People Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Larbi Ben M'Hidi -Oum El Bouaghi–

Faculty of Exact Sciences and Sciences of the Nature of Life

Department of Mathematics and Computer Science

Thesis submitted in partial fulfillement of Master's Degree in Computer Sciences

Option: Artificial Vision

Entitled

Semantic Segmentation With U-NET

Presented by: Chenouf Amira and Merzkane Oumayma

In front of the jury:

President: Dr. Zertal Soumia

Examiner: Dr. Khellas Kenza

Supervisor: Dr. Chibani Meriem

2022/2023

ACKNOWLEDGMENTS

*We would like to express my deepest gratitude to **Allah**, who has blessed me with the strength and perseverance to complete this thesis.*

*We would also like to extend my sincere appreciation to our supervisor, **Mme Chibani Meriem**, for her invaluable guidance, support, and encouragement throughout this project. Her expertise, feedback, and dedication have been instrumental in shaping the direction and quality of this work.*

We are also grateful to the examination committee for their time and effort in reviewing and evaluating this thesis. Their insightful comments and constructive criticism have greatly improved this research.

Finally, we would like to thank all the individuals who have contributed to the successful completion of this thesis, whether by providing technical assistance, moral support, or inspiration. We are truly indebted to all of you and appreciate your contribution towards achieving this milestone.

Dedication 1

*I would like to dedicate this thesis to my beloved **Parents**, who have always been my source of inspiration, motivation, and unwavering support throughout my academic journey. Their constant encouragement, guidance, and sacrifice have helped me achieve this significant milestone.*

*I also dedicate this work to my **brothers**, who have always been my pillars of strength and motivation. Their love, encouragement, and unwavering support have been the driving force behind my success.*

*Additionally, I would like to extend my heartfelt thanks to my cousin, **Kiki**, for always being there for me and providing me with the emotional support and encouragement I needed to keep going.*

*I would also like to dedicate this thesis to my tunte **Nassira**, for her unconditional love, support, and guidance, which have been instrumental in shaping my personality and helping me succeed in life.*

*I am also grateful to my binome, **Amira**, for her hard work, dedication, and collaboration on this thesis. Her insightful feedback and support have been invaluable, and I am proud to have had the opportunity to work with her on this project.*

*Finally, I would like to acknowledge **Rania** and **Hadjer** for their unwavering support and help throughout my academic journey.*

Oumayma

Dedication 2

*I dedicate this thesis to my **Mother**, who has always been my guiding light and source of inspiration. Her unwavering love, support, and encouragement have been the driving force behind my academic achievements.*

*I would also like to express my deepest appreciation to my **Father**, who has been my rock and constant support throughout my academic journey. His guidance, encouragement, and love have been essential in shaping my academic and personal growth.*

*I would also like to dedicate this work to my sister **Saliha**, who has been my constant companion and confidante. Her love, support, and understanding have helped me navigate through the challenges of life.*

*Furthermore, I am grateful to my **husband Allaoua**, who has been my biggest supporter and cheerleader. His love, patience, and encouragement have enabled me to pursue my academic goals while also managing my personal life.*

*I would also like to express my appreciation to my friend **Alima**, who has been with me through thick and thin. Her support, advice, and encouragement have been invaluable in keeping me motivated and focused.*

*Lastly, I would like to acknowledge my binome, **Oumayma**, for her hard work, dedication, and collaboration on this thesis. Her insights and contributions have been crucial in shaping the direction and quality of this work.*

To all of these important people in my life, I extend my heartfelt gratitude and deepest appreciation for their love, support, and encouragement, which have made this achievement possible.

Amira

Abstract

Semantic segmentation is a fundamental task in computer vision that involves assigning a specific label to each pixel in an image, it enables machines to understand the scene and extract meaningful information. In this project, we explore the application of the U-Net architecture for semantic segmentation, aiming to develop an efficient and accurate system for pixel labeling.

This project contributes to the field of semantic segmentation by investigating the application of the U-Net architecture. We demonstrate the effectiveness of the U-Net model in accurately segmenting images at the pixel level. The integration of deep learning techniques and the U-Net architecture holds great promise for advancing the field of computer vision and unlocking new possibilities in image understanding and analysis. The findings of this research open avenues for further advancements in semantic segmentation techniques by bridging the gap between theory and practical applications.

Keywords: Semantic segmentation, pixel labeling, U-Net architecture, deep learning techniques.

Résumé

La segmentation sémantique est une tâche fondamentale en vision par ordinateur elle consiste à attribuer une étiquette spécifique à chaque pixel d'une image, permettant aux machines de comprendre la scène et d'extraire des informations significatives. Dans ce projet, nous explorons l'application de l'architecture U-Net pour la segmentation sémantique, dans le but de développer un système efficace et précis pour l'étiquetage pixel par pixel.

Ce mémoire contribue au domaine de la segmentation sémantique en étudiant l'application de l'architecture U-Net. Nous démontrons l'efficacité du modèle U-Net dans la segmentation précise des images au niveau des pixels. L'intégration des techniques d'apprentissage profond et de l'architecture U-Net offre de grandes perspectives pour faire avancer le domaine de la vision par ordinateur et ouvrir de nouvelles possibilités dans la compréhension et l'analyse d'images. Les résultats de cette recherche ouvrent des perspectives pour de nouvelles avancées dans les techniques de segmentation sémantique, comblant ainsi l'écart entre la théorie et les applications pratiques.

Mots clés : La segmentation sémantique, l'architecture U-Net, l'étiquetage pixel par pixel, les techniques d'apprentissage profond.

الملخص

التجزئة الدلالية، وهي مهمة أساسية في رؤية الحاسوب، تنطوي على تعيين تصنيف محدد لكل بكسل في صورة، مما يمكن الآلات من فهم السياق واستخلاص معلومات ذات مغزى. في هذه الرسالة، نستكشف تطبيق بنية U-Net للتجزئة الدلالية، بهدف تطوير نظام فعال ودقيق لتسمية البكسل بشكل مفصل.

تساهم هذه الرسالة في مجال التجزئة الدلالية عن طريق دراسة تطبيق بنية U-Net ، و اثبات فاعلية نموذج U-Net في تجزئة الصور بدقة على مستوى البكسل. اندماج تقنيات التعلم العميق وبنية U-Net يحمل وعودًا كبيرة لتقدم مجال رؤية الحاسوب وفتح إمكانيات جديدة في فهم وتحليل الصور. نتائج هذا البحث تفتح آفاقًا للتطورات المستقبلية في تقنيات التجزئة الدلالية، مع ربط الفجوة بين النظرية والتطبيقات العملية.

الكلمات المفتاحية : التجزئة الدلالية، بنية U-Net، تسمية البكسل بشكل مفصل ، تقنيات التعلم العميق.

.

Table of Contents:

Liste of Figures:

Liste of Tables:

Liste of equals:

TABLE OF ACRONYMS :

AI	**A**rtificial **I**ntelligence
ML	**M**achine **l**earning
DL	**D**eep **l**earning
BC	**B**efore jesus **C**hrist
ANN	**A**rtificial **N**eural **N**etwork
LSTM	**L**ong **S**hort-**T**erm **M**emory
CNNs	**C**onvolutional **N**eural **N**etworks
RNN	**R**ecurrent **N**eural **N**etworks
ReLU	**R**ectified **L**inear **U**nit
TPU	**T**ensor **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
GUI	**G**raphical **U**ser **I**nterface
API	**A**pplication **P**rogramming **I**nterface

General Introduction

The advent of artificial intelligence has had a revolutionary impact in multiple fields, including that of computer vision. Among the major challenges of this field, semantic segmentation which occupies a central place. Its goal is to assign labels to each pixel in an image, which allows more accurate understanding and analysis of visual information. Thanks to semantic segmentation, it becomes possible to understand the details and subtleties of images, thus opening the way to new possibilities for interpreting and exploiting visual data.

Deep learning have played a key role in recent advances in semantic segmentation. It is based on deep artificial neural networks, called convolutional neural networks. These networks are able to learn complex visual features directly from the data, without requiring manual extraction of these features. Thanks to their ability to model hierarchical information, convolutional neural networks are particularly suitable for the task of semantic segmentation.

Among the most successful convolutional neural network architectures for semantic segmentation, U-Net stands out. Originally developed for biomedical image segmentation, U-Net has been widely adopted in other medical fields such as scanners, MRIs, X-rays and ultrasounds. What sets U-Net effective is its ability to capture both global contextual information and important local details. Its U-shaped architecture incorporates direct connections between the convolution layers and the deconvolution layers, allowing better preservation of information while ensuring a more complete understanding of the structure and characteristics of medical images.

Despite the significant progress made in the field of semantic segmentation with U-Net, there are still many challenges to overcome. The central issue lies in improving the accuracy and robustness of segmentation models, especially when faced with complex scenes and high resolution image data. Moreover, training these models often requires large sets of labeled data, which can be a costly and time-consuming task.

Semantic segmentation with U-Net and advances in artificial intelligence offer promising possibilities for improving the accuracy and efficiency of medical image analyses. However, further research is needed to overcome current challenges, especially with regard to obtaining high-quality annotated data and adapting models to anatomical variations and different medical imaging modalities. These efforts will help improve health care and facilitate clinical decision-making.

The problem addressed in this project is « How to overcome the limitations of semantic segmentation models based on artificial intelligence, more specifically deep learning using the U-Net architecture, to improve the accuracy and efficiency of segmentation in the field of medicine? ».

This project focuses on the application of U-Net architecture for semantic segmentation tasks. The aim is to develop an efficient and accurate system that can accurately segment objects and regions of interest in images. The research conducted in this manuscript explores the state of the art of deep learning, convolutional neural networks, and semantic segmentation techniques, with a particular emphasis on the U-Net architecture. Furthermore, we have presented the U-Net applications, its most famous versions and we have elaborated a comparaison between these latter.

Outline of the manuscript:

The remainder, of this manuscript is organized as follows :

- Chapter one, **"State of the Art",** we present a comprehensive overview of the current state of the art of deep learning, convolutional neural networks, and image segmentation. This chapter serves as a foundation, providing a concise summary of the fundamental concepts and principles related to these topics.
- Chapter two, **"U-NET Architecture",** focuses exclusively on the U-Net architecture. This chapter delves into the details of the U-Net model.
- The last chapter, **"Implementation",** we delve into the practical aspect of our research by conducting experiments and presenting the corresponding results. This section provides an in-depth exploration of our experimental methodology and the outcomes that we achieved.

Finally , a conclusion gives final remarks and suggests future works.

Chapter 01: State of the Art

1.1 Introduction

Artificial intelligence has become an essential part of modern technology, allowing machines to perform tasks that typically require humain like intelligence. One of the most prominent branches of artificial intelligence is machine learning, which has evolved into deep learning in recent years.

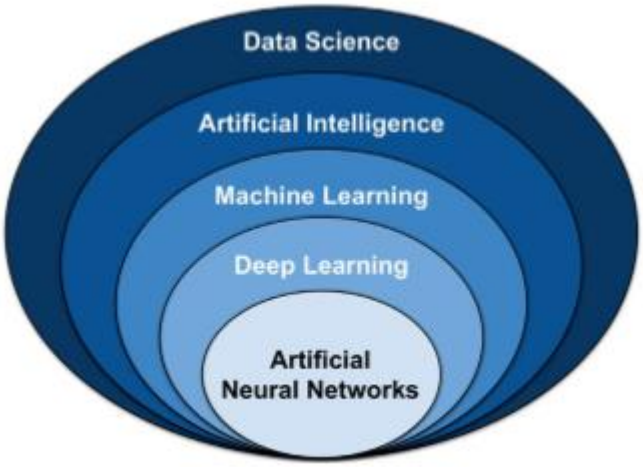


Figure 1.1: The relationship between Data Science, AI, ML, DL and ANN [1].

Image semantic segmentation is a critical task in computer vision, enabling machines to understand the content of an image at a pixel-level. The success of deep learning has significantly improved the accuracy of semantic segmentation, making it the most promising approach to solve this problem. Deep learning based segmentation models learn the features of images automatically, eliminating the need for hand crafted features.

This chapter aims to provide an overview of deep learning and its applications in image semantic segmentation. In the first section, we will introduce deep learning and its architecture. We will also discuss the various types of deep learning models, including supervised and unsupervised learning, along with their applications.

Section 2 will focus on CNNs, which are particularly effective for image segmentation tasks. We will introduce the layers of a typical CNN, including convolutional layers, pooling layers, and fully connected layers, and discuss how they contribute to the learning process. We will also provide an overview of some famous convolutional networks, such as AlexNet, ResNet, and ZFnet , that have achieved state of the art results on various image segmentation datasets.

In section 3, we will explore image segmentation in detail, including its various types. We will discuss how deep learning has revolutionized image segmentation and enabled significant improvements in accuracy. We will also highlight some of the most commonly used deep learning models for image segmentation.

1.2 Deep Learning

1.2.1 Definition

Deep learning is a subset of machine learning that utilizes artificial neural networks to process vast amounts of data. These networks are structured to mimic the complex and interconnected nature of the human brain. As new information is introduced, the neural connections between the networks nodes can adapt and grow, allowing the system to learn and improve its decision-making capabilities independently without the need for human intervention. This ability to autonomously learn and improve with experience makes deep learning a powerful tool for solving complex problems, improving performance, and making accurate predictions [2].

1.2.2 History

Year	Contributor	Contribution
300 BC	Aristotle	Introduction of associationism, beginning of the history of humans trying to understand the brain.
1873	Alexander Bain	Introduction of neural groupings as the first models of neural networks.
1943	McCulloch and Pitts	Introduction of the McCulloch-Pitts (MCP) model considered the ancestor of artificial neural networks.
1949	Donald Hebb	Considered the father of neural networks, he introduced Hebb's learning rule which would serve as the foundation for modern neural networks.
1958	Frank Rosenblatt	Introduction of the first perceptron.

1974	Paul Werbos	Introduction of back propagation.
1980	Teuvo Kohonen	Introduction of self-organizing maps.
1980	Kunihiko Fukushima	Introduction of the neocognitron, which inspired convolutional neural networks.
1982	John Hopfield	Introduction of hopfield networks.
1985	Hilton and Sejnowski	Introduction of boltzmann machines.
1986	Paul Smolensky	Introduction of harmonium, later known as restricted boltzmann machines.
1986	Michael I. Jordan	Definition and introduction of recurrent neural networks.
1990	Yann LeCun	Introduced LeNet and demonstrated the capabilities of deep neural networks.
1997	Schuster and Paliwal	Introduction of bidirectional recurrent neural networks.
1997	Hochreiter and Schmidhuber	Introduction of LSTM, which solved the vanishing gradient problem in recurrent neural networks.
2006	Geoffrey Hinton	Introduction of the deep belief network.
2009	Salakhutdinov and Hinton	Introduction of the deep boltzmann machines.
2012	Alex Krizhevsky	Introduction of AlexNet which won the ImageNet challenge.

Table 1-1: The major stages of deep learning [3].

1.2.3 The key differences between deep learning and machine learning approaches

Deep learning is a subfield of machine learning that utilizes neural networks (NN) consisting of multiple deep layers to process data. As the data flows through the neural network, it gradually learns more complex and abstract features by building upon simpler

features from the preceding layers. This is achieved by adjusting the weights of each link between neurons and applying real-number activations to each neuron, from the lower to the higher layers [4].

Machine learning attempts to extract new knowledge from diverse datasets that have been preprocessed and loaded into the system. In this approach, rules are formulated by the users and then fed into the system to guide the learning process. Occasionally, users may need to intervene ,treat and correct errors that the system has made during the learning process. This method requires significant user input and guidance to achieve accurate results [4].

Table (1.2) and figure (1.2) below provide a comparison of the characteristics of deep learning and machine learning techniques.

Machine Learning	**Deep Learning**
The application of artificial intelligence (AI) enables a system to learn and improve autonomously from experience.	Deep learning is an application of machine learning that utilizes advanced algorithms and deep neural networks to train models.
Huge data amounts.	Short datasets, as long as they are of good quality.
Computation-heavy.	Not always.
The ability to draw precise conclusions from raw data.	The accuracy of pre-processed data
Long training times can be a significant challenge for some machine learning models.	Can take a reduced time to train.
The inability to identify and interpret the specific features that individual neurons represent.	The clarity of the logic behind the decision of machine.
The ability to be applied in novel and unexpected ways.	Can be tailored to address specific problems.

Table 1-2: Comparison between ML and DL [4].

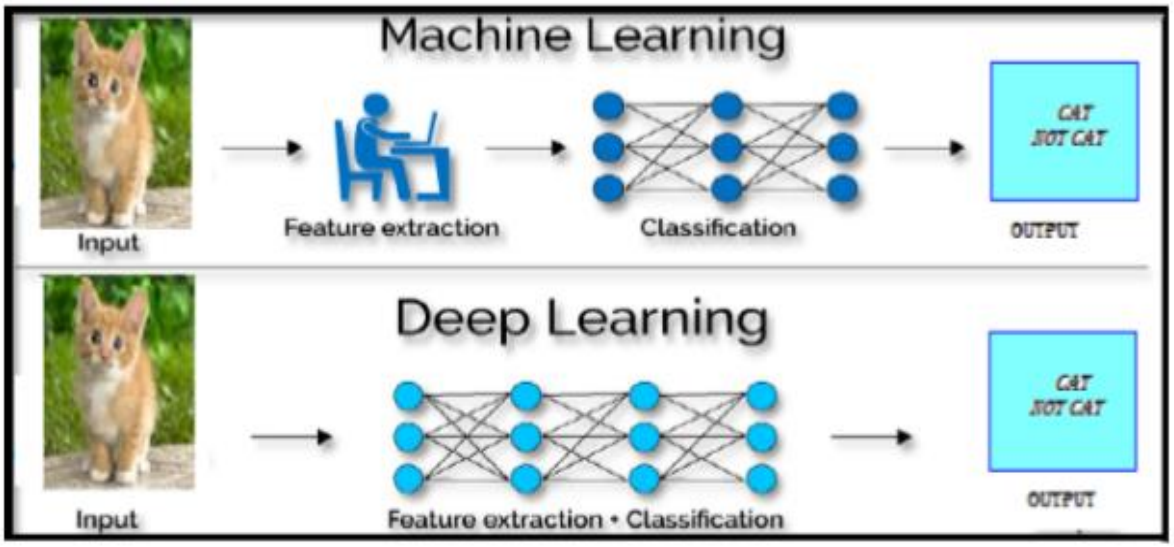


Figure 1.2: Deep Learning vs Machine Learning [5].

1.2.4 The process of deep learning

Deep learning, also known as artificial neural networks or neural networks, aims to replicate the human brain by using a combination of data inputs, weights, and biases, as illustrated in figure (1.3). By working together, these components enable accurate identification, classification, and description of objects within the data [6].

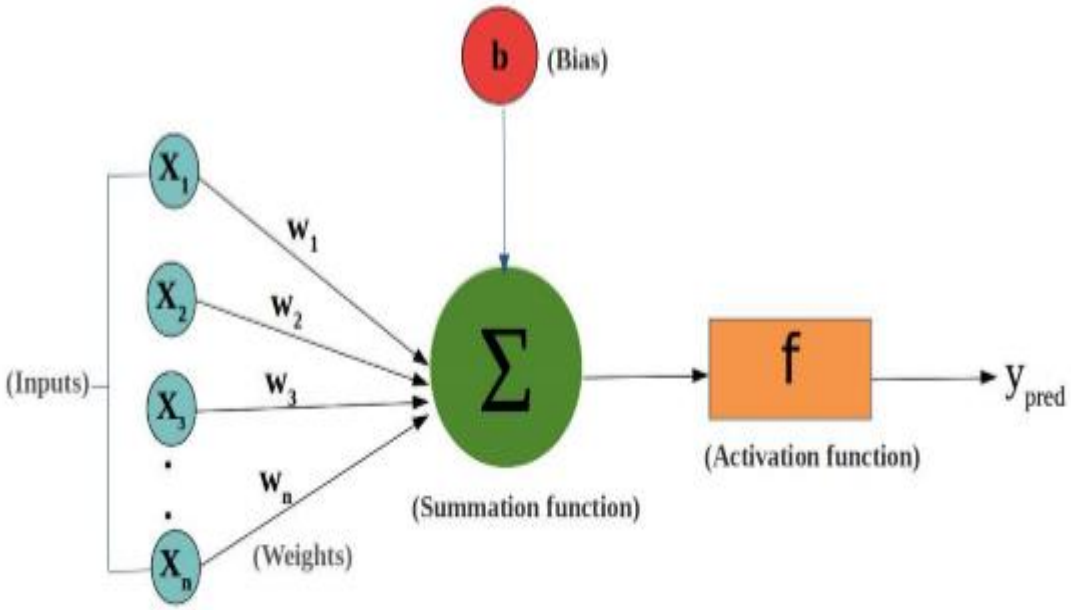


Figure 1.3: Inputs, weights and bias in a neural network [6].

Deep neural networks are composed of multiple interconnected layers of nodes, with each layer building upon the previous one to improve prediction accuracy or categorization. This process of passing information forward through the network is known as forward propagation. The input and output layers of a deep neural network are referred to as hidden layers. The input layer is where the model takes in data for processing, and the output layer is where the final prediction or classification is made, as illustrated in figure (1.4) [6].

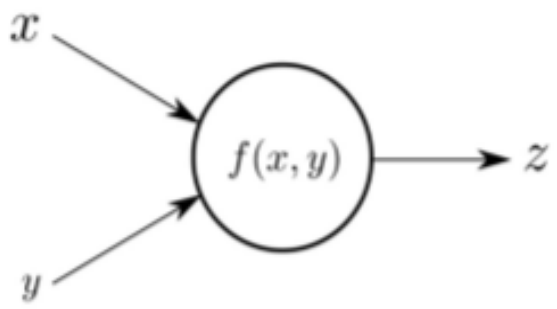


Figure 1.4 : Forward propagation process [6].

In addition to forward propagation, deep neural networks also utilize a process called backpropagation to adjust the weights and biases of the function by moving backwards through the layers and calculating errors in predictions. This is typically accomplished using algorithms like gradient descent. Figure (1.5) provides an illustration of this process. By leveraging both forward propagation and backpropagation, neural networks are able to make predictions and refine their accuracy over time, leading to improve performance on a range of tasks [6].

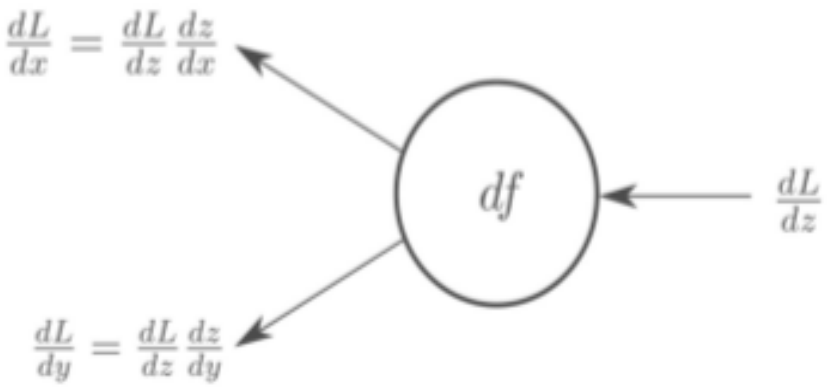


Figure 1.5: Backpropagation process [6].

1.2.5 The advantages and disadvantages of deep learning

1.2.5.1 The strength points

Deep learning has emerged as a powerful tool for processing complex data and performing tasks that considered impossible for machines to perform. In this section, we will discuss the strengths and advantages of deep learning.

- **Better results than with other machine learning methods**

Deep learning primary advantage lies in the quality of the outcomes it produces, particularly in areas like image recognition and processing, where it outperforms other forms of artificial intelligence [7].

- **Efficient execution of routine tasks without quality deviations**

Due to its reliance on continuous learning without signs of fatigue and consistent high-quality results, deep learning proves to be more efficient and faster compared to other traditional methods. Additionally, since the system is capable of self-training, it saves time and money while enhancing its functionality [7].

- **Processing unstructured data**

Furthermore, deep learning sets itself apart from other AI models in its ability to analyze unstructured data such as emails, photos, and documents, unlike those that only work with structured data such as phone numbers and addresses. This makes it more versatile and potentially more effective in handling a wide range of data types [7].

1.2.5.2 The limitations

- **Deep learning requires a lot of computing power**

Huge demand for computing power.On the one hand, it is necessary to ensure the maintenance of the artificial neural network, but also to deal with a very large amount of data that needs to be processed [7].

- **Expensive technical implementation**

The required computing power is directly proportional to the complexity and size of the data, making deep learning an expensive technology, mainly used by big data giants and in research [7].

- **Difficult or incomprehensible decisions**

Another issue with deep learning is the vast amount and complexity of data required for its operation, which makes it challenging to comprehend the rationale behind its decisions. Consequently, it cannot be integrated into applications that necessitate traceability, at least for now [7].

- **Requires a large database**

Deep learning heavily relies on large datasets to be effective. Despite the availability of libraries for artificial neural networks, there are still limitations to establishing deep intelligence, including the considerable time needed to develop the learning algorithms [7].

1.2.6 Deep learning applications

Deep learning has shown remarkable results in a wide range of fields, as we will discuss below [8] [9]:

- Facial recognition.
- Autonomous cars.
- Voice search and voice-activated assistants.
- Automatically add sounds to silent movies.
- Automatic translation.
- Image recognition.
- Automatic image description.
- Automatic colorization.
- Text sentiment analysis.
- Marketing research.

- Medicaldiagnosis (Detecting Developmental Delay In Children, Brain Cancer Detection).

Figure (1.6) shows other application areas for deep learning.

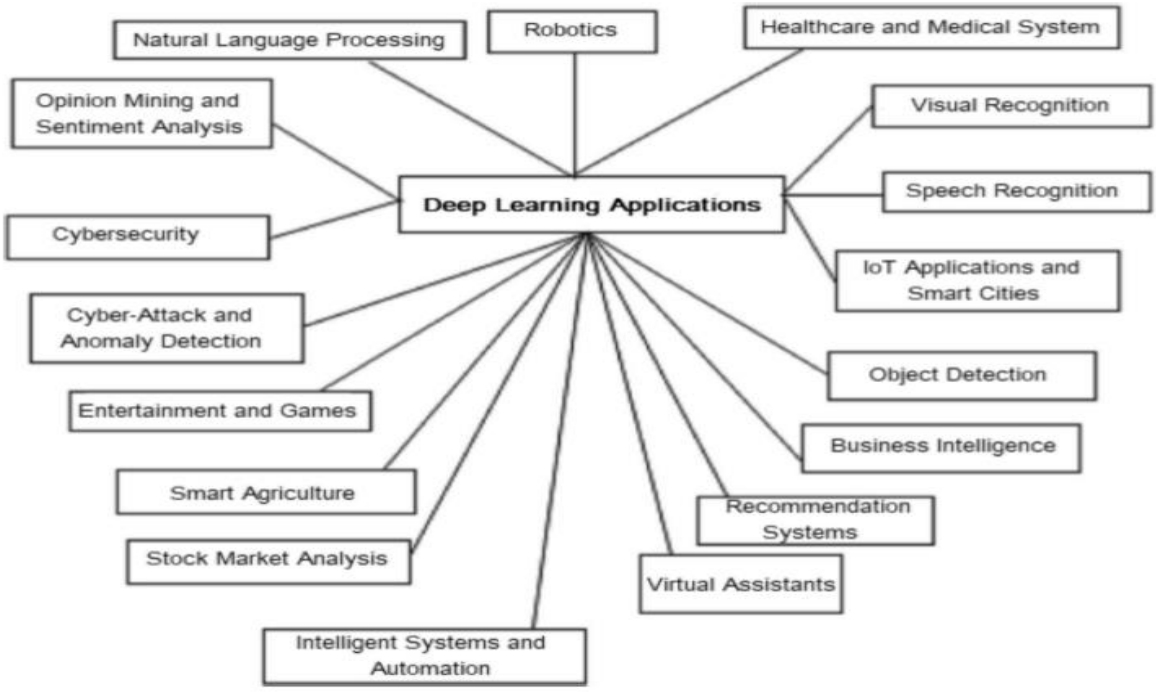


Figure 1.6: Deep learning applications [10].

1.2.7 Deep learning architectures

Deep learning is a vast field with a multitude of architectures and algorithms. The fundamental architecture underlying deep learning is the artificial neural network (ANN), which has undergone numerous modifications and variations over the years. In this section, we will explore some of the deep learning architectures that have emerged in recent times. These architectures are classified based on their learning methods, namely supervised and unsupervised learning [11].

1.2.7.1 Unsupervised deep learning

Unsupervised learning is a branch of deep learning that employs learning algorithms to analyze and group datasets that lack labels. In contrast to supervised learning, unsupervised learning algorithms work independently to discover the inherent structure of unlabeled data. This allows them to identify hidden patterns in data without human intervention. Some of the popular unsupervised learning models include [11]:

- Self-Organizing Maps.
- Autoencoders.

1.2.7.2 Supervised deep learning

Supervised learning is a method of learning that involves labeled datasets to train algorithms for classification or prediction tasks. These models rely on human intervention to label data and achieve higher accuracy rates than unsupervised learning. Some popular supervised learning models include [11]:

- Recurrent Neural Networks (RNN).
- Convolutional Neural Networks (CNN).

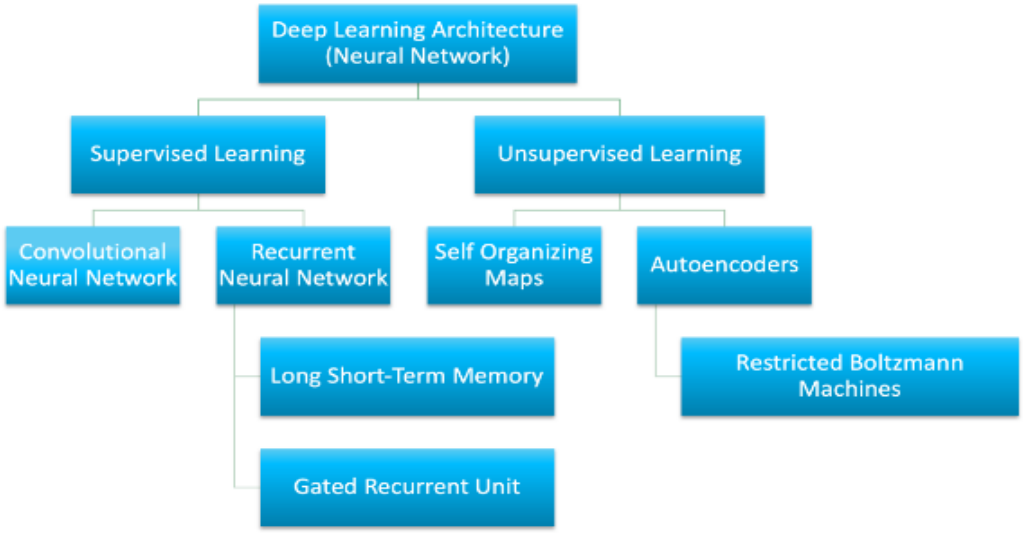


Figure 1.7: Different deep learning architectures [12].

1.3 Convolutional Neural Network (CNN)

1.3.1 Definition

A specialized deep learning algorithm, convolutional neural network (CNN) excels in tasks related to image processing and recognition. Comprising multiple layers, a CNN consists of convolutional, pooling, and fully connected layers [13].

The crucial component of a CNN is the convolutional layer, where filters are applied to the input image, extracting features such as shapes, textures, and edges. The feature maps produced by the convolutional layers are then passed through pooling layers, downsampling the spatial dimensions, while retaining the most relevant information. Finally, one or more

fully connected layers classify the image by making a prediction based on the features extracted [13].

1.3.2 Architecture

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

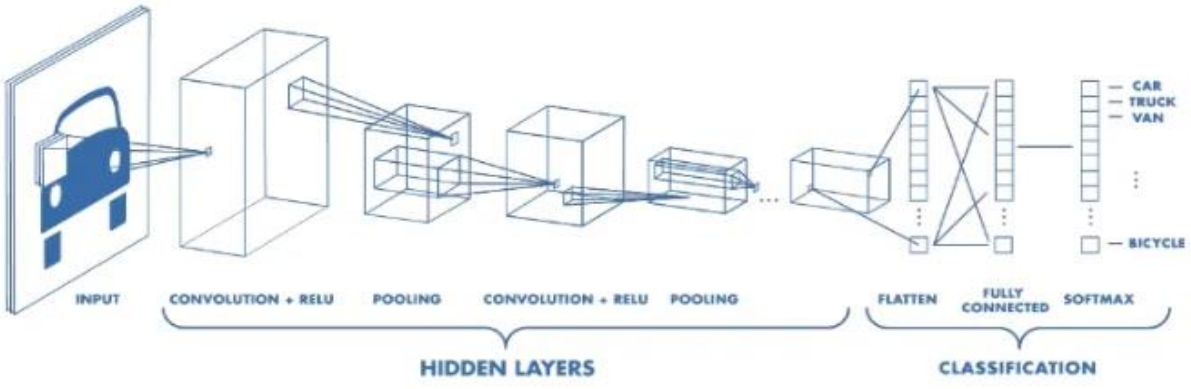


Figure 1.8: Architecture of a CNN [14].

1.3.2.1 Convolutional layer

In CNN, the convolutional layer plays a vital role in extracting relevant information from image data. The layer consists of learnable filters, which are essentially 2D arrays of weights. These filters slide over the input image to identify the presence of certain features such as edges or shapes. Convolution is applied to the image using the filter, which involves multiplying the input pixel values with the corresponding filter weights and summing up the results. This process is repeated with a stride, which is the number of pixels by which the filter shifts. The resulting dot products create a feature map or activation map, which contains the feature information extracted from the image using the filter. This process is repeated with multiple filters to extract different features from the input image. Finally, the feature maps are passed through fully connected layers for classification or prediction [15].

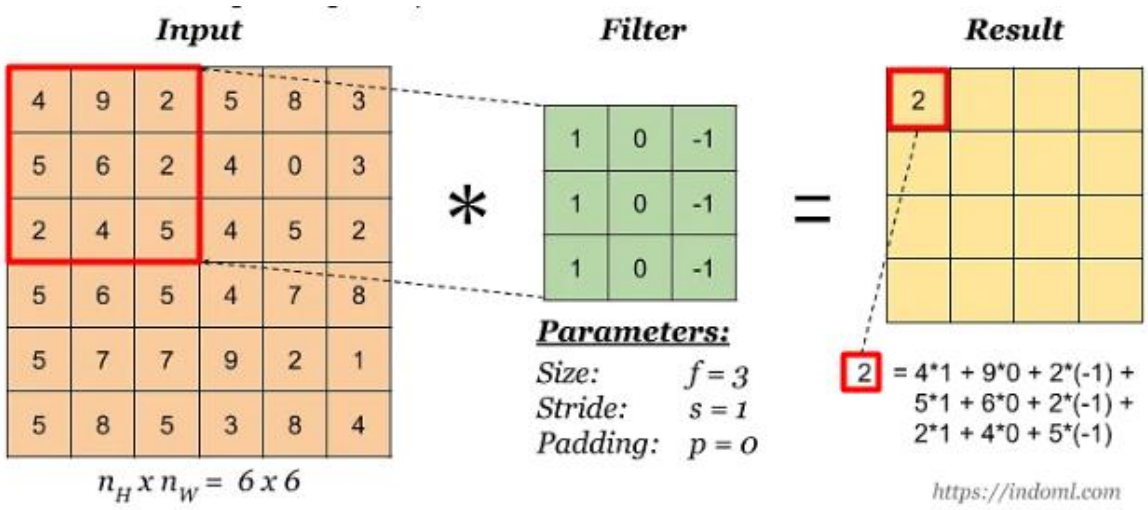


Figure 1.9 : The convolution operation [15].

As depicted in the image above, a filter or kernel of size 3x3 is applied to a sub-matrix of the same size from the image, and the resulting dot product is stored in the output matrix. The process is repeated by shifting the filter by 1 pixel, as demonstrated in the example using a stride of 1 [15].

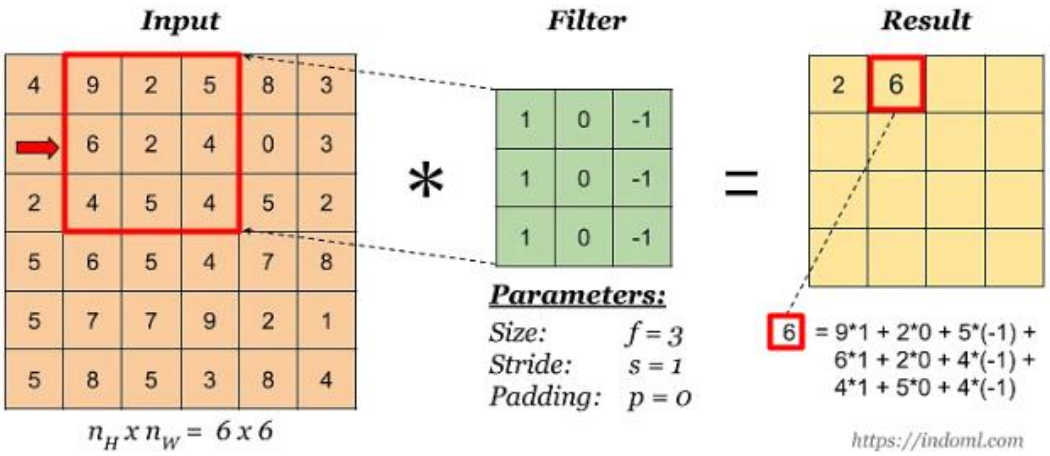


Figure 1.10 : The convolution operation [15].

The convolution operation is performed by moving the filter across the entire matrix to generate a feature map. The stride is an integer that determines the number of pixels by which the filter moves across the input matrix. A larger stride value results in a smaller output matrix. In some cases, the filter may not entirely fit the input image matrix, resulting in parts of the filter extending beyond the image as it moves across [15]. For example :

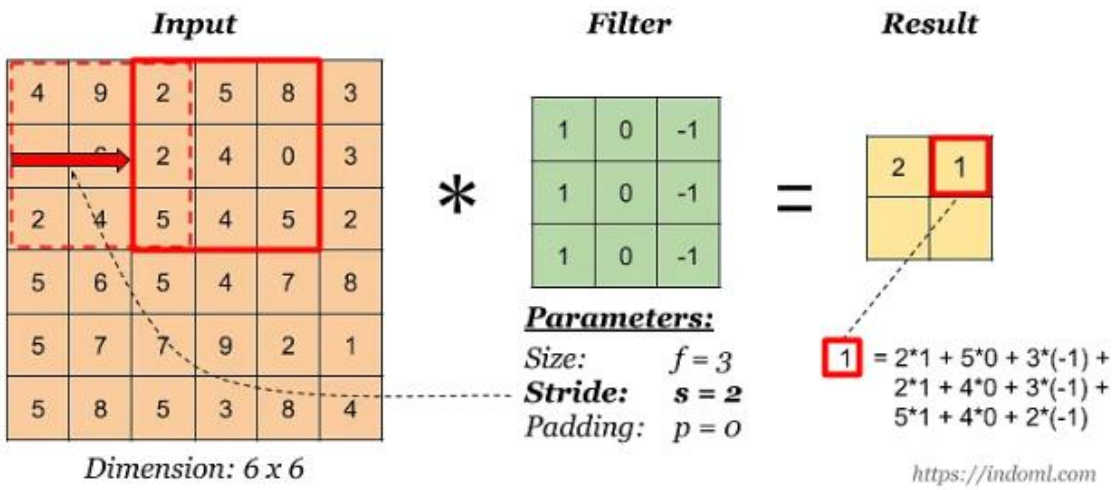


Figure 1.11 : The convolution operation [15].

In the above example, the stride of the filter is set to two. When the filter moves two steps forward in the next iteration, the last column of the filter will be out of the image pixels range. This issue can be resolved by using zero padding. Adding zero value elements around the input matrix ensures that the filter always has valid input pixels to interact with [15]. There are three types of padding :

- **Valid padding:** In this case, the last convolution is dropped/skipped if the filter is falling outside the input matrix [15].
- **Same padding:** In this case, the padding ensures that the input layer size is the same as the output layer [15].
- **Full padding:** As the name suggests, a border of zeros is added to the input matrix, increasing the output size [15].

Padding also plays a crucial role in maintaining the height and width of the feature map during the convolution operation. Without padding, the dimensions of the feature maps may decrease drastically due to filters, especially in deeper CNNs. After convolution, the rectified linear unit (ReLU) activation function is applied to the feature map to introduce non-linearity [15].

1.3.2.2 Pooling layer

In CNNs, pooling layers perform dimensionality reduction on the input data by utilizing a filter, but instead of using weights, they perform aggregation on the input pixels. The output of the pooling layers has a smaller size than the input, leading to the loss of some data, but this operation results in reduced feature complexity, making it less prone to overfitting, faster

computation, and better efficiency of the CNN. Max pooling and average pooling are the two types of pooling layers [15]:

- **Max pooling:** The input pixel with maximum value is saved in the output matrix [15].
- **Average pooling:** The average of input pixels is saved in the output matrix [15].

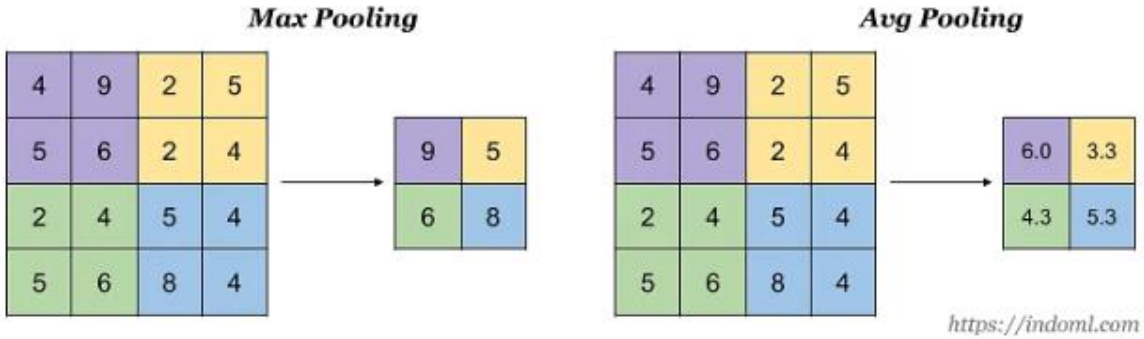


Figure 1.12: Example of max pooling and average pooling layers [15].

1.3.2.3 Fully connected layer

The convolutional layer is not fully connected as it only connects to a subset of pixels in the input image where the filter is applied. On the other hand, a fully connected layer is a type of layer where each node connects to all the previous nodes. The output of the convolutional and pooling layers are three-dimensional feature maps. These feature maps can be flattened to a one-dimensional vector and passed to the fully connected layer(s) for the final classification task. The softmax activation function is applied in this layer to predict the probabilities of classes for the final prediction [15].

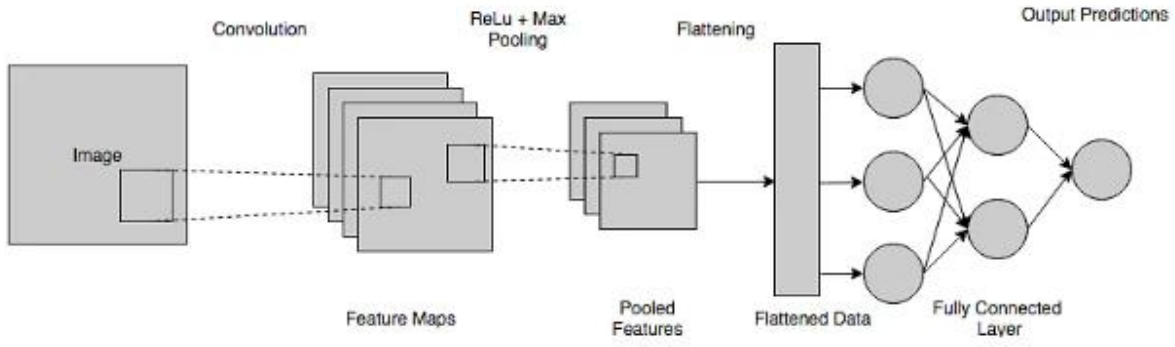


Figure 1.13: Fully connected layer [15].

1.3.3 Some famous CNN architectures

- **LeNet:** In the 1990s, Yann LeCun developed convolutional networks and successfully applied them to various tasks. The LeNet architecture, which is primarily used for reading postal codes and digits, is one of the most well-known applications of this technology [3].
- **AlexNet:** Developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, is credited with popularizing convolutional networks in computer vision. In 2012, the network was submitted to the ImageNet ILSVRC challenge and surpassed its competitors by a significant margin. Although it shared some architectural similarities with LeNet, AlexNet was deeper, larger, and utilized stacked convolutional layers (which was uncommon at the time, as it was typical to have only one convolutional layer followed by a pooling layer) [3].
- **ZFnet:** Matthew Zeiler and Rob Fergus won the ILSVRC challenge in 2013 with a convolutional network that they named ZFNet (short for Zeiler and Fergus Net). This network was an improvement over AlexNet and had its hyperparameters, such as the size of convolutional layers and kernel size, adjusted for better performance [3].
- **GoogLeNet:** In 2014, the ILSVRC challenge was won by a convolutional network created by Szegedy and his team at google. The key feature of this network was the inception module, which reduced the number of parameters required (4 million as compared to AlexNet's 60 million). Additionally, the module used global average pooling instead of the traditional fully connected layer, further reducing the number of parameters. The GoogLeNet architecture has since undergone several iterations, including Inception-v4 [3].
- **ResNet:** The winner of the ILSVRC 2015 was the Residual Network developed by Kaiming He et al. It utilizes skip connections and heavily employs batch normalization. Additionally, instead of the traditional global max pooling at the end of the network, it uses global average pooling [3].

1.4 Image Segmentation

1.4.1 Definition

Segmentation of an image involves partitioning it into regions that have similar properties, such as color, texture, or gray level. The objective is to isolate and distinguish

different visual components in the image. The ultimate goal of segmentation is to transform the image into a more meaningful and analyzable representation by simplifying or altering its original representation [16].

To achieve segmentation, certain guidelines should be followed as stated in [17]:

- The areas that are segmented should be uniform and homogeneous in terms of specific attributes such as gray level, standard deviation, and gradient.
- The internal parts of the segmented regions should be uncomplicated and free of small gaps.
- The neighboring regions must display significant differences in values concerning the feature being used for segmentation.
- The borders of each segmented area should be straightforward and accurately defined.

Formally, the segmentation of an image A into regions Ri, i = 1..n , is defined by the following properties [18] :

1. $U_{i=1}^{n} R_i = I$
2. $R_i \cap R_j = \emptyset \; ; \forall \, i,j \; such \; as \; i \neq j$
3. $P(R_i) = true; \forall \, i \in \{1, 2, \dots \dots, n\}$
4. $P(R_i \cap R_j) = false; \; \forall \, i,j \; such \; as \; i \neq j$
5. $R_i \; is \; connected \; component; \; \forall \, i \in \{1, 2, \dots \dots, n\}$

Équation 1 : Segmentation.

- P is a homogeneity predicate.
- The first condition indicates that the union of the regions brings us back to the starting image.
- The second indicates that two different regions are disjoint.
- The third expresses that the pixels belonging to a region must satisfy the homogeneity criterion.
- The fourth expresses the homogeneity criterion for segmentation into disjoint regions.

1.4.2 Semantic segmentation

1.4.2.1 Definition

Semantic segmentation is a type of deep learning algorithm that assigns a label or category to every pixel in an image, enabling the recognition of distinct categories of pixels. For instance, to navigate a road environment, an autonomous vehicle must identify vehicles, pedestrians, traffic signs, sidewalks, and other objects (see figure 1.14). This is achieved by the following steps [19]:

- Analyzing a set of images that have labeled pixels.
- Developing a semantic segmentation network.
- Training the network to classify images based on pixel categories.
- Evaluating the accuracy of the network

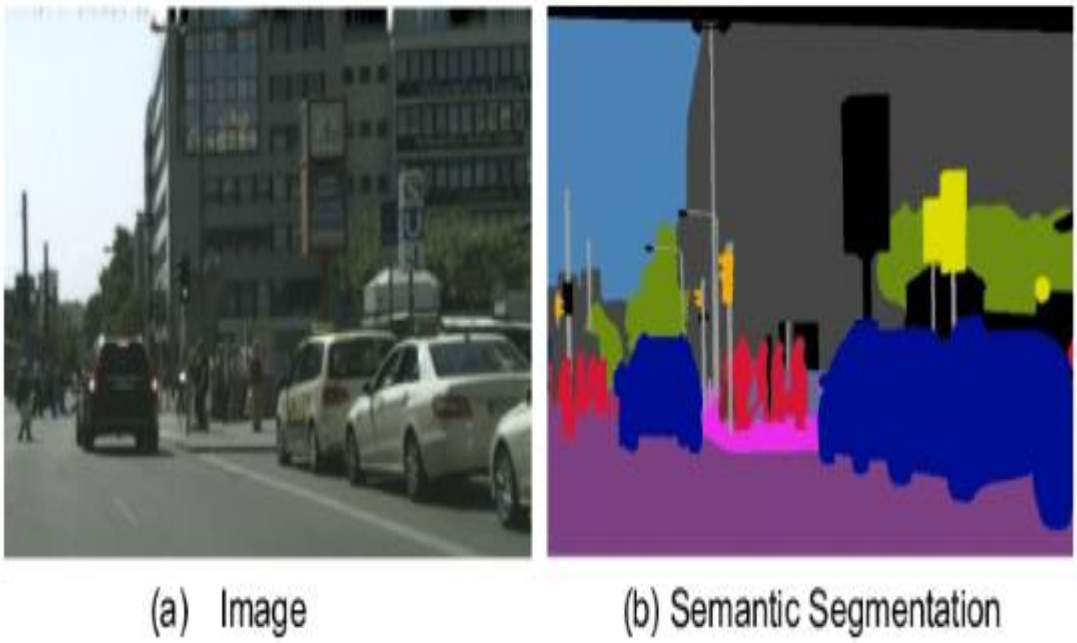


Figure 1.14 : Semantic segmentation [20].

1.4.2.2 Applications

Semantic segmentation labels the pixels of an image, which makes it useful in applications in various fields:

- **Autonomous driving**

The identification of a drivable path for vehicles by differentiating the road from obstacles such as pedestrians, curbs, poles, and other vehicles can be achieved through the use of semantic segmentation, as depicted in figure (1.15) [11].

Figure 1.15: Semantic segmentation for autonomous vehicles [11].

- **Medical image segmentation**

In medical imaging, semantic segmentation is utilized to detect significant features in scans, particularly to identify abnormalities such as tumors. The precision and recall rates of the algorithms are crucially important for such applications, as depicted in figure (1.16) [11].

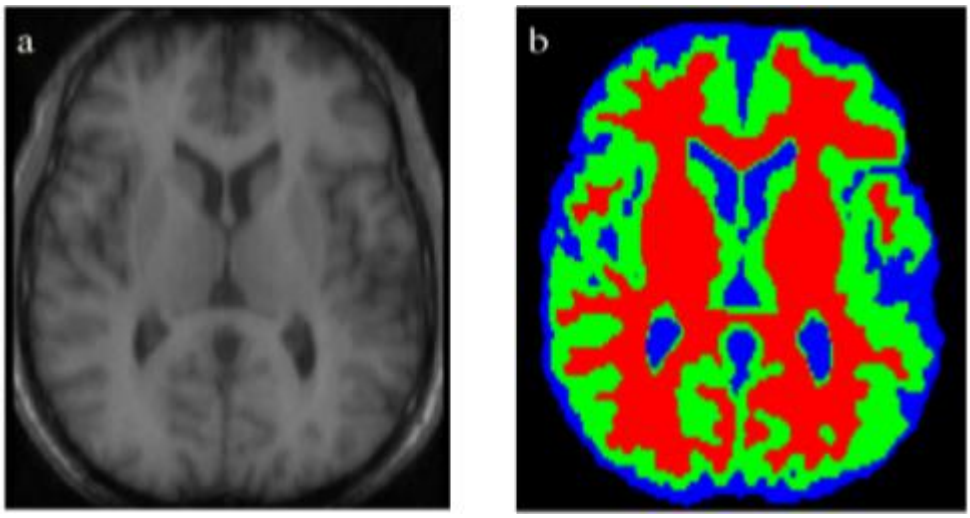


Figure 1.16: Segmentation of medical scans [11].

- **Scene understanding**

Semantic segmentation serves as a fundamental step for more advanced tasks like scene comprehension and visual question answering (VQA). The outcome of scene comprehension algorithms is generally a scene graph or a description of the image, as illustrated in figure (1.17) [11].

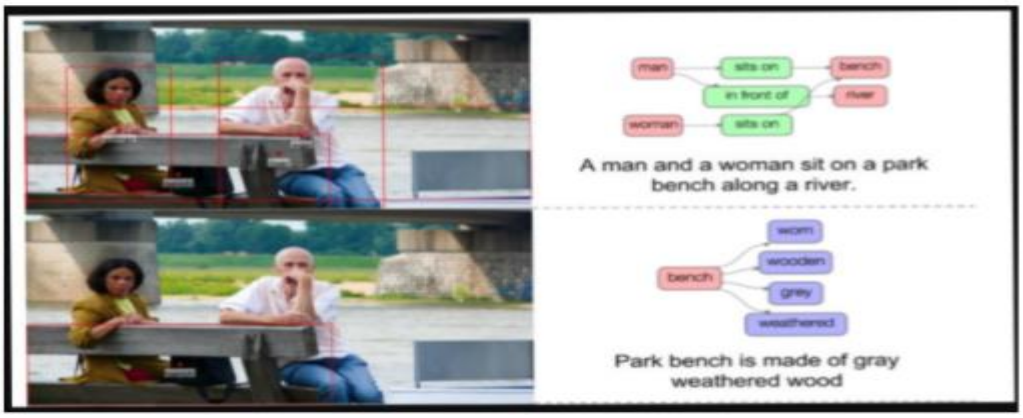


Figure 1.17: Scene understanding in action [11].

- **Fashion industry**

The fashion industry utilizes semantic segmentation to extract clothing items from images and recommend similar products from retail shops. Advanced algorithms can even manipulate specific clothing items within an image, allowing for virtual "re_dressing" capabilities, as depicted in figure (1.18) [11].

Figure 1.18 : Example of semantic segmentation used to redress a human based on text input [11].

- **Satellite and aerial drone imagery**

Semantic segmentation is used to identify mountains, rivers, deserts, roads and other terrain see figure (1.19) [21].

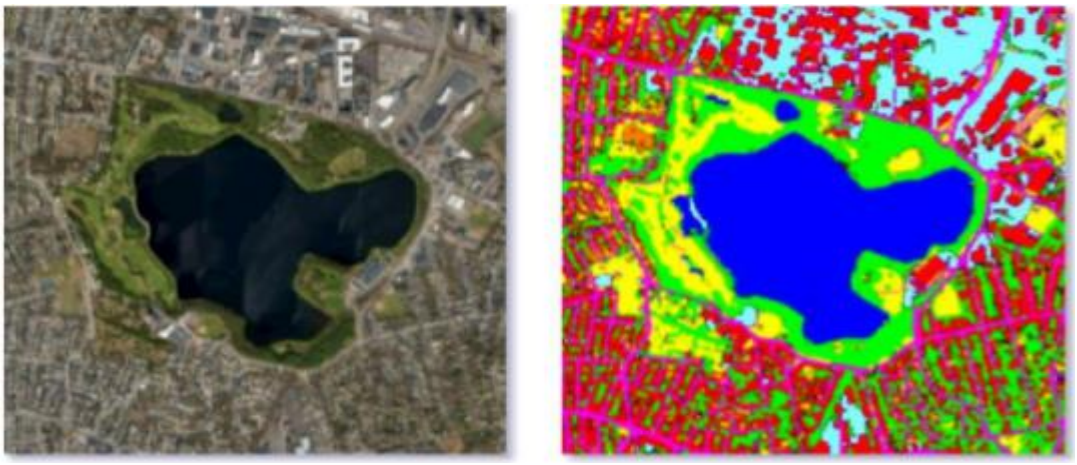


Figure 1.19 : Semantic segmentation of satellite/aerial images [11].

- **Industrial controls**

Semantic segmentation is used to detect defects in materials, such as the control of electronic components [21].

- **Robotic vision**

Semantic segmentation is used to identify objects and terrain and move around in them [21].

1.4.3 How does deep learning makes semantic segmentation more precise

Achieving accurate semantic segmentation is a challenging task in computer vision, even for experienced professionals in the field. However, various studies have demonstrated that deep learning techniques and models can be effective for this purpose. The theoretical and practical strategies for improving the quality and precision of semantic segmentation with deep learning are encouraging for the advancement of computer vision [22].

For achieving in-depth segmentation of images or videos, deep learning methods require a large amount of datasets. The use of deep learning models for semantic segmentation, like many other modern approaches, has resulted in achieving benchmark performances and being considered state of the art [19].

Deep learning and deep neural networks can be used to increase the accuracy of semantic segmentation [22].

1.5 Conclusion

In this chapter, we have provided an overview of the current state of deep learning, beginning with an introduction to the concept and the distinction between deep learning and machine learning. We have also discussed the advantages and disadvantages of this approach, as well as its architecture and how it works, including the architecture of convolutional neural networks. Additionally, we have covered the general idea of image processing, specifically focusing on segmentation and semantic segmentation, and how deep learning can improve the precision of semantic segmentation in various applications.

Moving forward, the next chapter will delve into the U-NET Architecture in more detail.

Chapter 02:U-NET Architecture

2.1 Introduction

There are various techniques available for addressing semantic segmentation challenges. Conventional methods involve identifying individual points, lines, or borders. Morphology can also be utilized, or alternatively, groups of pixels can be combined to solve the problem.

Nowadays, deep learning convolutional neural networks are extensively employed due to their ability to segment images, enabling more complex problems to be tackled. The U-Net model is one of the most commonly used neural networks for image segmentation. It is a fully convolutional neural network model and has demonstrated to be highly effective. Specifically, several studies have documented that U-Net based models attain remarkable performance in medical image segmentation.

In this chapter, we delve into the intricacies of the U-Net architecture, exploring its various versions, advantages, and applications. We begin by providing an overview of the original U-Net architecture and its key components. Furthermore, we explore the differents version of U-Net, its advantages, and the diverse domains where it excels.

Additionally, we investigate the broad range of applications where U-Net has demonstrated remarkable performance. From medical imaging tasks such as organ segmentation and tumor detection to computer vision applications like image restoration and object detection, U-Net has proven its versatility and efficacy across multiple domains.

2.2 What is U-Net

U-Net was designed by Olaf Ronneberger et al in 2015 at the university of Freiburg, Germany [23]. It is a convolutional neural network primarily developed for biomedical image segmentation. The network employs a fully convolutional architecture that has been adapted and expanded to function with a reduced number of training images while achieving more precise segmentation [24].

U-Net architecture is well-suited for semantic segmentation [25], especially for semantic segmentation challenges [26].

2.3 U-Net network architecture

The architecture of U-Net is characterized by a distinctive "U" shape, consisting of an encoder path on the left-hand side and a decoder path on the right-hand side [27]. The U-Net architecture features skip connections connecting corresponding layers in the encoder and

decoder paths. These connections enable the network to preserve low-level functionality for the ultimate prediction and enhance segmentation accuracy. Below is a more in-depth explanation of the structure:

2.3.1 Encoder network

The first section of the U-Net architecture is known as the contracting path, which utilizes a convolutional network architecture to recover an image context. This block is comprised of a combination of convolution layers and max pooling layers that capture the input image characteristics and reduce its size to decrease the number of network parameters. It involves the repetitive application of two 3x3 convolution layers, followed by a ReLU activation function and batch normalization. Next, a 2x2 max pooling operation is performed to gradually decrease the input image spatial resolution while simultaneously increasing the number of feature channels. This enables the extraction of increasingly abstract features from the input image, which is vital for precise segmentation [28].

2.3.2 Skip connections

In figure (2.1), the gray arrows illustrate the skip connections used to facilitate direct communication between the encoder and decoder layers [29]. These connections provide supplementary information that assists the decoder in producing more accurate semantic features. Additionally, they serve as shortcut connections that enable the indirect flow of gradients to earlier layers without any degradation.

In simpler terms, skip connections help to enhance the gradient flow during backpropagation, which aids the network in learning better representations [23].

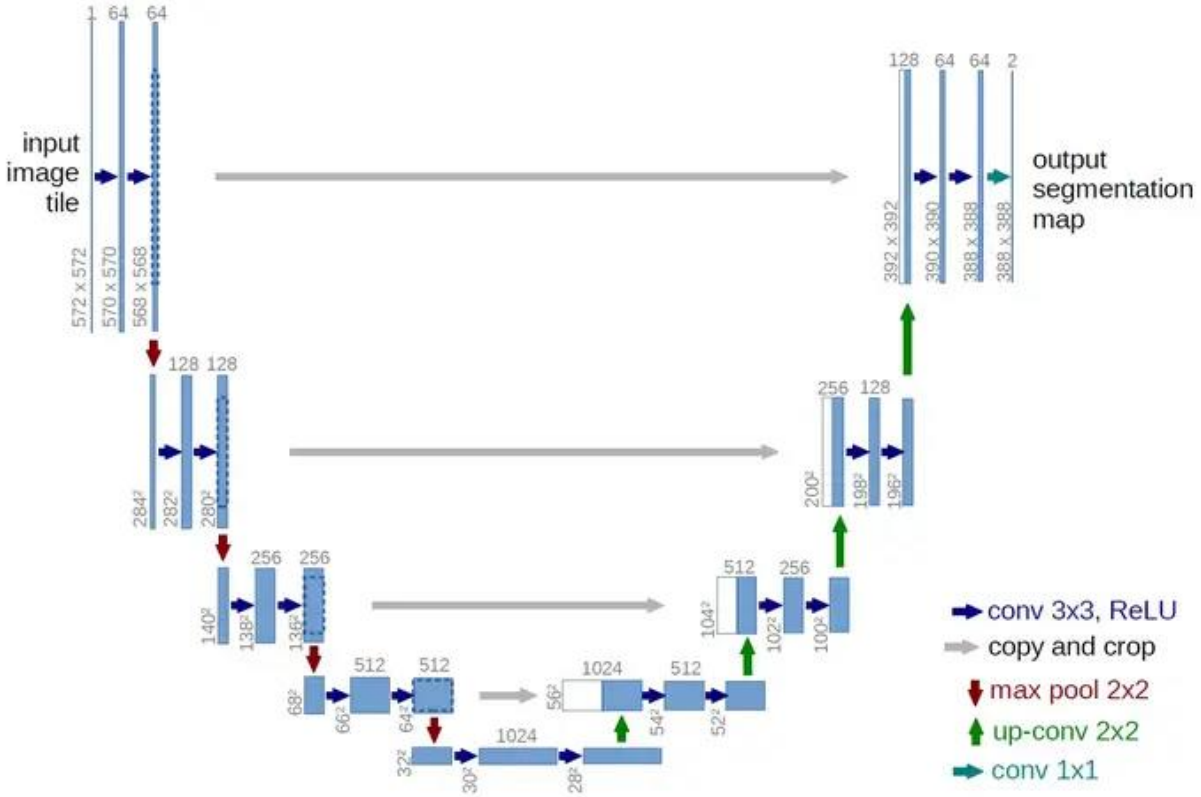


Figure 2.1: The U-Net network architecture [30].

2.3.3 Decoder network

The second section of the U-Net architecture is known as the expansive path, which processes the information received from the encoder and generates the final output. This section enables precise localization through transposed convolutions and allows for the determination of the input image initial size. The decoder block commences with an upsampling of the feature map, followed by a transposed 2x2 convolution layer. Next, two layers of 3x3 convolutions are employed, with each convolution being succeeded by a ReLU activation function. The output of the last decoder layer passes through a 1x1 convolution layer with a sigmoid activation function [28].

2.4 U-Net versions

U-Net is a well known convolutional neural network architecture that is widely used for the semantic segmentation of biomedical images. There exist several variations and versions of this network, including:

2.4.1 U-Net++

Another potent variation of the U-Net architecture, It utilises a dense grid of skip connections as an intermediary between the contracting and expansive paths. This facilitates

the network by transmitting additional semantic information between the two paths, which helps it to achieve more precise segmentation of images [31]. In the conventional U-Net architecture, the feature maps from the contracting path are concatenated directly into the corresponding layers in the expansive path. However, U-Net++, represented in figure (2.2), has multiple skip connection nodes between every corresponding layer. Each skip connection unit receives all feature maps from all previous units at the same level and an upsampled feature map from its immediate lower unit, resulting in each level being equivalent to a dense block. This configuration reduces the loss of semantic information between the two paths [31].

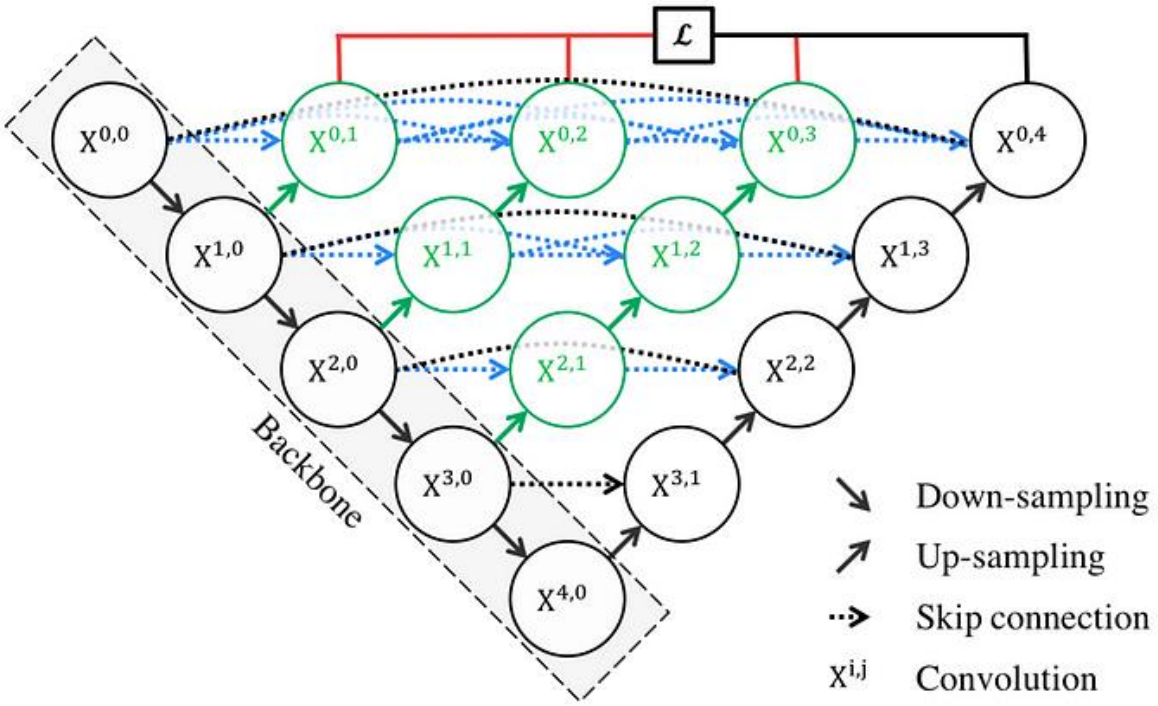


Figure 2.2: U-Net++ schematic representation [32].

2.4.2 U-Net 3+

U-Net++ was created as an improvement to U-Net by incorporating nested and dense skip connections. However, it did not effectively convey information from multiple scales and resulted in a complex and parameter-heavy network. To address these issues, Huang et al proposed U-Net 3+, which utilizes full scale skip connections and deep supervisions. By combining high and low-level semantics from feature maps of various scales, full scale jump connections can refine the results and improve accuracy. Deep supervision learns hierarchical representations from feature maps aggregated at multiple scales, and a hybrid loss function is

used to further refine the results, making it particularly effective for resolving organs of different sizes. Additionally, U-Net 3+ reduces network parameters with fewer channels compared to U-Net and U-Net++ [33].

Figure (2.3) illustrates the network structure of U-Net 3+.

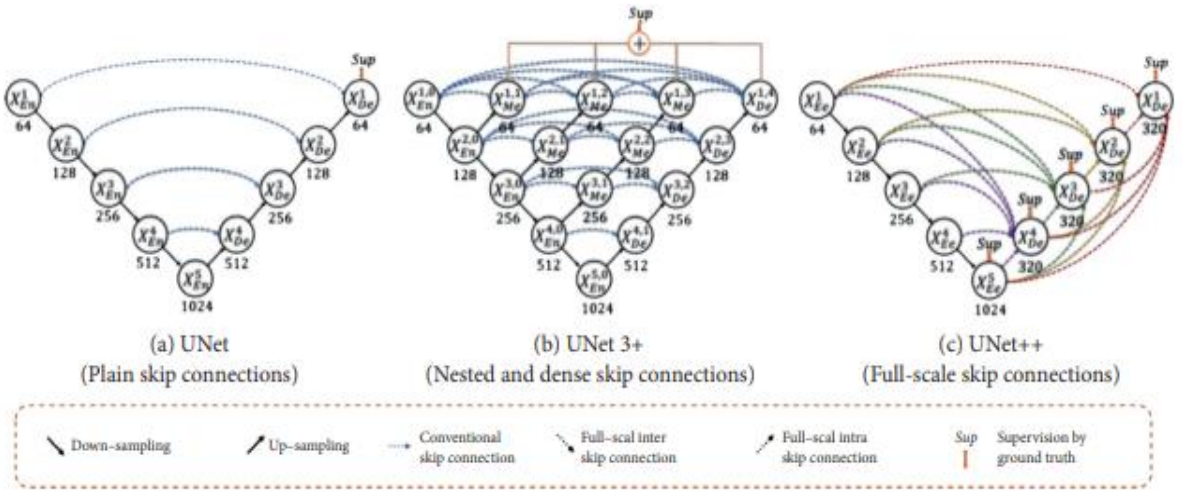


Figure 2.3: A graphic overview of U-Net, U-Net++, and U-Net 3+ [33].

2.4.3 Attention U-Net

The attention U-Net is designed to focus on specific objects of interest while disregarding irrelevant regions, which is a desirable characteristic in image processing networks. To accomplish this, the attention U-Net employs attention gates that discard irrelevant features. In this approach, each layer in the expansive path is equipped with an attention gate that filters out irrelevant features from the corresponding contracting path before concatenating the features with the upsampled features in the expansive path. This technique significantly enhances segmentation performance and does not add excessive computational complexity to the model, thanks to the repeated use of attention gates after each layer [31].

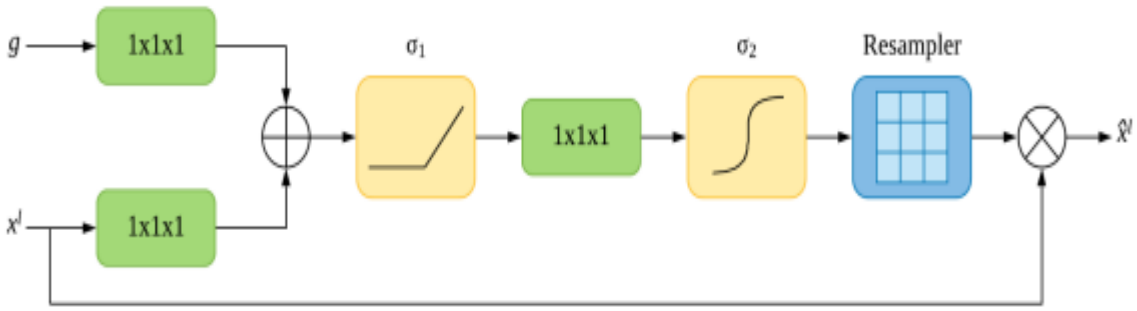


Figure 2.4: Additive attention gate schematic [31].

The attention gate in the network processes two signals, namely the input signal $x^l$and the gating signal g, both of which are passed through individual 1x1x1 convolutions. The outputs are then added and subjected to a series of linear transformations, which include a ReLU activation (σ1), a 1x1x1 convolution, a sigmoid activation (σ2), and an optional grid resampler. The original input is then concatenated to the output obtained from either the sigmoid unit or the resampler, as shown in figure (2.4) [31].

2.4.4 ResU-Net

ResU-Net is an innovative approach that leverages the performance improvement achieved by residual networks and incorporates it into the U-Net architecture to enhance the segmentation accuracy [34]. The figure (2.5) below is the architecture of ResU-Net.

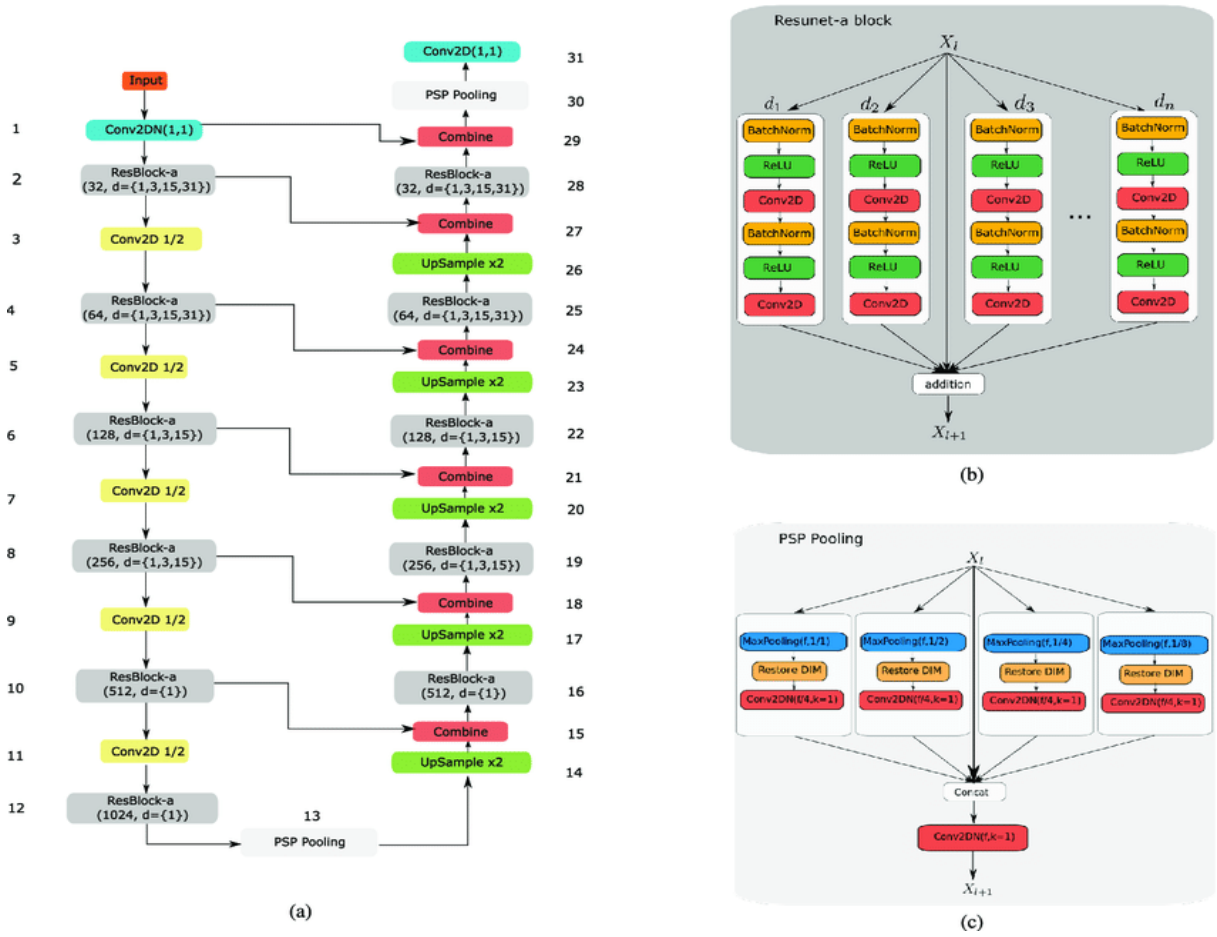


Figure 2.5: ResU-Net architecture [34].

2.4.5 DenseNet

The architecture of DenseNet involves two significant modifications to ResNet, a deep learning architecture. Firstly, each layer in a block receives the feature or identity map from all preceding layers. Secondly, instead of element wise addition, the identity maps are concatenated along the channel dimension into tensors. This means that the identity mapping of each layer depends not only on the previous layer but on all layers before it in the block, as illustrated in figure (2.6). By doing so, DenseNet can preserve all identity maps from prior layers and promote gradient propagation, resulting in higher accuracy with fewer computations. This allows for the creation of deeper models while reducing the number of channels in each layer [31].

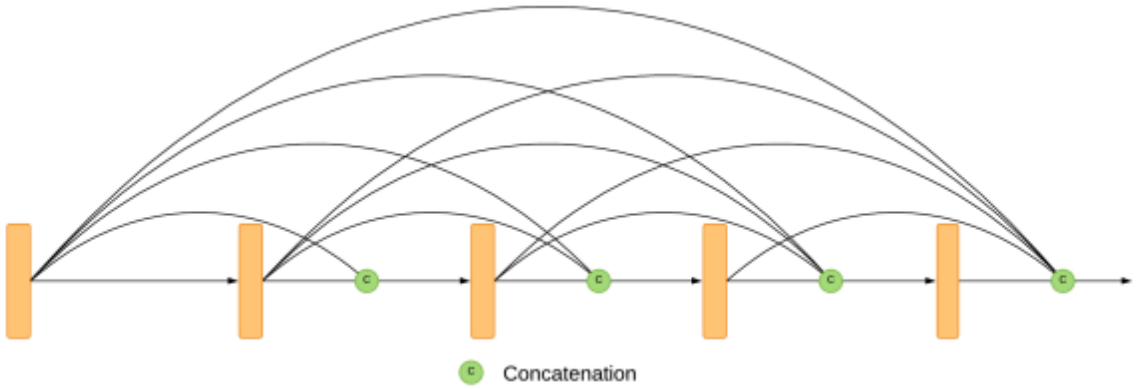


Figure 2.6: DenseNet architecture (A five-layer dense block) [31].

2.4.6 3D U-Net

The 3D U-Net is an extension of the original U-Net architecture that allows for volumetric segmentation in three dimensions. While still incorporating a contracting and expansive path, all 2D operations in the basic framework are replaced with 3D operations such as 3D convolutions, 3D max pooling, and 3D up-convolutions. As a result, the network can produce a 3D segmented image. This approach can achieve accurate segmentation results using minimal annotated examples, as 3D images often contain many repeated structures and shapes that allow for faster training with limited labeled data [31].

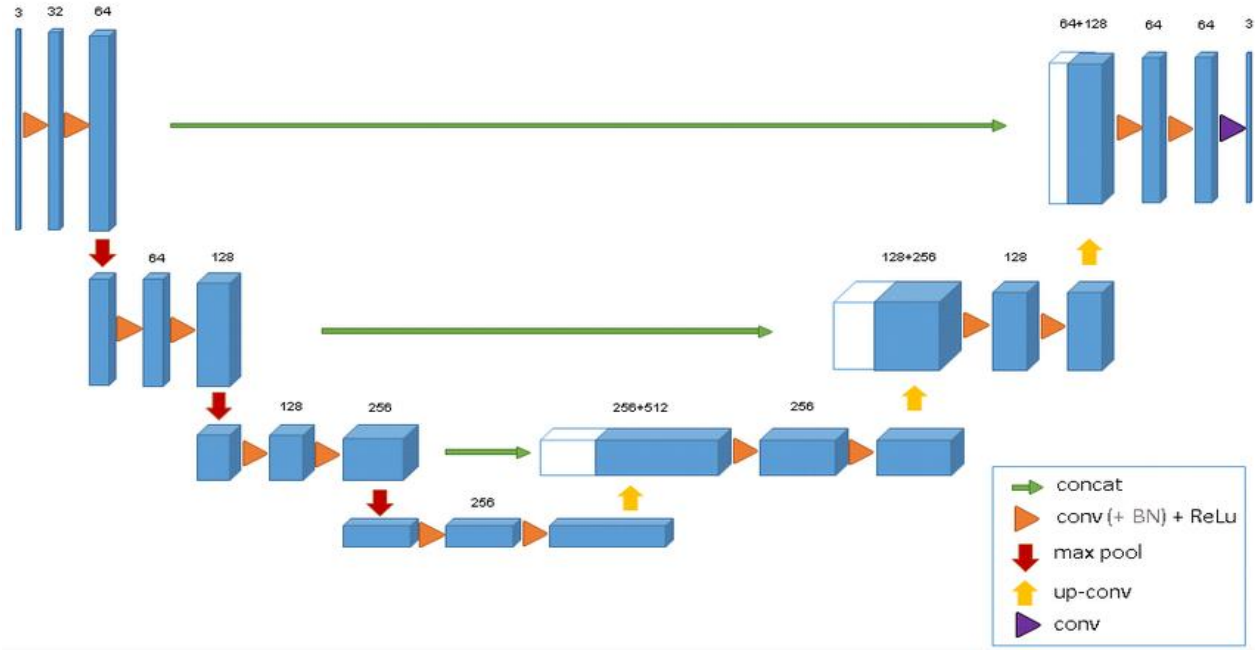


Figure 2.7: 3D U-Net architecture [28].

2.4.7 V-Net

The V-Net was originally designed for 3D or volumetric data, but it can still be utilized for 2D images. The key difference between U-Net and V-Net is that V-Net replaces the up-sampling and down-sampling pooling layers with a convolutional layer. The underlying concept behind the V-Net architecture is that using maxpool operations can result in significant loss of information. Thus, replacing it with a series of convolutional operations without padding can help in preserving more information during the segmentation process [34].

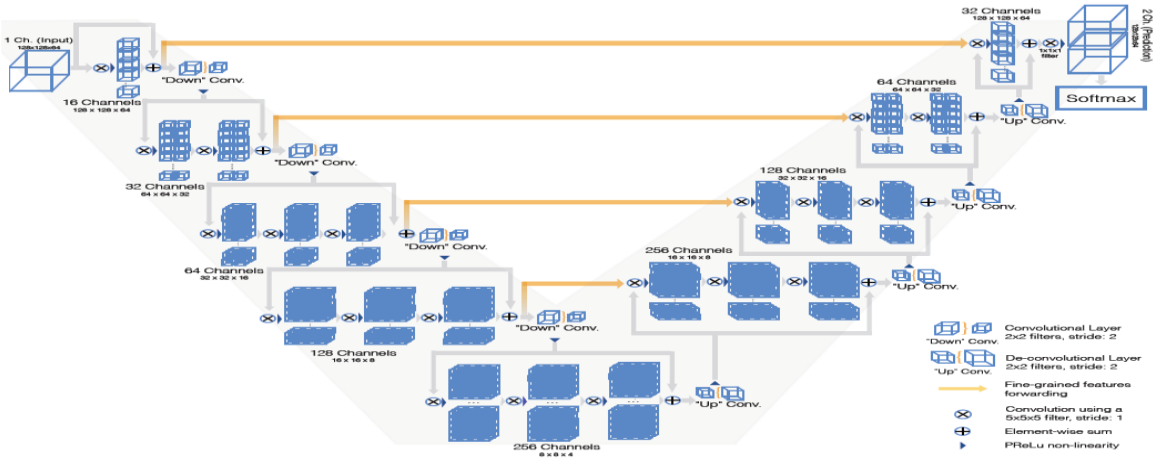


Figure 2.8: Schematic representation of V-Net architecture [28].

2.4.8 Recurrent U-Net

Recurrent U-Net has also been used for image segmentation tasks with sequential data, where each image is processed as a time step. In this case, the feedback loop provides the network with the ability to utilize temporal information between frames to improve segmentation accuracy. The recurrent connections also allow the network to learn longer-term dependencies between images, leading to more accurate segmentation results. However, the use of recurrent connections can increase the complexity of the model and require longer training times [31].

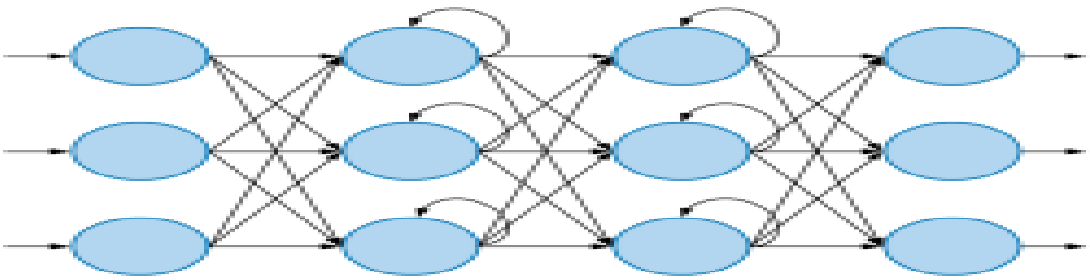


Figure 2.9: Recurrent U-Net architecture [31].

2.5 A comparaison between the U-Net versions

Architecture	Year of publication	Dimension	Main characteristics	Advantages	Disadvantages
U-Net	2015[23].	2D [33].	Basic U-Net architecture with two symmetrical parts (encoding and decoding) and a direct connection between them [29].	Easy to implement, ideal for image segmentation applications [28].	May lack accuracy for more complex segmentation tasks and larger images.
U-Net++	2018.	2D [33].	U-Net architecture with multiple resolution levels for each encoding and decoding branch, as well as hierarchical fusion of each resolution level.	Can improve segmentation accuracy for larger images and complex anatomical structures.	More complex to implement than basic U-Net architecture.
U-Net 3+	2020.	2D [33].	Improvement of U-Net architecture with three symmetric parts (encoding, processing, and decoding) and a combination of direct and convolutional	Can improve segmentation accuracy for larger images and complex anatomical structures.	More complex to implement than basic U-Net architecture.

			connections.		
Attention U-Net	2018.	2D [33].	U-Net architecture with attention modules added to improve segmentation accuracy by focusing on important regions of the image.	Can improve segmentation accuracy for larger images and complex anatomical structures.	More complex to implement than basic U-Net architecture.
ResU-Net	2019.	2D/3D.	U-Net architecture with residual blocks added to improve learning convergence and segmentation accuracy.	Can improve segmentation accuracy for larger images and complex anatomical structures.	More complex to implement than basic U-Net architecture.
DenseNet	2017.	2D/3D.	U-Net architecture with DenseNet blocks added to improve learning convergence and segmentation accuracy.	Can improve segmentation accuracy for larger images and complex anatomical structures.	More complex to implement than basic U-Net architecture.
3D U-Net	2016.	3D [33].	Version of U-Net architecture designed for segmentation of 3D volumetric images.	Ideal for segmentation applications of 3D volumetric images.	More complex to implement than basic U-Net architecture.

V-Net	2016.	3D.	U-Net architecture version with added volumetric convolution modules for segmentation of 3D volumetric images.	Ideal for segmentation applications of 3D volumetric images.	More complex to implement than basic U-Net architecture.
Recurrent U-Net	2019.	3D.	Recurrent layers.	Captures temporal dependencies in medical image segmentation tasks involving video or time series data.	More complex to implement than basic U-Net architecture.Requires more computational resources due to recurrent layers.

Table 2-1: Comparative analysis of performance among different versions of U-Net.

There are other variants such as : **SwinU-Net**, **TransU-Net**, **U²-Net**, **R2U-Net**, **CE-Net**,**3D U²-Net** (3D Universal U-Net), **Inception U-Net**, **Adversarial U-Net**, **nnU-Net** , **Parallel U-Net** [31], [33],[34], [35].

2.6 Advantages of U-Net

In the field of deep learning, it is necessary to use large data sets to train models. It can be difficult to assemble such volumes of data to solve an image classification problem, in terms of time, budget and hardware resources [26]. The U-Net architecture has several advantages that make it popular for image segmentation tasks:

- **Data Augmentation**

Data labeling also requires the expertise of several developers and engineers. This is particularly the case for highly specialized fields such as medical diagnostics. U-Net overcomes these problems, since it is effective even with a limited data set. This makes it

easier to train the model with augmented data, which can improve the generalization performance of the model. It also offers higher accuracy than conventional models [26].

- **U-Shape Architecture**

A classic autoencoder architecture reduces the size of input information and then subsequent layers. Decoding then begins, the representation of linear features is learned, and the track gradually increases. At the end of this architecture, the output size is equal to the input size.Such an architecture is ideal for preserving the initial size.The deconvolution is performed on the decoder side, which avoids the bottleneck problem encountered with a self-encoding architecture and thus avoids the loss of characteristics [26].

- There are several variants of this network adapted to different situations, some are very precise, others are very fast [28].
- U-Net is a conventional network architecture for fast and accurate image segmentation, especially in medical image analysis, where it has been used in medicine and abnormality and tumor detection [28].

2.7 Disadvantages of U-Net

- **A large number of parameters**

Due to the skip connections and extra layers in the expanding path, U-Net contains a large number of parameters. As a result, the model may be more susceptible to overfitting, particularly when working with small datasets [36].

- **High computational cost**

The skip connections in U-Net require additional computations, which can result in higher computational costs compared to other architectures [36].

2.8 A comparaison between the U-Net architecture and the fully convolutional networks (FCN)

The U-Net architecture is different from other image segmentation systems like fully convolutional networks in several ways. we mention some of the main differences between the two architectures:

	U-Net	FCN
Architecture	The U-Net architecture comprises two main parts: a contracting path and an expanding path that are connected by skip connections. The contracting path is responsible for feature extraction from the input image, while the expanding path performs upsampling of the feature maps and generates the final segmentation map.	FCN has a single encoder-decoder structure, where the encoder includes convolutional and max pooling layers to downsample the input image and extract features, and the decoder includes convolutional and upsampling layers to upsample the feature maps and produce the final segmentation map.
The number of parameters	Compared to FCN, U-Net architecture generally has a higher number of parameters because of the skip connections and additional layers in the expanding path. This can increase the risk of overfitting, especially when the model is trained on small datasets.	
Computational efficiency	FCN is generally considered more efficient than U-Net due to its smaller number of parameters and lack of additional computations for skip connections. This makes FCN a better choice for applications that require fast inference times or low computational resources.	
Performance	Overall, U-Net architecture often outperforms FCN in image segmentation tasks, especially when working with high-resolution images or datasets that have many classes. However, the performance of both architectures can differ based on the particular task and the quality of the training data.	

Table 2-2: Differences between U-Net architecture and fully convolutional networks for image segmentation [36].

2.9 Domaines of application

The U-Net architecture is commonly used in biomedical image segmentation. But it has also been applied to other areas of image segmentation such as route segmentation in autonomous driving, satellite image segmentation, and landscape image segmentation.

Overall, U-Net architecture has shown promising results in medical image segmentation and has the potential to aid in the diagnosis and treatment of various medical conditions.

Segmentation of medical images: The primary function of U-Net models is segmentation, which involves the separation and outlining of different objects in an image, rather than just classifying the entire image as a whole. This is especially important in medical imaging as accurate diagnosis of medical conditions requires careful examination of specific areas within an image. For example, identifying brain tumors requires the separation of tumors from the surrounding brain structures. U-Net has been widely used in medical imaging analysis due to its effectiveness and versatility [31]. The following table outlines the major image modalities where U-Net has been successfully applied for segmentation tasks.

Medical imaging modalities	Definition	Treated diseases	Version used
Magnetic Resonance Imaging (MRI)	MRI is a very popular radiology imaging technique used to take pictures of soft tissue inside the body. It is the most popular modality for image segmentation using U-net .It is a useful diagnostic tool	• Brain tumor. • Fetal brain. • Breast cancer. • Cardiovascular structures.	• 3D U-Net • Base U-Net • Parallel U-Net • U-Net++

	particularly for brain, heart and other soft tissues.		
Computed Tomography (CT)	CT is an imaging modality that uses X-rays to produce three-dimensional images of the internal structures of the body. It is often used for segmentation of lungs, bones and soft tissues.	• Liver cancer. • Lung cancer. • Bone cancer. • Cervical cancer.	• 3D U-Net. • U-Net++. • Base U-Net. • Base U-Net.
Ultrasound	Medical ultrasound is yet another noninvasive imaging technique for the analysis of internal structures. U-Net has been applied for the segmentation of	• Nerve segmentation. • Fetal head. • Gastrointestinal tumor. • Thyroid.	• Inception U-Net. • Base U-Net • Base U-Net • Residual U-Net

	various structures in ultrasound images.		
X-Ray	X-ray is a radiographic method used primarily to imaging hard tissue. U-Net models have been applied to X-rays to diagnose many diseases.	• Phalange bones. • Chest organs. • Surgical catheters. • Pelvic bones	• Base U-Net. • Base U-Net. • Residual U-Net. • Base U-Net.

Table 2-3: Exploring modalities for the application of U-Net in medical image segmentation[31].

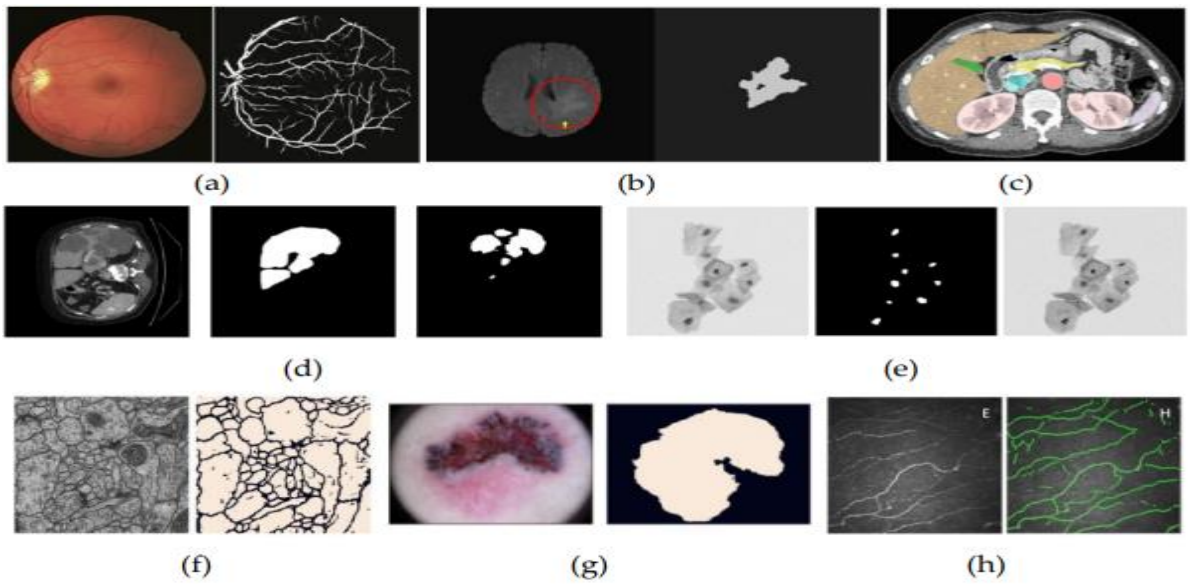


Figure 2.10: Examples of U-Net applications[31].

(a) Retinal vessel segmentation. **(b)** Brain tumor detection and segmentation. **(c)** Multi-organ abdominal segmentation (liver; spleen; left and right kidneys; pancreas; gallbladder; aorta; and inferior vena cava) on CT scans.**(d)** Liver tumor segmentation, left to right: original CT image, liver segmentation image, and lesion segmentation image .**(e)** Nuclei prediction, from left to right: original cell images, prediction of nuclei, labeling nuclei in the original images .**(f)** Cell segmentation .**(g)** Skin lesion segmentation . **(h)** Corneal nerve segmentation.

In summary, U-Net has been successfully applied to various medical imaging modalities for the segmentation of different structures, which demonstrates its versatility and effectiveness in this field.

2.10 Conclusion

In conclusion, the U-Net architecture has emerged as a powerful and versatile tool in the field of computer vision, providing a robust solution to various image analysis tasks. From its humble beginnings in biomedical image segmentation, U-Net has expanded its reach and found applications across diverse domains, ranging from autonomous driving to satellite imagery analysis.

Throughout this chapter, we have explored the U-Net architecture in depth, examining its key components and understanding its unique U-shaped design. We have also discussed the evolution of U-Net, including its different versions and variations that have been developed to address specific challenges and improve performance in different scenarios. We have also talked about his advantages, disadvantages and his main applications.

Chapter 03 : Implementation

3.1 Introduction

In this chapter, we delve into the conception and implementation of our system, providing a comprehensive overview of its architecture and functionality. We begin by presenting the overall system design, including the key components and their interactions.

Furthermore, we explore the technologies employed in the development of our project. We discuss the tools, frameworks, and programming languages utilized to build the system, providing insights into their relevance and advantages in achieving our goals.

One significant aspect of our project is the user interface application, which serves as the primary interaction point between users and the system. We showcase screenshots of our intuitive and user interface, illustrating its key features and functionality. This section offers a visual representation of how users can interact with the system and obtain the desired results.

Finally, we conclude this chapter by discussing the results obtained from the implementation and evaluation of our system. We analyze the performance metrics, such as accuracy and loss function, to assess the effectiveness of our approach.

Overall, this chapter provides a comprehensive insight into the conception, implementation, and evaluation of our system, highlighting the key architectural aspects, project details, employed technologies, and the achieved results.

3.2 Conception

3.2.1 Presentation of our system

Our system utilizes the U-Net architecture for image segmentation, with a focus on segmenting nuclei in medical images and the detection of skin cancer. The system goes through a preprocessing phase, where training images and masks are resized to a consistent size.

The U-Net architecture consists of a contracting path (encoding) and an expansive path (decoding), enabling the model to capture both global and local information. Each layer in the U-Net architecture serves a specific purpose, from convolutional layers with activation functions to dropout layers for regularization. The model is trained using the Adam optimizer and binary cross-entropy loss, with early stopping and model checkpointing to prevent overfitting and save the best model. The training process is visualized using TensorBoard.

The system achieves impressive results, as seen in the loss curve plot and the accuracy metric. The predictions on the training, validation, and test datasets demonstrate the effectiveness of the system in accurately segmenting nuclei and skin cancer. Overall, our system offers a robust and efficient solution for segmentation in medical images, with potential applications in various fields such as pathology and biomedical research.

3.2.2 Overall system architecture

The architecture of a neural network system plays a crucial role in determining its performance and ability to solve complex tasks. In this context, we have developed an innovative system architecture that encompasses both the training and testing phases. Our architecture is designed to learn from a given dataset during the training phase and make accurate predictions on unseen data during the testing phase. By effectively combining these two phases, our system aims to achieve robust and reliable performance.

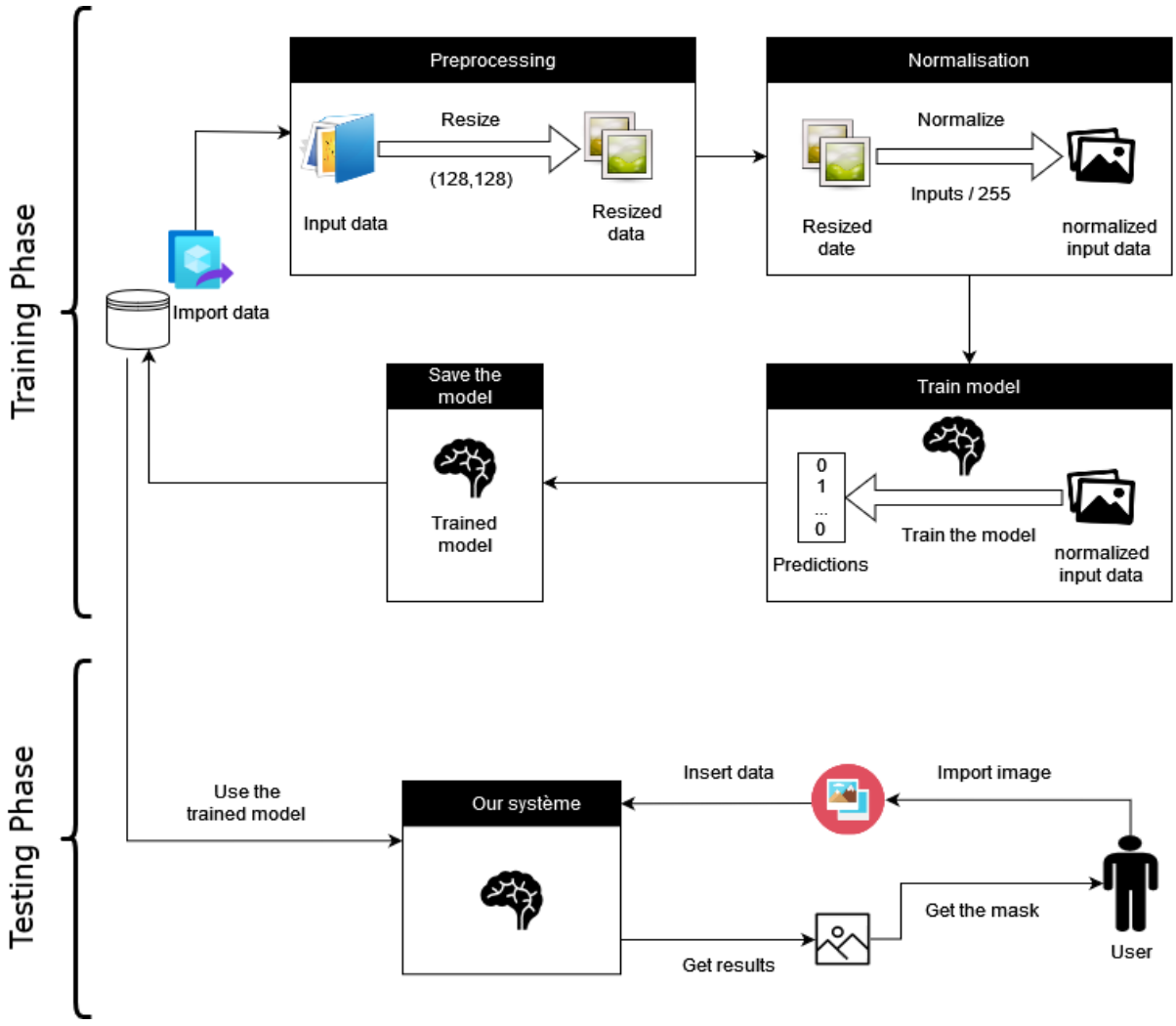


Figure 3.1: Overall system architecture

During the training phase of our architecture, the neural network model learns from a labeled dataset. This dataset consists of input data (features) and their corresponding target outputs. It consists of four modules, they handle image resizing, normalization, model creation and training, and model saving.

- **Image Resizing:**

In this module, the training images are resized to a specific width and height (128,128). The resizing ensures that all images have the same dimensions for consistency during training.

- **Image Normalization:**

After resizing, the images undergo normalization to bring the pixel values within the range of [0, 1]. This normalization is achieved by dividing the resized images by 255.

- **Model Creation and Training:**

In this module, the U-Net model is created using the TensorFlow Keras API. The model architecture includes contracting and expansive paths with convolutional layers, pooling layers, dropout layers, and activation functions. The fit function is then called to train the model using the training data.

- **Model Saving:**

After training, the trained model is saved to a file using the ModelCheckpoint callback. This allows us to store the model architecture and learned weights for future use or deployment.

In the testing phase of the system, users have the opportunity to interact with the trained model by providing an image for analysis. This process allows users to obtain valuable insights and predictions from the system. The user submits an image to the system, which then applies the trained model to the input image. Once the testing phase is complete, the system delivers the generated mask photo to the user, providing a visual representation of the identified regions or objects in the input image.

3.3 Working environment

3.3.1 Hardware

To carry out the implementation of this project, we have supplied a **acer** computer with specific technical specifications as presented in table 3-1.

Processor (CPU)	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
Installed memory (RAM)	4,00 Go (3,87 Go utilisable)
System type	Windows 10, 64 bits
Screen	LCD 1366 x 768

Table 3-1: Hardware parameters.

3.3.2 Software

For the implementation project that is in the field of artificial vision, and specifically the semantic segmentation project, python is a suitable choice of programming language. Specifically, the tensorflow library provides a range of functions and models for semantic segmentation tasks, making it a popular choice for this type of project.

The development environment for implementing a semantic segmentation project in python typically includes several essential tools. These tools can assist with data preprocessing, model training, evaluation, and visualization. Here are some commonly used tools for semantic segmentation projects:

3.3.2.1 Python language

In 1991, Guido van Rossum, a programmer, introduced python as an open-source programming language [37].

Python is a high-level programming language that is interpreted, object-oriented, and has dynamic semantics. It is widely used for rapid application development and as a scripting language to connect various components. Python simplicity and readability help reduce the maintenance costs of programs. It supports modular programming through its module and package system, promoting code reuse. Python interpreter and extensive standard library are freely available for major platforms, allowing easy distribution of python applications [38].

Python offers several advantages that contribute to its popularity [37], [39] :

- Simplicity and ease of use.
- Free and open-source.
- Interpreted language.
- Extensive library ecosystem.
- Portability.
- Supportive community.
- Object-oriented.
- High-level.
- Dynamically typed.

Despite the numerous advantages of the python programming language, there are a few drawbacks to consider [37]:

- Low speed.
- Inefficient memory consumption.
- Ineffective in programming for mobile devices.

Python language has several use cases in application development, some of which include the following examples [39]:

- Server-side web development.
- Automation with python scripts.
- Data science an machine learning.
- Software development.
- Software testing automation.

The used version is: python 3.10.9.

3.3.2.2 Anaconda distribution

Anaconda is an open-source distribution that caters to both python and R programming languages. It is widely utilized in the domains of data science, machine learning, and deep learning. Anaconda offers an extensive collection of over 300 libraries, making it a highly suitable choice for programmers engaged in data

science tasks. Its availability of tools and environments enhances efficiency in data analysis and scientific computing [40].

Anaconda simplifies package management and deployment processes. It provides a range of tools that facilitate data collection from diverse sources and enable the utilization of various machine learning and AI algorithms. Anaconda aids in creating manageable environments that can be easily set up and deployed for any project with a simple click of a button [40].There are several applications available in anaconda navigator, such as:

- JupyterLab.
- JupyterNotebook.
- Spyder.
- Pycharm.
- Orange 3.
- RStudio.
- Qt Console.
- R Studio.

3.3.2.3 PyCharm

PyCharm, widely recognized as a top python IDE, provides an array of impressive features. It excels in code completion and inspection, boasts a robust debugger, and offers compatibility with various web programming languages and frameworks. Developed by Jet Brains, a czech company renowned for creating integrated development environments for multiple web development languages, including PHP and JavaScript, PyCharm has garnered significant popularity [41].

This PyCharm tutorial caters to python developers aiming to master an IDE that encompasses a comprehensive set of tools for creating, debugging, and developing projects using different python frameworks. It is also suitable for enthusiastic learners who possess a basic understanding of any IDE [41].

PyCharm IDE provides a wide range of distinctive features that enhance the ease and efficiency of python programming. Here are some of its notable features [42]:

- **Intelligent code editor:** PyCharm offers intelligent code completion, analysis, and navigation capabilities, enabling programmers to write code faster and with greater ease.
- **Debugging and testing:** with support for various frameworks like pytest, doctest, unittest, and nose, PyCharm facilitates seamless debugging and testing of python code.
- **User-friendly interface:** PyCharm boasts an intuitive and customizable user interface. Users can personalize it by choosing from a vast selection of plugins and multiple color schemes.
- **Database and SQL support:** PyCharm provides support for different databases and SQL, making it a valuable tool for data analysis and visualization tasks.
- **Integrated tools:** PyCharm comes bundled with various integrated tools, including version control, code deployment, and task management, streamlining the development workflow.

3.3.2.4 Google collaboratory

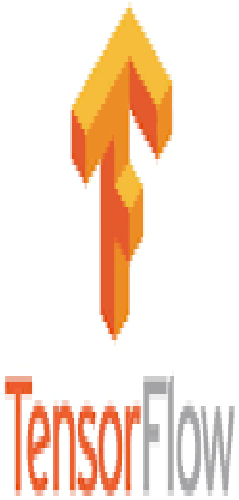
Google Collaboratory or Google Colab, is an executable document that can be used to store, write, and share programs that have been written via Google Drive. This software is basically similar to the free Jupyter Notebook in the form of a cloud that is run using a browser, such as Mozilla Firefox and Google Chrome. It allows users to run python code without the need for any other installation and setup processes. Instead, all the necessary settings and adjustments will be submitted to the cloud.Therefore, this application is a good place for programmers who want to hone their knowledge of Python. Apart from that, Google collaboratory is also well known for being able to drive the need for team collaboration. Where notebooks that will be created later can also be edited simultaneously by other team members, such as editing documents in Google Docs. The biggest advantage of Google collaboratory is that it has a collection of the most popular built-in machine learning libraries that you can easily load into your notebook [43].

Google Colab is a specialized application designed specifically for machine learning and deep learning tasks, offering a unique set of features that distinguish it from other coding software. Here are some notable features of Google Colab [43]:

- A comprehensive built in machine learning library.
- Cloud-based, so it doesn't take up space in computer memory.
- Data in google collaboratory can be accessed and edited easily.
- Simplify the process of collaboration between teams.
- Has GPU and TPU features that can be used for free.

3.3.2.5 Library used

In the realm of machine learning and deep learning, numerous libraries have gained substantial adoption, making significant contributions to our advancements. Within this project, we utilized a selection of crucial libraries, where in TensorFlow and Keras assumed pivotal roles in deep learning, OpenCV facilitated computer vision tasks, and Tkinter enabled the creation of user interfaces.

Library	Definition
TensorFlow 	TensorFlow, a widely-used machine learning and deep learning framework, was developed by the Google Brain Team and released on November 9, 2015. It is an open-source library that primarily utilizes the python programming language for numerical computation and data flow, simplifying and accelerating the process of machine learning [44]. TensorFlow is capable of training and executing deep neural networks for various tasks such as image recognition, classification of handwritten digits, recurrent neural networks, word embedding, natural language processing, video detection, and more. It can be deployed on multiple CPUs or GPUs, as well as mobile operating systems [44]. The name "**TensorFlow**" is derived from two words: "**Tensor**" and "**Flow**" A tensor refers to a multidimensional array, while "Flow" represents the way data flows through operations. In essence, TensorFlow is designed to manage the flow of data within operations performed on multidimensional arrays or tensors [44].

NumPy 	NumPy (Numerical Python) is the fundamental package for scientific computing in python. It is a python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, input/output operations, discrete fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [45].
Matplotlib 	Matplotlib is a python library that enables the creation of 2D graphs and plots through python scripts. It includes a module called pyplot, which simplifies the process of plotting by offering features to control line styles, font properties, axis formatting, and more. Matplotlib provides extensive support for a wide range of graphs and plots, including histograms, bar charts, power spectra, error charts, and more. By combining Matplotlib with NumPy, it offers a powerful open-source alternative to proprietary software like MATLAB. Moreover, Matplotlib can be seamlessly integrated with graphics toolkits such as PyQt and wxPython, further enhancing its capabilities and versatility [46].
Keras 	Keras is a python library that enables rapid development of deep learning models. It is an open-source framework aimed at beginners in the field of AI and offers a user-friendly API supporting various libraries for implementing recurrent or convolutional artificial neural networks, such as TensorFlow, Microsoft Cognitive Toolkit, PlaidML, or Theano [47]. The primary goal of Keras is to provide a convenient environment for quickly building artificial neural networks. It was introduced in 2015 and is the brainchild of François Chollet, a developer affiliated with Google. Keras is part of the larger oneiros project, which stands for open-ended neuro-electronic

	intelligent robot operating system [47].
OpenCV 	OpenCV(Open Computer Vision) is an extensive open-source library that focuses on computer vision, machine learning, and image processing. In today's systems, it has become increasingly vital for real-time operations. With OpenCV, it becomes possible to analyze images and videos to detect objects, faces, or even human handwriting. By integrating OpenCV with other libraries like NumPy, python becomes capable of efficiently processing the array structure of OpenCV for analysis purposes. The library utilizes vector space to identify patterns and extract various features from images, enabling the application of mathematical operations on these features [48].
Tkinter 	Tkinter is a widely used graphical user interface (GUI) package in python, serving as the default GUI module and the preferred choice for GUI programming. It simplifies the creation of GUI applications in python by being readily available within python's standard library. A notable advantage of tkinter is its cross-platform compatibility, allowing the same code to be seamlessly executed on windows, macOS, and linux operating systems [49].

Table 3-2: Library used.

3.4 Project explanation

3.4.1 **First dataset:** The nuclei in divergent images to advance medical discovery

This dataset contains a large number of segmented nuclei images. The images were acquired under a variety of conditions and vary in the cell type, magnification, and imaging modality (brightfield vs. fluorescence). The dataset is designed to challenge an algorithms ability to generalize across these variations [50].

Each image is represented by an associated ImageId. Files belonging to an image are contained in a folder with this ImageId. Within this folder are two subfolders [50] :

- **Images subfolder**

The "Images" subfolder stores the image file itself. It contains the actual visual representation of the image that users have uploaded or selected for processing. This subfolder holds the primary image data for each ImageId.

- **Masks subfolder**

The "Masks" subfolder is exclusively included in the training set. It contains the segmented masks that correspond to each nucleus within the image. Each mask in the "Masks" subfolder represents a distinct nucleus within the image, and the segmentation ensures that no pixel belongs to more than one mask. The purpose of these masks is to provide precise delineation of individual nuclei for training and analysis purposes.

By organizing the files in this manner, the application ensures that each image is associated with its respective ImageId and that the training set includes accurate segmented masks for further analysis and model training [50].

3.4.2 **Second dataset:** skin cancer

Training of neural networks for automated diagnosis of pigmented skin lesions is hampered by the small size and lack of diversity of available dataset of dermatoscopic images. The used dataset includes dermatoscopic images from different populations, acquired and stored by different modalities. Cases include a representative collection of all important diagnostic categories in the realm of pigmented lesions: Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv) and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, vasc) [51].

3.4.3 Preprocessing

In our model, the preprocessing consists of two steps: resizing and data normalization.

The resizing step involves adjusting the dimensions of the input images to a predefined size. This ensures that all images in the dataset have the same dimensions, which is necessary for training the model. In our case, the images are resized to an image width of 128 pixels and an image height of 128 pixels.

```
IMG_WIDTH = 128
IMG_HEIGHT = 128
IMG_CHANNELS = 3

TRAIN_PATH = '/content/drive/MyDrive/stage1_train/'
TEST_PATH = '/content/drive/MyDrive/stage1_test/'

train_ids = next(os.walk(TRAIN_PATH))[1]
test_ids = next(os.walk(TEST_PATH))[1]

X_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS), dtype=np.uint8)
Y_train = np.zeros((len(train_ids), IMG_HEIGHT, IMG_WIDTH, 1), dtype=np.bool)

print('Resizing training images and masks')
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
    path = TRAIN_PATH + id_
    img = imread(path + '/images/' + id_ + '.png')[:,:,:IMG_CHANNELS]
    img = resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant', preserve_range=True)
    X_train[n] = img  #Fill empty X_train with values from img
    mask = np.zeros((IMG_HEIGHT, IMG_WIDTH, 1), dtype=np.bool)
    for mask_file in next(os.walk(path + '/masks/'))[2]:
        mask_ = imread(path + '/masks/' + mask_file)
        mask_ = np.expand_dims(resize(mask_, (IMG_HEIGHT, IMG_WIDTH), mode='constant',
                                      preserve_range=True), axis=-1)
        mask = np.maximum(mask, mask_)

    Y_train[n] = mask
```

Figure 3.2: Preprocessing.

The data normalization step involves scaling the pixel values of the resized images to a specific range. In our case, the pixel values are divided by 255 to normalize them into the range of [0, 1]. This normalization process is important as it brings the pixel values to a standardized scale, making it easier for the model to learn and make accurate predictions.

```
inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)
```

Figure 3.3: Normalisation.

By performing resizing and data normalization as preprocessing steps, we ensure that the input data is consistent in size and range, which helps in training an effective and reliable model.

3.4.4 Create the model

Our model architecture follows a contracting and expansive path, allowing the model to capture both global and local features of the input images.

- **Contraction path (Encoder):**

The model uses a series of convolutional layers with a 3x3 kernel size, ReLU activation, and "same" padding to perform feature extraction.

Each convolutional layer is followed by a dropout layer (*tf.keras.layers.Dropout*) with a dropout rate of 0.1 to prevent overfitting. After each dropout layer, another convolutional layer with the same parameters is applied.

Max pooling (*tf.keras.layers.MaxPooling2D*) with a 2x2 pool size is performed after each pair of convolutional layers to down sample the feature maps.

```
#Contraction path
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.2)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.2)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(0.3)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)
```

Figure 3.4: Encoder.

- **Expansive path (Decoder):**

The model uses transpose convolutional layers *(tf.keras.layers.Conv2DTranspose*) to upsample the feature maps.

The upsampled feature maps are concatenated (*tf.keras.layers.concatenate*) with the corresponding feature maps from the contraction path to allow for information flow between the encoder and decoder. The concatenated feature maps go through a series of convolutional layers and dropout layers similar to the contraction path.

```
#Expansive path
u6 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)

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

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

u9 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1], axis=3)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
c9 = tf.keras.layers.Dropout(0.1)(c9)
c9 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)
```

Figure 3.5: Decoder.

The final convolutional layer uses a 1x1 kernel and sigmoid activation to generate the output predictions.

```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
```

Figure 3.6*:* Final convolutional layer.

3.4.5 Compilation the model

Our model is compiled using the Adam optimizer (*tf.keras.optimizers.Adam*) and binary cross-entropy loss and evaluated based on accuracy (*tf.keras.metrics.accuracy*).

```
model = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Figure 3.7: Compilation the model.

3.4.6 Training

Our model is trained using the fit function with the training data X_train and target data Y_train. We have split the training data into a training set and a validation set using a validation split of 0.1.

```
#Modelcheckpoint
checkpointer = tf.keras.callbacks.ModelCheckpoint('model_for_nuclei.h5', verbose=1, save_best_only=True)

callbacks = [
        tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
        tf.keras.callbacks.TensorBoard(log_dir='logs')]

history = model.fit(X_train, Y_train, validation_split=0.1, batch_size=16, epochs=26, callbacks=callbacks)
```

Figure 3.8: Training.

The epochs parameter specifies the number of times the model will iterate over the entire training dataset. In our case, the model will go through the training dataset **26** times during training and the batch_size parameter determines the number of samples that will be propagated through the network at each training step. In our case, the model will process **16** samples at a time before updating the weights. Batch processing is more efficient and allows for better utilization of computational resources.

Early stopping *(tf.keras.callbacks.EarlyStopping)* is applied with a patience of 2 to stop training if the validation loss does not improve.

3.4.7 Prediction and evaluation

After training, the model is used to make predictions on the training, validation, and test datasets. The predictions are thresholded using a threshold of 0.5 to obtain binary masks.

Some random samples from the training and validation sets are displayed using imshow from matplotlib.pyplot.

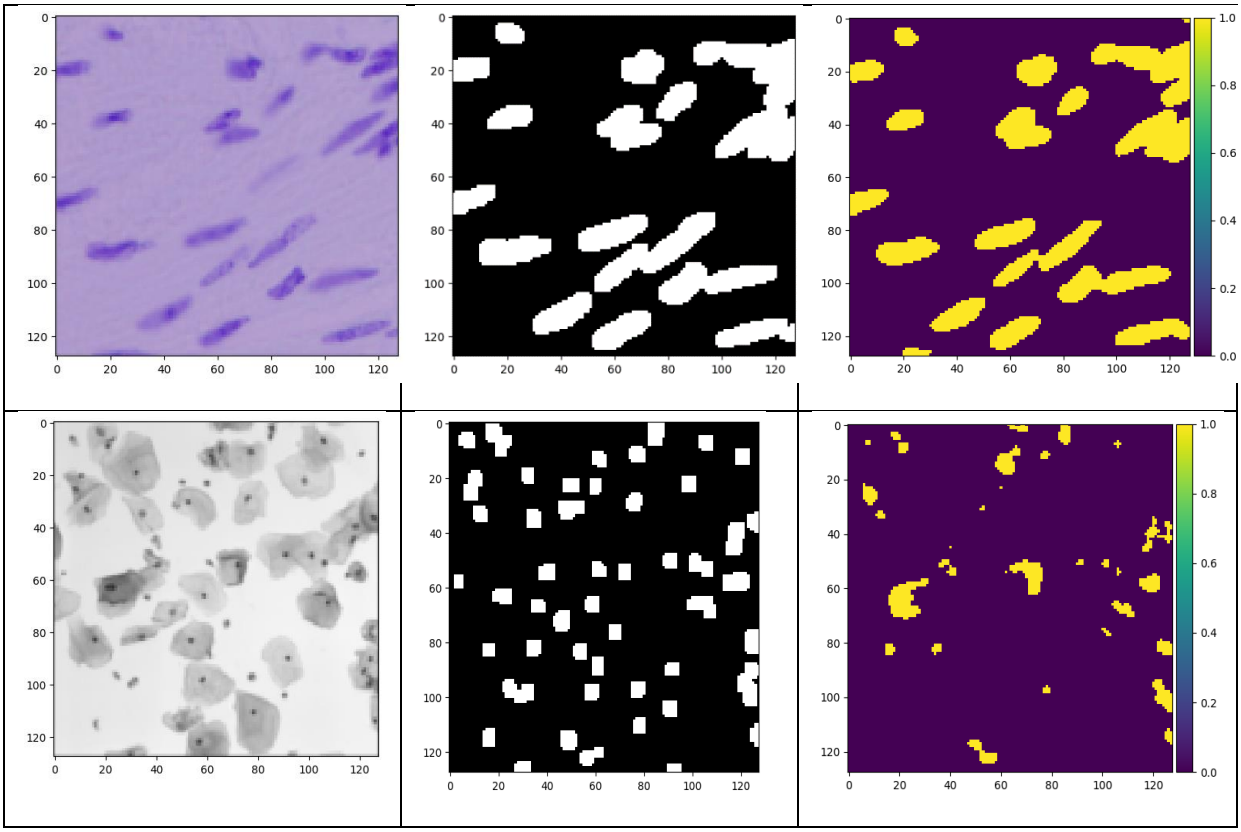


Table 3-3: Predictions from training set for nuclies dataset.

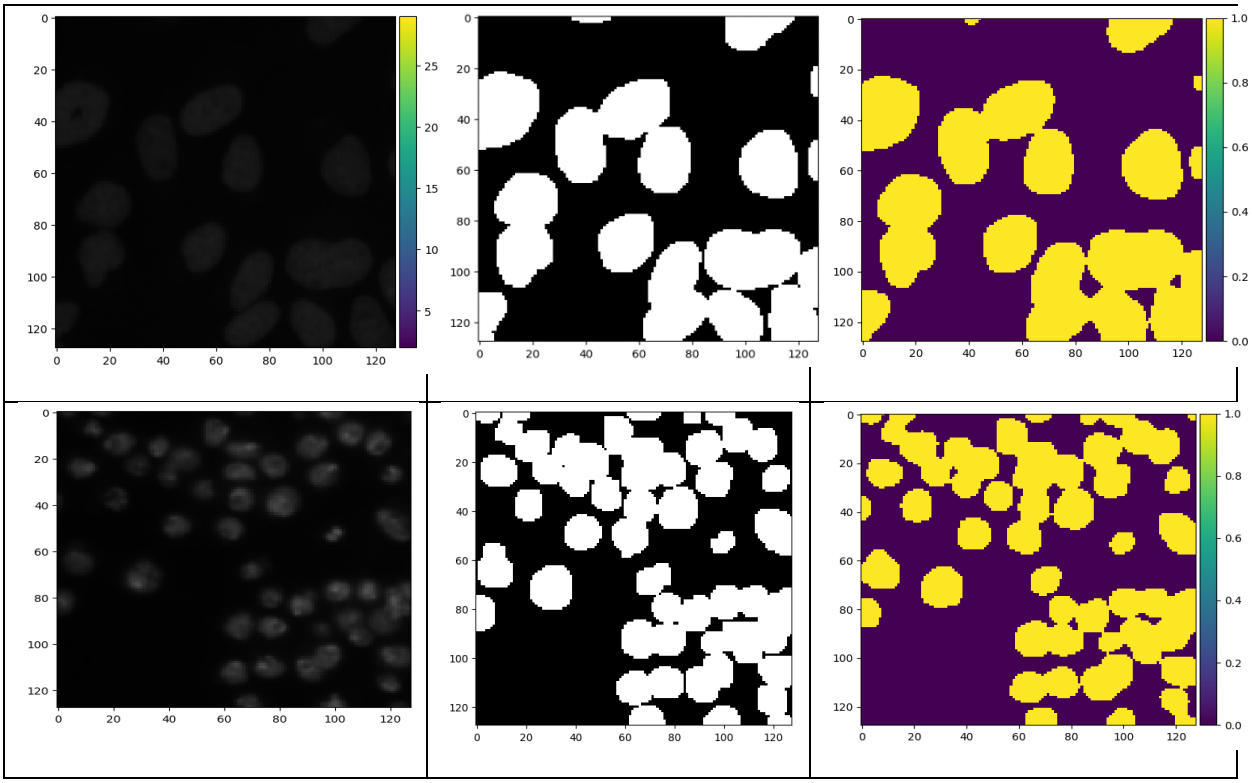


Table 3-4: Predictions from validation set for nuclies dataset.

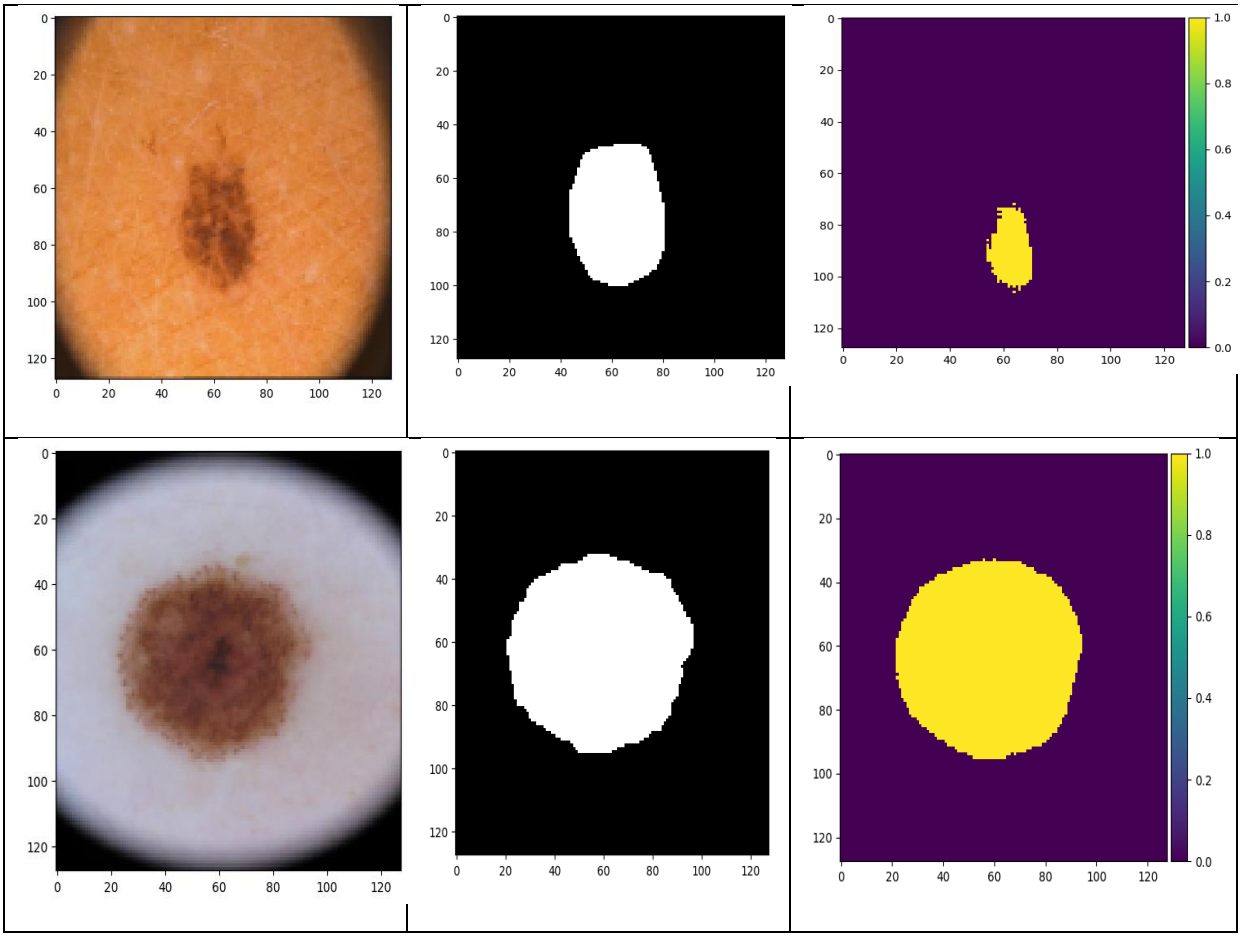


Table 3-5: Predictions from training set for skin cancer dataset.

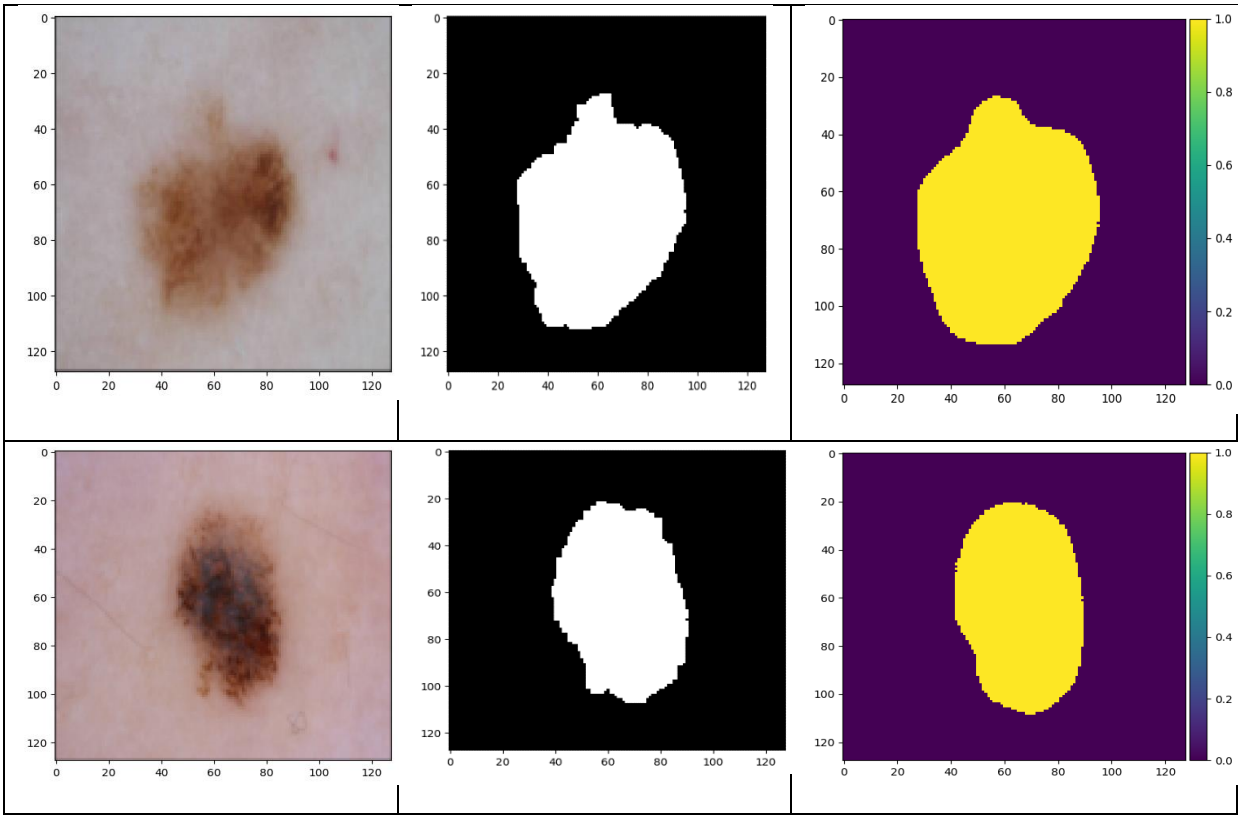


Table 3-6: Predictions from validation set for skin cancer dataset.

3.5 Results

3.5.1 Nuclies dataset

Upon training our model for 26 epochs, we achieved a training accuracy and validation accuracy of 0.96. In terms of loss, our model attained a training loss and a validation loss of 0.09.

```
_________________________________________________________________
Epoch 1/26
38/38 [==============================] - 139s 4s/step - loss: 0.6304 - accuracy: 0.7237 - val_loss: 0.5051 - val_accuracy: 0.7412
Epoch 2/26
38/38 [==============================] - 127s 3s/step - loss: 0.3581 - accuracy: 0.7952 - val_loss: 0.3885 - val_accuracy: 0.8311
Epoch 3/26
38/38 [==============================] - 129s 3s/step - loss: 0.2312 - accuracy: 0.9078 - val_loss: 0.2232 - val_accuracy: 0.9151
Epoch 4/26
38/38 [==============================] - 133s 4s/step - loss: 0.1578 - accuracy: 0.9403 - val_loss: 0.1664 - val_accuracy: 0.9322
Epoch 5/26
38/38 [==============================] - 130s 3s/step - loss: 0.1446 - accuracy: 0.9446 - val_loss: 0.1549 - val_accuracy: 0.9373
Epoch 6/26
38/38 [==============================] - 128s 3s/step - loss: 0.1254 - accuracy: 0.9522 - val_loss: 0.1313 - val_accuracy: 0.9498
Epoch 7/26
38/38 [==============================] - 127s 3s/step - loss: 0.1184 - accuracy: 0.9550 - val_loss: 0.1380 - val_accuracy: 0.9487
Epoch 8/26
38/38 [==============================] - 127s 3s/step - loss: 0.1144 - accuracy: 0.9565 - val_loss: 0.1312 - val_accuracy: 0.9498
Epoch 9/26
38/38 [==============================] - 130s 3s/step - loss: 0.1073 - accuracy: 0.9591 - val_loss: 0.1488 - val_accuracy: 0.9502
Epoch 10/26
38/38 [==============================] - 127s 3s/step - loss: 0.1131 - accuracy: 0.9568 - val_loss: 0.1255 - val_accuracy: 0.9505


Epoch 11/26
38/38 [==============================] - 130s 3s/step - loss: 0.1088 - accuracy: 0.9585 - val_loss: 0.1105 - val_accuracy: 0.9583
Epoch 12/26
38/38 [==============================] - 135s 4s/step - loss: 0.1016 - accuracy: 0.9613 - val_loss: 0.1086 - val_accuracy: 0.9586
Epoch 13/26
38/38 [==============================] - 126s 3s/step - loss: 0.0971 - accuracy: 0.9628 - val_loss: 0.1072 - val_accuracy: 0.9596
Epoch 14/26
38/38 [==============================] - 131s 3s/step - loss: 0.0953 - accuracy: 0.9628 - val_loss: 0.1048 - val_accuracy: 0.9605
Epoch 15/26
38/38 [==============================] - 130s 3s/step - loss: 0.0946 - accuracy: 0.9631 - val_loss: 0.1104 - val_accuracy: 0.9587
Epoch 16/26
38/38 [==============================] - 129s 3s/step - loss: 0.0945 - accuracy: 0.9635 - val_loss: 0.1000 - val_accuracy: 0.9616
Epoch 17/26
38/38 [==============================] - 128s 3s/step - loss: 0.0914 - accuracy: 0.9643 - val_loss: 0.1061 - val_accuracy: 0.9603
Epoch 18/26
38/38 [==============================] - 129s 3s/step - loss: 0.0917 - accuracy: 0.9639 - val_loss: 0.0954 - val_accuracy: 0.9633
Epoch 19/26
38/38 [==============================] - 129s 3s/step - loss: 0.0919 - accuracy: 0.9640 - val_loss: 0.1043 - val_accuracy: 0.9623
Epoch 20/26
38/38 [==============================] - 129s 3s/step - loss: 0.0937 - accuracy: 0.9639 - val_loss: 0.0964 - val_accuracy: 0.9629
19/19 [==============================] - 28s 1s/step
3/3 [==============================] - 3s 692ms/step
3/3 [==============================] - 3s 711ms/step
```

Figure 3.9: Model result.

3.5.2 Skin Cancer dataset

Upon training our model for 26 epochs, we achieved a training accuracy of 0.94 and validation accuracy of 0.93. In terms of loss, our model attained a training loss of 0.14and a validation loss of 0.16.

```
Epoch 1/26
33/33 [==============================] - 180s 5s/step - loss: 0.6752 - accuracy: 0.5778 - val_loss: 0.5502 - val_accuracy: 0.6948
Epoch 2/26
33/33 [==============================] - 177s 5s/step - loss: 0.5180 - accuracy: 0.6963 - val_loss: 0.5017 - val_accuracy: 0.6948
Epoch 3/26
33/33 [==============================] - 172s 5s/step - loss: 0.4458 - accuracy: 0.7456 - val_loss: 0.2844 - val_accuracy: 0.8896
Epoch 4/26
33/33 [==============================] - 173s 5s/step - loss: 0.3870 - accuracy: 0.8460 - val_loss: 0.3010 - val_accuracy: 0.8867
Epoch 5/26
33/33 [==============================] - 178s 5s/step - loss: 0.2661 - accuracy: 0.8980 - val_loss: 0.2699 - val_accuracy: 0.9062
Epoch 6/26
33/33 [==============================] - 174s 5s/step - loss: 0.2244 - accuracy: 0.9116 - val_loss: 0.2047 - val_accuracy: 0.9188
Epoch 7/26
33/33 [==============================] - 176s 5s/step - loss: 0.1921 - accuracy: 0.9228 - val_loss: 0.2106 - val_accuracy: 0.9244
Epoch 8/26
33/33 [==============================] - 179s 5s/step - loss: 0.1719 - accuracy: 0.9308 - val_loss: 0.1744 - val_accuracy: 0.9300
Epoch 9/26
33/33 [==============================] - 172s 5s/step - loss: 0.1662 - accuracy: 0.9331 - val_loss: 0.1772 - val_accuracy: 0.9304
Epoch 10/26
33/33 [==============================] - 172s 5s/step - loss: 0.1525 - accuracy: 0.9379 - val_loss: 0.1663 - val_accuracy: 0.9349
Epoch 11/26
33/33 [==============================] - 171s 5s/step - loss: 0.1537 - accuracy: 0.9395 - val_loss: 0.1689 - val_accuracy: 0.9346
Epoch 12/26
33/33 [==============================] - 174s 5s/step - loss: 0.1401 - accuracy: 0.9437 - val_loss: 0.1692 - val_accuracy: 0.9352
26/26 [==============================] - 38s 1s/step
3/3 [==============================] - 4s 1s/step
12/12 [==============================] - 17s 1s/step
```

Figure 3.10: Model result.

3.6 Discussions

This section is dedicated to the discussion of the results obtained when applying the U-Net architecture topology for the proposed system. The following figures illustrate the graphical representation of accuracy and loss in relation to the number of epochs. It should be noted that the blue curves represent the training accuracy, while the red curves represent the validation accuracy.

3.6.1 The accuracy

During the training phase of our semantic segmentation system using the U-Net architecture, we closely monitored the accuracy metric to assess the model's performance. The accuracy represents the proportion of correctly classified pixels in relation to the total number of pixels in the image.

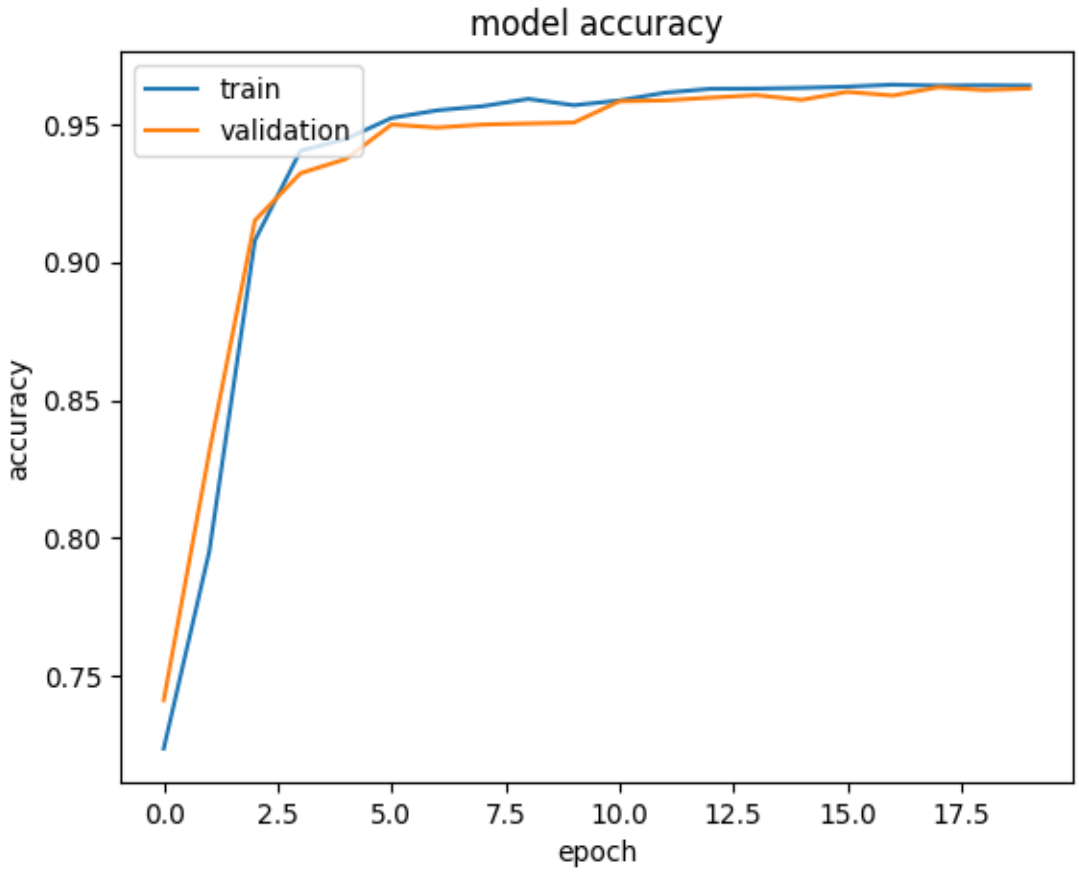


Figure 3.11: Accuracy for nuclies dataset.

At the initial stages of training, the accuracy started at a modest value of 0.72. This is expected, as the training progressed, we observed a steady improvement in the accuracy metric. After three epochs, we witnessed a significant leap in performance, with the accuracy stabilizing at an impressive value of 0.96 in validation and train curves.

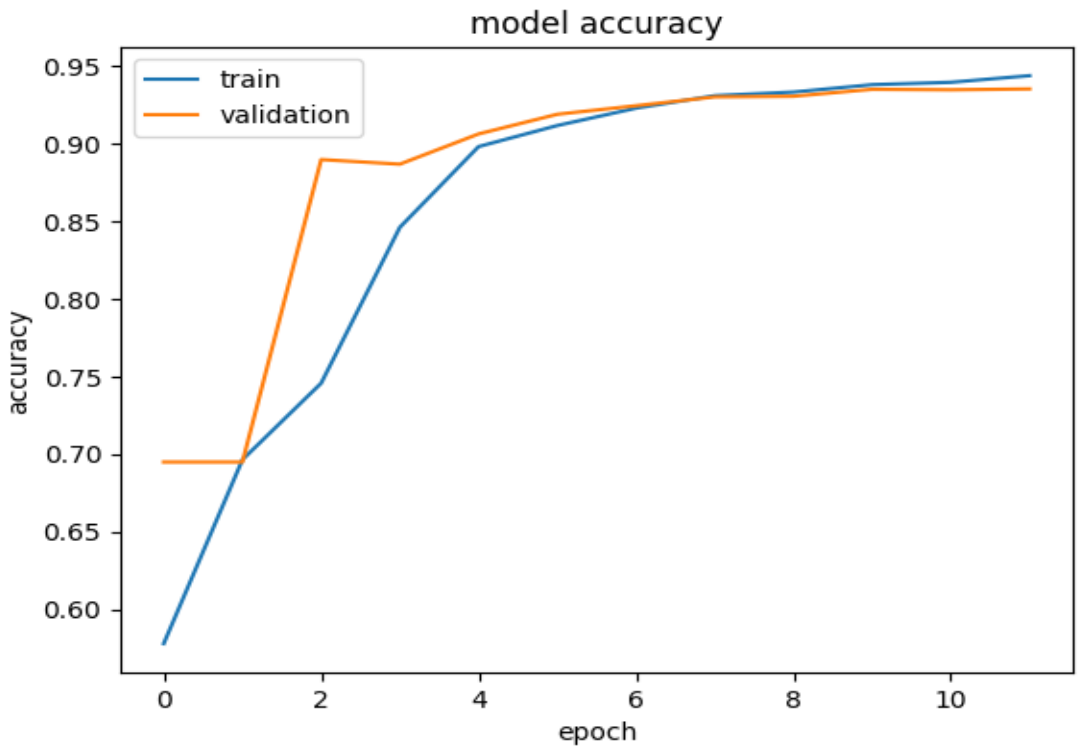


Figure 3.12: Accuracy for skin cancer dataset.

At the initial stages of training, the accuracy started at a modest value of 0.57. This is expected, as the training progressed, we observed a steady improvement in the accuracy metric. After four epochs, we witnessed a significant leap in performance, with the accuracy stabilizing at an impressive value of 0.94 in validation and train curves.

3.6.2 Loss

In addition to monitoring accuracy, the loss function is another crucial metric used to assess the performance of our semantic segmentation system using the U-Net architecture. The loss represents the discrepancy between the predicted segmentation and the ground truth labels for each pixel in the image.

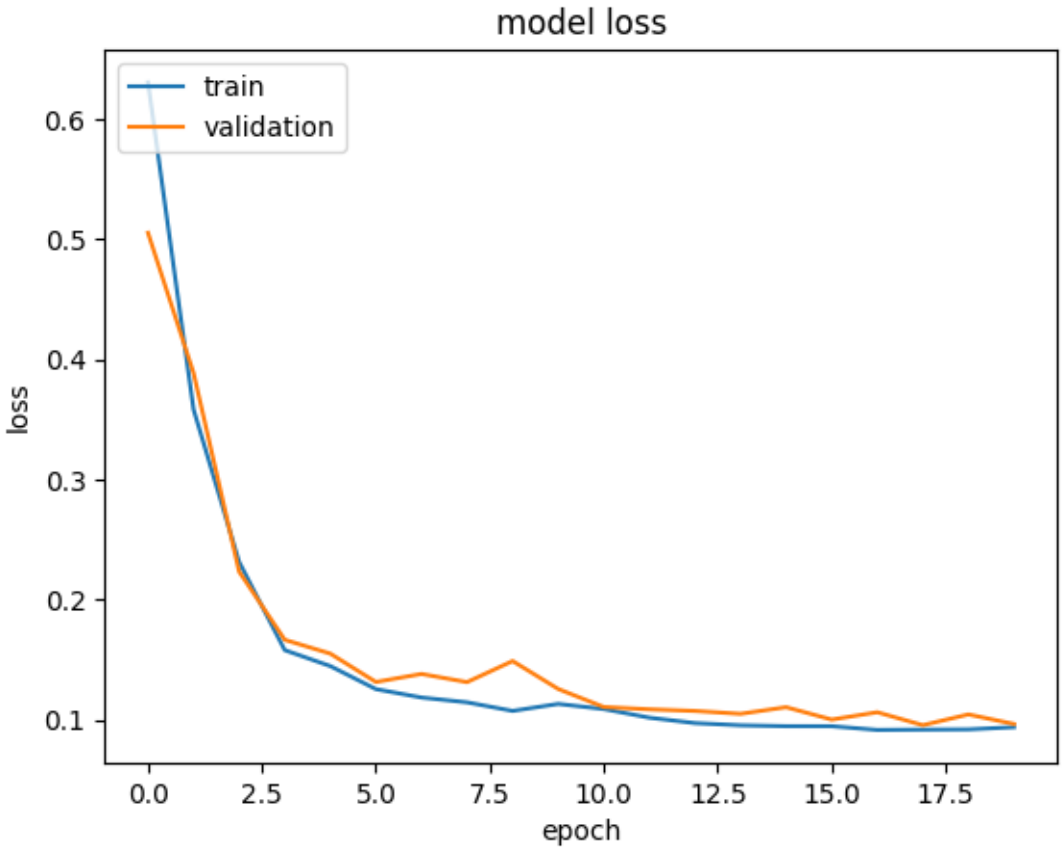


Figure 3.13: Loss for nuclies dataset.

We observe that increasing the number of epochs significantly decreases the loss function, where it can reach as low as 0.09 in both training and validation data. We note that in this metric, the lower the loss value, the better it is.

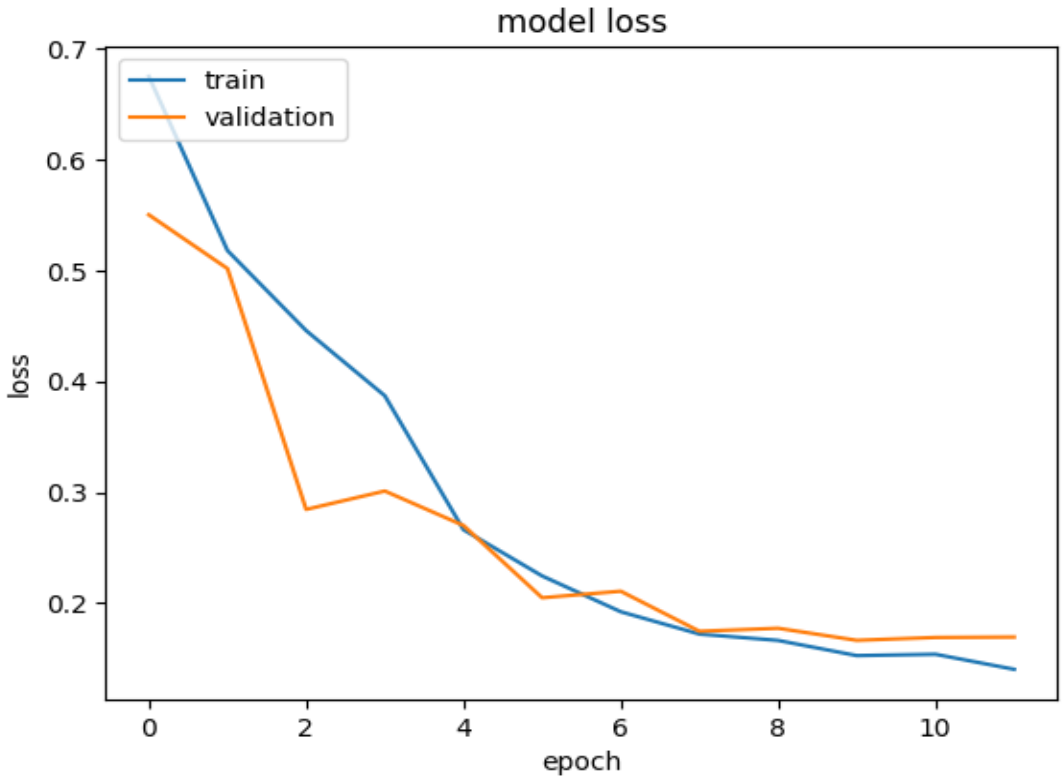


Figure 3.14: Loss for skin cancer dataset.

We observe that increasing the number of epochs significantly decreases the loss function, where it can reach as low as 0.14 in both training and validation data. We note that in this metric, the lower the loss value, the better it is.

3.6.3 Confusion matrix

The confusion matrix is a performance evaluation tool and statistics to assess the performance of a classification model. It is particularly useful for binary classification problems, where there are only two classes or categories.

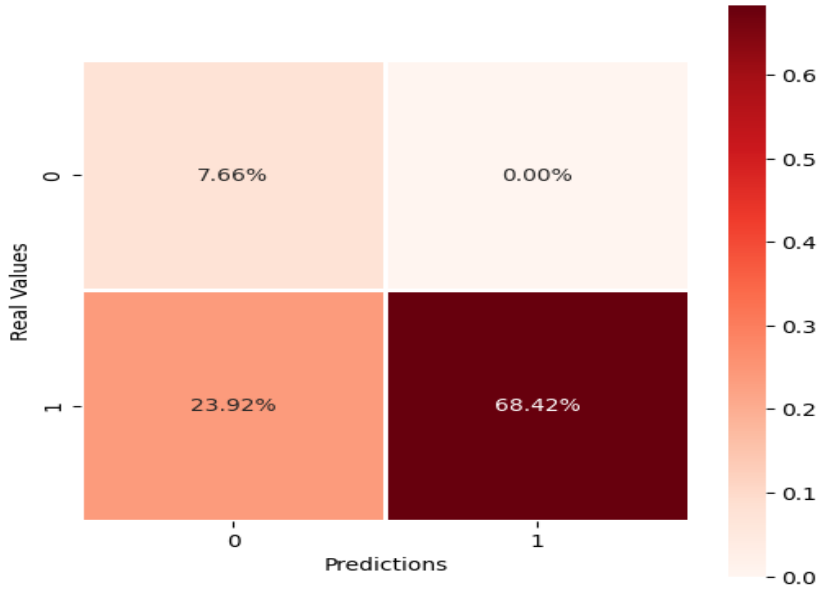


Figure 3.15: Confusion matrix for nuclies dataset.

- **True Positive (TP):** The model correctly predicted the positive class,TP=7.66%.
- **False Positive (FP):** The model incorrectly predicted the positive class when the actual class was negative.FP=23.92%
- **False Negative (FN):** The model incorrectly predicted the negative class when the actual class was positive.FN=0%.
- **True Negative (TN):** The model correctly predicted the negative class,TN=68.42%.

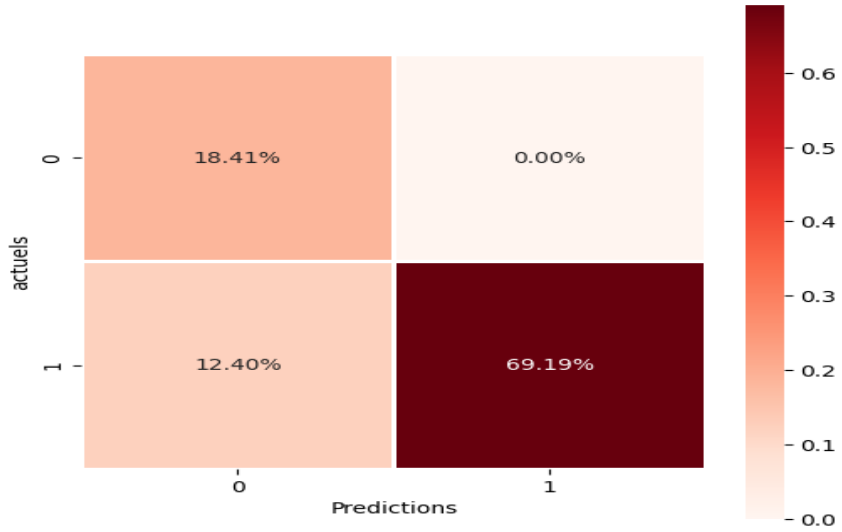


Figure 3.16: Confusion matrix for skin cancer dataset.

- **True Positive (TP):** The model correctly predicted the positive class,TP=18.41%.
- **False Positive (FP):** The model incorrectly predicted the positive class when the actual class was negative.FP=12.40%
- **False Negative (FN):** The model incorrectly predicted the negative class when the actual class was positive.FN=0%.
- **True Negative (TN):** The model correctly predicted the negative class,TN=69.19%.

3.7 Presentation of application

In this section, we present and describe, from a user's point of view, the features offered by our system. Each feature will be described with a screenshot and an explanation of how it works.

3.7.1 Home page

This is the first graphical interface that is displayed when launching the application. The first button on the home page is labeled **"Start"**. Its primary function is to allow users to access the main window of the application. By clicking this button, users are directed to the main interface, where they can interact with the core functionalities and features of the application. The second button on the home page is labeled **"Exit"**. Its primary purpose is to provide users with a means to exit or close the application.

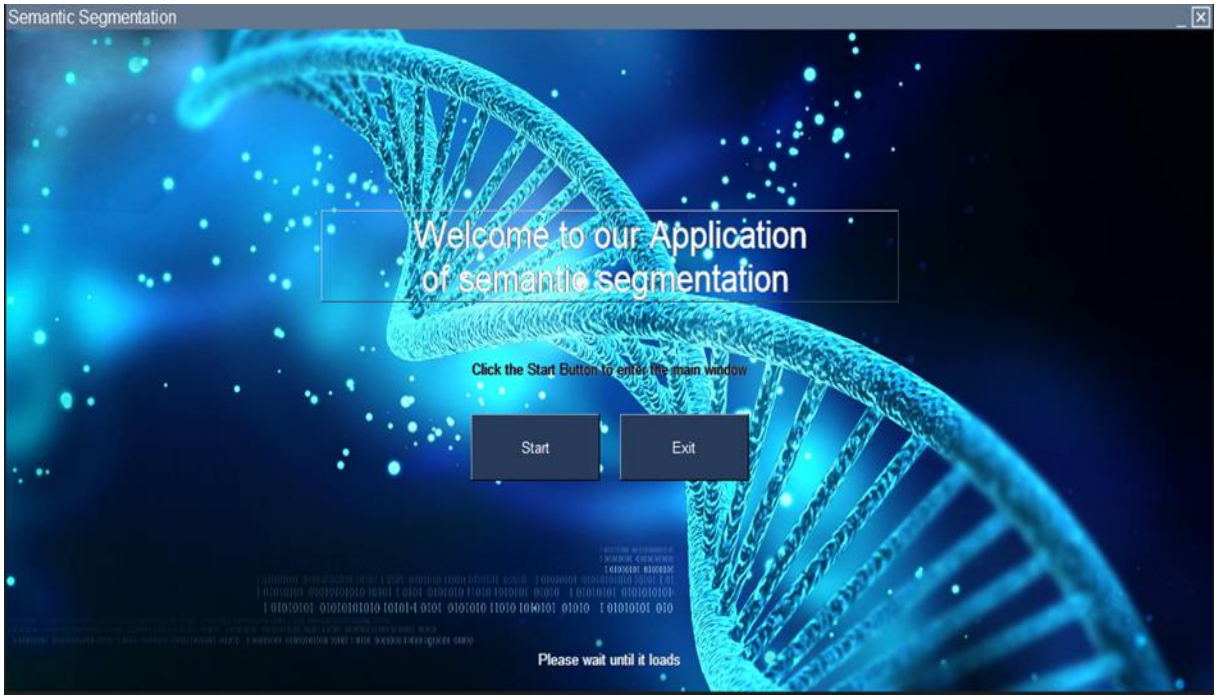


Figure 3.17: Home page.

3.7.2 Main page

The main page of our application is divided into four sections of buttons: the choose model buttons, the upload button, a button for segmentation and the show plots button. In addition to two image canvas: Test Image and Results.

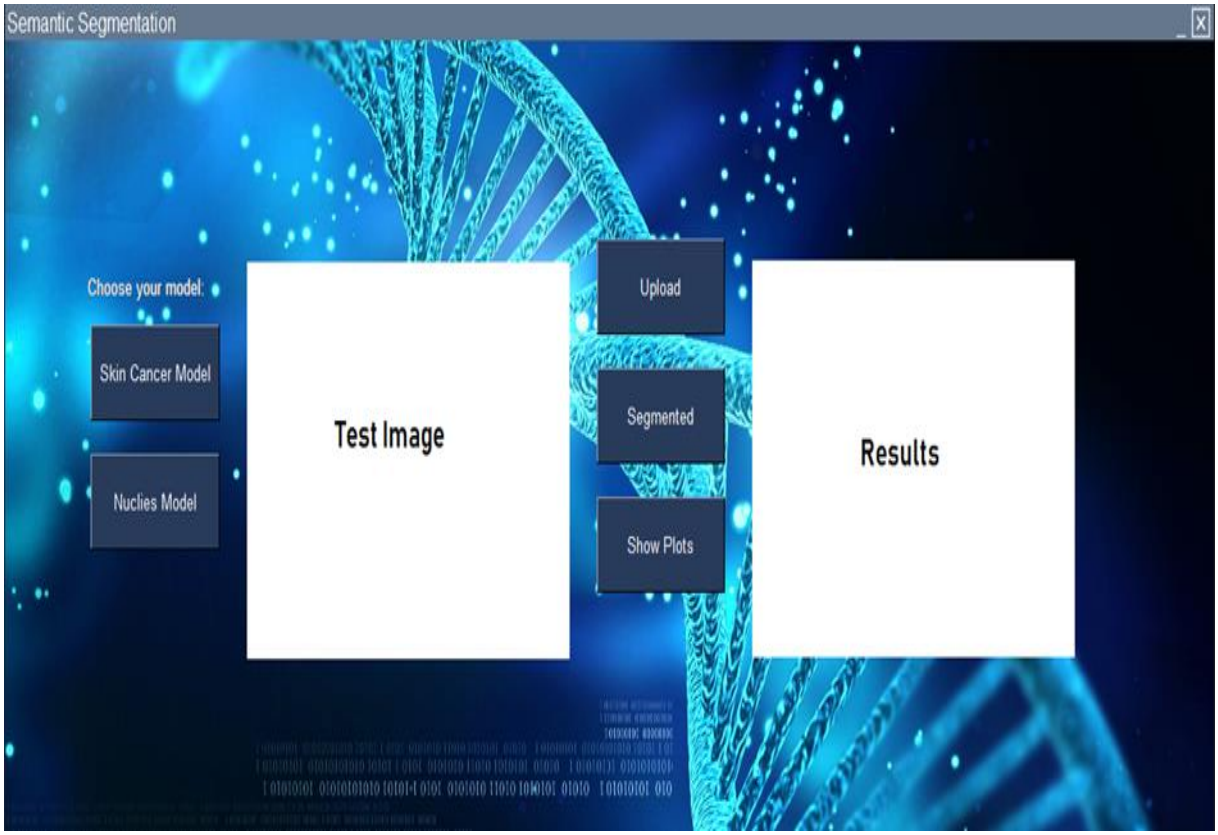


Figure 3.18: Main page.

- **Model selection buttons**

By simply clicking on the buttons of one of datasets **"Skin Cancer Model"** or **"Nuclies Model"** shows us a summary of the architecture and parameters of a trained model. It typically displays a concise overview of the model layers, the number of parameters in each layer, and the total number of trainable parameters in the model in the pycharm console.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_6 (InputLayer)         [(None, 128, 128, 3       0         []
                             )]

lambda_5 (Lambda)            (None, 128, 128, 3)       0         ['input_6[0][0]']

conv2d_95 (Conv2D)           (None, 128, 128, 16       448       ['lambda_5[0][0]']
                             )

dropout_45 (Dropout)         (None, 128, 128, 16       0         ['conv2d_95[0][0]']
                             )

conv2d_96 (Conv2D)           (None, 128, 128, 16       2320      ['dropout_45[0][0]']
                             )

max_pooling2d_20 (MaxPooling2D  (None, 64, 64, 16)     0         ['conv2d_96[0][0]']
)

conv2d_97 (Conv2D)           (None, 64, 64, 32)        4640      ['max_pooling2d_20[0][0]']

dropout_46 (Dropout)         (None, 64, 64, 32)        0         ['conv2d_97[0][0]']

conv2d_98 (Conv2D)           (None, 64, 64, 32)        9248      ['dropout_46[0][0]']
```

```
conv2d_113 (Conv2D)          (None, 128, 128, 1)  17        ['conv2d_112[0][0]']

==================================================================================================
Total params: 1,941,105
Trainable params: 1,941,105
Non-trainable params: 0
--------------------------------------------------------------------------------------------------
C:/Users/Lenovo/Desktop/result skin/12/model_for_SkinCancer.h5
model loaded to disk
```

Figure 3.19 : Model for skin cancer dataset.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_2 (InputLayer)         [(None, 128, 128, 3       0         []
                             )]

lambda_1 (Lambda)            (None, 128, 128, 3)       0         ['input_2[0][0]']

conv2d_19 (Conv2D)           (None, 128, 128, 16       448       ['lambda_1[0][0]']
                             )

dropout_9 (Dropout)          (None, 128, 128, 16       0         ['conv2d_19[0][0]']
                             )

conv2d_20 (Conv2D)           (None, 128, 128, 16       2320      ['dropout_9[0][0]']
                             )
```

```
 conv2d_37 (Conv2D)           (None, 128, 128, 1)       17        ['conv2d_36[0][0]']

==================================================================================================
Total params: 1,941,105
Trainable params: 1,941,105
Non-trainable params: 0
__________________________________________________________________________________________________
C:/Users/Lenovo/Desktop/Resultat_nuclies/20/model_for_nuclei.h5
model loaded to disk
```

Figure 3.20: Model for nuclies dataset.

- **Upload button**

The upload button allows users to easily access their image data and choose the specific image they want to work with. This button enhances user flexibility and facilitates seamless data integration into the application.

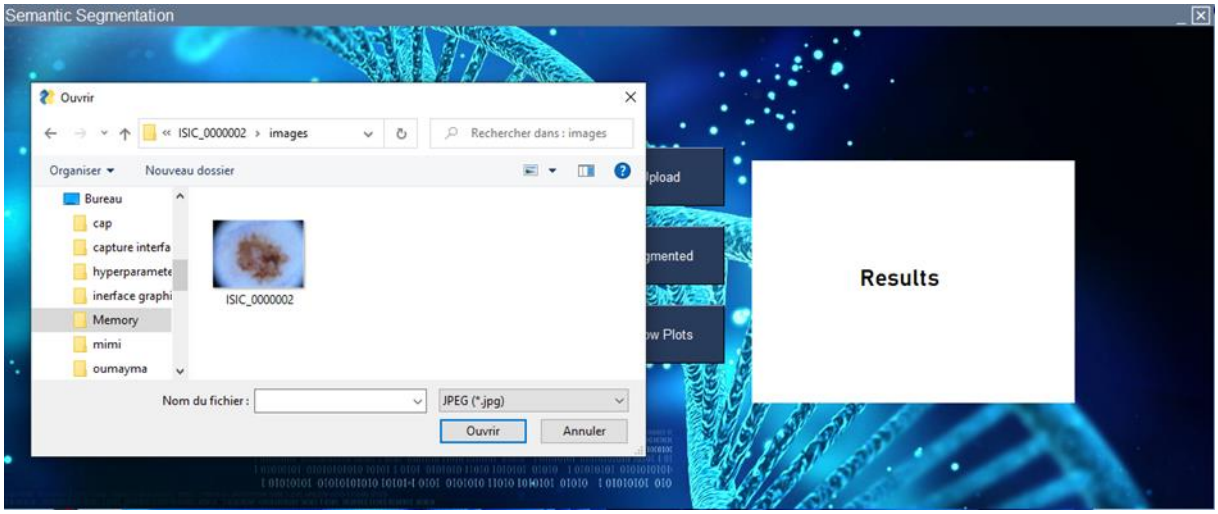


Figure 3.21 : Upload the image from skin cancer dataset folder.

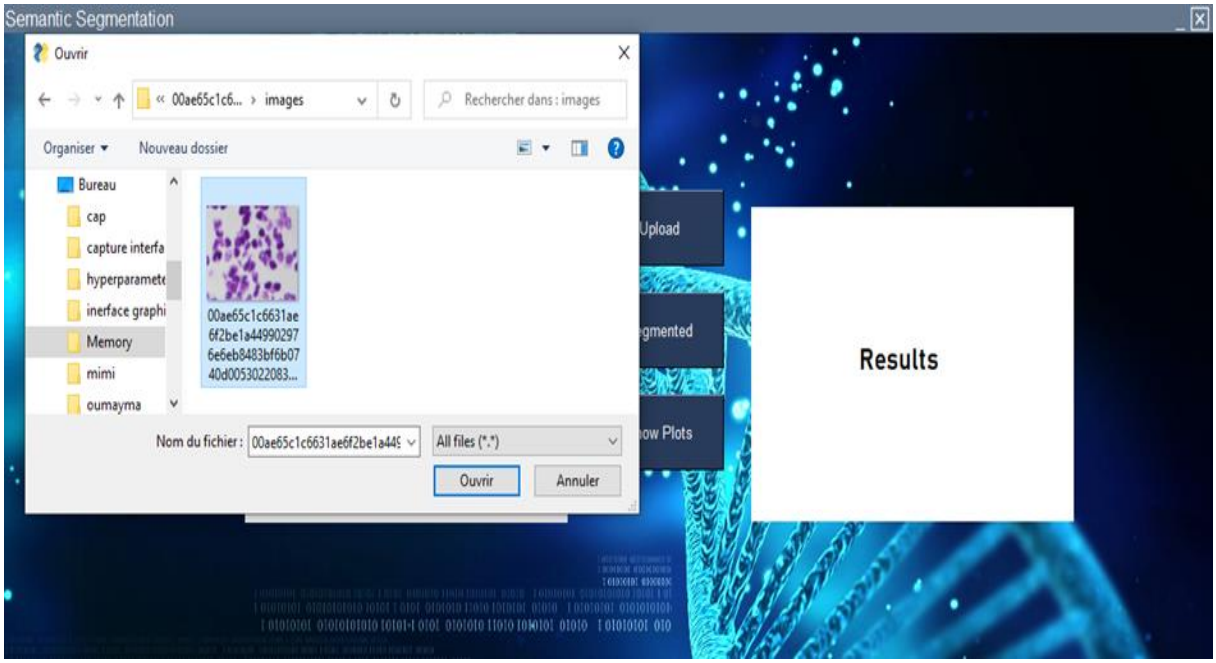


Figure 3.22: Upload the image from nuclies dataset folder.

- **Segmentation button**

This button triggers the image segmentation functionality of our application. When clicked, it analyzes the selected image using the underlying model and provides insights or outputs relevant to the images content or characteristics.

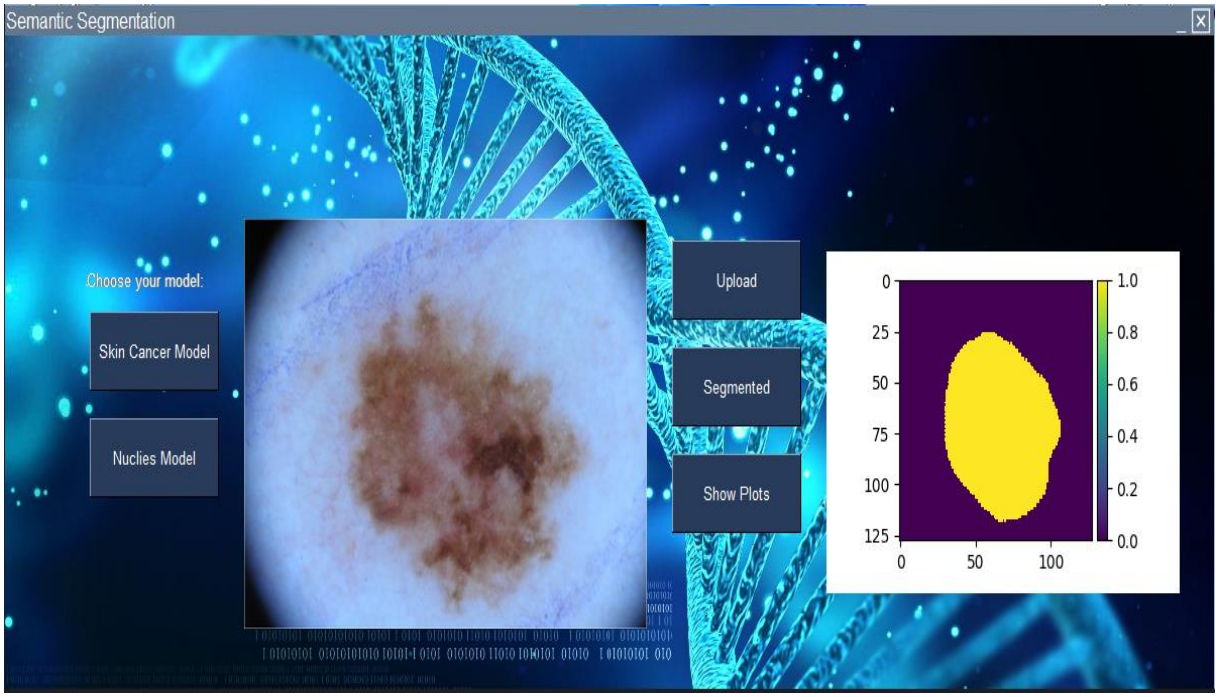


Figure 3.23: Semantic segmentation for skin cancer dataset.

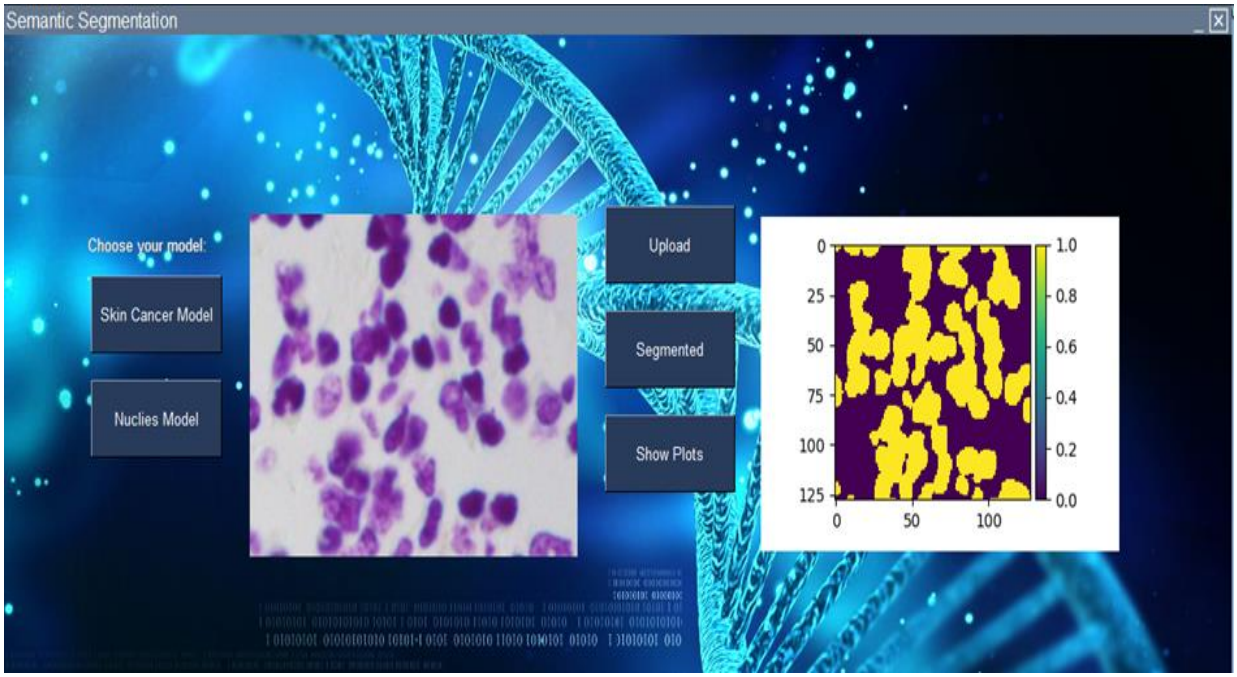


Figure 3.24 : Semantic segmentation for nuclies dataset.

- **Show plots button**

Clicking this button leads to another interface in which the results of the trained model are displayed from the accuracy, loss and confusion matrix providing users with feedback on the models performance.

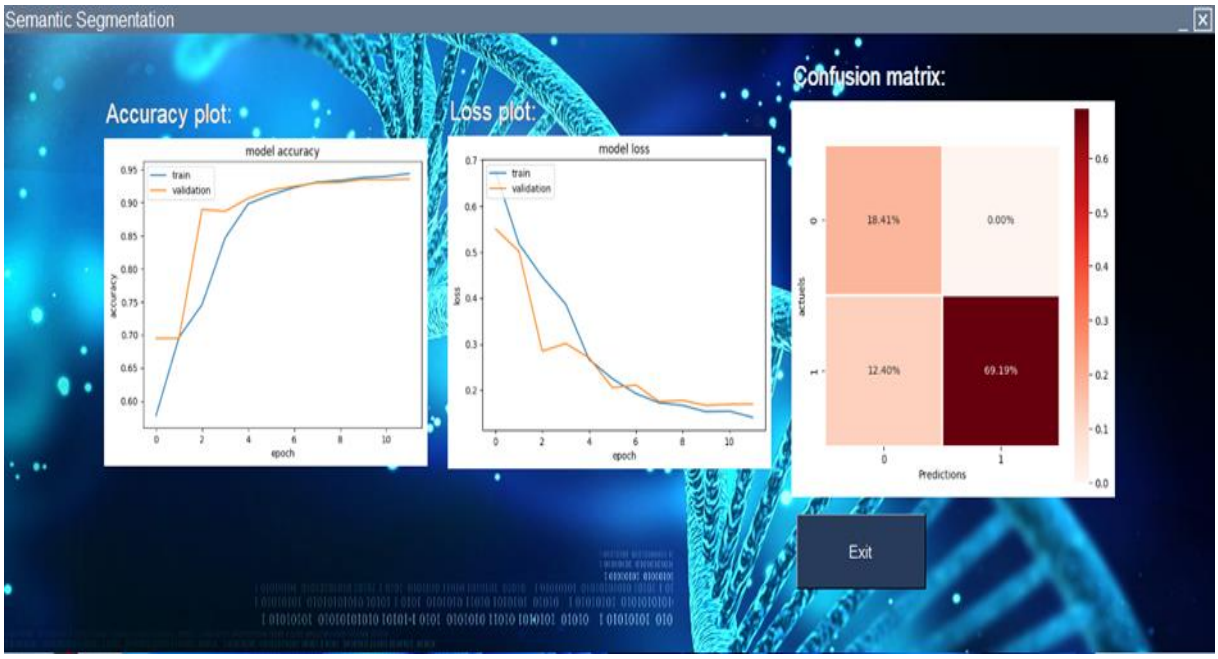


Figure 3.25: Plots of skin cancer dataset.

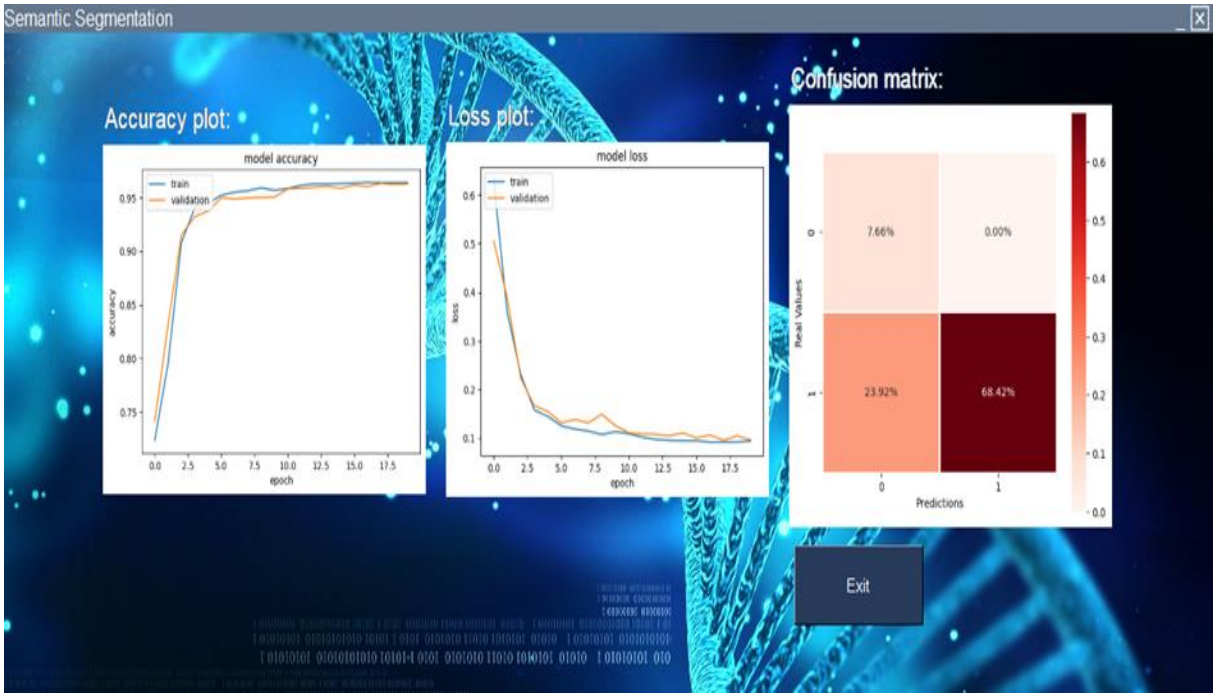


Figure 3.26: Plots of nuclies dataset.

3.8 Optimization of hyperparameters for nuclies dataset

3.8.1 Grid search

The grid search method **'GridSearchCV'** is used to estimate the best parameters of the model. Here's how it works:

- A keras regression model is created using the **'architecture_model()'** function. This model is used as an estimator in grid search.
- The parameters to be estimated are defined in the **activation**, **dropout_rate, optimizer** and **init_mode** lists.
- A **'GridSearchCV'** object is initialized with the model estimator, the parameters to estimate, the number of cross-validation (cv) folds, and other options like **'n_jobs'** which specifies the number of jobs to run in parallel (in this case, 1 for sequential execution).
- The **'fit()**' method is called on the **'GridSearchCV'** object with the training data **(X_train, Y_train)**. This runs the grid search by fitting the model for every possible combination of parameters and using cross-validation to assess performance.
- After the grid search is complete, the results are displayed by printing the mean score **'(mean_test_score)'**, standard deviation **'(std_test_score)'**, and corresponding parameters for each combination of parameters tested.

In summary, grid search is used to evaluate different combinations of parameters and find the best parameters for the model based on performance on the training data[52].

3.8.2 The search space

dropout_rate = [0.1, 0.2, 0.3, 0.4, 0.5]

activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']

init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']

optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']

3.8.3 Results summary

The performance of this method is reported in the tables below :

init_mode	**Cv=1**		**Cv=2**		**The Best**
	Loss	Accuracy	Loss	Accuracy	he_normal
Uniform	0.1965	0.9239	0.2380	0.8885	
lecun_uniform	0.1244	0.9528	0.1309	0.9487	
Normal	0.1205	0.9539	0.1841	0.9286	
Nero	0.6242	0.7664	0.6320	0.7373	
glorot_normal	0.1370	0.9467	0.1411	0.9450	
glorot_uniform	0.1116	0.9579	0.1407	0.9468	
he_normal	0.1064	0.9604	0.1285	0.9508	
he_uniform	0.1119	0.9589	0.1253	0.9530	

```
Best: -0.11745339632034302, using {'init_mode': 'he_normal'}
-0.21724506467580795 (0.020708031952381134) with: {'init_mode': 'uniform'}
-0.12765861302614212 (0.0032362714409828186) with: {'init_mode': 'lecun_uniform'}
-0.1523091271519661 (0.03177257627248764) with: {'init_mode': 'normal'}
-0.6281030476093292 (0.0039100944995588013) with: {'init_mode': 'zero'}
-0.13905376195907593 (0.00206814706325531) with: {'init_mode': 'glorot_normal'}
-0.12616674974560738 (0.014545243233442307) with: {'init_mode': 'glorot_uniform'}
-0.11745339632034302 (0.01104728877544403) with: {'init_mode': 'he_normal'}
-0.11857127398252487 (0.006693743169307709) with: {'init_mode': 'he_uniform'}

Training Time = 13461.07941031456 s
```

Table 3-7: The performance table for **‘init_mode’**.

activation	Cv=1		Cv=2		The Best
	Loss	Accuracy	Loss	Accuracy	relu
Softmax	0.5495	0.7664	0.5760	0.7373	Best: -0.11824231967329979, using {'activation': 'relu'} -0.5627454817295074 (0.0132806599140167 24) with: {'activation': 'softmax'} -0.36439502239227295 (0.023462802171707153) with: {'activation': 'softplus'} -0.18442784994840622 (0.004081137478351593) with: {'activation': 'softsign'} -0.11824231967329979 (0.006555888801813126) with: {'activation': 'relu'} -0.1595837101340294 (0.025228135287761688) with: {'activation': 'tanh'} -0.5169495195158375 (0.028890267014503 48) with: {'activation': 'sigmoid'} -0.46623268723487854 (0.0010230541229248047) with: {'activation': 'hard_sigmoid'} -0.34375111758708954 (0.039004817605018616) with: {'activation': 'linear'} Training Time = 12030.256384372711 s
Softplus	0.3409	0.8654	0.3879	0.8457	
Softsign	0.1885	0.9287	0.1803	0.9306	
Relu	0.1117	0.9588	0.1248	0.9514	
Tanh	0.1344	0.9477	0.1848	0.9277	
Sigmoid	0.5458	0.7664	0.4881	0.7005	
hard_sigmod	0.4652	0.7664	0.4673	0.7288	
Linear	0.3047	0.8875	0.3828	0.8573	

Table 3-8: The performance table for **'activation'**.

dropout_rate	Cv=1		Cv=2		The Best
	Loss	Accuracy	Loss	Accuracy	0.1
0.1	0.1068	0.9604	0.1266	0.9514	Best: -0.11674807593226433, using {'dropout_rate': 0.1} -0.11674807593226433 (0.0098988600075244 9) with: {'dropout_rate': 0.1} -0.13825958222150803 (0.023009620606899 26) with: {'dropout_rate': 0.2} -0.1330990046262741 (0.006130501627922058) with: {'dropout_rate': 0.3} -0.128941148519516 (0.0021196454763412476) with: {'dropout_rate': 0.4} -0.16058000922203064 (0.006448030471801758) with: {'dropout_rate': 0.5} Training Time = 5367.952988624573 s
0.2	0.1613	0.9502	0.1152	0.9552	
0.3	0.1270	0.9522	0.1392	0.9458	
0.4	0.1311	0.9538	0.1268	0.9523	
0.5	0.1670	0.9379	0.1541	0.9386	

Table 3-9: The performance table for **'dropout_rate'**.

Optimizer	Cv=1		Cv=2		The Best
	Loss	Accuracy	Loss	Accuracy	Adam
SGD	0.4234	0.7678	0.5244	0.7538	Best: -0.11399243026971817, using {'optimizer': 'Adam'} -0.4738868772983551 (0.050471872091293335) with: {'optimizer': 'SGD'} -0.13698451220989227 (0.00855022668838501) with: {'optimizer': 'RMSprop'} -0.6090573370456696 (0.012608855962753296) with: {'optimizer': 'Adagrad'} -0.7050100564956665 (0.03563582897186279) with: {'optimizer': 'Adadelta'} -0.11399243026971817 (0.01497969776391983) with: {'optimizer': 'Adam'} -0.13993649929761887 (0.010929502546787262) with: {'optimizer': 'Adamax'} -0.11424735188484192 (0.003559298813343048) with: {'optimizer': 'Nadam'} Training Time = 12133.101943969727 s
RMSprop	0.1284	0.9491	0.1455	0.9421	
Adagrad	0.6217	0.7672	0.5964	0.7366	
Adadelta	0.7406	0.4525	0.6694	0.7371	
Adam	0.0990	0.9625	0.1290	0.9490	
Adamax	0.1290	0.9513	0.1509	0.9442	
Nadam	0.1107	0.9602	0.1178	0.9541	

Table 3-10 : The performance table for **'optimizer'**.

The selection of the hyperparameters values, such as the dropout rate, activation functions, optimizer and initialization modes, is based on previous studies and the existing body of work in the literature. These values have been widely explored and found to be effective in various deep learning tasks. By leveraging the knowledge gained from previous research, we can make informed choices for these hyperparameters to improve the performance of our model.

It is important to note that the structure of the network, including the number of neurons and layers, is not optimized in this context. We have utilized a predefined model architecture known as U-Net, which has proven to be successful in image segmentation tasks. The U-Net architecture is designed specifically for this task, and extensive research has already been conducted to determine its optimal structure. Hence, our focus is on tuning the hyperparameters to enhance the model's performance while keeping the network structure fixed.

3.9 Conclusion

In conclusion, we have presented and explored the conception, implementation, and evaluation of our system. Through a detailed examination of the system architecture and project explanation, we have provided a comprehensive understanding of its design and functionality.

Throughout the project, we utilized various technologies to develop a robust and efficient solution. The chosen tools, frameworks, and programming languages played a crucial role in achieving our project goals and ensuring the system's effectiveness.

Furthermore, the evaluation of our systems performance provided valuable insights. By analyzing the results and discussing the obtained accuracy and validation scores, we were able to assess the system's effectiveness and identify areas for potential improvement.

General Conclusion

In this project, our focus has been on semantic segmentation using the U-Net architecture, with the goal of creating a precise and efficient system for pixel-level image labeling. Through an in-depth exploration of the latest advancements in image segmentation, deep learning, and convolutional neural networks. We have specifically examined the U-Net architecture with its unique characteristics such as skip connections and symmetrical design, the U-Net architecture has demonstrated its effectiveness in the field of semantic segmentation.

In the practical part, this project demonstrated the effectiveness of the U-Net architecture using two different datasets, one for nuclies segmentation and the other for skin cancer segmentation. The results obtained showed satisfactory performance in terms of accuracy and loss, thus validating the effectiveness of the U-Net architecture in these specific areas. In addition, a graphical user interface has been developed to facilitate interaction with the semantic segmentation system. This allows users to explore and analyze results intuitively, which is essential for practical use of the U-Net architecture in real applications.

However, there are still interesting prospects for future work:

- First, it would be beneficial to further explore the performance of the U-Net model using larger and more diverse datasets, to validate its robustness and potential in different contexts.
- In addition, improvements can be made to the graphical interface: implement real-time segmentation for example.
- Finally,comparing the results of U-Net with other versions, like 3D U-Net, is an important area for future research, particularly in tasks that deal with volumetric data such as 3D medical image segmentation.By comparing U-Net with 3D U-Net, you can investigate and compare their results on 2D and 3D segmentation tasks.

In conclusion, the exploration of semantic segmentation with the U-Net architecture has provided valuable knowledge and insights. The combination of theoretical understanding, practical implementation, and comprehensive evaluation has laid the groundwork for future advancements in the field. As technology progresses and more sophisticated techniques emerge, we anticipate even greater breakthroughs in semantic segmentation, contributing to a wide range of applications and domains.

Bibliography:

[1] : Choi, RY. Coyner, AS. Kalpathy-Cramer, J. Chiang, MF.and Campbell, JP. (February27,2020). Introduction to machine learning, neural networks, and deep Learning. Translational Vision Science & Technology. 9(2). pp.1-12. https://pdfs.semanticscholar.org/6f91/5a026d320ecd1a893e3de8c4bb951b4bd62d.pdf .

[2]: Ayad N, Aggoun M A. (2021)." Generating X-Ray Image Of Covid-19 Using Generative Adversarial Networks (GANS) And Covid-19 Detection Using Convolutional Neural Networks (CNN) ". (Diplôme de Master), Université Larbi Ben M'hidi Oum El Bouaghi.

[3] : D Y Moualek. (2017)." Deep Learning pour la classification des images ". (Diplôme de Master), Université Abou Bakr Belkaid-Tlemcen.

[4]: Zahidi, Y. El Younoussi, Y. and Al-Amrani, Y. (2021). Arabic Sentiment Analysis based on Neural Network Models: Overview and Comparison. In Proceedings of the 2nd International Conference on Big Data, Modelling and Machine Learning. pp. 77-80. https://www.scitepress.org/Papers/2021/107287/107287.pdf.

[5] : Ounissi M. Harnane Z I. (2020)." Modélisation et classification avec Deep Learning Application à la détection du Coronavirus Covid-19 ". (Diplôme de Master), Université Larbi Ben M'hidi Oum El Bouaghi.

[6]: Boubaker I. Assas A. (2022)."Deep Learning based approach for predicting depression using Twitter data ". (Diplôme de Master), Université Larbi Ben M'hidi Oum El Bouaghi.

[7] : retengr. (Jan 22, 2021). Deep Learning : définition, applications, avantages et inconvénients. https://www.retengr.com/2021/01/22/deep-learning-definitions-applications-avantages-inconvenients/ .

[8] : BOUAZIZ, D E. (2020)." Sentiment analysis with deep learning ". (Diplôme de Master). Université Larbi Ben M'hidi Oum El Bouaghi.

[9] : Larbes, A. Amrani, O. (2021)." Une approche Deep Learning pour l'analyse des Sentiments". (Diplôme de Master). Université Larbi Ben M'hidi Oum El Bouaghi.

[10]: Sarker,IH. (18 August 2021). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. SN Computer Science. 2(6).pp.1-20. https://doi.org/10.1007/s42979-021-00815-1 .

[11] : Mammeri, O. Medkour, R. (2022)." Semantic Segmentation". (Diplôme de Master). Université Larbi Ben M'hidi Oum El Bouaghi.

[12] : Samaya Madhavan, M. Tim Jones. (25 Jan 2021). Deep learning architectures. https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ .

[13]: GeeksforGeeks. (3 Feb 2023). Convolutional Neural Network (CNN) in Machine Learning. https://write.geeksforgeeks.org/posts-new?interview-experience&ref=GLBIE.

[14]: Mayank Mishra. (26 Aug 2020). Convolutional Neural Networks, Explained.

https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 .

[15]: projectpro, (26 Apr 2023). Introduction to Convolutional Neural Network Architecture.

https://www.projectpro.io/article/introduction-to-convolutional-neural-networks-algorithm-architecture/560 .

[16] : Roumili, L. Yahiaoui, Y. (2022)." Le Hessian et le Tensor Voting dans la Segmentation des Images Médicales ". (Diplôme de Master). Université Abderrahmane Mira-béjaia.

[17] : ROUAGUED, S. (2019)." Extraction des zones d'image à partir de couche d'images médicale". (Diplôme de Master). Université Mohamed Khider – BISKRA.

[18] : Bakhti, S. Boudjellal, H. (2022)." La segmentation des images par la méthode de classification "Fuzzy C-Means" FCM ". (Diplôme de Master). UNIVERSITE MOHAMED BOUDIAF - M'SILA.

[19] : MathWorks. Segmentation sémantique. https://fr.mathworks.com/solutions/image-video-processing/semantic-segmentation.html .

[20] : Ilyes Talbi.(24 Dec, 2021). Qu'est-ce que la segmentation d'images ?. https://larevueia.fr/quest-ce-que-la-segmentation-dimages/ .

[21] : Boutalba Med A. (2020)." Segmentation des images de drones par apprentissage profond ou Deep Learning ". (Diplôme de Master). UNIVERSITE BADJI MOKHTAR - ANNABA.

[22]: Naveen Joshi. (3 sep 2021). How Deep Learning Makes Semantic Segmentation More Precise.https://www.linkedin.com/pulse/how-deep-learning-makes-semantic-segmentation-more-precise-joshi .

[23] : Nikhil Tomar. (19 jan 2021). What is UNET? . https ://medium.com/analytics-vidhya/what-is-unet-157314c87634.

[24]: Educative. (2023). What is U-Net ?. https://www.educative.io/answers/what-is-u-net.

[25]: Conor O'Sullivan. (8 mars 2023). U-Net Explained: Understanding its Image Segmentation Architecture. https://towardsdatascience.com/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a.

[26] : DataScientest. (9 juin). U-NET : le réseau de neurones de Computer Vision. https://datascientest.com/u-net.

[27] : Saadet Aytaç Arpacı. Songül Varlı. (21 December, 2021). « LUPU-Net : a new improvement proposal for encoder-decoder architecture ». INTERNATIONAL ADVANCED RESEARCHES and ENGINEERING JOURNAL. 9(3). p.352-361.

[28]: Nada Belaidi. (8 mars 2022). U-Net : le réseau de neurones populaire en Computer Vision.https://blent.ai/blog/a/unet-computer-vision .

[29] : Margaret Maynard-Reid. (21 févr. 2022). U-Net Image Segmentation in Keras. https://pyimagesearch.com/2022/02/21/u-net-image-segmentation-in-keras/ .

[30] : RACHEL STCLAIR. (13 févr. 2023). What is UNet? How Does it Relate to Deep Learning?. https://www.aiplusinfo.com/blog/what-is-unet-how-does-it-relate-to-deep-learning/#:~:text=UNet%20is%20a%20powerful%20deep,spatial%20resolution%20of%20the%20features.

[31]: Nahian Siddique. Sidike Paheding. Colin P. Elkin. Vijay Devabhaktun.(03 June 2021). «U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications». Institute of Electrical and Electronics Engineers. (Volume: 9).pp. 82031 – 82057. https://ieeexplore.ieee.org/document/9446143 .

[32]: Leo Wang. (22 jul 2022). UNet++ Clearly Explained — A Better Image Segmentation Architecture. https://pub.towardsai.net/unet-clearly-explained-a-better-image-segmentation-architecture-f48661c92df9 .

[33] : Xiao-Xia Yin , Le Sun, Yuhan Fu, Ruiliang Lu, and Yanchun Zhang.(15 April 2022). «U-Net-Based Medical Image Segmentation». Journal of Healthcare Engineering. (Volume: 2022).pp.1 – 16. https://doi.org/10.1155/2022/4189781. https://www.hindawi.com/journals/jhe .

[34]: Vishal Rajput. (1 apr 2022). Attention U-Net, ResUnet, & many more. https://medium.com/aiguys/attention-u-net-resunet-many-more-65709b90ac8b

[35] : Chao Huang ,Hu Han, Qingsong Yao , Shankuan Zhu , and S. Kevin Zhou .(4 Sep 2019). «3D U² -Net: A 3D Universal U-Net for Multi-Domain Medical Image Segmentation ».pp.1-9. https://arxiv.org/pdf/1909.06012 .

[36] : Data Science Blogathon.(6 janv. 2023). Top 8 Interview Questions on UNet Architecture. https://www.analyticsvidhya.com/blog/2023/01/top-8-interview-questions-on-unet-architecture/ .

[37] : Bastien L.(28 avril 2023). Python : tout savoir sur le principal langage Big Data et Machine Learning. https://www.lebigdata.fr/pythonlangage#:~:text=utiliser%20Python%202.-,Le%20langage%20Python%20pour%20le%20Big%20Data%20et%20le%20Machine,les%20interfaces%20graphiques%20d'applications.

[38]: python. What is Python? Executive Summary .https://www.python.org/doc/essays/blurb/.

[39] : the Amazon Web Services site (the “AWS Site”).(September 30, 2022)Qu'est-ce que Python ? – Le langage Python expliqué. https://aws.amazon.com/fr/what-is/python/#:~:text=Python%20est%20un%20langage%20interpr%C3%A9t%C3%A9,les%20erreurs%20dans%20le%20code.

[40] : Mohammad Waseem.(14 mars 2023). Python Anaconda Tutorial : Everything You Need To Know . https://www.edureka.co/blog/python-anaconda-tutorial/ .

[41] : Great Learning.(22 Dec 2022). PyCharm Tutorial for Beginners Definition, Importance, Tools & Features. https://www.mygreatlearning.com/blog/pycharm-tutorial/ .

[42] : Bollyinside. (13 avr 2023).PyCharm Definition and Meaning | Wiki bollyinside. https://www.bollyinside.com/what-is/wiki/pycharm-2 .

[43] :vviia.(1 May 2022). Get to know Google Colab: Starting from the definition, how to use it, to its benefits. https://medium.com/@vviia/get-to-know-google-colab-starting-from-the-definition-how-to-use-it-to-its-benefits-4b190cdbe9ba?source=user_profile---------18---------- .

[44] : Javatpoint. What is Tensorflow | TensorFlow Introduction. https://www.javatpoint.com/tensorflow-introduction .

[45] :NumPY. What is NumPy?. https://numpy.org/doc/stable/user/whatisnumpy.html .

[46] : Tutorialspoint. Python Data Science – Matplotlib. https://www.tutorialspoint.com/python_data_science/python_matplotlib.htm .

[47] : Antoine Crochet-Damais. (22 juil 2022). Keras : tout savoir sur l'API de deep learning open source. https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1501863-keras/ .

[48] : GeeksforGeeks. (26 oct 2022). OpenCV – Overview. https://www.geeksforgeeks.org/opencv-overview/ .

[49] : Studytonight. Introduction to Python Tkinter Module. https://www.studytonight.com/tkinter/introduction-to-python-tkinter-module .

[50] : Boos Allen Hamilton.(16 jan 2018). 2018 Data Science Bowl. https://www.kaggle.com/c/data-science-bowl-2018 .

[51]: Kaggle. Skin Cancer MNIST: HAM10000. https://www.kaggle.com/datasets/kmader/skin-cancer-mnist-ham10000 .

[52] : Bahi, H A A. (2021)." Optimization of hyperparameters of ANNs – Application to the second virial coefficiant (B) of fluid mixture". (Diplôme de Master). Université Larbi Ben M’hidi Oum El Bouaghi.