ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA

SCHOOL OF ENGINEERING

Department of
Computer Science - Science and Engineering
DISI

Bachelor's Degree in Computer Engineering

Bachelor's Thesis
in
Automatic Controls

Development of a Neural Network-Based Software Stack for Automatic Segmentation of Ultrasounds

Candidate:

Caterina Leonelli

Advisor:

Prof. Giuseppe Notarstefano

Co-Advisors:

Prof. Francesco Ursini
Prof. Andrea Camisa
Ing. Lorenzo Sforni

Academic Year
2021–2022

Contents

Abstract

The purpose of this thesis is to implement and analyze a use case of Deep Learning techniques for the semantic segmentation of ultrasound images. The dataset used consists of hand ultrasound images obtained from the database of the Rizzoli Orthopedic Institute. The data consists of hand images, where each image has been manually annotated with three different labels. The three classes are: bone, synoviuml cavity, and tendon.

To train the Deep Learning models, a convolutional neural network (CNN) architecture of the U-net type has been used. The input for the network is an image with dimensions 224 x 224 x 3, and the output is a 3-dimensional vector corresponding to the predicted class probabilities for each of the three classes. The trained model was then evaluated on a test set of 60 images. The results showed that the network was able to achieve an accuracy of over 90.0% on both the training and test sets. Furthermore, since the dataset is imbalanced, other prediction evaluation metrics besides accuracy, such as the F1-score, were necessary, achieving an average value greater than 90.0%.

Introduction

Motivations and State of the Art

Within the field of Machine Learning, and more specifically Deep Learning, machines can be trained to classify a set of input data. Particularly, when the data consists of 2D images, there exist architectures that train the machine to classify each pixel of an input image. This task, referred to as "segmentation," thus allows training a machine to recognize semantically distinct areas within the image. This can be applied to various types of images, especially in the medical domain. In this thesis, semantic segmentation is applied to a set of metacarpal ultrasound images for the recognition of the synoviuml cavity, bone, and tendon. Specifically, changes in the volume of the synoviuml cavity can be recorded and analyzed to prevent and/or treat rheumatoid arthritis. In fact, when arthritic pathology is active, anatomical alterations can occur in the bone, tendon, and synoviuml cavity, which can be pathognomonic of rheumatoid arthritis. Currently in Italy, this type of diagnosis is generally performed by a rheumatologist who, with experience and a clinical eye, assesses the severity of the damage. Manually segmenting ultrasound images depends on the skill and experience of the operator, potentially leading to subjective diagnosis. Moreover, for a professional in the field, reviewing a large volume of clinical images is time-consuming. Therefore, semantic segmentation could play an important role in facilitating differential diagnosis from other joint pathologies.

Structure and Organization

Chapter 1 provides a concise introduction to the basic concepts of Machine Learning. It explains what a Machine Learning algorithm is and then delves into its subsection called Deep Learning. Chapter 2 covers the task of semantic segmentation in more detail, using Deep Learning techniques and the U-net neural network as an example. Chapter 3 presents the implementation of multiclass semantic segmentation applied to the metacarpal ultrasound dataset from the *Rizzoli* Orthopedic Institute. Chapter 4 showcases the tests conducted using the implementation described in the previous chapter.

Chapter 1

Basic Concepts of Machine Learning

1.1 Machine Learning Algorithms

An algorithm that is capable of learning from new input data is referred to as a "Machine Learning" algorithm [1]. Mitchell (1997) [2] proposes a definition of *learning* applicable to an algorithm:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

1.1.1 Types of Tasks

Machine Learning tasks are the various objectives that an algorithm can have and are described in terms of the steps the algorithm must take to process an example. An example is a collection of quantitatively measured features (or *features*): it is represented as a vector $\mathbf{x} \in \mathbb{R}^n$ where each component x_i is another feature itself. Examples of features in an image could be the pixel values that compose it. We will refer to a collection of examples as a dataset or *data points*. The most common Machine Learning tasks include:

- Classification: This task involves the algorithm determining the category to which a given input data belongs. In this type of task, the algorithm must find a function $f : \mathbb{R}^n \longmapsto \{1...k\}$ such that $y = f(x)$, where $\mathbf{x}$ is the input data vector to be classified, and $\{1...k\}$ represents the set of categories. Thus, the machine must produce an output prediction $f(x)$ chosen from the set of available categories, which is as similar as possible to the actual associated category y. Typically, the set of categories is represented numerically or can also be represented by a vector containing probabilities for each class. For example, consider the MNIST dataset [3] as an input set, consisting of images of handwritten digits from 0 to 9. The ultimate goal of the machine is to

determine the represented digit in each image. The machine could be set up to assign a category from 0 to 9 for each image and input vector $\mathbf{x}$ alternatively, it could be configured to provide, for each input vector $\mathbf{x}$, the probability that $\mathbf{x}$ belongs to each category, resulting in a vector $\mathbf{y}$ of size 9 where each *i-th* component is the probability that $\mathbf{x}$ belongs to the *i-th* category.

- Regression: In this task, the machine predicts a continuous numeric output given some type of input. It aims to find a function $f : \mathbb{R}^n \longmapsto \mathbb{R}$. For instance, given the features of a property, the machine could predict its market price. Note that regression differs from classification in that the output is not from a finite set; it can be a numeric value within the set of real numbers. In other words, classification predicts a discrete variable, while regression predicts a continuous variable. It's worth noting that, as discussed in the previous point, classification can output a probability even if it is a continuous variable; however, such a task is not considered a regression task. Similarly, regression might produce a discrete value as output, but this occurs when the output belongs to the set of integers. It is important not to confuse these two tasks, as their implementations have distinct characteristics.

- Transcription: In this task, the machine analyzes input data and generates a textual description of the processed information as output. An example of this task is speech recognition, where human spoken language is recognized and subsequently processed to be converted into textual form.

1.1.2 Performance Measurement

A Machine Learning algorithm must be quantitatively evaluated. Often, this measurement is specific to the chosen task. For instance, in classification and transcription tasks, the accuracy of the model is commonly measured. Accuracy is defined as the ratio of the number of correct predictions to the total number of predictions made. Its counterpart is the *error rate*, which is the ratio of incorrect predictions to the total number of predictions made.

It is a good practice to evaluate the model on data that it hasn't processed during training. Therefore, a test set, consisting of data distinct from those used in training, should be prepared for the machine to make predictions on. Choosing the appropriate measures to evaluate the performance of a model is crucial, and it heavily depends on the application scenario; see Section 1.4.3 for more on this.

1.1.3 Experience - Supervised and Unsupervised Learning

Machine Learning algorithms are classified into supervised and unsupervised types. The difference between these approaches lies in what is referred to as the algorithm's "experience." An algorithm can learn in a *direct* manner, where it receives direct feedback on what output it should produce, or in an *indirect* manner, where it is not provided with any feedback on the desired output.

A supervised algorithm processes a dataset that contains a set of features, and each example in the dataset is associated with a label or target. For example, in the case of classification with the MNIST dataset [3], each image is associated with a number indicating the represented digit. The term "supervised learning" implies a scenario where an instructor or teacher shows the algorithm what it should achieve.

In contrast, an unsupervised algorithm processes a dataset with numerous features and must discern the useful properties of its structure solely from the data. Continuing with the earlier analogy, an unsupervised algorithm lacks any guidance on how to process the data; it must deduce its approach. For instance, consider a machine learning to play chess. It can be trained with direct information, such as a list of correct and incorrect moves, or with indirect information-playing a series of games without specifying correct or incorrect moves. In this latter case, the only feedback the machine receives is the result of the game (win or loss).

1.2 Capacity, Overfitting, and Underfitting

The goal of a Machine Learning algorithm is to learn how to perform a defined task with a specific dataset (referred to as the training dataset, training set, or train set), such that it can perform the same task on new, unseen data. The ability to perform well on previously unseen data is known as "generalization capacity" or *generalization*.

Particularly, the aim is not just to reduce the error of predictions made during training (known as *training error*), as is typically done in simple optimization problems. The objective is also to minimize the error on test data (i.e., the *test error*). The main challenge here is that the test set is not observable or known a priori. However, the field of statistical learning theory asserts that if the training set and the test set are constructed using certain criteria, we can make assumptions and use these to modify the algorithm to improve it.

Some of these assumptions (also called *i.i.d assumptions*) may include:

- Examples in each dataset are independent of each other.
- The training set and the test set have identical distributions.

With these assumptions, we can assign the same probabilistic distribution to each individual example, resulting in what can be called a "probabilistic framework." This framework and the i.i.d assumption allow us to mathematically study the relationships between test error and training error. Specifically, it can be stated that to achieve good performance from a Machine Learning algorithm, it is crucial to minimize both the training error and the gap between training error and test error. The values of these two factors indicate whether the algorithm is exhibiting *underfitting* or *overfitting* behavior. Underfitting occurs when the model cannot sufficiently reduce the training error, while overfitting occurs when the gap between training error and test error is too large. These behaviors are related to the model's *capacity*.

The capacity of a model refers to its ability to accommodate a wide range of functionalities. This means that a Machine Learning algorithm with low capacity may not perform

the task well, resulting in a high training error. Conversely, an algorithm with high capacity may perform well on the training error but might fail to apply what it has learned to the test set. Consequently, this leads to a larger gap between training error and test error. The goal of a Machine Learning algorithm is to have an appropriate capacity for the task assigned by the training set. This is achieved when the machine appropriately and calibratedly selects the solution space, i.e., the functions $f(x)$. Various approaches express preferences among different solutions, falling under what is known as regularization techniques.

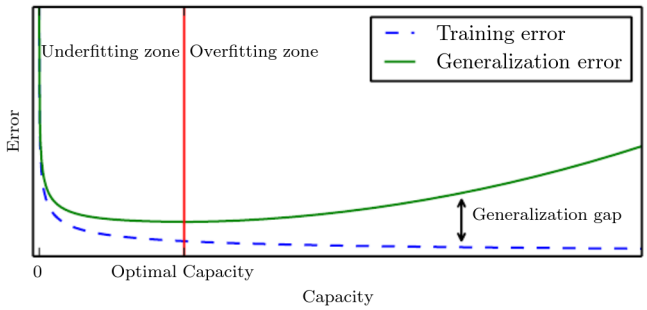

Figure 1.1: Graph depicting the relationship between training error, test error, and algorithm capacity [2]

1.3 Hyperparameters and Validation Set

To control the behavior of a Machine Learning algorithm, specific parameters called *hyperparameters* are often used. These parameters are typically adjusted externally and are not learned by the algorithm during training. Using hyperparameters, the model's capacity can be controlled. If hyperparameters were calculated by the algorithm like training parameters, it could lead to overfitting of the hyperparameters themselves. Therefore, it is a common practice to utilize another set of input data, called the *validation set*, to help determine the best values for these hyperparameters.

The validation set is usually created from the training set. Around 20% of the initial training set is used to form the validation set, while the remaining 80% is used for the actual training set. To avoid ambiguity in the notation of these two types of datasets, the term "training set" will be used hereafter to refer to the data used to train the model parameters during training, while the "validation set" is the dataset used to estimate generalization error or test error after or during the training phase, for the purpose of updating hyperparameters.

1.4 Deep Learning

Deep learning is defined as a subclass of Machine Learning techniques that leverage many layers of non-linear information processing. It is primarily used for supervised or unsupervised learning of features, involving tasks like transformation, analysis, and classification of

patterns within data [4]. Deep learning is thus an approach within Artificial Intelligence that employs an Artificial Neural Network, also known as a Neural Network.

The concept of a neural network was first proposed in 1944 by Warren McCullough and Walter Pitts, researchers at the University of Chicago [5]. A neural network consists of a large number of densely interconnected active nodes, resembling a simplified depiction of the human brain composed of neurons. Most nodes are organized into layers, where a single node can be connected to multiple nodes in subsequent layers. Each node receives data processed by the preceding nodes, applies its processing, and sends the new data to the nodes of the next layer.

For each incoming connection, a node assigns a set of "weights." When the network is active, the node receives different data and applies the assigned weight to it. These weighted products are then summed, and if the result exceeds a certain threshold (called the "threshold"), it is passed to the next layer; otherwise, it is discarded. During training, the weights and thresholds are continuously adjusted to reduce the error in the algorithm's predictions [6].

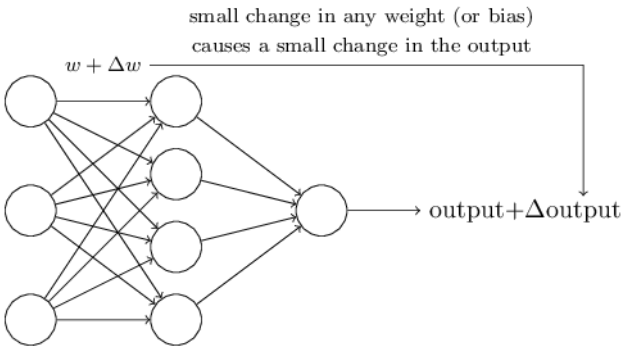

Figure 1.2: Diagram depicting the relationship between weight changes from one layer of the neural network to another and the change in output [7]

Considering the example of the MNIST dataset [3], a potential neural network utilizing that input data could be structured as follows:

1.4.1 Deep Feedforward Networks

Deep Feedforward Networks, also known as Feedforward Neural Networks or Multilayer Perceptrons (MLPs), aim to approximate the function $f(x)$ to the true prediction y. For instance, in a classification task, the neural network's goal is to calculate $f(x)$ such that, given $\mathbf{x}$ as the input vector and y as the correct category value, $y = f(x)$ holds true. Achieving an exact equality is often challenging, so the ultimate objective of this neural network type is to find a function $f(x, \theta)$ that, given certain additional input parameters θ, closely approximates y.

The model is termed "feedforward" because information flows from the input vector $\mathbf{x}$

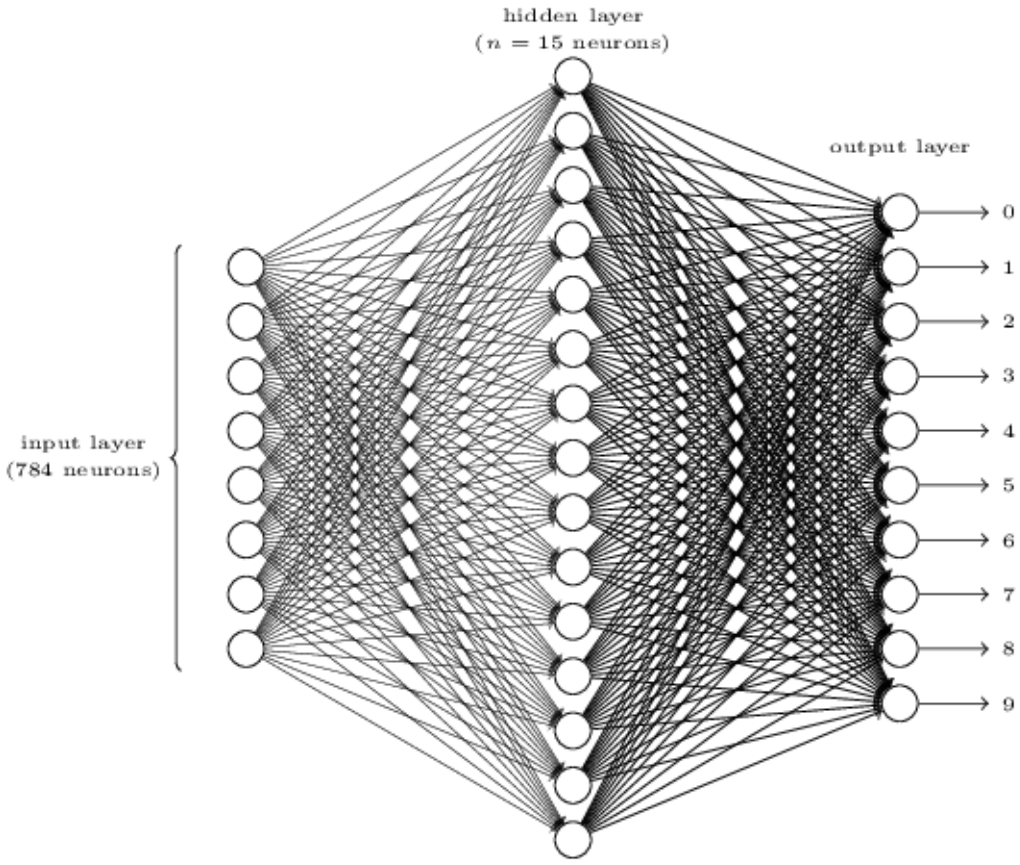

Figure 1.3: Example of a feedforward neural network processing MNIST dataset [7]

through the intermediate layers used to define $f(x, \theta)$ and ultimately to the output y. The input layer is the initial layer of the neural network containing the input data. The number of nodes in this layer equals the number of features in the dataset. It remains unchanged during training.

Fully Connected (FC) Layer: Each neuron in an FC layer is connected to all nodes in the previous layer. The number of nodes in this layer is a user-defined parameter that can be modified during training.

Convolutional Layer: The convolutional layer is often used to process 2D image data. It performs convolution operations.

Pooling Layer: The pooling layer is typically used after a convolutional layer to reduce input dimensions and model complexity. It involves an operation that takes a set of input values and produces a single value, such as max pooling.

Dropout Layer: The dropout layer is often used after a fully connected layer to reduce model complexity and mitigate overfitting [8]. It involves randomly deactivating nodes during training.

Output Layer: The output layer is the final layer of the neural network and contains the output data. The number of nodes in this layer equals the number of classes in the dataset. It remains unchanged during training.

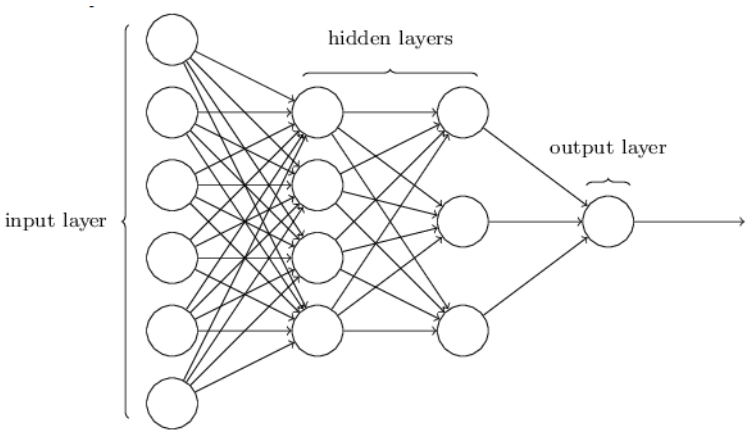

Figure 1.4: Diagram illustrating a neural network divided into input layer, hidden layers, and output layer [7]

1.4.2 Common Layers in Deep Feedforward Networks

- **Input Layer**: The first layer of the neural network containing input data. The number of nodes in this layer matches the number of features in the dataset. This layer remains fixed during training.
- **Fully Connected (FC) Layer**: Each neuron in an FC layer is connected to all nodes in the previous layer. The number of nodes in this layer is a parameter that can be

modified during training.

- **Convolutional Layer**: Often used to process 2D image data through convolution operations.
- **Pooling Layer**: Typically used after a convolutional layer to reduce input dimensions and model complexity. It involves operations like max pooling.
- **Dropout Layer**: Frequently used after a fully connected layer to reduce model complexity and overfitting risk. It involves randomly deactivating nodes during training.
- **Output Layer**: The final layer of the neural network containing output data. The number of nodes matches the number of classes in the dataset. This layer remains fixed during training.

1.4.3 Imbalanced Classification

Under optimal conditions, a dataset exhibits homogeneous classification, meaning that there is approximately an equal number of specific elements for each class. However, in practice, the original dataset often suffers from class imbalance, where different classes have unequal representation. Over the years, many techniques have been proposed to address this issue, and a comprehensive treatment of these techniques is provided by He, Haibo and Garcia et al. in [9]. In this thesis, to tackle the imbalanced dataset problem during training, weights calculated using the function have been incorporated.

1.4.4 Transfer Learning and Fine Tuning

Transfer Learning is a Machine Learning method in which a learning model developed for an initial specific task is reused for a second task. Essentially, the idea is to leverage the features learned during the first task, such as various weights of the cost function or accuracy function, and apply them to the second task's training. As a result, the second training becomes faster and more effective. Most Transfer Learning solutions involve aligning the input space of the source dataset with that of the target dataset. When the source model's dataset (i.e., the model where the first learning task was performed) significantly differs from the second learning task's dataset (target dataset), the Fine Tuning technique can be employed. This approach enhances learning efficiency through Transfer Learning while specializing in the target dataset through Fine Tuning.

Fine Tuning typically involves the following four steps:

1. Pretrain a neural network model, i.e., the source model, on a source dataset (e.g., the ImageNet dataset).

2. Create a new neural network model, i.e., the target model. Copy all parameters from the source model except for the output layer. It is assumed that these model parameters contain knowledge learned from the source dataset and that this knowledge is transferable to the target dataset. It is also assumed that the output layer of the source model is closely related to the labels of the source dataset, and therefore, it is not used in the target model.

3. Add an output layer to the target model, with the number of outputs matching the categories in the target dataset. Initialize the parameters of this layer randomly.

4. Train the target model on the target dataset. The output layer is trained from scratch, while the parameters of all other layers are optimized based on the source model's parameters.

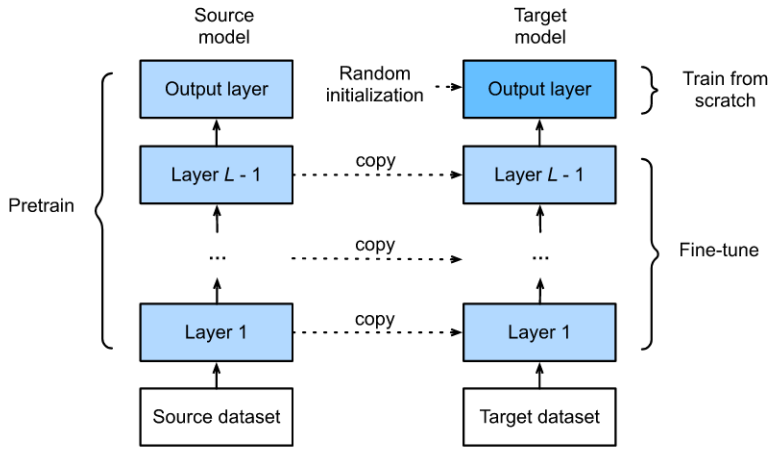

Figure 1.5: Diagram depicting the steps to apply Fine Tuning technique

Chapter 2

Segmentation using Deep Learning and Creation of Ultrasound Image Dataset

2.1 Multiclass Semantic Segmentation for Object Recognition

The field of Computer Vision aims, from a biological perspective, to develop computational models of the human visual system, while from an engineering standpoint, it seeks to build autonomous systems that can perform some of the tasks accomplished by the human visual system. These two objectives are closely linked, as studying the properties of the human visual system often inspires engineers designing autonomous Computer Vision systems, and vice versa, artificial vision algorithms can provide insights into how the human visual system works [10]. These goals are quite complex because, unlike what the human visual system does, a Computer Vision system must tackle an "inverse problem": that is, interpreting the world humans see from one or more images, evaluating properties such as shape, lighting conditions, and color distribution. This is why these systems must rely on models from the theory of physics or probability or rely on Machine Learning techniques with large datasets [11].

One of the typical tasks in Computer Vision is to determine whether an input image contains a specific object or feature. The most common subcategories of recognition tasks are:

- **Image Classification**: The system aims to understand the entire image as a whole, with the goal of classifying it by assigning a specific label.
- **Object Detection**: Involves locating an object within an image, encompassing both classification and localization. It is used to analyze more realistic scenarios where multiple objects may exist in an image [12].

- **Semantic Segmentation**: Semantic segmentation involves dividing an image into regions that belong to different semantic classes. Unlike object detection, semantic segmentation recognizes and understands what is in the images at the pixel level. Its labeling and predictions are thus at the pixel level [13]. Based on the number of categories, semantic segmentation can be:
  - Binary: Two categories are used, where generally one represents the object category to be identified, while the other represents everything else (the background).
  - Multi-class: More than two categories are used to identify multiple specific objects within the image.
- **Instance Segmentation**: An extension of semantic segmentation, where the goal is to distinguish instances of the same class into different segments. Instance segmentation can be defined as the technique to simultaneously solve both the object detection and semantic segmentation problems. Automating this process is challenging because the number of instances is not known in advance, and evaluating obtained instances is not based on pixel-level, as it was in semantic segmentation.

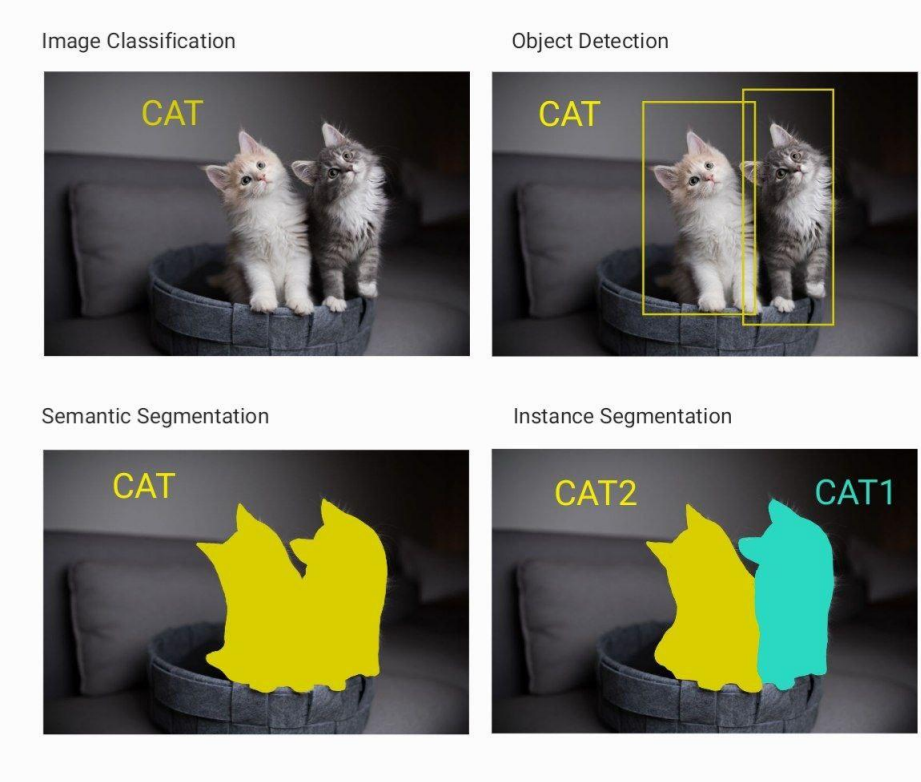

Figure 2.1: Most common recognition tasks

2.2 Dataset Generation

The articulation of the hand consists of 27 bone components that divide the hand and wrist into three joint compartments (carpus, metacarpus, and phalanges) and five joint sections:

- Wrist joints
- Carpal joints
- Metacarpal joints
- Finger joints

In a normal joint, the articular surfaces are enclosed within a capsule lined by a membrane called the synoviuml membrane. The articular surfaces are covered by a smooth surface, the articular cartilage, which allows smooth movement between them. The joint space is "lubricated" by a fluid called synoviuml fluid, which facilitates optimal sliding. When the cartilage is consumed or altered due to physiological wear or abnormal joint usage (repetitive movements in work activities), or due to traumas or fractures affecting the joint surfaces, or due to other inflammatory causes affecting the synoviuml fluid, such as primary osteoarthritis or rheumatoid arthritis, the affected joint often becomes stiff and painful, sometimes altering the axis of the finger, leading to deformity [14]. Therefore, when a patient reports hand pain with undefined location and characteristics, an ultrasound examination is recommended, as it allows visualizing tissues that cannot be analyzed by other methods such as X-rays or magnetic resonance imaging (MRI). Ultrasound is a medical diagnostic imaging technique that uses ultrasound waves and is based on the principles of emitting echoes and transmitting ultrasonic waves. In particular, each ultrasound is generated by a transducer in contact with the patient's skin using a gel. The transducer emits ultrasound with a frequency determined by the operator, considering that a higher frequency provides higher image resolution but is more effective in imaging superficial layers of the joint. The ultrasound wave emitted by the transducer, when it encounters bone tissue, cannot penetrate deeper and bounces back. The same transducer is capable of capturing the returning signal, which is then processed by a computer and displayed on a monitor. Thus, ultrasound is always an operator-dependent procedure, as it requires specific manual skills and observation abilities, as well as image understanding and clinical experience.

Before defining the Machine Learning model to be used, it is important to analyze the available data. In the case of semantic segmentation tasks, it's useful to evaluate factors like lighting and contrast levels of the images, their resolution, and the presence of artifacts. For instance, one might decide to crop images to enhance the objects to be classified, as cropping reduces the number of pixels belonging to one or more less important classes (such as the background). Another crucial consideration is the number of images. If the dataset is small, data augmentation techniques are often applied to increase the number of images. The available dataset consists of 60 images depicting sagittal sections of the metacarpus, captured by Dr. Luana Mancarella at the Rizzoli Orthopedic Institute in Bologna. The

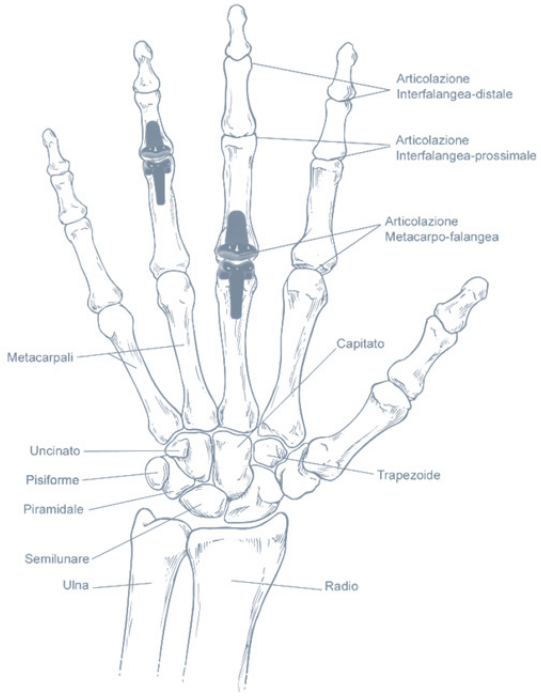

Figure 2.2: Anatomy of the human hand [14]

images show features such as bone, tendon, synoviuml cavity, and articular cartilage, as depicted in the figure below.

For annotating the images, the Apeer software [1] was used. This software allows users to draw masks on uploaded images and export annotations in TIFF format. Apeer is a web application supported by common browsers and is user-friendly, thanks to an intuitive dashboard and comprehensive documentation. For ease of implementation, the images were converted to PNG format, and three distinct datasets were created:

- Bone-Tendon Dataset
- synoviuml-Tendon Dataset
- synoviuml-Bone Dataset

Each ultrasound is associated with a single mask image that represents the desired output of the machine. Some examples of image-mask pairs are shown below.

[1] link to Apeer web application: https://www.Apeer.com/app

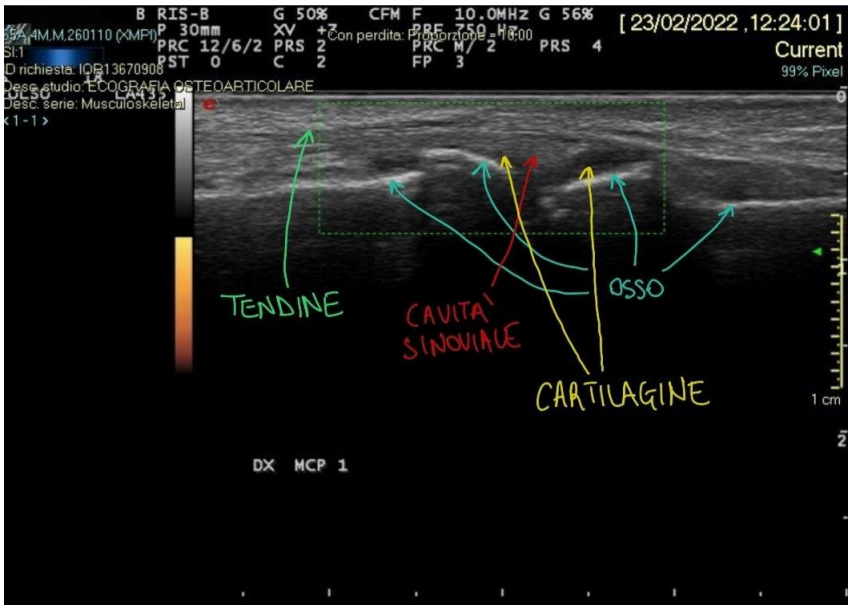

Figure 2.3: Elements within a generic ultrasound image from the dataset

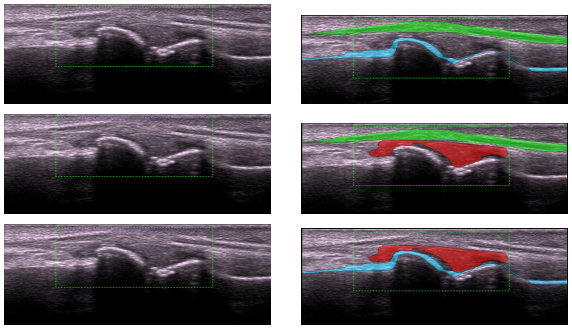

Figure 2.4: On the left, cropped ultrasound images of the metacarpus. On the right, corresponding mask drawings.

Details about the dataset formation process are summarized in Table 2.1.

Description	Value
Number of collected images	2148
Number of analyzed patients	80
Number of metacarpal images	620
Number of annotated images	60

Table 2.1: Table of details describing the dataset formation.

The dataset exhibits class imbalance, where each image has one or more classes with a significantly lower number of pixels compared to others. This phenomenon is further discussed in Section 1.4.3. In the case of the Rizzoli ultrasound images, in the following mask: The number of background pixels is much higher compared to the pixels of the other

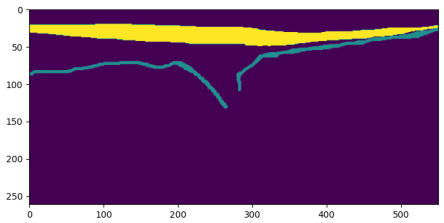

Figure 2.5: Example of a mask from the Rizzoli Orthopedic Institute ultrasound images showing class imbalance.

two classes. Specifically, the background occupies about 90% of the total pixel count, bone occupies 5%, and tendon occupies 5%. Imbalanced classification can pose issues during training: in the example above, if the machine is trained without applying techniques to address imbalanced classification, it may result in predictions of all background, achieving a high accuracy value. For instance, if the machine predicts that all pixels are "background," it will achieve an overall accuracy of 90%.

More detailed technical information about the images in each dataset is provided below.

Bone-Tendon Dataset	Background	Bone	Tendon	Total
Class Count	0	1	2	
Image Count	48	48	48	48
Total Images in Train Set	12	12	12	12
Total Images in Validation Set	12	12	12	12
Total Images in Test Set	3	3	3	3
Total Pixels in an Example Image	46710	1248	2218	50176
Average Pixel Count in Dataset	2770454	50380	189726	3010560
Average % of Pixels per Image	92.03%	1.67%	6.30%	100.00%
Weights Assigned for Each Class	0.36	19.92	5.29	

Table 2.2: Details about the Bone-Tendon dataset composition.

synovium-bone Dataset	Background	Synovium	Bone	Total
Class Count	0	1	2	
Image Count	48	48	48	48
Total Images in Train Set	12	12	12	12
Total Images in Validation Set	12	12	12	12
Total Images in Test Set	3	3	3	3
Total Pixels in an Example Image	45342	3854	980	50176
Average Pixel Count in Dataset	2646475	116783	46598	3010560
Average % of Pixels per Image	87.9%	3.88%	1.55%	100.00%
Weights Assigned for Each Class	0.35	8.02	20.1	

Table 2.3: Details about the synovium-bone dataset composition.

synovium-tendon Dataset	Background	Synovium	Tendon	Total
Class Count	0	1	2	
Image Count	48	48	48	48
Total Images in Train Set	12	12	12	12
Total Images in Validation Set	12	12	12	12
Total Images in Test Set	3	3	3	3
Total Pixels in an Example Image	41045	3854	5277	50176
Average Pixel Count in Dataset	2468047	115217	176416	3010560
Average % of Pixels per Image	82.00%	3.83%	5.86%	100.00%
Weights Assigned for Each Class	0.37	7.98	5.21	

Table 2.4: Details about the synovium-tendon dataset composition.

2.3 U-Net Architecture

For semantic segmentation tasks, it's important to carefully choose the network architecture based on the specific dataset. The U-Net is a fully convolutional network designed for image segmentation. Although originally developed for biomedical image segmentation, it can be applied to various image segmentation tasks. The network is based on an encoder-decoder architecture with skip connections between corresponding encoder and decoder layers. The network is trained end-to-end and can be used for image segmentation tasks with minimal preprocessing and post-processing. For more details on this type of network, refer to Section 3.3.

Chapter 3

Implementation

3.1 User Interface

The software can be utilized through a command-line interface. The interface has been created using the *argparse* library in Python, which allows the developer to define a program and the required arguments. *argparse* then parses these arguments from `sys.argv`, the list of command-line arguments passed to a Python script. The argparse module also automatically generates help and usage messages and raises errors when users provide invalid arguments to the program. To launch the application, execute the command:

```
python3 main.py
```

The user interface consists of two programs:

- **Train and Predict**: Allows training a neural network and performing prediction tests.
- **Only Predict**: Enables running prediction tests on an already trained neural network.

For each of the programs, various parameters and options can be specified as shown in the figures.

```
> python3 main.py -TP --help
usage: TP [options]

optional arguments:
  -h, --help            show this help message and exit
  -TP, --train-and-predict
                        choose train and predict program
  -P, --only-predict    choose only predict program
  -v, --version         show program's version number and exit
  -e EPOCHS, --epochs EPOCHS
                        number of epochs to train a model
  -pat PATIENCE, --patience PATIENCE
                        number of patience for earlystopping callback during fit
  -mon MONITOR, --monitor MONITOR
                        Metric to monitor for earlystopping and modelcheckpoint callbacks during fit
  -c [COMMENT], --comment [COMMENT]
                        general comment printed to log file
  -d DIR_DATASET, --dir-dataset DIR_DATASET
                        absolute directory path where dataset is stored
  -m {UNET,TRANSFER_LEARNING_VGG16,TRANSFER_LEARNING_VGG19}, --model-name {UNET,TRANSFER_LEARNING_VGG16,TRANSFER_LEARNING_VGG19}
                        name of the model to train
```

Figure 3.1: Options and parameters for the "Train and Predict" program

```
> python3 main.py -P --help
usage: P [options]

optional arguments:
  -h, --help            show this help message and exit
  -TP, --train-and-predict
                        choose train and predict program
  -P, --only-predict    choose only predict program
  -v, --version         show program's version number and exit
  -mw MODEL_WEIGHTS, --model-weights MODEL_WEIGHTS
                        absolute file path where weights of a model are saved
  -c [COMMENT], --comment [COMMENT]
                        general comment printed to log file
  -m {UNET,TRANSFER_LEARNING_VGG16,TRANSFER_LEARNING_VGG19}, --model-name {UNET,TRANSFER_LEARNING_VGG16,TRANSFER_LEARNING_VGG19}
                        name of the model to load
  -d DIR_DATASET, --dir-dataset DIR_DATASET
                        absolute directory path where dataset is stored
```

Figure 3.2: Options and parameters for the "Only Predict" program

3.2 Dataset Description and Loading

The ultrasound images used in this thesis are present in the Orthopedic Institute Rizzoli's database in DICOM format. The DICOM format is a standard format for exchanging medical images and is often chosen for its widespread use and ability to contain additional information, such as image acquisition date, patient name, performing physician's name, diagnosing physician's name, and more. Although the *pydicom* module exists, which allows reading DICOM files and accessing the contained information, for implementation simplicity, the images were exported from the database in TIFF format and then converted to PNG. Once the images were collected and the format was defined, an analysis of the dataset was performed to determine the appropriate preprocessing steps. Since the segmentation classes contain elements in an imbalanced manner, it was decided to crop the images to reduce the pixel count of the most prevalent class (background), thus performing downsampling. The images were saved in RGB format, and this color format was retained as the neural networks used in this thesis were implemented using the "Keras" library [15] based on TensorFlow [16], which supports three-channel image inputs, such as RGB. Subsequently, associated masks were created using the Apeer web application. The masks exported from Apeer are already normalized, and each pixel is represented by an integer value ranging from 0 to N-1, where N is the number of classes. As the created datasets have 3 classes, each pixel can take one of the following values: 0, 1, 2. For instance, in the bone-tendon dataset, the value 0 represents the background, the value 1 represents the bone, and the value 2 represents the tendon. Thus, a dataset consisting of 60 images of size 547x261 pixels was obtained, with each image having an associated mask of the same size. The OpenCV library [17] and its *cv2* module in Python were used to load the dataset, enabling the loading of images and associated masks in PNG format and converting them to NumPy arrays. NumPy arrays are multidimensional arrays that allow the representation of matrices and vectors. Upon loading the images, resizing was performed on both images and masks to ensure they all have the same size of 224x224, necessary for the input layer of the VGG neural network.

3.3 Model Creation - VGG Networks

In the realm of Machine Learning, one or more neural networks must be employed. As previously defined, a neural network is a mathematical model composed of artificial "neurons" that aims to resemble a biological neural network. For tasks involving images, convolutional neural networks are commonly used. The U-Net [18] was developed by Olaf Ronneberger et al. with the goal of improving the efficiency of biomedical image segmentation. Its architecture consists of two parts: the first is called the encoder or contraction path, which locates significant context features in the image as its resolution decreases; the second part is called the decoder or expanding path, which accurately localizes image details as the resolution increases. Since the decoder is applied subsequently to the encoder, and the layers of the encoder typically mirror those of the decoder, the architecture assumes a "U" shape, hence its name.

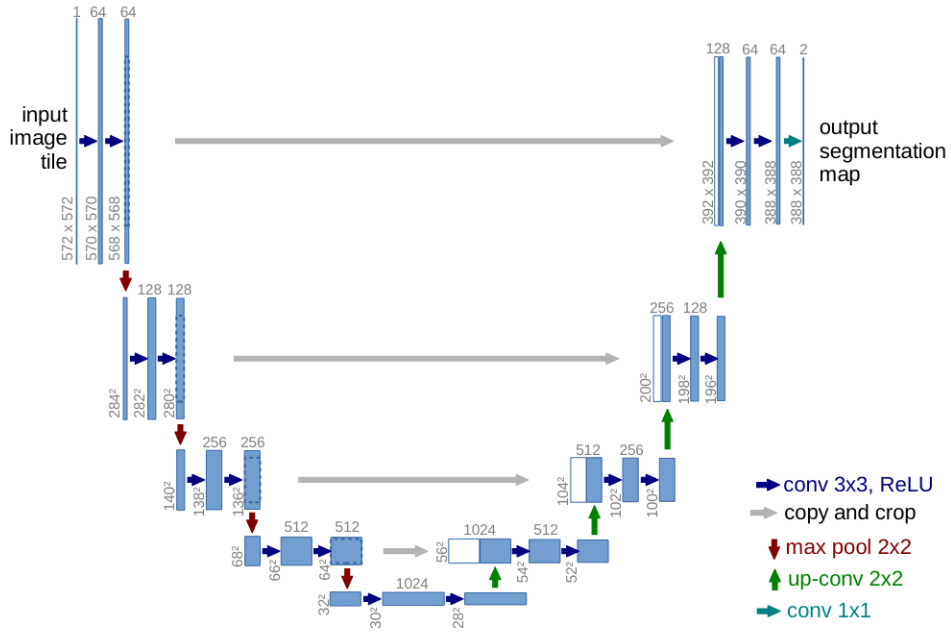

Figure 3.3: U-Net architecture (example for 32x32 pixels with the lowest resolution). Each blue rectangle corresponds to a multi-channel feature map. The number of channels is indicated above the rectangle. The x-y dimension is provided at the lower-left corner of the rectangle. White rectangles represent feature maps. Arrows indicate various operations as specified in the legend. [18]

3.3.1 VGG Network

A VGG block consists of a sequence of convolutions with a 3 x 3 kernel and padding of 1 (maintaining height and width), followed by a 2 x 2 max-pooling layer with a stride of 2 (halving height and width after each block) [13]. The VGG network can be divided into two parts: the first primarily consists of convolutional and pooling layers, and the second comprises fully connected layers. The convolutional layers are grouped into non-linear transformations that maintain dimensionality, followed by resolution reduction, as illustrated in the figure.

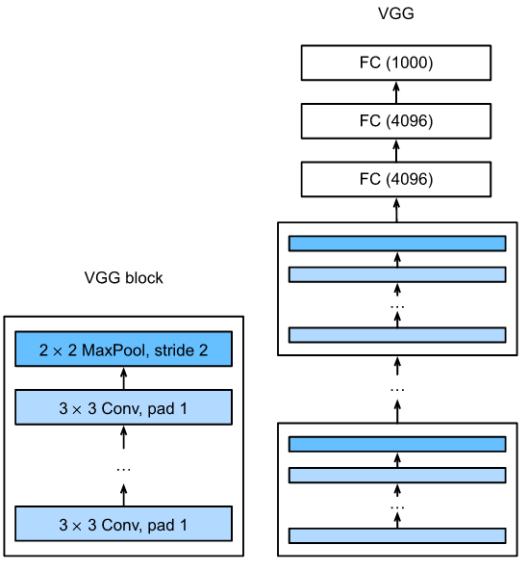

Figure 3.4: Block diagram of a VGG network [13]

The original VGG network had 5 convolutional blocks, with the first two having a single convolutional layer, and the last three containing two convolutional layers each. The first block has 64 output channels, and each subsequent block doubles the number of output channels until it reaches 512. Since this network uses 8 convolutional layers and 3 fully connected layers, it is often referred to as VGG-11 [13]. The convolutional part of the network connects different VGG blocks in succession, and this convolution grouping has remained relatively unchanged over the past decade, despite significant modifications in specific choices of operations. As such, the term "VGG" refers to a family of networks rather than just a specific manifestation; in this regard, the paper by Simonyan and Zisserman (2014) [19] describes various VGG variants.
The models used in this thesis employ the VGG neural network architecture [19]. Created by Karen Simonyan and Andrew Zisserman in 2014, the VGG network is a convolutional neural network that has achieved good results in various fields, such as image classification. In particular, the VGG network reflects the concept of depth: the initial layers are larger in size and have a small number of kernels, and as the network progresses, the number of kernels

increases while the dimensions decrease. In this thesis, VGG-16 and VGG-19 networks were used, differing in the number of layers: 16 and 19, respectively. The Keras library was employed to create each model, enabling the creation of neural networks in a simple and fast manner. Since this thesis focused on medical image segmentation, modification of the VGG16 neural network architecture was necessary, specifically by removing the top layers (fully connected layers) and adding a symmetric decoder block to the encoder block. Thus, the architecture takes on the "U" shape of the U-Net. During model loading, pre-trained weights from ImageNet [20] were loaded. ImageNet is a dataset of 14 million images divided into 1000 classes.

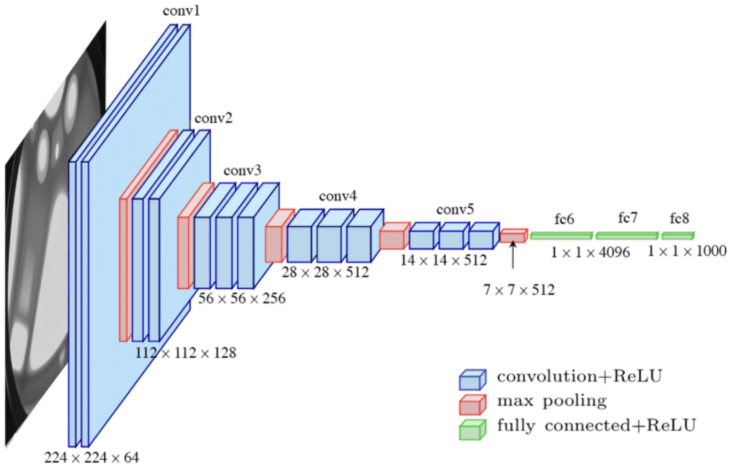

Figure 3.5: Architecture of the VGG16 neural network used for image classification[1]

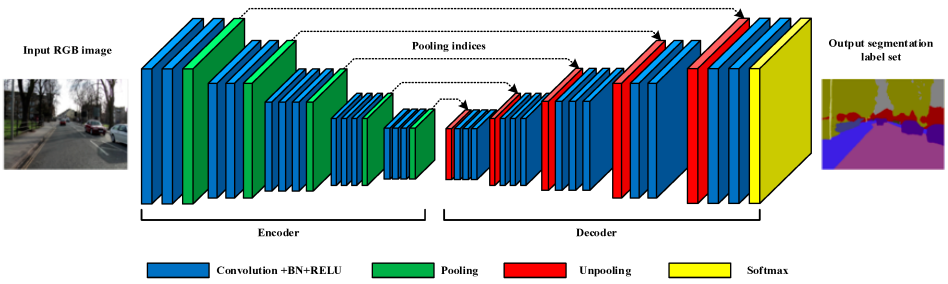

Figure 3.6: Architecture of the VGG16 neural network used in this thesis for semantic segmentation[2]

[1]Image link: https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484
[2]Image link: https://wikidocs.net/164366

3.4 Training

For model compilation, the decision was made to utilize the Adam optimizer [21], which is an optimizer that employs the "momentum" technique to accelerate the training process. Furthermore, the Categorical Crossentropy loss function was selected, which is a loss function used for multiclass classification.

During the training phase, the previously created dataset was utilized, divided into two parts: 80% of images for training and 20% of images for validation.

Also, during model compilation, the metrics to evaluate the training progress must be specified. Specifically:

- **Accuracy**: A metric commonly used for classification, defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

  It represents the sum of correctly predicted values divided by the sum of all values. While intuitive, it is less useful with imbalanced datasets, as in the case under consideration.

- **IoU (Intersection Over Union)**: IoU is an index of overlap intersection, used for binary semantic segmentation. It is calculated as:

$$IoU = \frac{TP}{TP + FP + FN} \tag{3.2}$$

  Here, TP is the number of correctly classified pixels, FP is the number of pixels falsely classified as positive, and FN is the number of pixels falsely classified as negative. In the case of multiclass semantic segmentation, IoU is calculated for each class.

- **MeanIoU**: A metric used for multiclass semantic segmentation, calculated as:

$$MeanIoU = \frac{1}{N} \sum_{i=1}^{N} IoU_i \tag{3.3}$$

  where N is the number of classes.

- **Precision**: A metric used for multiclass classification, defined as:

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

  It represents the sum of correctly predicted values divided by the sum of all values predicted as positive. For multiclass tasks, various types of average precision can be calculated.

- **Recall** or **Sensitivity**: A metric used for multiclass classification, defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

  It represents the sum of correctly predicted values divided by the sum of all true positive values. Similar to Precision, different types of averages can be calculated for multiclass tasks.

- **AUC on PR curve (Area Under Curve)**: Precision and Recall are useful measures of prediction success, particularly for imbalanced classes. Precision is a measure of result relevance, while Recall is a measure of how many relevant results are returned. The PR curve represents the trade-off between Precision and Recall for various thresholds, with the Area Under Curve (AUC) indicating the classifier's ability to distinguish between classes.

- **F1 score**: A metric used for multiclass classification, defined as:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{3.6}$$

  It represents the harmonic mean between Precision and Recall. As Precision and Recall are calculated in a binary manner, an F1 score is obtained for each class. For multiclass tasks, various types of average F1 score can be calculated.

Due to the use of the Fine Tuning technique, training is divided into two phases: the initial training phase of the pre-trained model, and the fine-tuned model training phase. Refer to the Fine Tuning section for more details. During training, the TensorFlow API *ModelCheckpoint* is used to save model weights that achieve the best value of a variable named "monitor." This allows using the weights for the "only predict" program. The "monitor" variable is assigned as an input parameter by the user, based on the metric they wish to minimize or maximize. The "monitor" variable is also used for *EarlyStopping*, which halts training if the monitored metric does not improve for a user-specified number of epochs. All training parameters and results are then saved in a .json file, used for model testing.

3.5 Predictions

Predictions are made on the test dataset using the fine-tuned model. Subsequently, to assess the quality of predictions, metrics defined in the previous section are calculated, including:

- **Multiclass Confusion Matrix**: This matrix displays the frequency with which data has been classified into a class different from the correct one. In other words, it provides a visual representation of True Positives (TP), True Negatives (TN), and False Negatives (FN) predictions. The matrix has predicted classes on the horizontal axis and true classes on the vertical axis.

- **Classification Report**: A table showing precision, recall, F1-score, and support for each class, provided by the Sklearn library [22].

Using the *pyplot* module of *matplotlib* [23], predicted images are saved, as well as plots of training, validation, and test metrics.

Chapter 4

Test Results

Tests were conducted using different images than those in the training set to effectively evaluate the algorithm. In fact, if predictions were made using the same images as input to training, it would lead to overfitting behavior. Overfitting occurs when the model becomes too closely tied to the input data's characteristics during training, resulting in relatively good performance with that data, but poor performance with new data during testing.
All tests were conducted on the same physical machine located in the CASY laboratory, specifically a workstation with an octa-core Intel i9 processor, 64 GB RAM, and 2 NVIDIA QUADRO P4000 GPUs. During each training run, using the Tensorflow library, the dataset was divided into three parts. For all three datasets:

- Train set: 48 images + 48 masks
- Validation set: 12 images + 12 masks
- Test set: 3 images + 3 masks

The main configuration parameters for each training run are:

- Number of epochs: 160 epochs for training and 40 for fine-tuning
- Image dimensions: (224, 224)
- Number of classes = 3
- Batch size = 8
- Optimization algorithm = "Adam"
- Loss function = "Categorical Crossentropy"

For training evaluation, accuracy and loss function graphs are presented, with values plotted for each epoch of training. For evaluating the neural network's performance during testing, confusion matrix graphs and classification reports are provided, containing the metrics described in Chapter 3.

4.1 Predictions with VGG16 U-Net in Transfer Learning and Fine Tuning

Predictions on the bone-tendon dataset

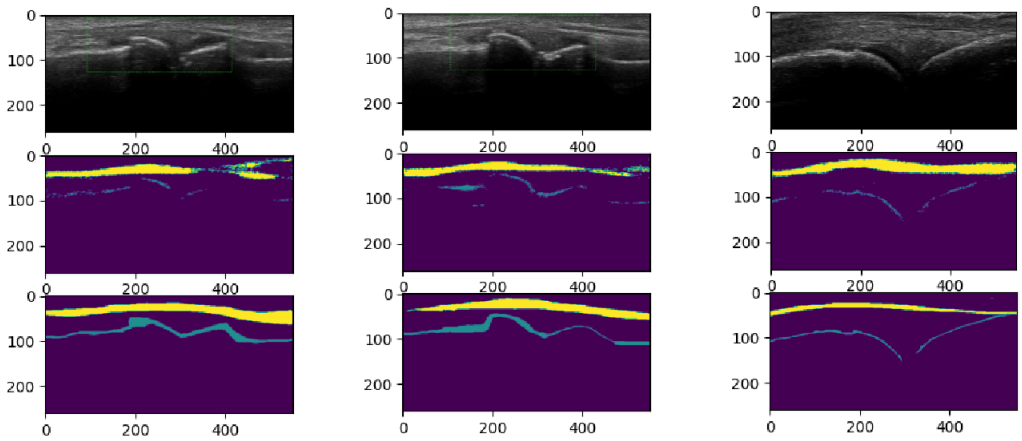

Figure 4.1: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the bone-tendon dataset

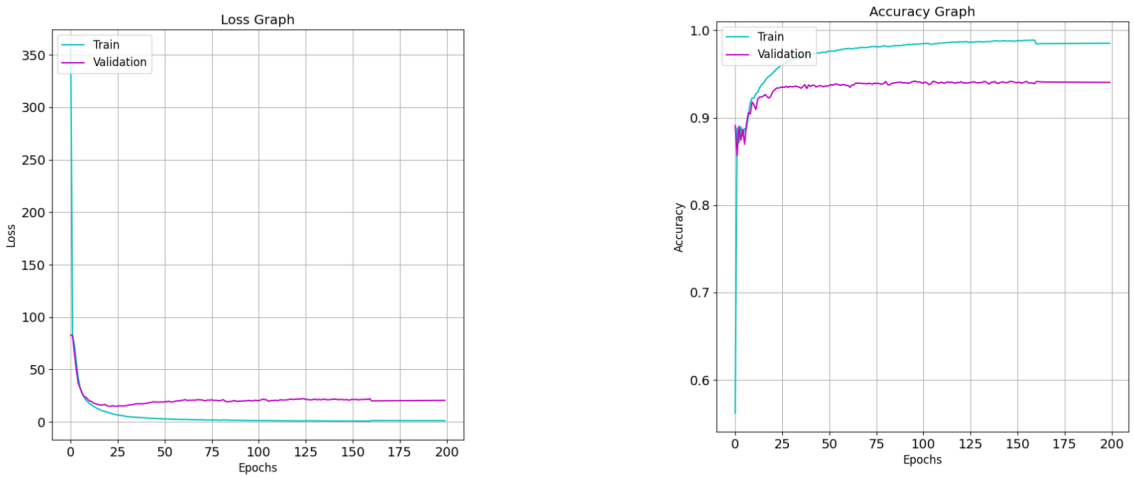

Figure 4.2: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the bone-tendon dataset

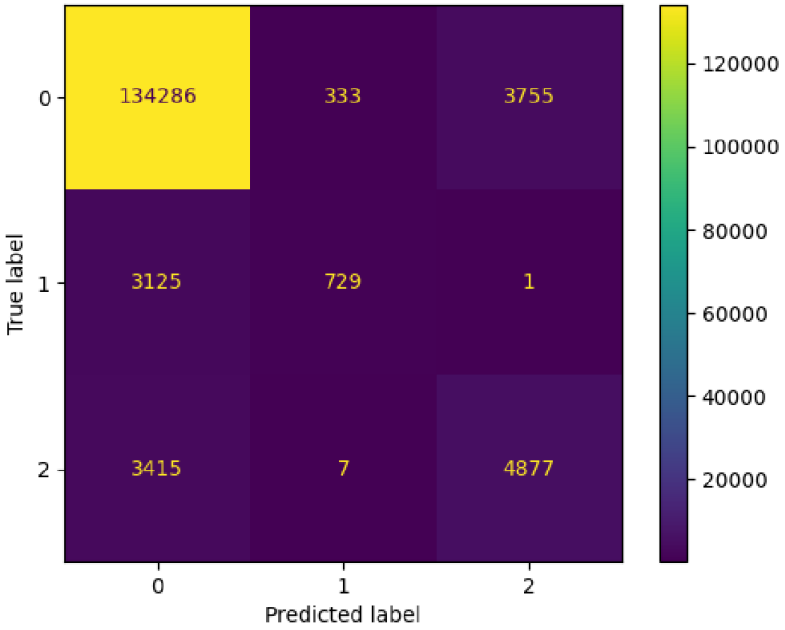

Figure 4.3: Multiclass confusion matrix graph for predictions using the VGG16 network on the bone-tendon dataset

	precision	recall	f1-score
Background	95%	97%	96%
Bone	68%	19%	30%
Tendon	56%	59%	58%
accuracy	93%	93%	93%
macro avg	73%	58%	61%
weighted avg	93%	93%	92%

Table 4.1: Classification report table for predictions using the VGG19 network on the bone-tendon dataset

Predictions on the synovium-bone dataset

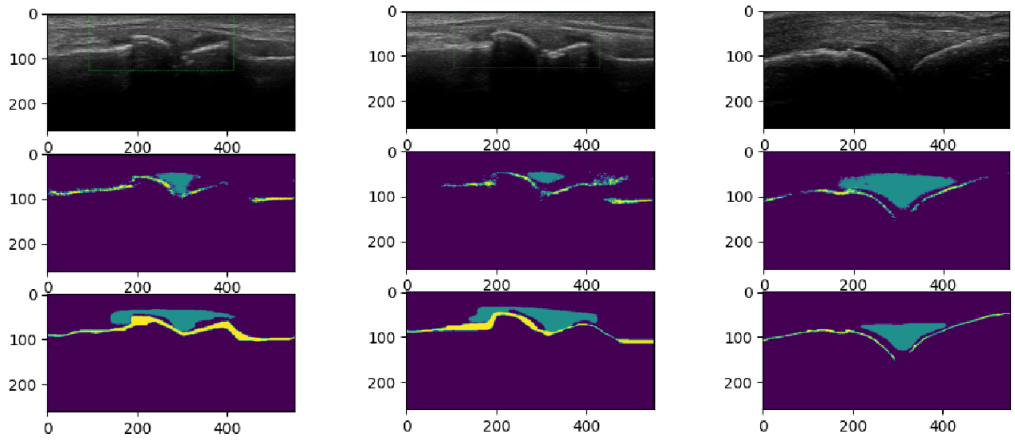

Figure 4.4: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the synovium-bone dataset

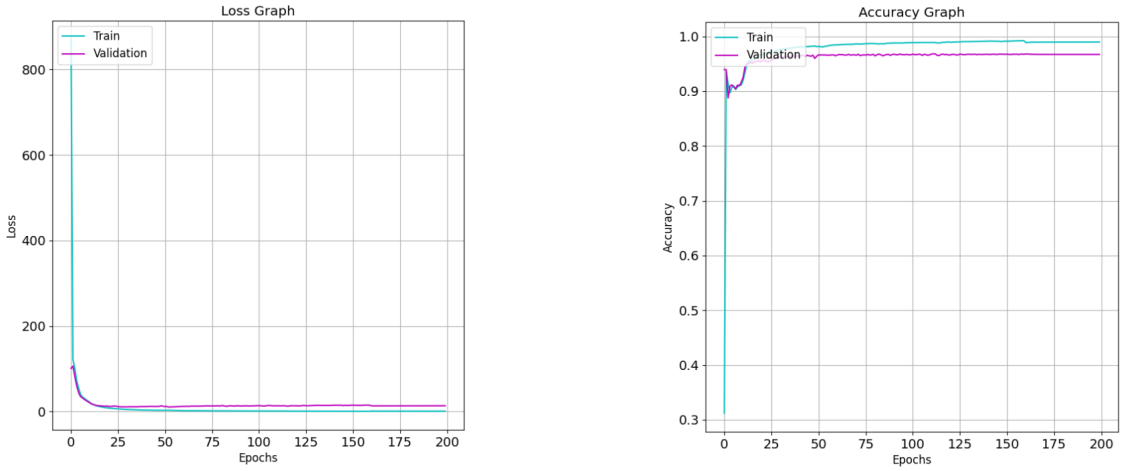

Figure 4.5: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the synovium-bone dataset

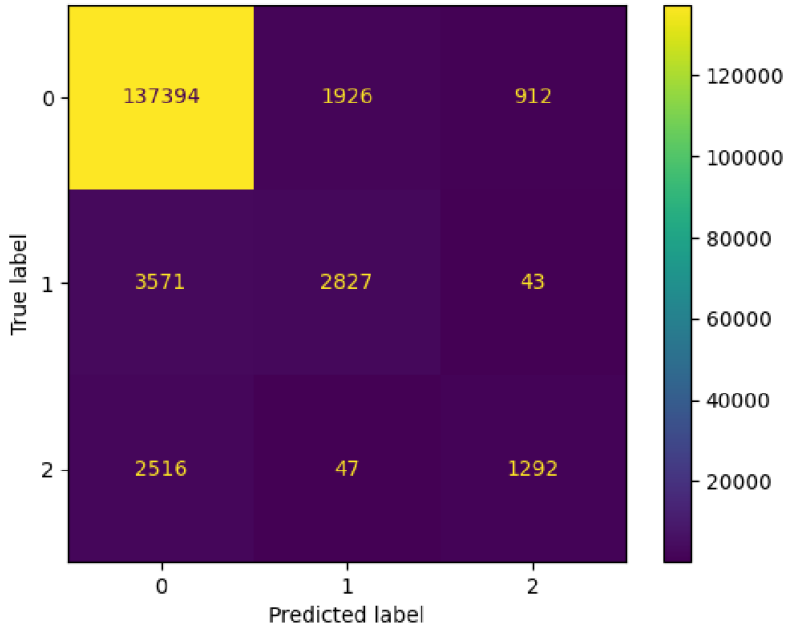

Figure 4.6: Multiclass confusion matrix graph for predictions using the VGG16 network on the synovium-bone dataset

	precision	recall	f1-score
Background	96%	98%	97%
synoviuml	59%	44%	50%
Bone	57%	34%	42%
accuracy	94%	94%	94%
macro avg	71%	58%	63%
weighted avg	93%	94%	93%

Table 4.2: Classification report table for predictions using the VGG19 network on the synoviuml-bone dataset.

Predictions on the synovium-tendon dataset

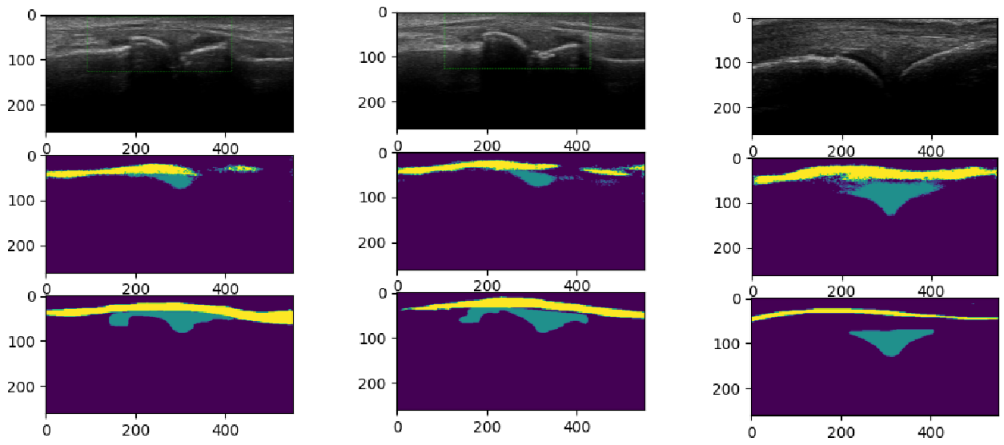

Figure 4.7: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the synovium-tendon dataset

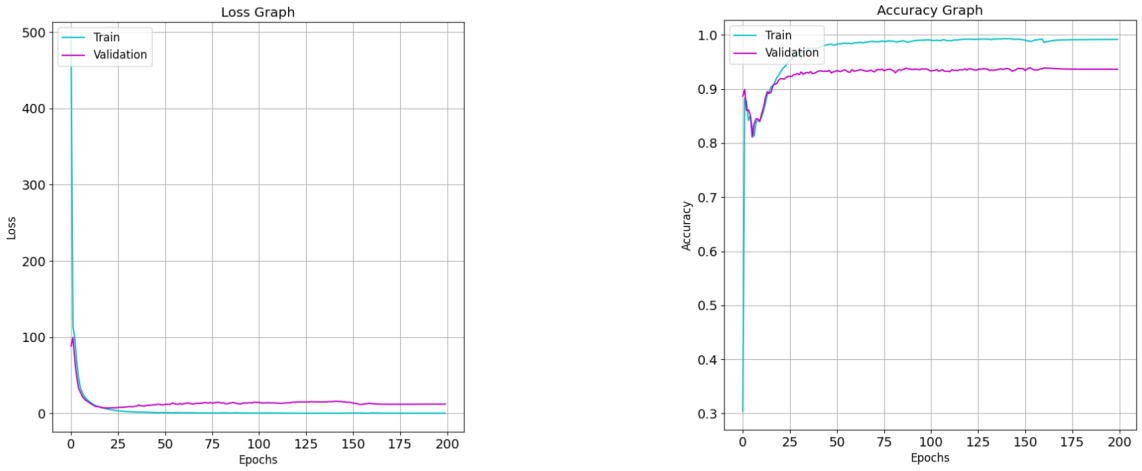

Figure 4.8: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the synovium-tendon dataset

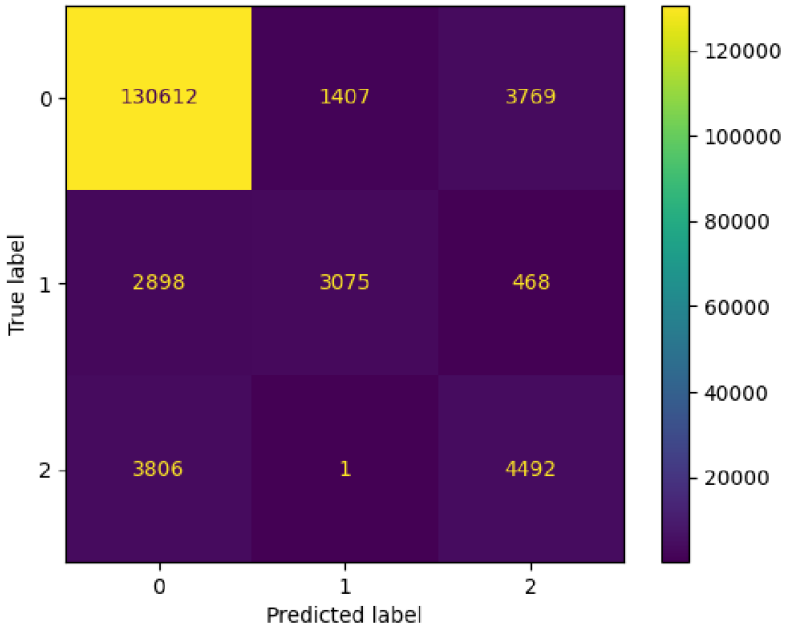

Figure 4.9: Multiclass confusion matrix graph for predictions using the VGG16 network on the synovium-tendon dataset

	precision	recall	f1-score
Background	95%	96%	96%
Synovium	69%	48%	56%
Tendon	51%	54%	53%
accuracy	92%	92%	92%
macro avg	72%	66%	68%
weighted avg	92%	92%	92%

Table 4.3: Classification report table for predictions using the VGG19 network on the synovium-tendon dataset.

4.2 Predictions with VGG19 U-Net in Transfer Learning and Fine Tuning

Predictions on the bone-tendon dataset

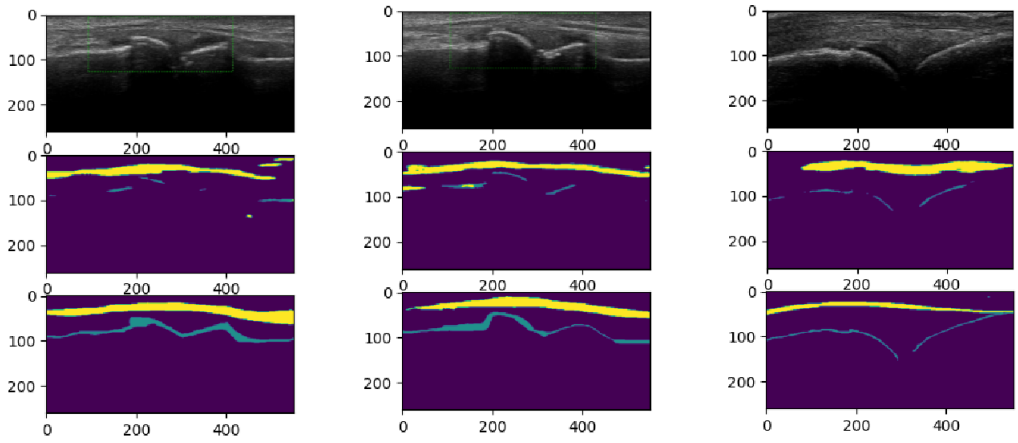

Figure 4.10: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the bone-tendon dataset

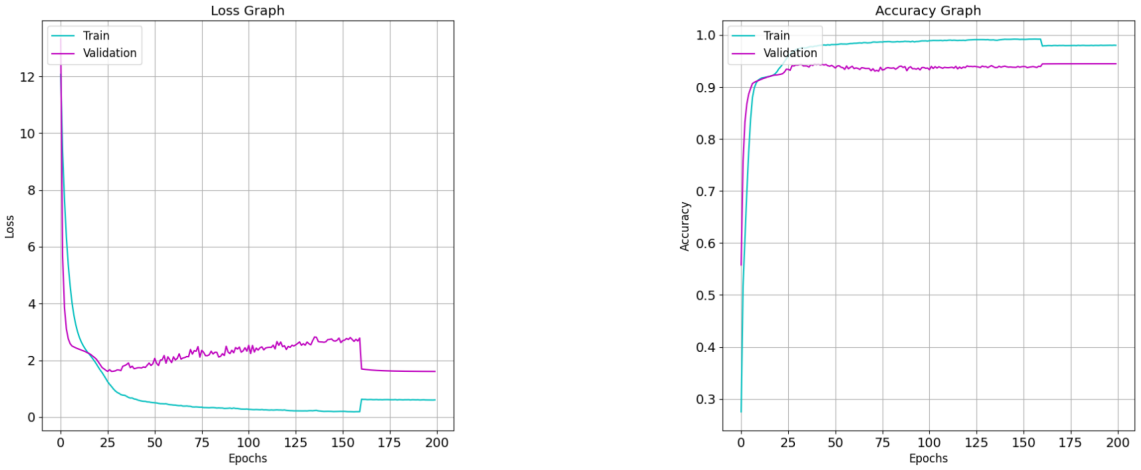

Figure 4.11: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the bone-tendon dataset

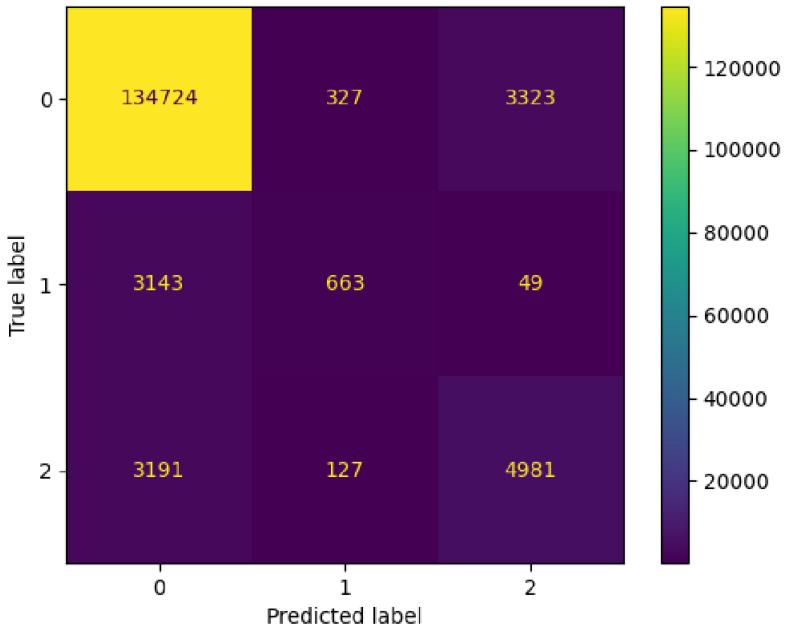

Figure 4.12: Multiclass confusion matrix graph for predictions using the VGG19 network on the bone-tendon dataset

	precision	recall	f1-score
Background	96%	97%	96%
Bone	59%	17%	27%
Tendon	60%	60%	60%
accuracy	93%	93%	93%
macro avg	71%	58%	61%
weighted avg	93%	93%	93%

Table 4.4: Classification report table for predictions using the VGG19 network on the bone-tendon dataset.

Predictions on the synovium-bone dataset

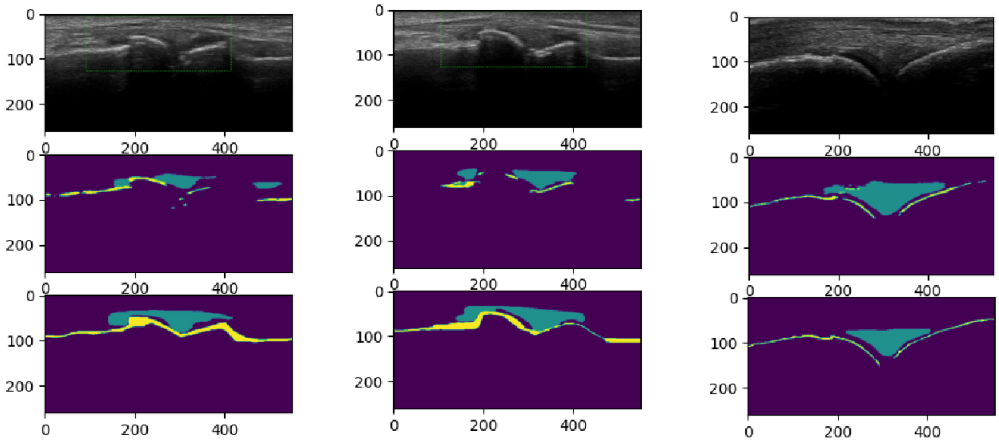

Figure 4.13: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the synovium-bone dataset

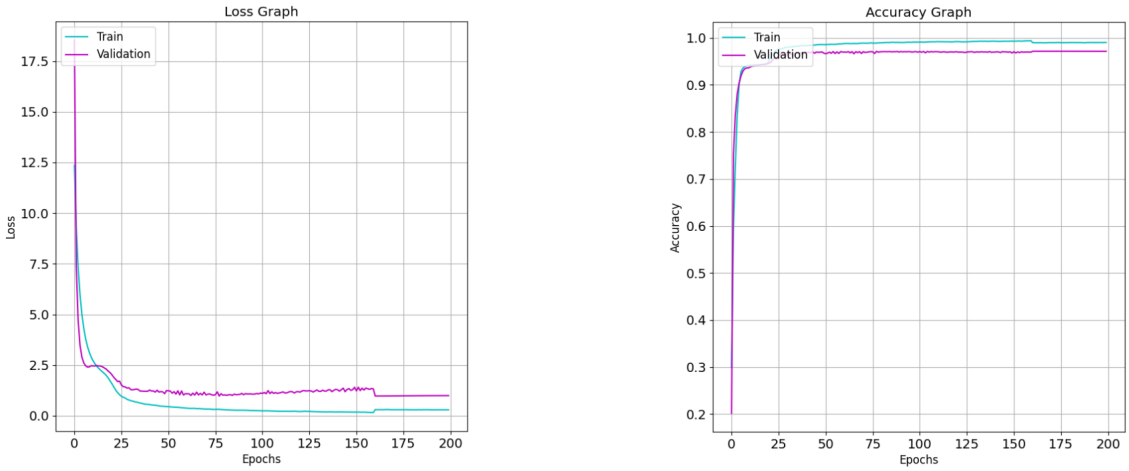

Figure 4.14: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the synovium-bone dataset

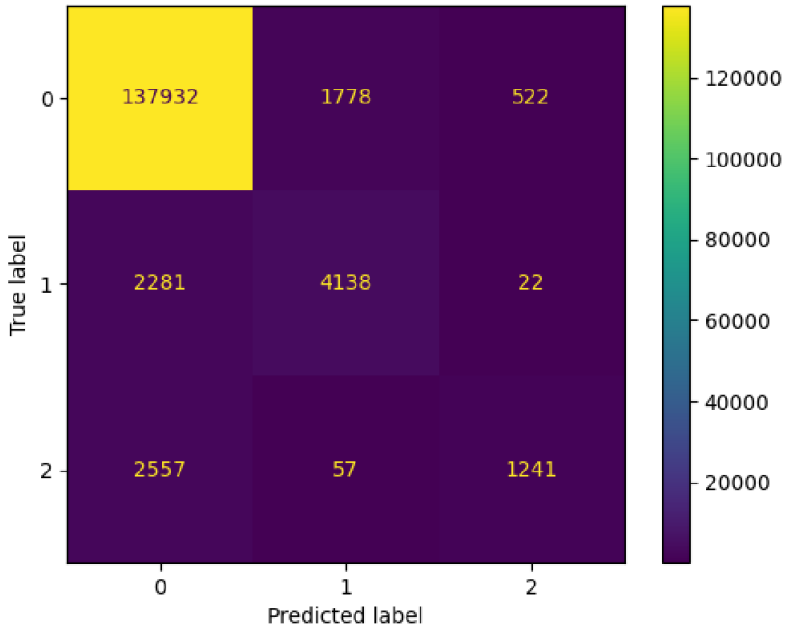

Figure 4.15: Multiclass confusion matrix graph for predictions using the VGG19 network on the synovium-bone dataset

	precision	recall	f1-score
Background	97%	98%	97%
Sinovia	69%	64%	67%
Osso	70%	32%	44%
accuracy	95%	95%	95%
macro avg	78%	65%	69%
weighted avg	95%	95%	95%

Table 4.5: Classification report table for predictions using the VGG19 network on the synovium-tendon dataset.

Predictions on the synovium-tendon dataset

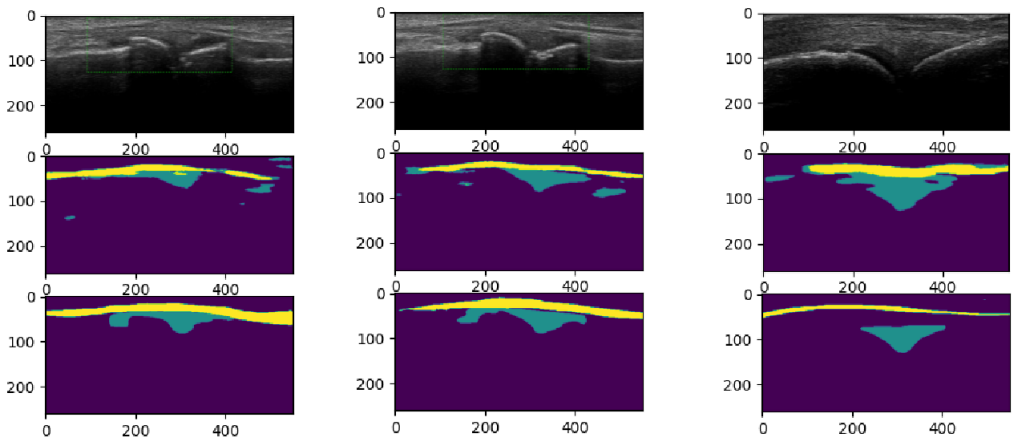

Figure 4.16: From top: Input ultrasound image, machine-generated prediction, associated mask for the input image in the synovium-tendon dataset

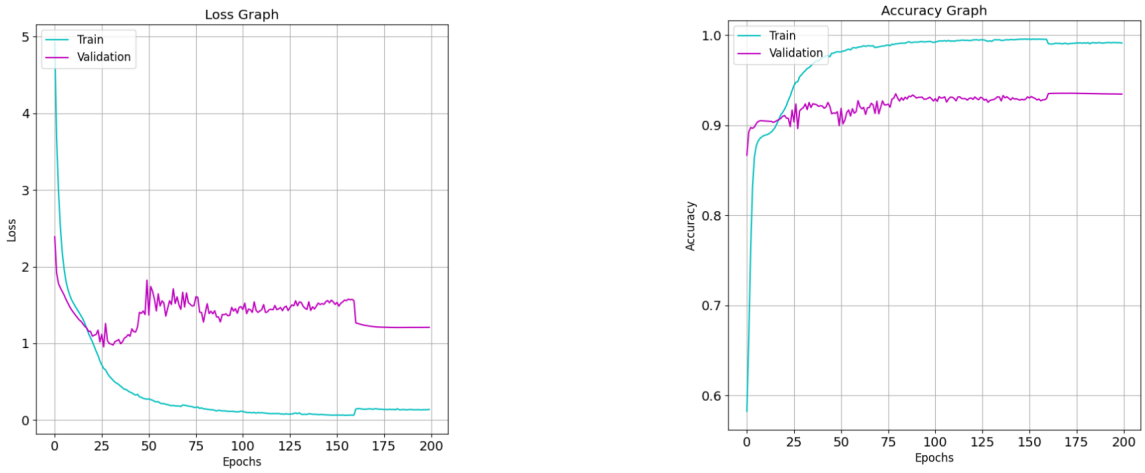

Figure 4.17: From left: Loss function graph and accuracy function graph for predictions using the U-Net Base network on the synovium-tendon dataset

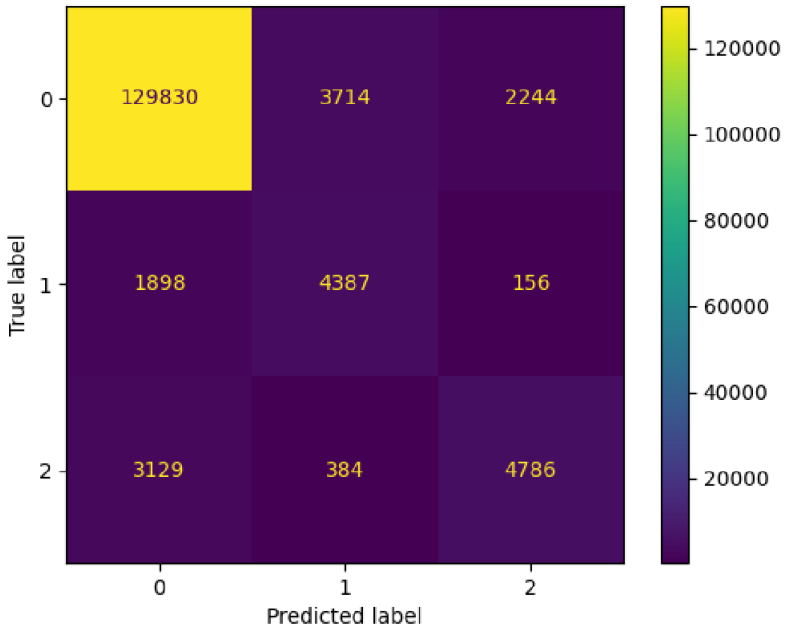

Figure 4.18: Multiclass confusion matrix graph for predictions using the VGG19 network on the synovium-tendon dataset

	precision	recall	f1-score
Background	96%	96%	96%
Sinovia	52%	68%	59%
Tendine	67%	58%	62%
accuracy	92%	92%	92%
macro avg	72%	74%	72%
weighted avg	93%	92%	92%

Table 4.6: Classification report table for predictions using the VGG19 network on the synovium-tendon dataset.

4.2.1 Final Comments on the Conducted Tests

Analyzing the conducted tests, it can be observed that the chosen networks are particularly well-suited for processing medical images. Despite the limited number of images in the dataset, the produced metrics are satisfactory. In general, the accuracy is greater than 90%, and the F1-score reaches levels exceeding 70%. Increasing the dataset using data augmentation techniques could significantly improve this latter figure. This would also enable more consistent training with a much larger number of epochs. In detail, examining the validation accuracy and loss function graphs reveals that these metrics improve during

the Fine Tuning phase, particularly with a notable improvement observed in the network using VGG19. Regarding the predicted images, it is noticeable that VGG19 produces outputs with more defined edges compared to those produced by VGG16. However, VGG19 tends to confuse pixels of classes 1 and 2 more than VGG16, as can be seen in the confusion matrix.

Chapter 5

Conclusions

This thesis has explored Deep Learning techniques for semantic segmentation of ultrasound images. Specifically, the case of multiclass semantic segmentation was analyzed, where the machine's goal was to identify the areas representing bone, tendon, and synoviuml cavity in the input ultrasound images. The results have demonstrated and confirmed that Deep Learning techniques can be utilized for this task, achieving satisfactory performance even considering the limited size of the dataset under examination. Regarding this latter aspect, various techniques for dataset expansion and data augmentation can be explored as future developments to enhance the robustness of the considered network training.

Chapter 6

Acknowledgments

I would like to dedicate this small space to the individuals who have provided me with invaluable support throughout this remarkable academic journey.
I extend my gratitude to my advisor and professor, Giuseppe Notarstefano, who offered me the opportunity to undertake an internship where I learned new topics and allowed me to write this thesis, enabling me to delve further into the acquired knowledge.
I am also thankful to my co-advisor, Francesco Ursini, for his valuable guidance and timely suggestions on the necessary modifications for the thesis.
I thank my co-advisors, Andrea Camisa and Lorenzo Sforni, who have been mentors and guides, readily equipping me with tools, improvements, and helpful suggestions for the elaboration of this work.

I express my gratitude to my numerous colleagues and, above all, friends who have shared joys and challenges during these years spent together. In a significant manner, I thank Andrea, Mirko, and Vittorio, with whom I have shared great moments of laughter at the bar and gelateria in Borgo Panigale.
I appreciate Patrick and Lorenzo, who, with their playful and patient demeanor, have been valuable teachers.
I acknowledge Leonardo for providing me with shelter during my initial time in Bologna and for expanding our network of friendships with his enthusiasm.
A huge thank you goes to Zak and Steve, who not only tolerated me but also took care of me, accepting me for who I am and teaching me the importance of kindness and unconditional love towards others.
I am grateful to Emanuele, whose spirit of goodness reminds me of the beauty of helping others.
A special and proud thanks go to my group of friends and today also Engineers: to Serena, whose kindness and open-mindedness encourage me to strive for the same; to Alice, an enterprising and fascinating person who quickly found harmony with me, even when we delved into the most intricate discussions of physics and calculus; and finally to Anna, with whom I have nurtured a shared adventurous spirit from the very first day, which still

accompanies us in the most vibrant and emotional experiences.
Above all, I thank my dearest friend, who started as a perfect companion for chocolate muffin binges and revealed herself as a strong and unwavering presence to lean on during my darkest moments.

Without all of you, I would have fewer beautiful stories to tell and remember.

I extend my thanks to my entire family, which remains tightly-knit and will always be a strong point of reference for me.
I appreciate Paolo and Lisa as examples of professional Engineers who inspired me to pursue this career. In particular, I thank Lisa for her inspiration as both a Woman and an Engineer.
I acknowledge my grandmothers as examples of strong female figures, each in her own way, teaching me perseverance, steadfastness, and the importance of not giving up in the face of challenges.
I thank my cousin Virginia for adding color to the months during the past pandemic and for giving me a reason to smile and stay active every day.
Thanks to my parents, who allowed me to study by believing in me and who will continue to support me in my future endeavors.
I would like to thank my mother and father once again for their commitment in helping me and my sister build the unique bond that connects us. Thus, I also thank Ada, who remains an essential part of my life and without whom I would not have been able to define myself.

In conclusion, I sincerely express my pride in having met all of you and being able to thank each and every one of you. Thank you.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] T. M. Mitchell and T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1.

[3] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[4] L. Deng and D. Yu, "Deep learning: Methods and applications.," *Found. Trends Signal Process.*, vol. 7, no. 3-4, pp. 197–387, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/ftsig/ftsig7.html#DengY14.

[5] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[6] L. Hardesty, "Explained: Neural networks - ballyhooed artificial-intelligence technique known as âdeep learningâ revives 70-year-old idea.," *MIT News*, 2017.

[7] M. A. Nielsen, *Neural networks and deep learning*, misc, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[8] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf.

[9] H. He and E. Garcia, "Learning from imbalanced data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 9, pp. 1263–1284, 2009, ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.239. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5128907&tag=1.

[10] T. S. Huang, "Computer vision: Evolution and promise," 1996.

[11] R. Szeliski. "Computer vision algorithms and applications." (2011), [Online]. Available: http://dx.doi.org/10.1007/978-1-84882-935-0.

[12] S. Wang and Z. Su, *Metamorphic testing for object detection systems*, 2019. DOI: 10.48550/ARXIV.1912.12162. [Online]. Available: https://arxiv.org/abs/1912.12162.

[13] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[14] D. P. P. Borelli, "L'artrosi delle dita," [Online]. Available: chirurgiadellamanobrescia.it/patologie-mano/curare-artrosi-delle-dita/.

[15] F. Chollet *et al.* "Keras." (2015), [Online]. Available: https://github.com/fchollet/keras.

[16] M. Abadi, A. Agarwal, P. Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[17] Itseez, *Open source computer vision library*, https://github.com/itseez/opencv, 2015.

[18] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. DOI: 10.48550/ARXIV.1505.04597. [Online]. Available: https://arxiv.org/abs/1505.04597.

[19] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[21] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.