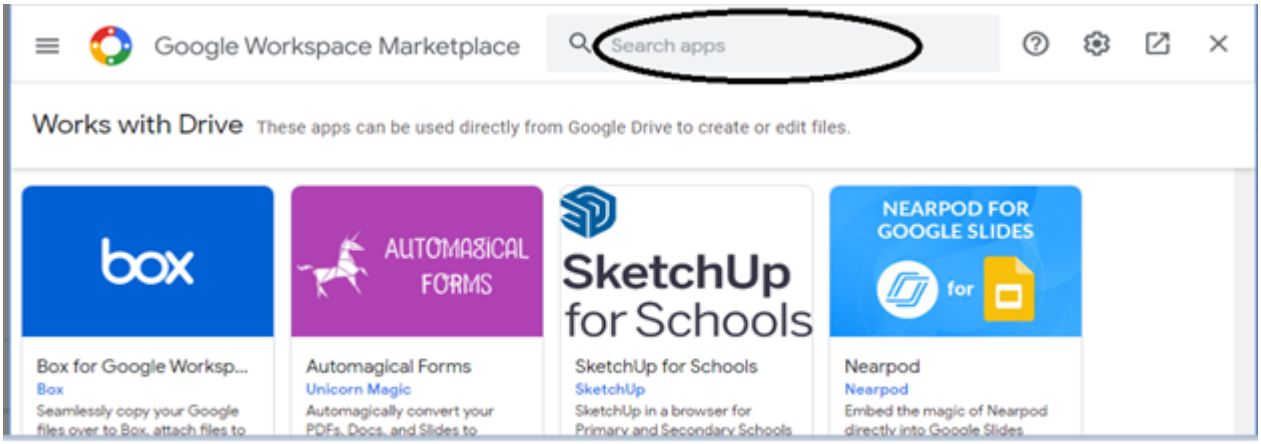


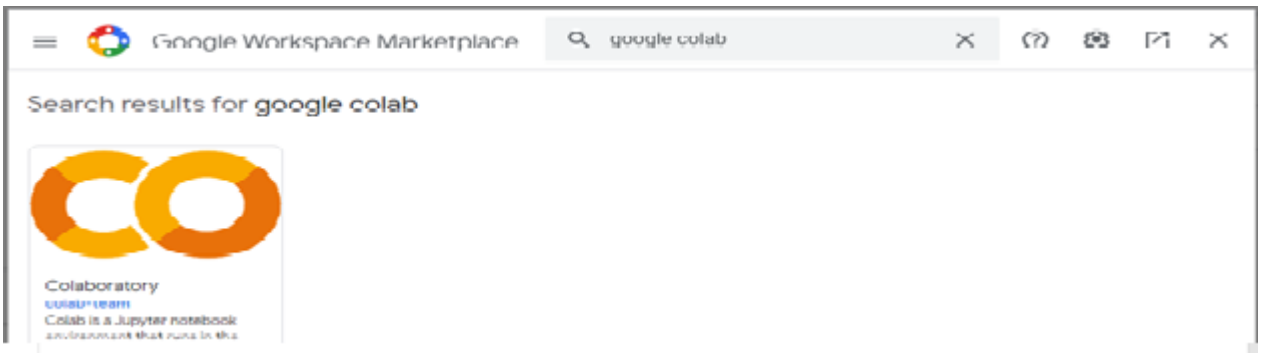
3.2 create and configure google colab

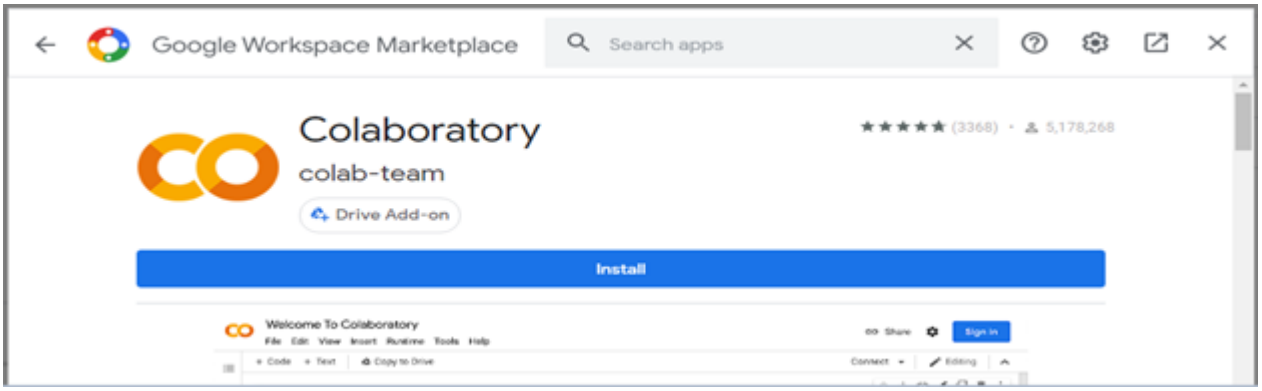
Now enter the first folder that we created, we do the following steps:

1. **Right click** in the blank inside the folder or choose **new** button.
2. Choose **More** ⇒ Connect **more apps**.
3. A window will appear in the center of the screen. Search for **google colab** in zone of search.
4. Click on **Colaboratory** ⇒ then **install**.


ccc - Google Drive
drive.google.com/drive/folders/1VIBfZZSEAtu9Qi0rA4mPkvGSGzoxtKaQ
Drive
Search in Drive
ccc
Folder
File upload
Folder upload
Google Docs
Google Sheets
Google Slides
Google Forms
More
Storage
6.59 GB of 15 GB used
Buy storage
Google Drawings
Google My Maps
Google Sites
Document Viewer for Google Drive
Google Apps Script
Google Jamboard
Connect more apps


Google Workspace Marketplace
Search apps
Works with Drive
These apps can be used directly from Google Drive to create or edit files.
box
Box for Google Worksp...
Box
Seamlessly copy your Google files over to Box, attach files to
AUTOMAGICAL FORMS
Automagical Forms
Unicorn Magic
Automagically convert your PDFs, Docs, and Slides to
SketchUp for Schools
SketchUp for Schools
SketchUp
SketchUp in a browser for Primary and Secondary Schools
NEARPOD FOR GOOGLE SLIDES
for
Nearpod
Nearpod
Embed the magic of Nearpod directly into Google Slides

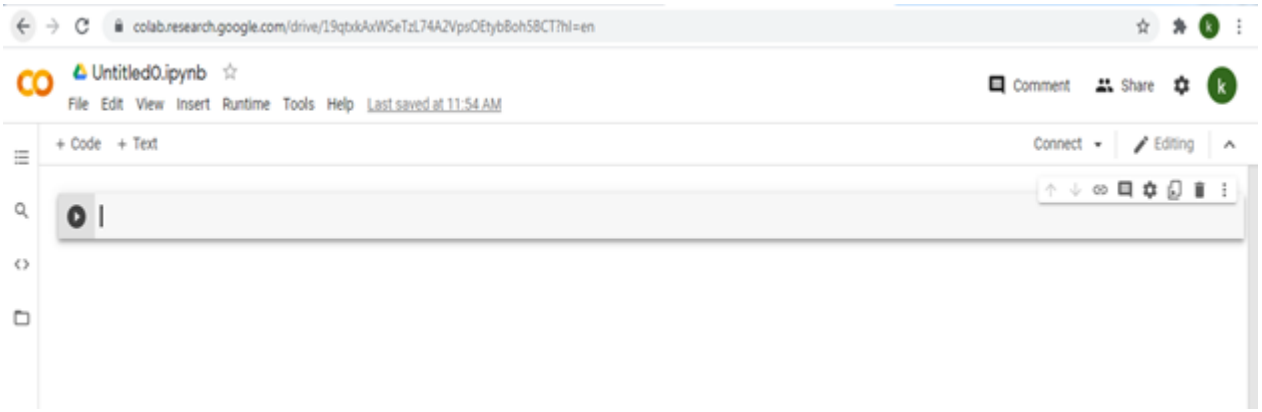




After completing these steps, now we are ready to create new google colab project, using the following steps:

1. **Right click** in the blank inside the folder or choose **new button**.
2. Choose **More** ⇒ **Google Colaboratory**.

A new window will appear like this one

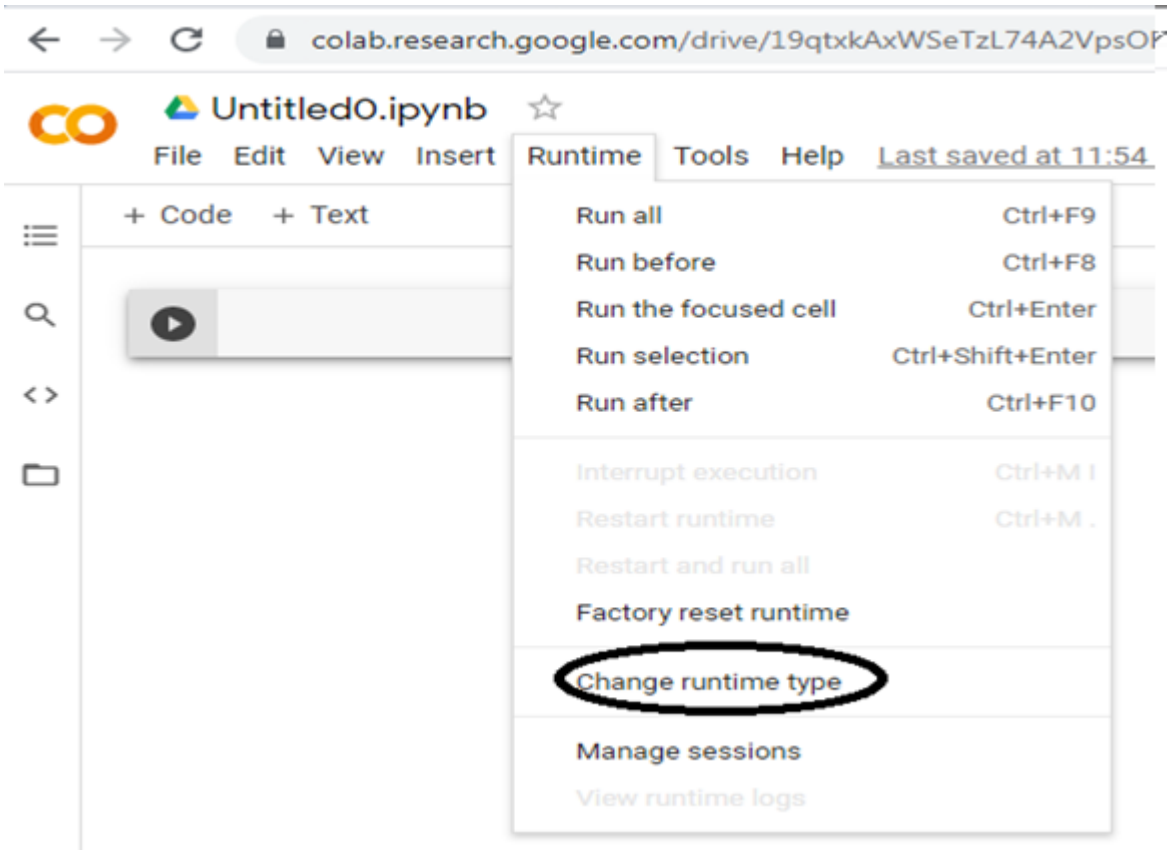


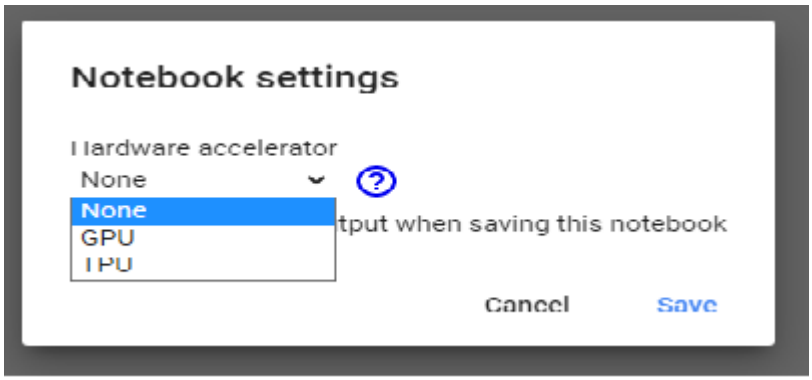
We set our google into GPU runtime compiler as follows:

1. From menu bar choose **Runtime** ⇒ click on **Change runtime type**.

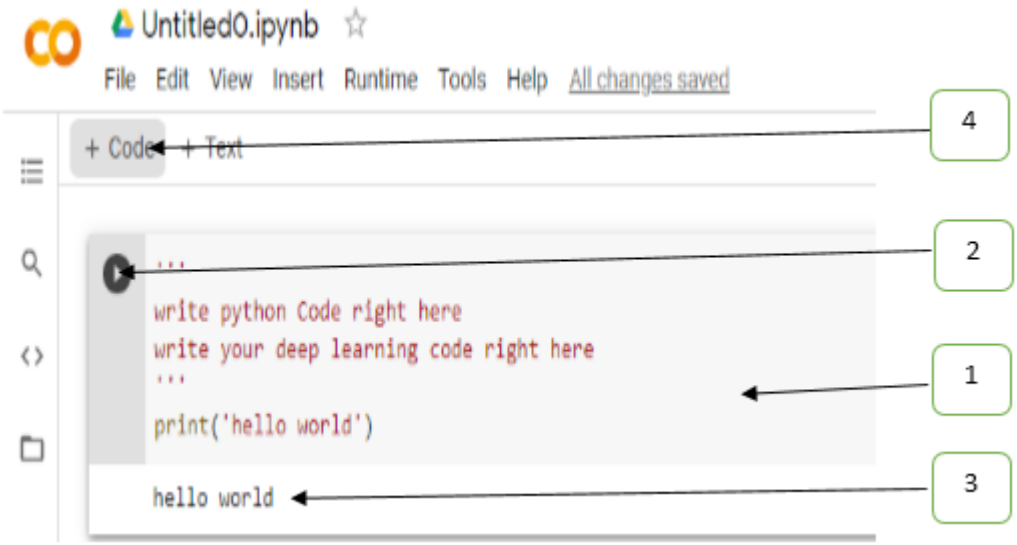
2. Then click on **Notebook settings** (Hardware accelerator) ⇒ choose **GPU**.

3. Click **save** to activate GPU runtime accelerator.





Now google colab window is ready to build deep learning models (U-Net Model in our case). All we need is to write and run our code in zone of code.



Note that:

1. (1): Represents zone of coding instruction.
2. (2): Execute the code inside this zone only.
3. (3): Results of execution.
4. (4): Add new zone of coding instruction.

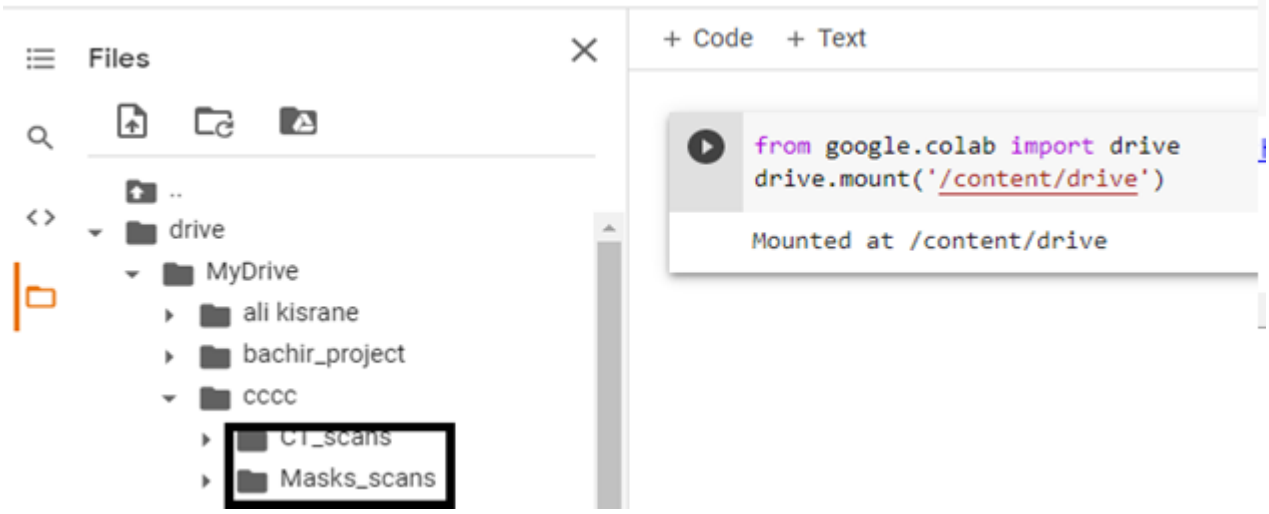
After these whole operations, we need to connect google drive with google colab for the reason that we need our uploaded dataset (Covid-19 CT scans with their masks) to build our U-Net model. For that we will write and run the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

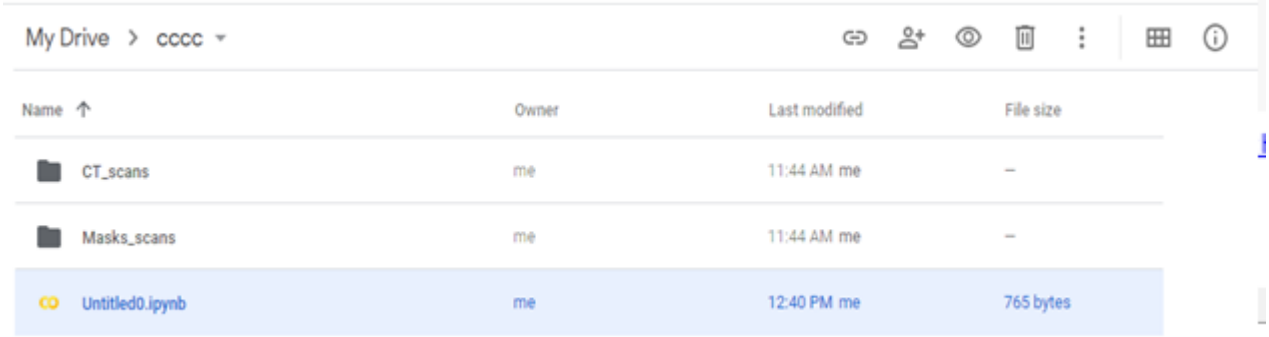
```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/aut

Enter your authorization code:
4/1AX4XfWhkgkgePa4kEaB
```

After run this code, we click on Go to this URL in a browser link to allow the access for google colab to our drive folders. C**opy and paste the authorization code** then click **enter key**.

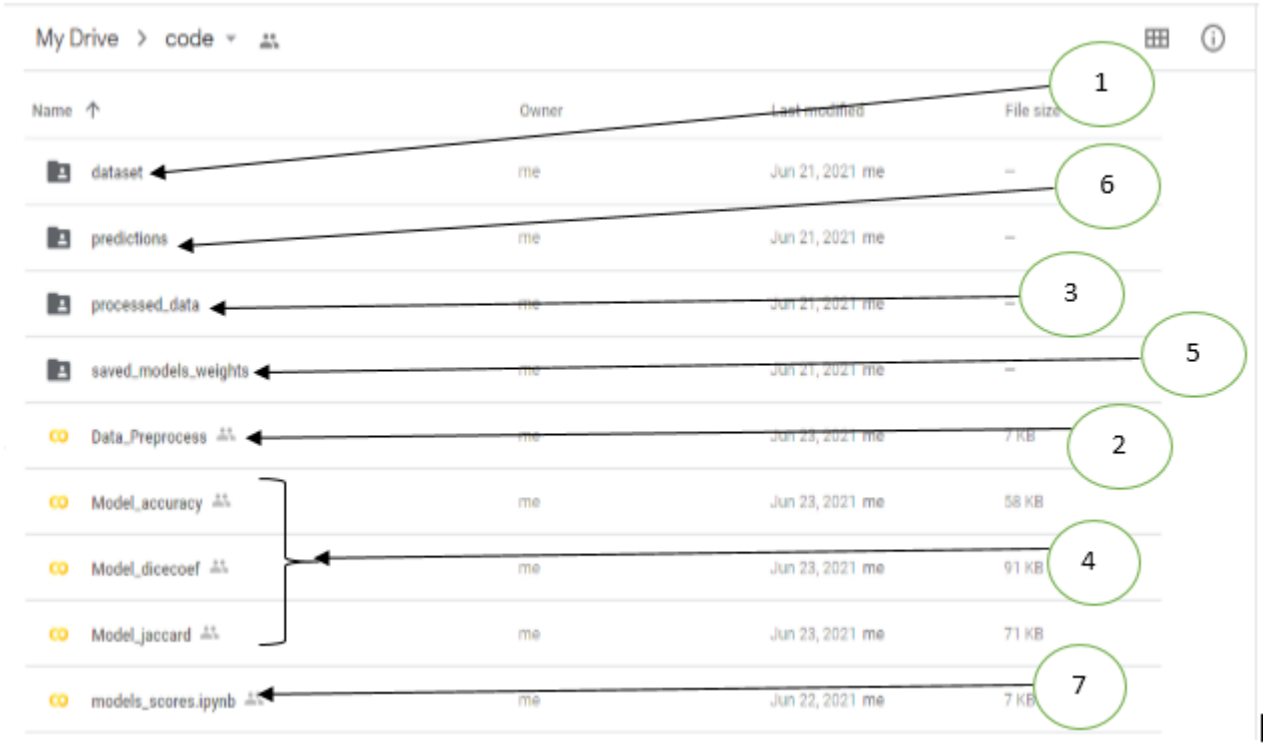
Now we can access to our Covid-19 dataset by using the path of each folder of these following folders that we need in our project:



Finally, to save the google colab code project, just click CTRL+S. This will create **Untitled0.ipynb** saved project.



3.3 General description



1. (1): First uploaded dataset which containing **CT_scans** with thier **infection mask**.
2. (2): Represents python code for data preprocess **(Images and masks preprocess)**. In this case we load and read the first uploaded dataset (number 1) then applying our data reshaping and normalization methods.
3. (3): Represents result of these prevoius operations **(saved in folder called processed_ data)**.
4. (4): Represents our training U-Net models code using different evaluation metric (Accuracy, IoU, Dice).
5. (5): Represents saved weights of our U-Net models.
6. (6): Represents our U-Net models prediction results in test set (saved as array of images **.npy**).
7. (7): Represents our UNet models scores (see. Table. 2.1).

Bibliography

[1] Yanhui Guo, Amira S. Ashour, "Neutrosophic sets in dermoscopic medical image segmentation". in Neutrosophic Set in Medical Image Analysis, 2019. URL: https://www.sciencedirect.com/topics/engineering/medical-image-segmentation

[2] Adel Oulefki, Sos Agaian, Thaweesak Trongtirakul, Azzeddine Kassah Laouar, "Automatic COVID-19 lung infected region segmentation and measurement using CT-scans images", 2 Nov 2020. URL: https://pubmed.ncbi.nlm.nih.gov/33162612/

[3] Adnan Saood, HatemAdnan, "COVID-19 lung CT image segmentation using deep learning methods: U-Net versus SegNet". 09 Feb 2021. URL:https://bmcmedimaging.biomedcentral.com/articles/10.1186/s12880-020-00529-5

[4] Yan Hao, Song Yuheng, "Image Segmentation Algorithms Overview". 7 Jul 2017 . URL: https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf

[5] Dr Daniel J Bell and Ass. Pr. Mirjan M. Nadrljanski et al, "Computed tomography". URL: https://radiopaedia.org/articles/computed-tomography

[6] Dr Daniel J Bell and Dr Zemar Vajuhudeen et al, "Electronvolt (unit),". URL: https://radiopaedia.org/articles/electronvolt-unit

[7] "Biological Materials Information". URL: https://www.globalspec.com/learnmore/specialized_industrial_products/pharmaceutical_biotechnology/biotechnology/biological_materials

[8] Rajeev Tiwari, Aayushi Priya, "A Review on Segmentation Techniques in Medical Images". Feb 2017. URL: https://www.researchgate.net/publication/332595639_A_Review_on_Segmentation_Techniques_in_Medical_Images

[9] J. Vergés–Llahi, "A Color Image Segmentation Algorithm". URL: https://www.tdx.cat/bitstream/handle/10803/6189/07Jvl07de11.pdf?sequence=7

[10] Yadwinder Kaur, Dilpreet Kaur, "Various Image Segmentation". 5 May 2014. URL: https://ijcsmc.com/docs/papers/May2014/V3I5201499a84.pdf

[11] S. Ferrari, "Image segmentation". 2011-2012. URL: http://homes.di.unimi.it/ferrari/ImgProc2011_12/EI2011_12_16_segmentation_double.pdf

[12] Alireza Norouzi, Mohd Shafry Mohd Rahim , Ayman Altameem, Tanzila Saba, "Medical Image Segmentation Methods, Algorithms,and Applications". 12 Jul 2014. URL: https://www.researchgate.net/publication/263608069_Medical_Image_Segmentation_Methods_Algorithms_and_Applications

[13] F. Albregtsen, "REGION EDGE BASED". 21 Sep 2011. URL: https://www.uio.no/studier/emner/matnat/ifi/INF4300/h11/undervisningsmateriale/INF4300-2011-f04-segmentation.pdf

[14] D. Withey, Z. Koles, "A Review of Medical Image Segmentation: Methods and Available Software". 2008.

[15] D. Marshall, "Region Growing". 1994-1997. URL: https://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node35.html

[16] Dong Gook Lee, Ji Ho You, Sung Geun Park, Seung Hyub Baeck, and Hyun Jik Lee, "Influence of Pretreatment on Adhesion Quality of Supercritical-fluid-deposited Cu Film on Si". Sep 2019. URL: https://www.researchgate.net/publication/335735186_Influence_of_Pretreatment_on_Adhesion_Quality_of_Supercritical-fluid-deposited_Cu_Film_on_Si

[17] Zhensong Chen, Zhiquan Qi, Fan Meng, Limeng Cui, Yong Shi, "Image Segmentation via Improving Clustering Algorithms with Density and Distance". In Information

Technology and Quantitative Management (ITQM 2015). doi: https://doi.org/10.1016/j.procs.2015.07.096

[18] Himanshu Mittal, Avinash Chandra Pandey, Mukesh Saraswat, Sumit Kumar, Raju Pal, Garv Modwel, "A comprehensive survey of image segmentation: clustering methods, performance parameters, and benchmark datasets". 09 Feb 2021. URL: https://link.springer.com/article/10.1007/s11042-021-10594-9

[19] AMZA CATALIN, "A REVIEW ON NEURAL NETWORK-BASED IMAGE". De Montfort University, Mechanical and Manufacturing Engineering. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.1365&rep=rep1&type=pdf

[20] Rajarshi Bhadra, "What is a Neural Network?". 5 Sep 2019. URL: https://towardsdatascience.com/what-is-a-neural-network-a02b3c2fe3fa

[21] MARSHALL HARGRAVE, "Deep Learning Definition". URL: https://www.investopedia.com/terms/d/deep-learning.asp

[22] "What Is Deep Learning? 3 things you need to know." URL: https://www.mathworks.com/discovery/deep-learning.html

[23] Witold Pedrycz, Shyi-Ming, Chen Deep Learning: Concepts and Architectures (Studies in Computational Intelligence, 866), Springer; 1st ed. 2020 edition (November 13, 2019), 29 Oct 2019

[24] A. Sufian, "Conceptual model of CNN". Jan 2020. URL: https://www.researchgate.net/figure/Conceptual-model-of-CNN41_fig2_337401161

[25] Rikiya Yamashita, "Convolutional neural networks: an overview and application in radiology". 22 Jun 2018. URL: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

[26] A. Sufian, Farhana Sultana, Amlan Chakrabarti, Anirudha Ghosh, "Fundamental Concepts of ConvolutionalNeural Network". In book: Recent Trends

and Advances in Artificial Intelligence and Internet of Things (pp.519-567), Jan 2020. URL: https://www.researchgate.net/publication/337401161_Fundamental_Concepts_of_Convolutional_Neural_Network

[27] Zhao Guyu, "Convolution process.," Sep 2019. URL: https://www.researchgate.net/figure/Convolution-process_fig2_335878018

[28] S, Si., M.T. Budhi Irawan, M.T. Casi Setiningsih S.T, Muhamad Yani, "Application of Transfer Learning Using Convolutional Neural NetworkMethod for Early Detection of Terry's Nail". in Journal of Physics: Conference Series, May 2019. URL: https://www.researchgate.net/publication/333593451_Application_of_Transfer_Learning_Using_Convolutional_Neural_Network_Method_for_Early_Detection_of_Terry's_Nail

[29] John A. Bullinaria, "Recurrent Neural Networks", 2015. URL: https://www.cs.bham.ac.uk/~jxb/INC/l12.pdf

[30] Kaushik Das, "HOW RECURRENT NEURAL NETWORK (RNN) WORKS", 05 Nov 2020. URL: https://dataaspirant.com/how-recurrent-neural-network-rnn-works/

[31] L. S. S. L. a. Y. Z. Xiangbin Liu, "A Review of Deep-Learning-Based Medical Image", 25 Jan 2021. URL: https://www.mdpi.com/2071-1050/13/3/1224/pdf

[32] Harshall Lamba, "Understanding Semantic Segmentation with UNET", 17 Feb 2019. URL: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

[33] Arya, "explain fully convolutional network and Unet", URL: https://zhuanlan.zhihu.com/p/65398511

[34] Kuan Wei, "Understand Transposed Convolutions", 29 Jul 2020. URL: https://towardsdatascience.com/understand-transposed-convolutions-and-build-your-own-transposed-convolution-layer-from-scratch-4f5d97b2967

[35] aishwarya mali, "Image Segmentation using U-net", 07 Mar 2018. URL: https://ashm8206.github.io/2018/03/07/Ultrasound-Image-Segementation-Using-Unet.html

[36] Saman Sotoudeh Paima, Ali Bashirgonbadi, Mehran Naghibi,Hamid Soltanian-Zadeh, Navid Hasanzadeh, "Segmentation of COVID-19 Infections on CT: Comparison of Four UNet-Based Networks", 08 Jan 2021. URL: https://ieeexplore.ieee.org/document/9319412

[37] Narges Saeedizadeh, Shervin Minaee, Rahele Kafieh, Shakib Yazdani and Milan Sonka, "COVID TV-UNet: Segmenting COVID-19 Chest CT Images Using Connectivity Imposed U-Net", 24 Jul 2020 . URL: https://arxiv.org/pdf/2007.12303.pdf

[38] Jason Brownlee, "How to Manually Scale Image Pixel Data for Deep Learning", In Deep Learning for Computer Vision, 25 Mar 2019. URL: https://machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning/

[39] Hélène Laurent, Christophe Rosenberger, Baptiste Hemery, "Evaluation Metric for Image understanding", IN IEEE International Conference on Image Processing (ICIP),, Nov 2009, Cairo, Egypt. pp.4381 - 4384, 10.1109/ICIP.2009.5413548. hal-00958185

[40] Jeremy Jordan, "Evaluating image segmentation models", 30 May 2018. URL: https://www.jeremyjordan.me/evaluating-image-segmentation-models/

[41] Jeremy Jordan, "An overview of semantic image segmentation", 21 May 2018. URL: https://www.jeremyjordan.me/semantic-segmentation/#loss

[42] Christophe Pere, "What are Loss Functions?", 17 Jun 2020. URL: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904

[43] Satyam Kumar, "Overview of various Optimizers in Neural Networks", 09 Jun 2020. URL: https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5

[44] "COVID-19 CT Lung and Infection Segmentation Dataset", Coronavirus Disease Research Community - COVID-19, 20 Apr 2020. URL: https://zenodo.org/record/3757476#.YNO7utUzbIU

[45] Jason Dsouza, "What is a GPU and do you need one in Deep Learning?", In Deep Learning, everyone seems to recommend using a GPU. What is it, can you do without one, and who is it exactly for?, 25 Apr 2020. URL: https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa