POLITECNICO DI TORINO

Master Degree in Data Science and Engineering

Master Degree Thesis

Deep Autoencoder networks and Adversarial architectures for Retinal Vessel Segmentation from Optical Coherence Tomography Angiography images

Supervisor
Prof. Kristen Mariko Meiburger

Candidates
Paola Privitera
Chiara Tomei

Academic Year 2021-2022

Abstract

The retinal vessel segmentation is of great importance in the diagnosis of numerous diseases including diabetic retinopathy, atherosclerosis and hypertension. The structure of the vasculature is characterised by vessels of different thicknesses and lengths, and, given the strong imbalance between the vascular part and the background, this segmentation task is rather complex, especially if the specialist is not supported by Artificial Intelligence algorithms.

In this work, the dataset OCTA-500 has been used to perform a supervised segmentation. The portion of the dataset that has been selected contains 2D images from the projection map of volumetric images obtained by Optical Coherence Tomography (OCT) and Optical Coherence Tomography Angiography (OCTA), with associated segmentation masks. These two imaging technologies are rapid and non-invasive techniques that allow for microscopic imaging of both structural and vascular features of the retina. In particular, OCTA can provide a qualitative and quantitative assessment of the retinal and choroidal microvascularisation.

After an overview of the current segmentation techniques, it has been decided to employ algorithms based on the use of deep neural networks. Following the application of several preprocessing steps, the analysis has been initially performed through the use of a standard Autoencoder architecture. Subsequently, what is considered as one of the state-of-the-art of segmentation has been applied, i.e. the U-Net structure. Moreover, a comparative analysis has been carried out with some new versions of it, such as the Attention U-Net and the Attention Residual U-Net. Finally, the potential of adversarial training has been explored to understand the characteristics of GAN architectures in the segmentation task, in particular with the use of the Pix2Pix GAN.

Ad hoc metrics have been chosen in order to measure the performance of such a complex task and to train the model in the most correct way. The obtained results show a great enhancement of the performance when using the OCTA projection maps instead of the OCT ones. Indeed, for each implemented model, the Dice score related to the vascular class is characterized by an increase up to 10%. Another significant improvement is the one related to the use of maximum projection maps instead of full projection maps. The best configuration found is the one related to the use of augmented OCTA images through the U-Net network, reaching values of 87.6% for the Dice minority score and 77.93% for the IoU minority score. On the other hand, using the maximum projection maps with the same settings, values of 90.31% Dice and 83.32% IoU have been reached.

Contents

Chapter 1

Medical image segmentation

The segmentation technique, in the medical field, is crucial for several diagnostic and analytical activities. Segmentation allows the isolation of organs, abnormalities, tumours and numerous diseases. Using this technique, specialists are able to confine a particular region of interest (ROI) inside the images, to measure tissues, cells, abnormalities and contours, to facilitate the diagnosis and the choice of treatments and dosages. For a long time, these needs have been addressed by manual human work, resulting in very laborious activities. Today, with the introduction of Artificial Intelligence (AI), increasingly advanced models have made possible the achievement of high-performance results.
Medical image segmentation is a special form of image segmentation, however, it is considered rather difficult due to the peculiarity of medical images, which are often characterised by irregular shapes, high variety depending on the image collection techniques or different patients, unpredictable factors and a sparsity of samples that makes segmentation a very challenging task [1].

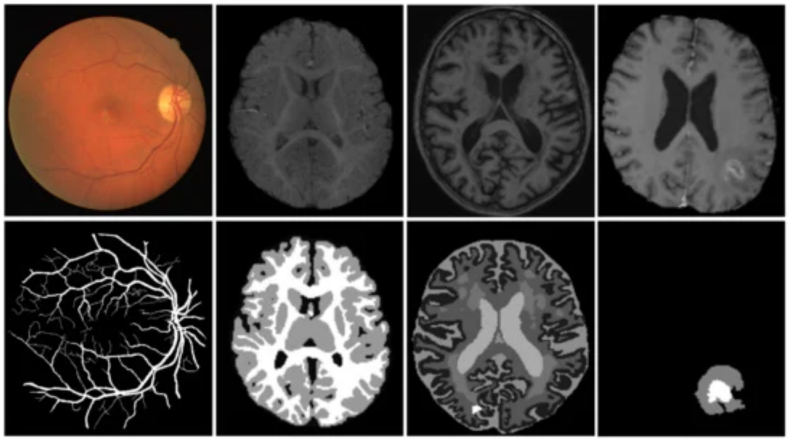


Figure 1.1: Examples of different kinds of medical image segmentation (obtained from [37])

1.1 Imaging techniques for retinal vessel segmentation

Retinal vascular segmentation is used to highlight retinal vessels which are essential in the diagnosis of several diseases, including diabetic retinopathy (DR), retinal venous occlusion (RVO), uveitis and others. In detail, researches have shown a correlation between vessel density and DR. In particular, diabetic patients are characterised by a lower retinal vessel density and fractal dimension than healthy subjects. It has also been proved that the smaller the vessel density, the more severe is the disease. The same is true for subjects with RVO for which it is appreciable a reduction of the capillary density, regarding both superficial and deep ones, or the presence of abnormal microvascular branches. Moreover, this type of segmentation can be useful in the detection of choroidal neovascularisation, frequent in patients affected by uveitis [2].

The main complexity of retinal vascular segmentation is related to the thin shape of blood vessels, which are often irregular and difficult to detect. Moreover, the quality of the images on which this technique is performed is crucial to ensure good results, since noisy images with poor illumination or incorrect angulation can easily lead to rough errors in segmentation and thus also in the diagnosis.

Various technologies can be used to obtain images on which to perform segmentation. Widely used techniques includes fluorescein and indocyanine green angiography, but these examinations are quite invasive, dye-based, and time-consuming, as well as involving the risk of possible adverse allergic reactions. Flourescein Angiography (FA) is a technique that can only detect superficial blood vessels, without being able to analyse the deep structure of capillaries. These limitations have driven research towards something more effective and less invasive, such as Optical Coherence Tomography Angiography (OCTA), an imaging technique that can effectively obtain both retinal and choroidal circulations by using a less invasive approach [3].

1.1.1 Flourescein angiography

The **Flourescein Angiography (FA)** technique was first introduced in 1961 by Novotny and Alvis, revolutionising the study of retinal diseases by using intravenous injections of a noniodinated contrast agent: flourescein sodium ($C_{20}H_{12}Na_2O_5$) [4]. Flourescein absorbs blue and green lights and emits green light signals, allowing for a reasonably detailed mapping of the retinal capillary circulation. However, it has been noticed that the visibility of capillaries rapidly decreases with the increasing of the distance from the foveal centre, especially for deep capillaries. Furthermore, only 40% of capillaries below 4.5 μm can be detected by FA [3].

Figure 1.2: Flourescein angiography of a human eye (obtained from [38])

One way to enhance FA performance is to use **Adaptive Optics Imaging**, which can significantly improve the resolution of the images by highlighting previously non-visible details. However, Adaptive Optics Imaging techniques are highly expensive and limited to small portions of the image, and so not widely used. With FA, rather limited results are obtained in terms of choroidal circulation information. This is because the dye used during the process obscures the details of the deeper carotid artery. For this reason, it is customary to use another reagent, the **Indocyanine Green (ICG)**, which instead works on infrared frequencies, enabling a more effective visualization of medium to large choroidal vessels.

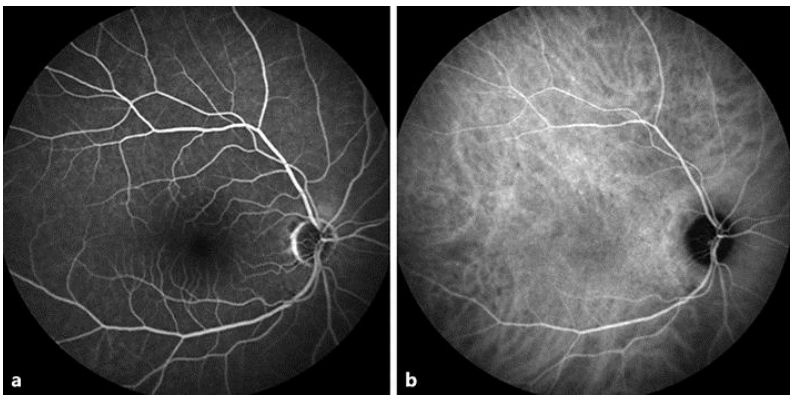


Figure 1.3: Comparison between Flourescein dye (a) and Indocyanine Green dye (b) of the right eye showing normal retinal and choroidal vessels (obtained from [5])

Despite its downsides, FA has been used for ages thanks to its ability in detecting dynamic information on blood transit, the widespread use of the equipment, the capacity of capturing a large part of the region of interest in a single image and, finally, the relative speed, which allows vessel detection in a few fractions of a second.

However, apart from the positive sides that allowed FA to be the golden standard for a long time, there are medical safety reasons for which researches have geared towards

new acquisition techniques. Indeed, the intravenous injection required for FA frequently causes allergic reactions, some of them quite severe. Examples of adverse reactions to the reagents are vomiting and nausea, but also serious episodes related to respiratory and cardiac difficulties and even death. Hence it is not recommended and considered avoidable for several categories of patients such as pregnant women and children. It is with the advent of Optical Coherence Tomography (OCT) that science has made a significant progress to reduce the problems associated with Flourescein Angiography.

1.1.2 Optical Coherence Tomography

Optical Coherence Tomography (**OCT**) is an imaging technique that allows the visualisation of the intermediate and deep capillary network, by using coherent reflected light to capture volumetric structural data of retina with micrometer-resolution [6], [7].

OCT interferometrically measures the amplitude and delay of reflected or backscattered light [3]. Light is a fundamental part of this technique and, due to its wave-nature, it is characterised by a certain amplitude and wavelength. A beam of light is scanned over the retina or over the front of the eye and the depth is calculated basing on the interference of the reflected or backscattered light, and on the light that has travelled a certain predefined path. The coherence is a peculiarity of light that occurs when two waves have the same frequency, the same waveform and their phase difference is constant. Tomography involves the use of penetrating waves in order to create sectional and therefore volumetric imaging. Indeed, OCT is an optical imaging modality that can perform cross-sectional image of biological tissues within less than 10 μm axial resolution using light waves.

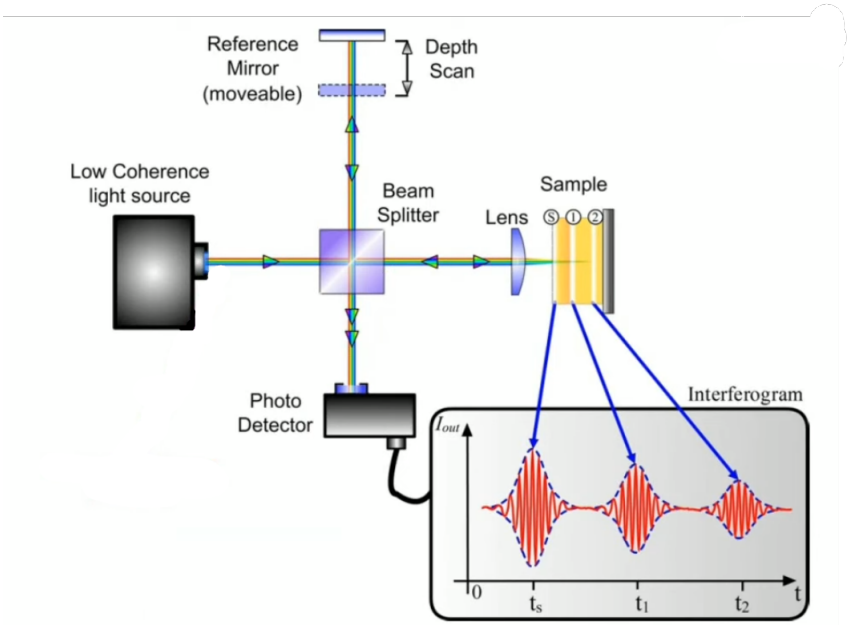


Figure 1.4: Operating scheme of OCT technique (obtained from [39])

More in detail, as can be seen in Figure 1.4, a light beam of 820 nm, close to infrared light, is projected. The beam is split into Probe Beam and Reference Beam through the Beam Splitter. The Probe Beam reaches the target tissue which is the retina, while the Reference Beam reaches the reference mirror placed at a known distance. Then, the echo time delay of the light reflected from the layers of the target tissue is compared with the echo time delay of the light reflected from the reference mirror. A positive interference is produced when the lights reflected from both the target tissue and the reference mirror arrive simultaneously. On the other hand, a negative interference is obtained when the two waves arrive in counterphase and thus, when there is a difference between the two phases of π, where the factor defining the phase is $2\pi f$, with f frequency. At the end, the interference is measured by a photo-detector which produces a range of time delays in order to compare the differences. In this way, the interferometer integrates several data points of over 2 mm depth to construct a tomogram of retinal structures.

The procedure for the OCT exam is quite simple, the patient's pupil is first dilated with mydriatics (tropicamide) and then he is asked to look into the internal fixation target light in the ocular lens. After that, the scanning beam is positioned over the target area and the scans are obtained. To produce the final image, several data points are integrated by the interferometer to construct the tomogram of the target area. Finally, the tomogram is displayed in greyscale or false colours on a high-resolution computer screen.
Figure 1.5 provides an example of the scan obtained at the end of the process.

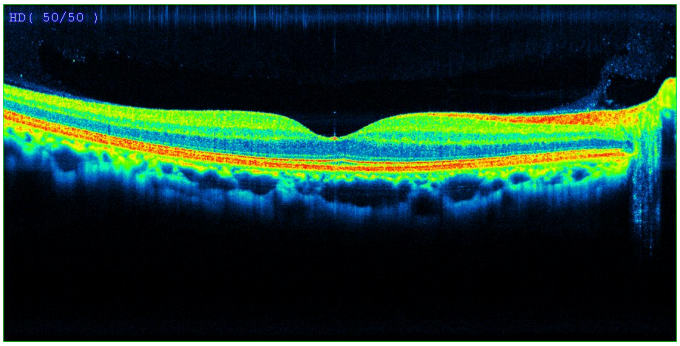


Figure 1.5: Colour OCT scan (obtained from [40])

- The red and yellow colours represent the areas of maximum reflection and optical backscattering.
- The blue and black colours represent the areas of minimum reflection and optical backscattering.

There exist several techniques through which OCT can be performed. Time Domain OCT, Spectral Domain OCT and Spectral Swept Source OCT.

- The **Time Domain OCT** exploits a monocromatic light. Constructive interference is observed as maximum intensity when the optical path of both waves are exactly the same. By moving the mirror it is possible to see the different layers one at a time, as shown in Figure 1.4. Light is pushed in very micro pulses ($1 \cdot 10^{-15}$ sec).

This is a slow technique, based on the mechanical movement of a mirror, and so also the eye movement is captured, inducing artifacts and being able to acquire only 400 scans per second.

- In the **Spectral Domain OCT** a broadband light is used. This technique is based on the transformation from time to frequency, and thus it depends on the Fast Fourier Transform (FFT). If there are several waves of lights, which are characterised by different lengths and frequencies, by passing them together, a combination through time can be obtained, as shown in Figure 1.6. By changing the receiver to frequency instead of time, only three bands will be received, one band at a certain frequency for each original wave. Any compound waves is composed of different waves and, by applying the Fourier transform, it is possible to show how many waves are there and the reflection of each frequency separately. By doing so, a combination of waves can be divided into separate ones. Thus, in this case, there is not a dependence on the mirror movement but on the velocity of light, meaning that this imaging technique is very quick and allows reaching 26000-40000 scans/sec. Moreover, as can be easily noticed in Figure 1.7, resolution is better, with 5 μm compared to the previous technique with 10 μm. This very fast capturing of the image allows to have the OCT Angiography (OCTA).

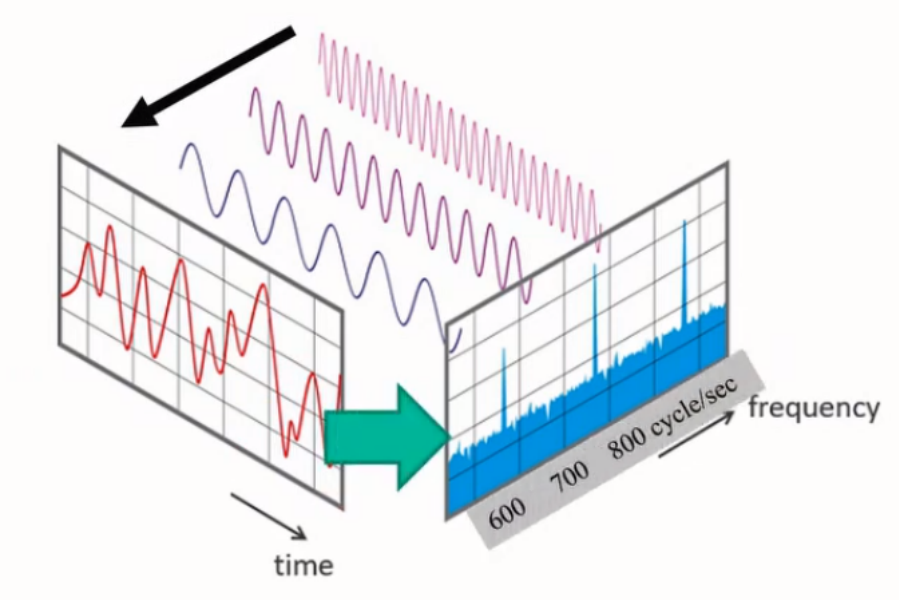


Figure 1.6: Frequency transformation for Spectral Domain OCT (adaptation of image obtained from [41])

- The **Spectral Swept Source OCT** uses a broadband light but, instead of separating the waves at the detector level as for Spectral Domain OCT, here the separation is applied at the level of the source of the light itself. Through the use of this technique, it is possible to show deeper structures, such as the laminar cribrosa or the

deep choroidal vessels.

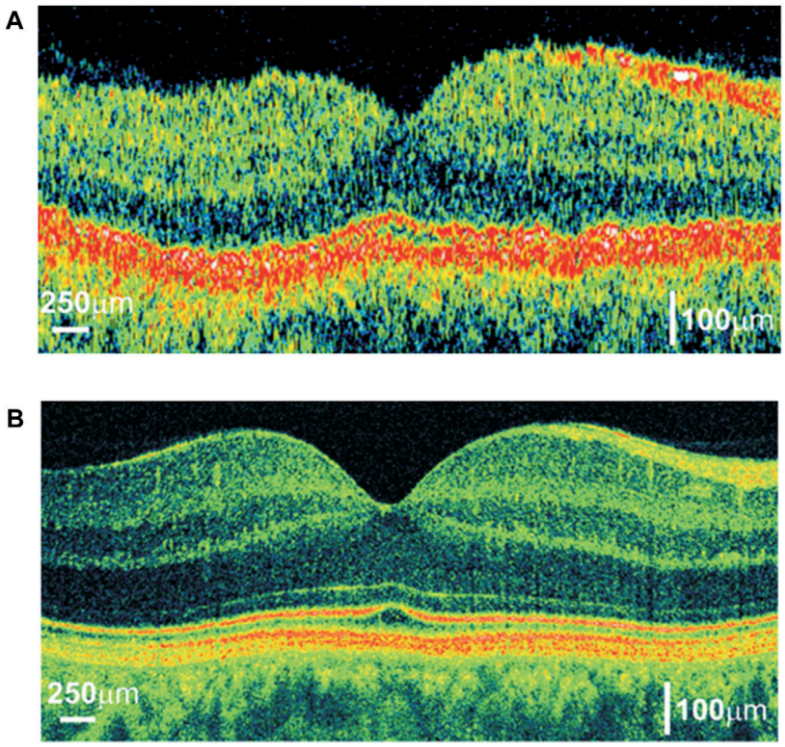


Figure 1.7: Comparison of OCT cross-sectional imaging of the human eye in vivo: (A) Time Domain OCT with an axial resolution of approximately 10 μm and speed allowing for the acquisition of 400 optical A-scans per second; (B) Spectral Domain OCT with an axial resolution equal to 2 μm and speed allowing for a measurement of 30000 optical A-scans per second (obtained from [9])

1.1.3 Optical Coherence Tomography Angiography

Optical Coherence Tomography Angiography (**OCTA**), like OCT, is a rapid and non-invasive technique. It uses the reflectivity of light on the surface of moving red blood cells to accurately render blood vessels. When performing OCT, only the structural information of the retina can be obtained, whereas with OCTA it is possible to get more specific information on the retinal tissue vascular network. Movement is at the heart of this technique as, unlike static tissues, the vascular part is in constant motion thanks to the presence of blood. It is therefore sufficient to take several images of the same portion of tissue at different time fractions and the difference will be due to the passage of blood, thus allowing vessels to be identified. The dissimilarity between pairs of consecutive images is then calculated basing on the phase of the Fourier transformed OCT data.

During an OCT scan there are several scanning directions that lead to the creation of the complete tomogram (Figure 1.8). An A-scan is an unidimensional axis scan that acts on the x-axis. A B-scan is a two-dimensional scan consisting of a series of A-scans

and lies on the plane made up by the x-axis and y-axis. Finally, a set of B-scans provides the depth information and thus creates the 3D volume tomogram, including the z-axis.

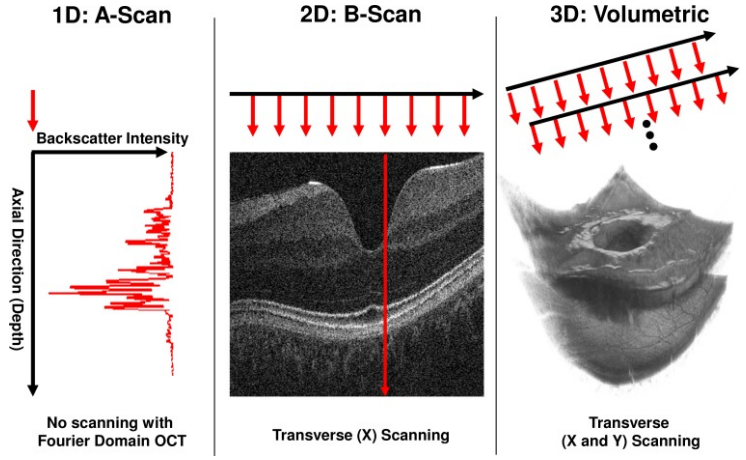


Figure 1.8: Schema of OCT scanning and scanner coordinate system. Left: 1D acquisition (A-scan). A single depth profile is acquired to measure backscattered intensity vs. axial dimension (depth). Middle: 2D imaging (B-scan). The OCT beam is scanned in a transverse direction while A-scans (red arrows) are captured. Right: 3D acquisition. Multiple B-Scans are acquired such that A-scans are sampled on a 2D grid in the transverse plane (obtained from [10])

As can be seen from Figure 1.9, numerous B-scans (N1, N2, N3) are performed from the same source. The differences and various correlations are then calculated pair by pair and then combined at the L3 level in the final OCTA. The procedure is repeated multiple times to create the volumetric image. The acquisition time (TS) is determined by the number of A-scans required to create a single B-scan. There is then a time required to come back to the initial position, which is called fly-back-time (TF). These two time parameters are crucial to define OCTA sensitivity and the saturation behaviour [3].

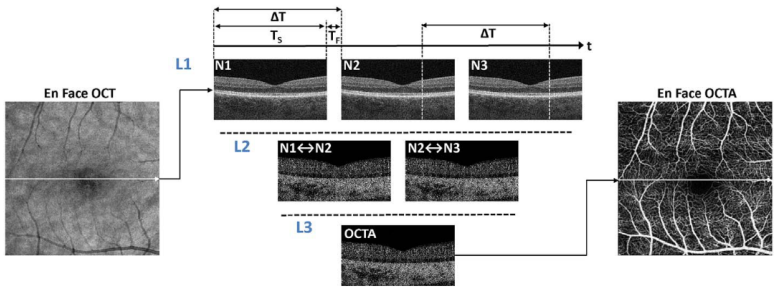


Figure 1.9: Simplified schema of how Optical Coherence Tomography Angiography works (obtained from [3])

The capacity of having a good representation of blood vessels without the use of

reagents which can be potentially harmful to the patients, is a great advancement in imaging techniques used for vascular segmentation. Indeed, it greatly reduces the procedure time and avoids invasive injections. Moreover, the radiation emitted by laser is non-ionised and therefore rather harmless to humans.

With OCTA it is therefore possible to obtain high-contrast, very sharply defined images that perfectly indicate the retinal microvasculature. OCT and OCTA techniques are often used simultaneously, in order to capture as much information as possible regarding the images observed. The structural data provided by OCT are indeed intrinsic to the images produced by OCTA. The disadvantages of using this new technique are certainly the technical and computational resources required. In fact, since it is necessary to acquire the same source several times, a very high A-scan speed is required. Furthermore, although the technology is based on the presence of blood, it is not able to provide quantitative information on the actual presence of blood or about the blood flow itself, but it is limited to the detection and delineation of blood vessels. Finally, it is necessary to note the presence of some artifacts derived from the OCTA technique which, in some cases, can lead to misleading results.

However, despite its weaknesses, this technique is considered an extremely powerful tool for the analysis and diagnosis of numerous pathologies.

Chapter 2

Segmentation techniques

Vascular segmentation, as anticipated, is a rather complex task and retinal vascularisation of images is a key component for retinal image analysis. In recent years, increasingly complex approaches based on sophisticated convolutional neural network architectures have pushed performance to new heights. However, beside that, there exist many different segmentation techniques that can be distinguished in two main categories: traditional and deep learning segmentation techniques. In the next paragraphs, a brief overview of the functioning of these techniques is given.

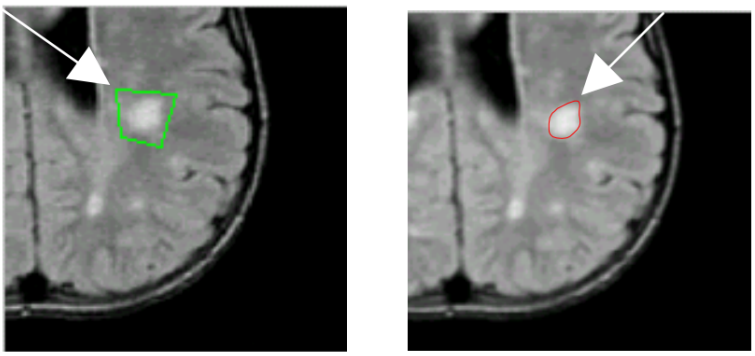
2.1 Traditional segmentation techniques

2.1.1 Model-based approaches

Model-based approaches rely on explicit vessel models to perform the vasculature extraction. The most common are deformable models which can be divided in two main groups: parametric deformable models and geometric deformable models.

Parametric deformable models

The snake technique is one of the principal approaches characterising the parametric deformable models. It is based on the concept of *active contours*, which are parametric curves that deform basing on the influence of internal and external forces. A snake consists of a set of control points, i.e., *snaxels*, that are connected together and are associated with an energy that increases or decreases according to the forces to which each snaxel is subject to. The internal forces smooth out the contours, while the external forces drive the snake to form the more defined contours of the figure that has to be segmented, such as lines and angles. One of its weaknesses, being a parametric model, is the need for initial parameters often given by the user, on which the outcome of the performance heavily depends.



(a) initial contour (b) final contour

Figure 2.1: Segmentation of a tumour in an MRI brain data performed through parametric active contour model (obtained from [14])

Deformable geometric models

Deformable geometric models try to solve problems related to the initial parameterisation. In particular, curves and surfaces evolve using only geometric measures, resulting in a parameterisation-independent evolution. Therefore, evolving curves and surfaces can be implicitly represented as a set of levels of a higher-dimensional function. In this way, changes in topology can be automatically handled without the use of parameters [11]. However, the main disadvantage is that they are characterised by higher computational costs than snakes.

2.1.2 Pattern-based models

The aim of pattern-based models is to automatically find and detect the structures and features of the objects to segment, which in this case correspond to the blood vessels.

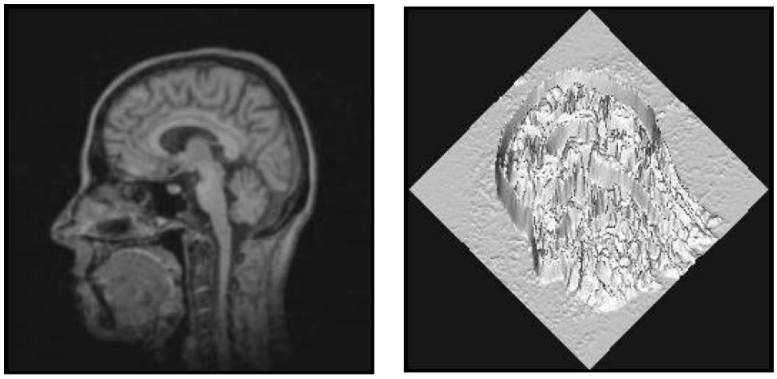
Skeleton technique

One of the most common pattern recognition technique is the Skeleton approach. The objective is to find the skeleton of the object to be segmented, gradually thinning the subject, in order to find the essence of its shape. Various methods are used to extract the internal structure, for example: "*(i) apply thresholding and then object connectivity, (ii) thresholding followed by a thinning procedure, and (iii) extraction based on graph description*" [12]. Finally, the blood vessel centerlines found through the application of this method are then connected together in order to recreate the trees and the network of the vascular tissue.

Ridge-based technique

Another popular method is the ridge-based technique. Given the nature of vascular tissue, image ridges are nothing more than indicators of vessels. Each image is considered to have

three dimensions, the two relating to greyscale and the one relating to intensity ridge, which approximates the skeleton of the objects. Figure 2.2 illustrates the transformation from a 2D image to the 3D map applied to an MRI image. Starting from any pixel, the method consists in finding the intensity vertices, the ridges that indicate the local intensity peaks, i.e., the ridge points. This is done by scaling upward in the direction of the maximum gradients of the surface to find peaks and locate ridges [12].



(a) (b)

Figure 2.2: (a) An MRI slice and (b) Corresponding 2D intensity map in 3D (obtained from [12])

Region growing technique

Region growing is a technique that involves, starting from a seed point, the grouping of pixels according to their nature. In particular, if two pixels are close or have similar characteristics (e.g., intensity value), they are grouped together because it is probable that they are included in the same object. This model is very dependent on the initial seeds chosen and often requires several stages of post-processing to be refined. Sometimes, the region growing technique is used in combination with the ridge-based model to achieve better segmentations [12].

2.2 Deep learning models

Various deep learning techniques can be used for the purpose of segmentation. In particular, the use of CNNs (Convolutional Neural Networks) is particularly suitable for the analysis of images that are characterised by a very high number of features, since it is possible to perform an automatic feature extraction. Neural networks are able to perform pixel-based segmentation, classifying each pixel as belonging to the reference class or to the background.

Most studies focus on the use of specialised networks for the segmentation task or, more in general, widely used in the field of imaging, such as: U-Net, VGG, ResNet.

The downsides of the extensive use of deep learning are the computational resources required, the huge amount of data needed to train a model, as opposed to other simpler

techniques (e.g., shallow learning), and the complexity of the networks themselves, which are often poorly comprehensible [13].

Deep learning models can address both supervised and unsupervised problems.

Supervised learning involves the use of labels and thus the availability of segmentation masks that allow the model to be trained on the basis of certain ground truths. This is not always possible, especially in the medical field, where the availability of samples is often limited. However, it is currently the most reliable technique that allows the creation of highly robust models.

Unsupervised learning refers to problems in which the ground truth masks are not available. In these cases, the transfer learning technique can be applied, consisting in the use of neural networks previously trained on data with the same domain and thus able to generalise appropriately, or on data of another domain, and then proceeding with a domain adaptation technique. The latter technique is particularly delicate in a medical context as the creation of artifacts is very high.

Chapter 3

Dataset OCTA-500

The dataset chosen for this project is the OCTA-500, whose authors are Mingchao Li, Yerui Chen, Keren Xie, Songtao Yuan, Qiang Chen, including the authors of the two reference papers: *Image Projection Network: 3D to 2D Image Segmentation in OCTA Images* [7] and *IPN-V2 and OCTA-500: Methodology and Dataset for Retinal Image Segmentation* [8]. The access to the dataset can be requested from: https://ieee-dataport.org/open-access/octa-500. All the retinal images used in the project refer to the cited dataset.

All the OCT and OCTA images are generated from the same commercial 70 kHz spectral domain OCT system with a center wavelength of 840 nm (RTVue-XR, Optovue, CA). In the dataset, the OCTA volumes are obtained from multiple OCT volumes through the split-spectrum amplitude-decorrelation (SSADA) algorithm. Each OCT volume has a size of 640 $px \times 400\ px \times 400\ px$ corresponding to a 2 $mm \times 6\ mm \times 6\ mm$ volume centered at the retinal macular region. The OCTA volume size is 160 $px \times 400\ px \times 400\ px$, then stretched into a size of 640 $px \times 400\ px \times 400\ px$, using bilinear interpolation, to match the OCT images. The different images are taken from eyes that were imaged from Jiangsu Province Hospital between March 2018 to September 2018 [7].

The dataset (Figure 3.1) contains 500 subjects with 2 field of view (FOV) types, including OCT and OCTA volumes that represent a micron-level resolution to present the three-dimensional structure of the retinal vascular, 6 types of projections, 4 types of text labels and 2 types of pixel-level labels. In particular, 300 subjects with FOV size of 6 $mm \times 6\ mm \times 2\ mm$ and volume 400 $px \times 400\ px \times 640\ px$ are represented by the sub-dataset OCTA_6M. The remaining 200 subjects in the OCTA_3M dataset have FOV dimensions 3 $mm \times 3\ mm \times 2\ mm$ and volume 304 $px \times 304\ px \times 640\ px$. It has been decided to carry out the analysis on this second part of the database in order to be able to handle images with a smaller original size more easily and requiring fewer resources, but certainly more in-depth analyses can be conducted even taking into account the 300 images contained in OCTA_6M.

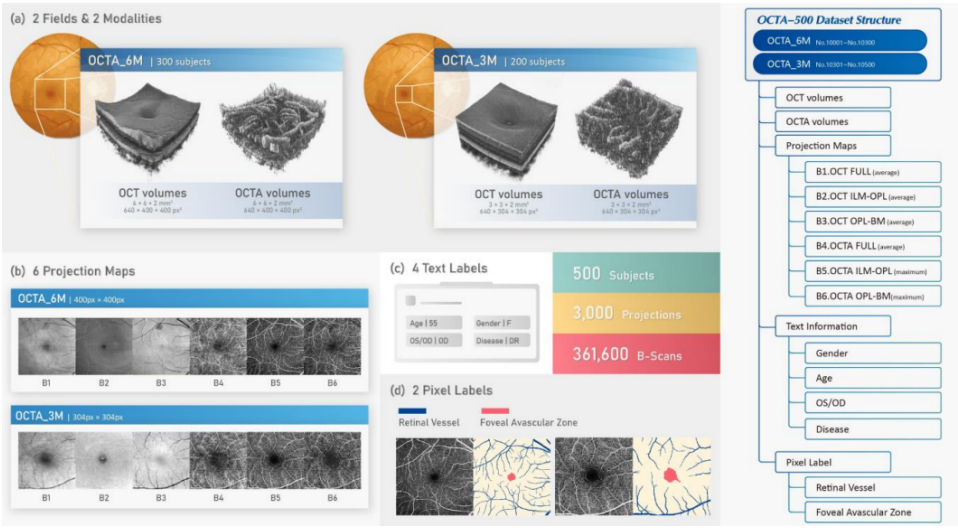


Figure 3.1: Proposed OCTA-500 dataset (obtained from [8])

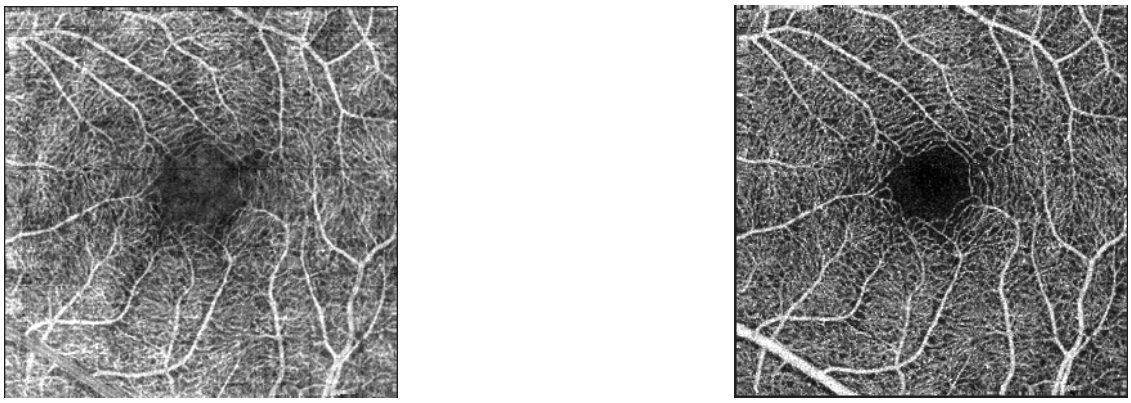
In the reference paper, the construction of the binary target segmentation masks is described as follows: "*The ground truth of Retinal Vessel segmentation is drawn on the OCTA maximum projection maps between the ILM layer and OPL. [...] Ten students and three experts participated in the manufacture and revision of the ground truth. The ground truth drawing of the RV is as follows: (1) The large blood vessels in the inner retina are the segmentation targets in this study, which have a relatively obvious vascular topology and high signal intensity. (2) Capillary plexus, lesion signals and background noise are excluded. The capillary plexus in the inner retina with 6 mm × 6 mm field is different from the large vessel target, because it has no clear vascular topology and relatively low signal intensity.*" [7]

In the first paper, the authors presented an innovative image projection network (IPN) capable of achieving 3D-to-2D image segmentation in OCTA images. With their technique, they managed to obtain the 2D segmentation mask from a 3D input data with optimal results. For this project, however, it has been decided to directly use the 2D images from the projection maps provided by the database. This choice has been made in order to be able to focus less on the manipulation of 3D images and more on the application of the techniques usually used on datasets such as DRIVE or STARE, created via different imaging techniques, on images derived from OCT and OCTA. In both papers, experiments had also been carried out on projection maps alone, so a comparison data was available.

The projection maps that have been taken into consideration are the two on which also the reference papers carried out experiments.

- In particular, the first is the **Full Projection Map** (B1 and B4), which directly averages the 3D OCT and the 3D OCTA volumes along the axial direction, providing a view of both retina and choroid. This type of projection map is the one on which the main analysis has been focused.

- The other projection map is the **Maximum Projection Map** (B5) of the inner retina from the OCTA images, created taking the maximum along the projection direction from the *internal limiting membrane* (ILM) layer to *outer plexiform layer* (OPL), which can clearly show the vascular morphology of the inner retina. In the maximum projection map, the blood flow signals of choroid and the artifacts are removed, making the retinal vessel more clearly as visible in the figure of comparison between FPM and MPM 3.2. The authors of the database used a a public layer segmentation software (OCTExplorer 3.8) in order to generate the maximum projection maps.



(a) Full Projection Map

(b) Maximum Projection Map

Figure 3.2: Comparison between full projection map (a) and maximum projection map (b) (obtained from [42])

The decision to focus on the full projection maps was motivated by the desire to first deal with the baseline proposed by the papers and therefore with images that are more difficult to segment and that have much more noise than maximum projection maps. However, following the first experiments, it has been decided to apply the models also on the maximum projection maps on which, as expected, far superior results have been achieved, reaching metrics at 90% and in line with the reference papers, as analysed in the conclusions of this project.

Chapter 4

Preprocessing

For vascular segmentation of the ocular retina, it has been decided to use the 3 mm $\times$ 3 mm (3M) projection maps provided by the OCTA-500 dataset, focusing on segmentation from 2D projection images. The volumetric data can be projected from different retinal layers to facilitate the individual visualisation of each retinal plexus. In particular, as anticipated, full projection maps have been used in the present work, but also a brief comparison with the maximum projection maps has been carried out. The original dataset has been created to perform two main tasks, **RV segmentation** (retinal vessel segmentation) and **FAZ segmentation** (foveal avascular zone). The FAZ area is a blood-free zone, often correlated with diseases such as diabetic retinopathy and retinal vein occlusion. However, in this study, it has been decided to implement only the first type of segmentation, which is useful for the detection of numerous retinal diseases.

In order to make proper use of the segmentation models, an exploratory phase of the data has been carried out, followed by a preprocessing phase, used to standardise the samples, highlight their main characteristics and synthetically augment them trying to reduce phenomena such as overfitting.

4.1 Loading of the dataset

The full implementation code is available in the GitHub repository: https://github.com/tomeichiara/RetinalVesselSegmentation. All necessary libraries and packages are listed in the *Import* section of each notebook. All experiments have been performed with Python version 3.7.13, 12 GB RAM and Colab GPU.

All the experiments have been carried out separately on OCT and OCTA images in order to compare their performance, strengths and weaknesses. Future developments can certainly include a merging of the two types of projection maps to obtain more information. The noticeable differences in the representation of the two imaging techniques can be evaluated by looking at Figure 4.1.

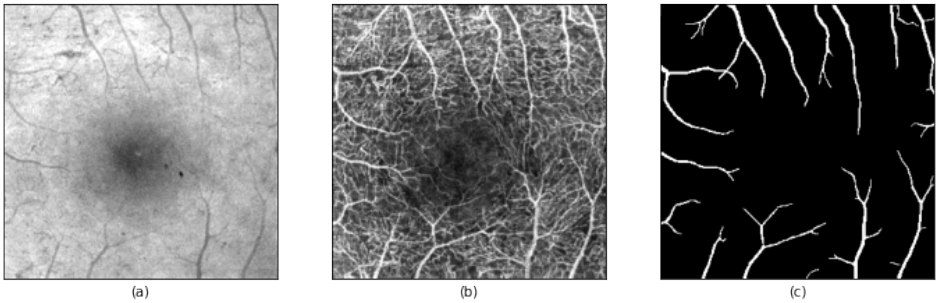


Figure 4.1: OCT and OCTA comparison. Figure (a) shows the OCT retina image while (b) the OCTA one. The segmentation mask (c) is the same for both since the source is identical (obtained from [42])

In order to perform basic database operations such as loading and reading files, the free library *OpenCV* has been chosen.
Functions such as (4.1) have been used in case of particular computational needs or related to the demands of the network models used in the current work.

Code Listing 4.1: Resize

```
img = cv2.resize(img, (256,256), interpolation = cv2.INTER_CUBIC)
```

In particular, it has been preferred to keep the images of their original size (304×304) when the models allowed this. As an example, in case of architectures like the autoencoder and the U-Net, the network depth and symmetry did not require the use of predefined dimensions. This has allowed, in most cases, to work with the images at their original resolution, even in the case of augmented datasets. Indeed, a slight drop in performance has been noticed when switching to a more common size of 256×256.

After reading the images, it has been decided to save them in *numpy arrays* to make them easier to be handled.

Figure 4.2: Grey FAV area of target image (obtained from [42])

In order to perform only RV segmentation, the original target images containing the segmentation masks have been modified to eliminate the grey region dedicated to FAV segmentation, visible in Figure 4.2.

After having verified the presence of only three colours in the segmentation masks (i.e., white, black and grey), a thresholding function has been used to equalise the grey part to the black background, so that the information on the foveal avascular zone (4.2) was ignored during the segmentation process. On the other hand, the original images derived from the full projection map have been preserved in their authentic version.

Code Listing 4.2: Thresholding for FAV removing

```
(thresh, blackAndWhiteImg) = cv2.threshold(img, 180, 256, cv2.
                                    THRESH_BINARY)
```

The general procedure to adequately train the segmentation models is to partition the dataset into specific subsets: training, validation, and test. Firstly, the dataset has been split into train and test sets, by maintaining the 80% for the former and the 20% for the latter. Secondly, in order to validate the great amount of parameters available in the different deep learning models, the training set has been further split into a pure training part and a validation set, keeping the 4:1 proportion. Note that the splits have been randomly generated but kept the same for all the methods and for the different runs, in order to have consistent results among the different trials.

4.2 Normalisation

Pixels assume values ranging from 0 to 255, where each number represents a particular colour. However, when using this notation with such high numerical values on neural networks, it is probable to run into various problems. To solve this, the entire numpy vector has been simply normalised in order to have values from 0 to 1, by dividing the original numbers by 255, allowing to obtain the exact same image but with a different range of values. By doing so, the computational process will be simpler, leading to have a better control of the "dynamic range" of the activations at different layers. This, in turn, helps the optimisation process to converge in a more rapid and stable manner. This type of normalisation, as well as the other preprocessing steps envisaged, are not based on the mean and variance of the dataset, nor on any prior knowledge related to it. Therefore, all preprocessing methods can be implemented even on test data while leaving the verification phase as agnostic and independent as possible.

4.3 Equalisation

It is important to apply a preprocessing equalisation of the images as variations in colours, luminosity, tones, or contrasts, due to different acquisition methods or different patients, may deteriorate performance.

In the current work, two techniques have been implemented to improve contrast in images, the Histogram Equalisation and the Contrast Limited Adaptive Histogram Equalisation (CLAHE).

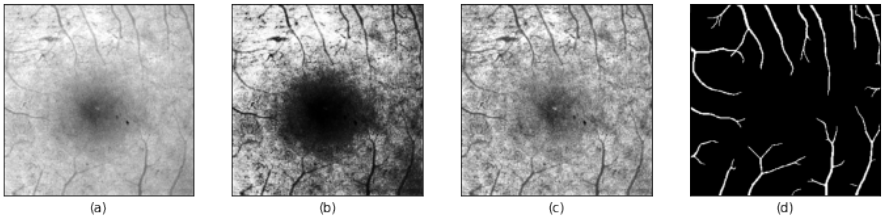


Figure 4.3: OCT visual representation of different preprocessing techniques. Figure (a) is the original image source, Figure (b) refers to the equalised one and in Figure (c) the CLAHE technique is applied. Figure (d) represents the segmentation mask which is the same for each of the previous image, since the ground truth is independent on the type of technique

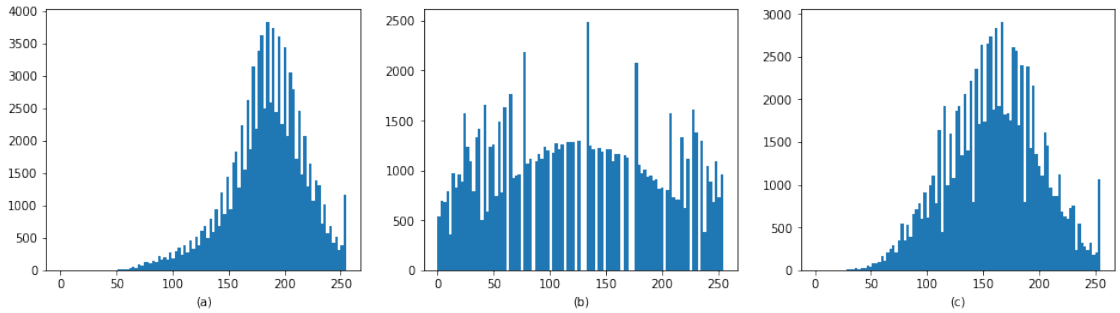


Figure 4.4: OCT histograms of different preprocessing technique. Figure (a) is the histogram related to the original image, Figure (b) the equalised one and Figure (c) shows the one obtained after the application of CLAHE

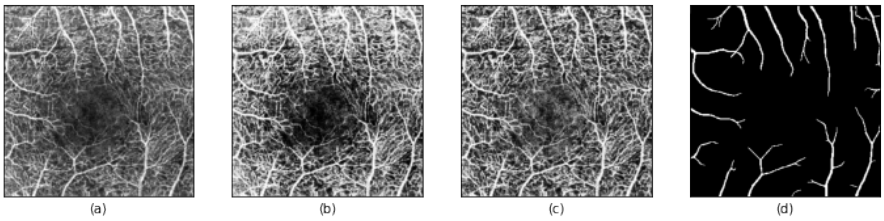


Figure 4.5: OCTA visual representation of different preprocessing techniques. Figure (a) is the original image source, Figure (b) refers to the equalised one and in Figure (c) the CLAHE technique is applied. Figure (d) represents the segmentation mask since the ground truth is the same for any type of representation

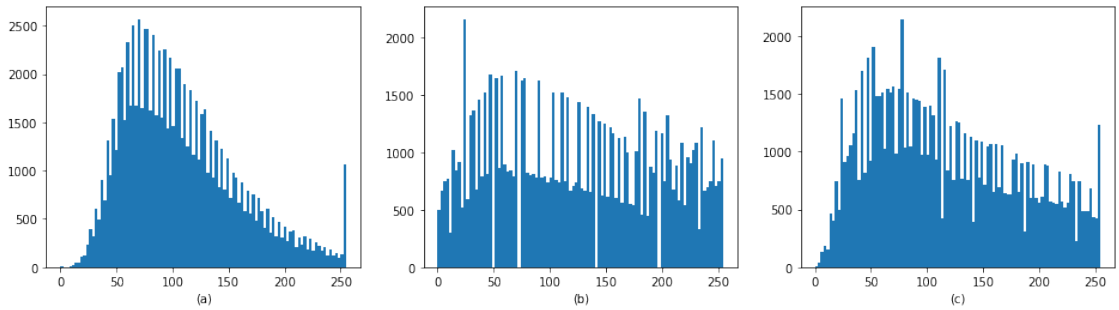


Figure 4.6: OCTA histograms of different preprocessing techniques. Figure (a) is the histogram related to the original image, Figure (b) the equalised one and Figure (c) shows the one obtained after the application of CLAHE

4.3.1 Histogram Equalisation

In a poorly contrasted image a large number of pixels occupies a small portion of the available range of intensities. Through histogram modification, each pixel is reassigned to a new intensity value so that the dynamic range of grey levels is increased [16]. The first equalisation technique is the **Histogram Equalisation**, which is a way of stretching the histogram to include all ranges in the image histogram.

The histogram is indeed used to represent the intensity of image pixels. In particular, in the histograms shown in Figure 4.4 and 4.6, the x-axis represents the scale of the tone, white on the right and black on the left, while the y-axis indicates the number of pixels in the image for that particular tone.

Histogram equalisation is used to improve image contrast and it is one of the most useful forms of nonlinear contrast enhancement [15]. To do this, the pixel density, which may be very unbalanced on only certain values, is spread out over the entire x-axis, stretching out the intensity range of the image. Indeed, in Figure 4.4 (a), values are concentrated in a rather unbalanced manner, while in Figure 4.4 (b) they are spread over the most underpopulated regions. By looking at the corresponding graphic representations of pixels (Figure 4.3 a-b), it is also evident how the contrast is significantly increased.

The histogram equalisation technique consists in mapping a specific distribution into another one, which is more uniform and stretched out. From the details found in the *OpenCV* library, used to perform this transformation, it has been seen that, in order to accomplish the equalisation effect, it is needed to remap the cumulative distribution function *cdf*. For a certain histogram $H(i)$, its cumulative distribution $H'(i)$ is:

$$H'(i) = \sum_{0 \leq j < i} H(j) \tag{4.1}$$

In order to use this remapping function, it is needed to normalise 4.1 such that the maximum value is the maximum value for the intensity of the image. Then, the following procedure can be used to obtain the intensity values of the equalised image:

$$equalized(x, y) = H'(src(x, y)) \tag{4.2}$$

4.3.2 Contrast Limited Adaptive Histogram Equalisation

The image enhancement technique is useful for reaching good quality results in processing medical images. The histogram equalisation uses the global contrast of the image, and this may lead to too bright and too dark regions. By doing so, most of the information may be lost, especially when the histogram is not confined to a particular region. For this reason, the **Contrast Limited Adaptive Histogram Equalisation (CLAHE)** technique has been applied. It solves the problem by dividing the image into small tiles and then equalizing the histogram within each tile, which by default has a size of 8x8, when using the *OpenCV* library. Each pixel is transformed according to a new distribution that follows the behaviour of its neighbourhood region. Indeed, the transformation function is proportional to the *cdf* of pixel values in the neighbourhood. In some cases, pixels close to the image boundaries, like the ones in the blue region of Figure 4.7, must be treated in a particular way. For instance, the image can be extended by mirroring part of it, in such a way to extend borders and allow the new transformation to be computed.

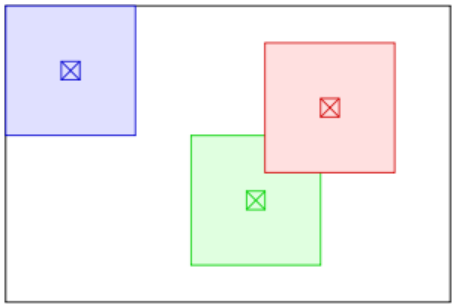


Figure 4.7: Pixels near image boundaries (obtained from [43])

However, this application could lead to an amplification of the contrast and the noise for certain parts of the image, especially where the histogram is highly concentrated. For this reason, a **limitation of the contrast** is applied to reduce the phenomenon of noise amplification. In particular, when the histogram of a specific tile is above a certain limit (which corresponds to 40 for the *OpenCV* library), those pixels are clipped and uniformly distributed to other bins before applying histogram equalisation. Through CLAHE is possible to produce images in which the noisy content of an image is not enhanced, but in which sufficient contrast is provided for the visualisation of the structures within the image [16].

4.3.3 Results

From the results obtained, there is a slight increase in the performance when CLAHE technique is applied. This difference is more evident in the OCTA dataset (Table 4.2) and in particular in the performance related to the minority class, which corresponds to the vascular part of the image. This may be due to the ability of CLAHE to adequately highlight and contrast the vascular part from the background noise, but without going to exacerbate the contrast that could lead to an excessive overlapping of the foveal area and, in general, to the creation of artifacts.

	Mean IoU	IoU minority	Dice minority
No Preprocessing	0.8028	0.6336	0.7757
Histogram Equalization	0.8025	0.6330	0.7752
CLAHE	0.8035	0.6347	0.7765

Table 4.1: OCT performance with U-Net

	Mean IoU	IoU minority	Dice minority
No Preprocessing	0.8678	0.7545	0.8601
Histogram Equalization	0.8646	0.7487	0.8563
CLAHE	0.8704	0.7603	0.8638

Table 4.2: OCTA performance with U-Net

4.4 Data augmentation

In medical image segmentation, the lack of samples is a real and crucial issue. The quality of segmentation often depends on the available dataset and the quantity of labelled datasets to train the models with. Data augmentation and resampling processes help to overcome this issue and to avoid phenomena such as **overfitting**, which is particularly common in the absence of sufficiently large training datasets. Indeed, the lack of data frequently leads the model to not generalise well. Moreover, the variance of training results, applied on different datasets, is really evident and the network will only be able to properly learn the specific data of the small training set, meaning that it will not be able to generalise the learning procedure. This problem is maximised with the use of complex neural networks, which require the training, and thus the updating, of thousands of parameters. Indeed, it is well known how, as the complexity of the model increases, and thus the number of weights to be updated increases, the network needs an exponentially larger database to avoid incurring the problems of overfitting. It is therefore necessary to consider data augmentation activities to build a robust model and an adequate pipeline.

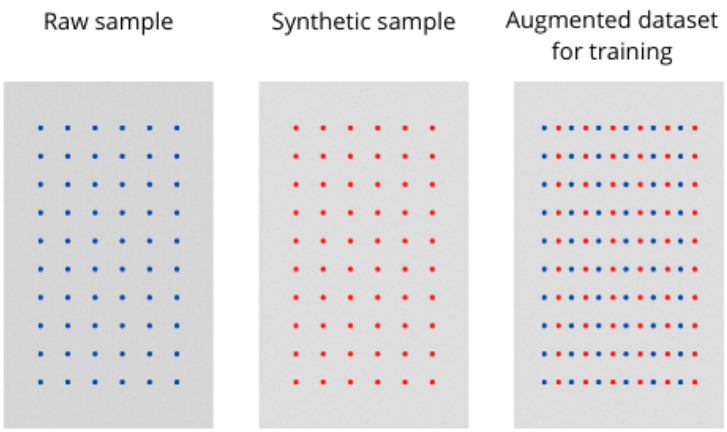


Figure 4.8: Visualisation of Data Augmentation technique

However, data augmentation in the medical field is a rather complex operation that has to be carried out carefully. Usually, small transformations are used during the training phase to add variability and artificially augment the size of the training set. The most common methods used to increase the dataset are geometric transformations such as rotations and reflections, mechanical transformations like elastic or optical deformations, and colour space transformations. In the current case, it is essential to apply the exact same transformation not only to the original image, but also to the mask label (i.e., the ground truth), as otherwise it would lead to misleading learning and would no longer correspond to the image that have to be segmented.

4.4.1 Augmentation techniques applied

The resampling operations have been implemented only on the training set and never on the test or validation part, in order not to compromise the reality of the data, because the algorithm must remain blind to the data that will be provided as testing ones. For this project, it has been used *Albumentations*, an open source Python library for fast and flexible image augmentations: https://github.com/albumentations-team/albumentations. Albumentations implements a large variety of image transform operations that are optimized for performance, and also provides a concise, yet powerful image augmentation interface for different computer vision tasks, including object classification, segmentation, and detection. It is considered faster than any other image augmentation tool and, with a low effort, it is possible to compose a large variety of operations into more complex preprocessing pipelines [17].
Different data augmentation techniques have been applied, striving to choose those with a low impact on the original information stored in the images, thus avoiding the creation of artifacts or new synthetic misleading samples, which is crucial when dealing with medical images, where precision plays a fundamental role.

The transformations that have been tried are the following:

- **Horizontal flip**: the image is flipped horizontally around the y-axis. The flipping rearranges the pixels while protecting the features of the image.

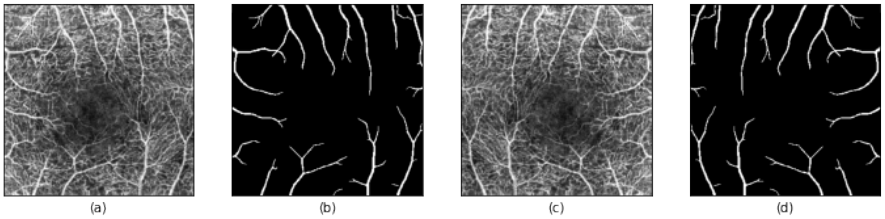


Figure 4.9: Horizontal flip on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Vertical flip**: the image is flipped vertically. It corresponds to composition of a rotation of 180° and a horizontal flip.

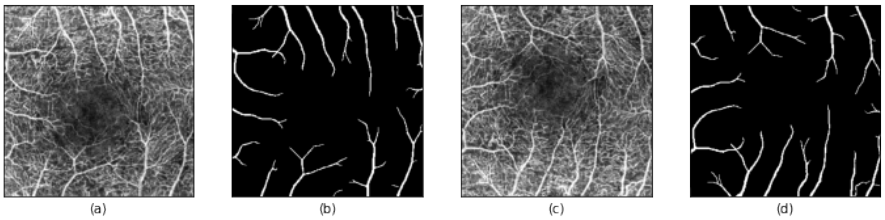


Figure 4.10: Vertical flip on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Random rotation**: the image is randomly rotated at 90°. It is possible to use this angle due to the square size of all the images in the dataset. Indeed, in case of rectangular images, the image dimension would not be preserved after rotation. In such cases, only a 180° rotation is suggested.

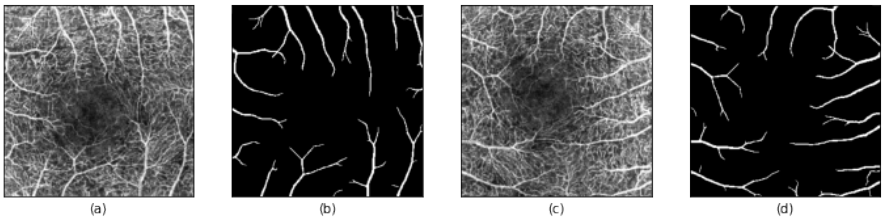


Figure 4.11: Random rotation of 90° on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Transpose**: the transposition of an image is performed by swapping the x-axis and y-axis of its representation.

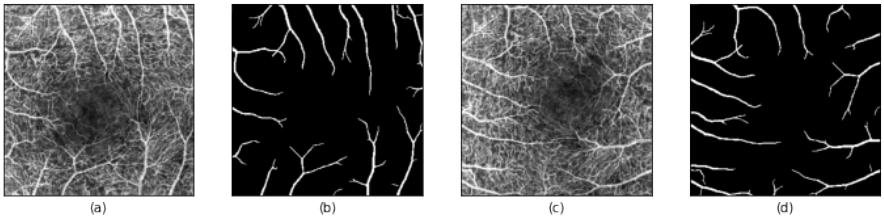


Figure 4.12: Transpose transformation on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Elastic transformation**: a distortion is added to the original image. It is part of the spatial-level, non-rigid transformations. The global elastic deformation affects a relatively large area of the image, leaving the image overall smooth and continuous [18]. The deformation generates a coarse displacement grid with a random displacement for each grid point. This grid is then interpolated to compute a displacement for each pixel in the input image. The input image is then deformed using the displacement vectors and a spline interpolation.

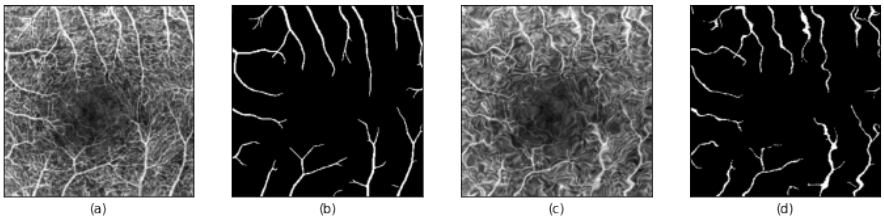


Figure 4.13: Elastic transformation on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Grid distortion**: it is another type of non-rigid transformation based on an imaginary grid of the image.

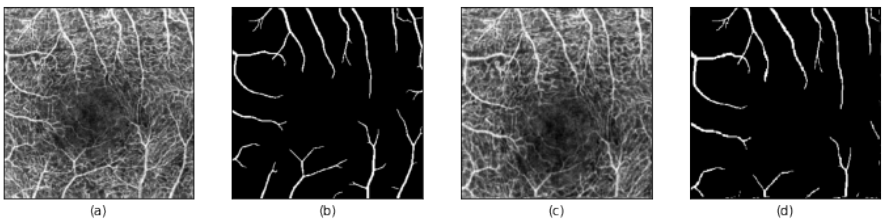


Figure 4.14: Grid distortion on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Optical distortion**: it is a geometric distortion where there is a deviation from a rectilinear projection. It is a form of optical aberration. In this case, the vertical lines appear to curve outwards, like a barrel.

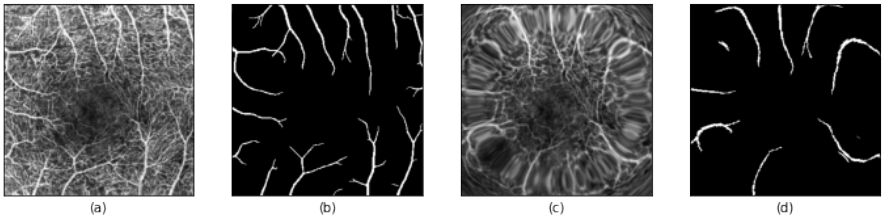


Figure 4.15: Optical distortion on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Random contrast**: a random enhancement contrast is applied, where the contrast is the degree of separation between dark and light regions of the image. It is a form of pixel-based transformation, where the action is about colours and luminosity and not about space.

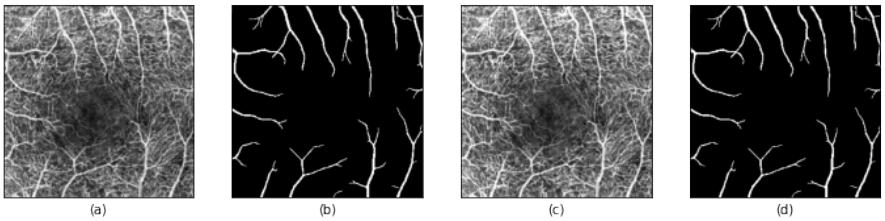


Figure 4.16: Random Contrast on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Gaussian blur**: the image is blurred, meaning that the noise level of the image is reduced by applying a filter which, in this case, is a Gaussian filter with a random kernel size. Also this one is a pixel-based transformation.

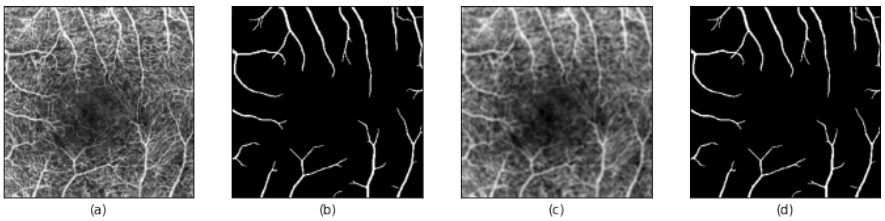


Figure 4.17: Gaussian blur on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

Two composite transformations have been chosen. They randomly put together different types of transformations to add variability to the dataset. When applying the composition, each transformation takes the output of the previous one as input.

- **Composition type 1**: the first composition includes the **Random Gamma** transformation (which affects the brightness of the image), one transformation among

elastic, grid or optical distortions, and finally a rotation.

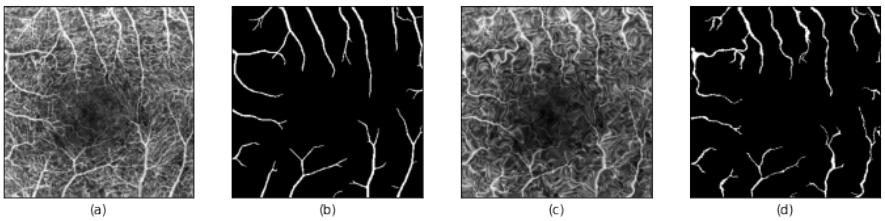


Figure 4.18: Composition of type 1 on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

- **Composition type 2**: the second type of composite transformation is created by combining the **Sharpen transformation** (which increases the level of detail in the image, performing the opposite function to blurring) and the Random Contrast transformation.

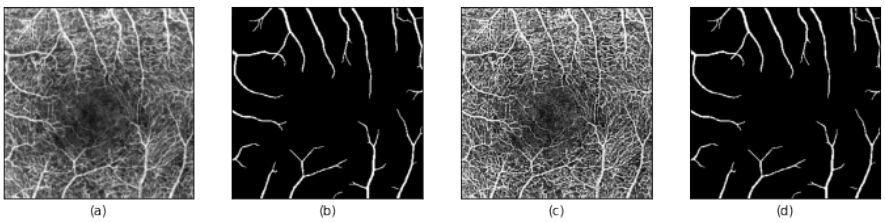


Figure 4.19: Composition of type 2 on OCTA image. Figures (a) and (b) correspond to the original representation, while (c) and (d) to the transformed one

Augmentation downside

As previously mentioned, the use of some data augmentation techniques can lead to the creation of artifacts which can be particularly dangerous in the medical field. For this reason, it has been decided to avoid preprocessing techniques such as image **patches**. When cropping the original image, especially for the purpose of segmentation, it is highly suggested to include the entire subject of segmentation in the crop, by choosing an adequate size of the patch. In case of a vascular segmentation, the part to be segmented is a complex and articulated network that is perpetuated throughout the image. Therefore, choosing a crop of an adequate size to contain the entire vasculature would be impossible, while choosing smaller crops would certainly lead to the loss of the overall sense of the vascular network. Furthermore, some crops would contain completely black spaces representing only the background, and this could lead to disturbing information for the training. One method to overcome this could be to create large enough crops to maintain an adequate portion of the network within them and, at the same time, apply some form of **Patch Smart sampling** [19] in order to implement preliminary object detection and eliminate those patches that are not meaningful to the model.
Moreover, the size of the available images in the dataset (304x304) has been considered

adequate for processing, while special attention would be given to patches in case of particularly large images, such as the whole slide images characterising certain medical imaging techniques.

Apart from the cropping technique, other data augmentation methods may be unsuitable for medical databases and, for this reason, it has been decided to avoid using techniques involving a particular local displacement, which would have introduced overlap and discontinuity of the vessels network. Therefore, less invasive techniques have been applied or, as in the case of elastic transformation, techniques that involved the entire image, thus maintaining the continuity of the vasculature [18].

Augmentation results

Table 4.3 shows the results obtained with the original dataset and the augmented ones, when running one of the best models with the best combination of hyperparameters. In particular, the original dataset refers to the dataset preprocessed with CLAHE and no resampling operation. Augmentation1 dataset, instead, includes the application of all the augmentation activities explained above. Note that, in order to maximize the data available for the training phase, a different proportion has been chosen for splitting the dataset, leaving only 10% of the samples available for testing instead of 20%. The Augmentation1 dataset contains a total of 1944 training data. Finally, Augmentation2 dataset is a particular version of the augmented dataset where have been chosen only the transformations that maximized performance and decreased the risk of misleading results. In detail, the selected augmentation techniques are Horizontal flip, Vertical flip, Transpose, Elastic transformation, Grid distortion, Gaussian blur and the Composition of Sharpen and Random contrast transformations, resulting in a total of 1296 training samples for the Augmentation2 dataset.

	Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
Original	150	0.01	8	0.8704	0.7603	0.8638
Augmentation 1	30	0.01	16	0.8649	0.7478	0.8557
Augmentation 2	30	0.01	16	0.8809	0.7793	0.8760

Table 4.3: Performance comparison on OCTA dataset. Deep network model used for the training: U-Net

Overall the model achieves the best performance when trained with the dataset augmented with the tightest selection of transformations (i.e., Augmentation2). However, note how the number of epochs is very different between the original dataset and the augmented datasets, and this is due to temporal and computational limitations of the experiments. Therefore, it is not excluded that better results can be achieved by running the network for a larger number of epochs, thus allowing the model to improve the training tasks and learn more and better even from a noisier dataset.

Chapter 5

Evaluation metrics and losses

5.1 Evaluation Metrics

Image segmentation and in particular vascular segmentation consist in determining the class of each individual pixel in the image, detecting if it is a background or a foreground pixel. In order to effectively evaluate the tuning of the different hyperparameters and the goodness of the neural networks, it is essential to introduce some evaluation metrics, paying particular attention to some characteristic aspects of image segmentation. Indeed, by nature, these problems are highly unbalanced because most of the pixels usually belong the background and, for this reason, the pixel-wise accuracy is not always the most reliable evaluation metric and may lead to overestimation and loss of significance [8].

5.1.1 Traditional evaluation metrics

Traditional evaluation metrics involve comparing the expected class label to the predicted one and, therefore, are based on the concepts of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN). In detail,

- TP: samples for which the prediction is positive and the true label is positive;
- FP: samples for which the prediction is positive but the true label is negative;
- TN: samples for which the prediction is negative and the true label is negative;
- FN: samples for which the prediction is negative but the true label is positive.

In this specific work, "*TP identifies that pixel is a vessel in both the segmented and ground truth image while in TN, the pixel is non-vessel in segmented and ground truth images. FP identifies that pixel is a vessel in the segmented image but non-vessel in observer marked image, also in FN, the pixel is a vessel in ground truth while non-vessel in the segmented image.*" [20]

1. **Accuracy**: it represents the fraction of predictions correctly classified by the model.

$$Accuracy = \frac{\text{correctly classified samples}}{\text{total number of samples tested}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

It is the most popular metric for model evaluation, but it is not reliable when dealing with significantly unbalanced datasets because it is not able to distinguish between the numbers of correctly classified samples of different classes. Indeed, predicting the majority class would always lead to a high value, even if the predictions of the minority class are all wrong.

2. **Precision**: it determines the percentage of correctly labelled positive instances out of all positive labelled instances.

$$Precision = \frac{\text{samples correctly assigned to positive}}{\text{total number of samples assigned to positive}} = \frac{TP}{TP + FP} \tag{5.2}$$

In other words, it is a measure of how often the model is correct when it predicts positive.

3. **Recall**: it indicates the percentage of correctly labelled positive instances out of all instances that are actually positive.

$$Recall = \frac{\text{samples correctly assigned to positive}}{\text{total number of samples actually belonging to positive}} = \frac{TP}{TP + FN} \tag{5.3}$$

It is a measure of how often the model predict positive samples among all true positive sample.

4. **F1 score**: it is the harmonic mean of precision and recall and it increases with both precision and recall.

$$F1\ score = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.4}$$

As previously explained, accuracy is not the best choice in case of unbalanced datasets, because the inaccuracy of minority classes would be overshadowed by the accuracy of majority classes. On the other hand, the recall metric can be useful because a high value would indicate a good ability in correctly identifying pixels as vascular part. However, by having only the recall indication, it would be impossible to understand if the model actually correctly predicts the vessel part or if there is a particular disproportion in classifying everything as vessel part no matter what it really is. For this reason the F1 score is usually a good trade-off because it incorporates both the information of the recall index and the precision one.

5.1.2 IoU and Dice coefficients

In addition to traditional evaluation metrics, there exist other metrics which are specific for the segmentation task.
The first is the **Intersection over Union** (IoU score or Jaccard score) (5.5) which calculates the intersection between the predicted area and the ground truth one, over the union. The aim is to maximize the intersection, where prediction and label overlap (i.e., True Positives), and to reduce the union, that refers to the area in which the prediction

is outside the label. This metric provides a value in the range [0, 1]: in particular, 0 when none of the prediction and the label are matching, 1 if the entire segmentation is correctly predicted.

$$IoU_{score} = \frac{TP}{TP + FP + FN} \tag{5.5}$$

$$IoU = \frac{OverlapArea}{UnionArea}$$

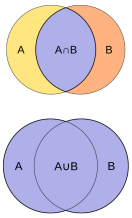


Figure 5.1: IoU graphical representation

Another very common metric used in the field of image segmentation is the **Dice score**, which actually corresponds to the F1 score and thus it takes into account both precision and recall (5.6). When viewed from a more graphic point of view, it is twice the intersection over the sum between union and intersection.

$$Dice_{score} = \frac{2 \cdot TP}{(TP + FP) + (TP + FN)} \tag{5.6}$$

5.2 Losses

The loss function represents a leak, it indicates the way in which the errors made by the algorithm are penalised and how the algorithm learns from them. Indeed, the aim is to change the network weights such that the loss is minimised. In detail, the loss is used to compute the gradient descent and proceed with the backpropagation by multiplying the derivative of the loss by the learning rate and subtracting this value from the weight of the network in the previous step. Therefore, the choice of the loss function has a significant influence on the network learning.

5.2.1 Binary Cross Entropy loss

Binary Cross Entropy (BCE) loss, or Log loss, is considered as a baseline for many different classification and segmentation tasks. The formula for the BCE loss can be seen in equation (5.7).

$$BCE_{Loss} = -\sum_{n=1}^{N}[y_n log(\hat{y_n}) + (1 - y_n) log(1 - \hat{y_n})] \tag{5.7}$$

It represents the probability of having classified the class correctly plus the probability of having classified it incorrectly. The y letter refers to the true label image, y_n refers to a single element of that label, $\hat{y}$ refers to the prediction of the output image and $\hat{y_n}$ to a single element of that prediction. "*In this equation, it can be seen that BCE, while incorporating an element of probability, smoothed out by the log component, still awards both true positives and true negatives, while penalizing false positives and false negatives [...] this can lead to simplistic solutions to segmentation when the data is significantly sparse, by labelling all output as background.*" [21]

5.2.2 Mean Squared Error loss

Mean Squared Error (MSE) is a loss that calculates the sum of the squared differences between the true label and the estimated one (5.8), trying to penalise larger errors more heavily.

$$MSE_{Loss} = \sum_{n=1}^{N} \frac{(y_n - \hat{y_n})^2}{N} \tag{5.8}$$

By its nature, it is rather close to the concept of accuracy. It is easy to implement and widely used, however in the segmentation domain it can have some problems, for instance related to the fact that it gives more importance to low level features rather than high level ones, and therefore it usually provides blurry images. A technique to compensate for this problem is the use of adversarial loss, which will be explored in the later chapter on Pix2Pix implementation, or the use of Jaccard loss.

5.2.3 Jaccard loss

The loss chosen to be used for the majority of the models implemented in this work is the Jaccard Loss, derived from the IoU score, since it has proven to be a quite stable

loss. The model does not directly predict 0s or 1s, but a probability for each pixel to be vascular or background. For this reason, the IoU formula cannot be just based on its discrete version, because TP is the number of pixels where both the prediction and ground truth are 1. The predictions of the model are never actually 1, but probability values in the range (0, 1) due to the sigmoid activation in the final upsampling layer of the networks.

Therefore, for optimization purposes it has been necessary to rewrite the equation in terms of the soft probabilities and to use an approximation of IoU. This gives the equation for the approximation (5.9):

$$IoU_{Loss} = -\sum_{n=1}^{N} \frac{\hat{y_n} y_n}{\hat{y_n} + y_n - (\hat{y_n} y_n)} \tag{5.9}$$

Chapter 6

Segmentation Algorithms

6.1 Traditional Autoencoder

The autoencoder is a particular type of neural network that consists of two main parts, the **encoder** and the **decoder**. Through the first component it is able to encode the input object in a **latent space**, by creating a compressed representation of it. The latent space is clearly visible in Figure 6.1 as a bottleneck that ensures that all data are encoded within this space. From the data that has been encoded, the decoder has the task of reconstructing the original object from the compressed information available in the latent space. This presupposes the ability of the decoder to recreate the object itself, without having access to the original information but only to a synthetic and compressed representation of the source data [22].

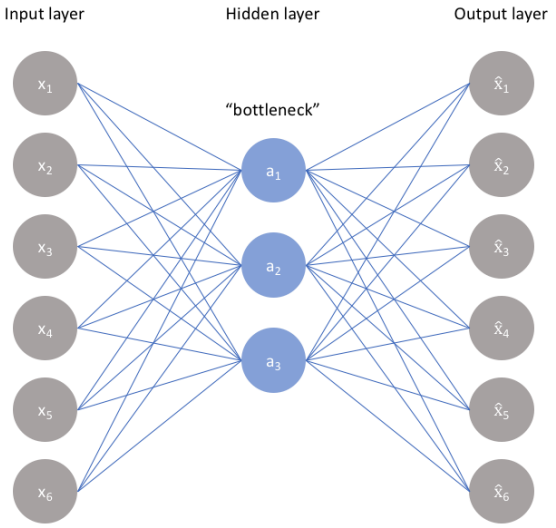


Figure 6.1: Autoencoder structure (adaptation of image obtained from [23])

The whole point of training an autoencoder is that supposedly the latent space has a small number of parameters (very few information to be learnt). So, it tries to compress

the information in input into a smaller representation such that the network can learn what an image is, without necessarily having to learn the noise in it. However, it is not a matter of image compression but the main ability of an autoencoder is to perform an *unsupervised feature extraction*, ensuring that the features encoded in the latent space are those necessary and essential for the reconstruction of the image itself. Moreover, a latent space is needed because it allows keeping similar objects close to each other and different objects far apart. Indeed, the better the latent space behaves, the easier is to get significant results. Autoencoders have usually bad looking latent spaces where the information is not put in a clear manner. This problem can be addressed by **Variational Autoencoders** (VAEs), which instead force the latent space to behave in a more controlled way.

One of the main advantages of the autoencoder is that unpaired data can be used to perform several tasks, i.e., without the need of having a label associated to the image or the object in input. If the purpose is image reconstruction, it will be sufficient to input an image x to the encoder and compare the reconstructed $\hat{x}$ with the image x itself. Then, the network will be driven to minimise what is defined as the reconstruction error, thus minimising the difference between the reconstructed image and the real one:

$$minL(x, \hat{x}) \tag{6.1}$$

6.1.1 Different uses

As previously anticipated, one of the simplest uses of an autoencoder is the reconstruction of the source image, however also more complex tasks can be performed.

- One of the applications of autoencoders is the **denoising task**. Given a noise-free image, the procedure consists in synthetically add noise to it and give this new noisy image as input to the encoder. Then, the network is trained and, through the reconstruction loss, it learns to produce images that tend to the original, noise-free image. In this way, after the training phase of the network, it will be possible to give the encoder any noisy image and obtain as output a completely denoised one.

- Another possible purpose for the use of an autoencoder is **domain adaptation**, the basis of the concept of **transfer learning**. Through domain adaptation, the model should be able to learn robust or high-level features, which are required in order to be able to apply transfer learning. Domain adaptation utilises both source and target data for learning, even though their distributions are not the same. The goal is to share the knowledge among source and target data but only for related tasks or domains [24]. A typical application refers to the case of confocal microscope images and electron microscope images. Indeed, confocal images are more common and easier to collect compared to electron microscope ones. In order to train the system, it is necessary to provide as input to the network the confocal images and also the corresponding electron ones. If the data are enough and the model is able to generalise, it can be obtained a system that, given as input a confocal microscope image, is able to build the equivalent electron microscope image.

- A simple but effective task that autoencoders can address is the **image colourisation** (Figure 6.2). The concept of image colouring is similar to the denoising task, except that, instead of reconstructing a clean image without noise, here the purpose is to obtain a coloured image in output, starting from the same black and white image received in input. After a reasonable training phase, the network should be able to generalise on unseen images and to compress the information in the latent space in such a way to obtain, from any input image in black and white, the corresponding coloured image.
  The most common method of displaying colour images is RGB, but other approaches are also possible, such as Lab Colour Space, which is usually preferred for switching between greyscale and colour, thanks to its lower channel complexity.

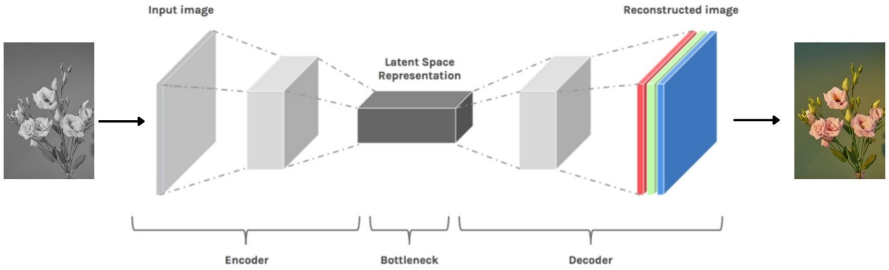


Figure 6.2: Example of autoencoder for image colourisation (adaptation of image obtained from [44])

6.1.2 Autoencoder implementation

As anticipated, the segmentation technique involves the classification, and thus the assignment of a class label, of each pixel in the image. In the current work, a binary classification was required to recreate the mask of the retinal vascular tissue. Therefore, the input to the autoencoder is the original retinal image, OCT or OCTA, while the prediction should converge to the binary mask available as ground truth.

In detail, the structure of the autoencoder used to perform the segmentation is composed by the layers that can be observed in Figure 6.3.

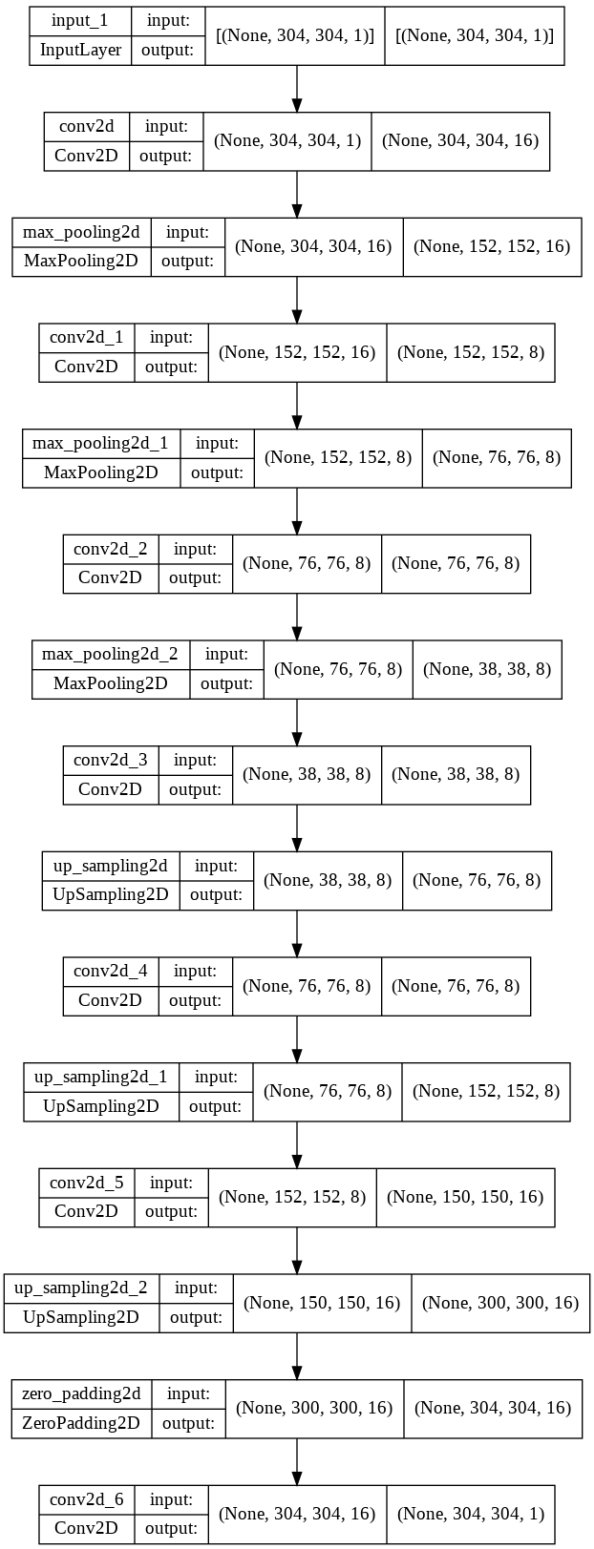


Figure 6.3: Autoencoder layers

It has been decided to implement an autoencoder with **convolutional** layers for an optimal feature map extraction. In the encoder part, layers of fundamental importance are those represented by the **max pooling** operation [26], which has the role of calculating the maximum value between the patches of the feature map, necessary to create a downsampled feature map. A combination of the convolutional and pooling operations leads to the central latent space, where the information is compressed. Then, there is the decoder part, in which convolutional layers are followed by **upsampling** layers. The upsampling activity allows the decoder to restore images from low resolution feature maps.

There exist various types of upsampling and the most common are *Nearest interpolation*, *Bilinear interpolation* or *Cubic interpolation*. With Nearest interpolation, the necessary pixels are recreated from the neighbour pixels. With Bilinear interpolation, new pixel values are estimated by averaging the two nearest pixels, while Cubic interpolation estimates the volume of the closest pixels. Another technique usually implemented to upsample is *Transposed Convolution*, a machine learning approach that consists in training specific parameters and, thus, learning the upsampling technique in an end-to-end manner thanks to the backpropagation steps. This last method can be very powerful because it can learn parameters to get a good upsampling task. However, at the same time, it is prone to create artifacts, especially in an easy network like the traditional autoencoder. For this reason, it has been decided to use the Nearest interpolation upsampling technique in the autoencoder architecture.

Code Listing 6.1: Autoencoder implementation code

```
input_img = keras.Input(shape=(304, 304, 1))

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img
                                                                 )
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.ZeroPadding2D(padding=(2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x
                                                                         )

opt = keras.optimizers.Adam(learning_rate=lr)
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer=opt, loss=[jaccard_coef_loss], metrics=[
                                                              jaccard_coef])
```

After each convolutional layer, an **activation function** is applied. The purpose of activation layers is to add non-linearity to the network, as without it the complexity of the

mapping functions that can be expressed by the network would be greatly reduced. Every tensor coming out from the convolutional layer is passed to the activation function, that changes the values of the matrix and therefore the complexity of the mapping functions. Moreover, the activation layer can also be used to normalise values when needed. The activation function used in each intermediate convolutional layer is the *Rectified Linear Unit* (ReLU) function, which returns value 0 for negative inputs while preserving positive inputs [25]. This function does not saturate values and always grows, which instead is not granted when using Sigmoid activation. The ReLU is also used to overcome the vanishing gradient problem because, at large input and output variations, the derivative of the gradient would not go to zero anyway, as it would happen by using the Sigmoid function.

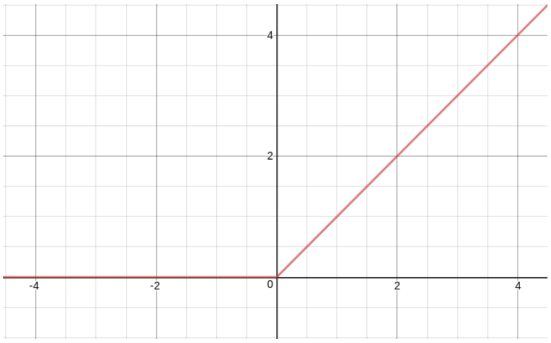


Figure 6.4: Line plot of ReLU for negative and positive values (obtained from [25])

The vessel segmentation task is a binary semantic segmentation and for this reason only an output is needed: the probability that a pixel belongs to a vessel or that it belongs to the background. For example, if a threshold of 0.5 is set and the output gives a probability of 0.1, it means that the current pixel should be considered as background, while if it is above 0.5 it is probably a vessel. Then, these probabilities should be converted into classes, and this is done by using $y_{predThresholded} = y_{pred} > 0.5$. To obtain this probability value, the *Sigmoid* (6.2) has been chosen as final activation function.

$$S = \frac{1}{1 + e^{-x}} \tag{6.2}$$

It is a S-shaped function that maps the input into a value in the range 0 to 1. Indeed, the Sigmoid function is used for models whose aim is to predict the probability as output which, in this specific case, corresponds to the probability of belonging to the vascular class or to the background one. As can be seen from the implementation code (6.1), *Adam optimizer* has been used to optimize the gradient descent. It is a very useful optimization algorithm when dealing with a large amount of parameters and data. In particular, it is a stochastic gradient descent method that is based on the adaptive estimation of the first-order and the second-order moments. The parameters β_1 and β_2 control the weight decay rates of the moving averages [27].

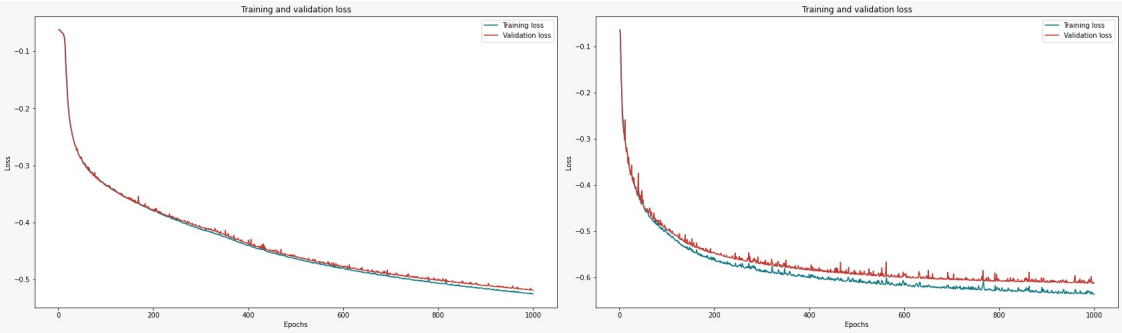
The loss function and the evaluation metrics that has been implemented are the Jaccard Loss and the Jaccard Coefficient, extensively described in chapter 5.1.2.

6.1.3 Results

In order to evaluate the performance of this first implemented model, various parameters such as *batch size*, *learning rate* and *epochs* have been tuned, for various versions of the dataset. What is referred below as the Original Dataset is the dataset composed of the 200 original images, divided into train, validation and test sets, to which the CLAHE technique has been applied. The dataset defined as Augmented is the version to which oversampling has been applied by using the transformations selected in chapter 4.4.1 - (**Augmentation results**). Finally, it has been analysed the difference of the model when trained with the OCTA database or with the OCT version, showing a significant improvement in the performance with OCTA under all conditions investigated.

Original Dataset

The first thing that has been explored is the change in loss as the learning rate increases, as shown in Figure 6.5, by using a batch size of 8 and 1000 epochs for all runs. As can be easily noticed, the losses are characterised by different behaviours when changing the learning rate.



(a) (b)

Figure 6.5: Tuning of learning rate. Training and validation losses from OCTA **original** dataset not augmented.
Batch size = 8, Epochs = 1000. (a) LR = 1^{-4}; (b) LR = 1^{-3}

In Figure (a), with a learning rate of 1^{-4}, there is a rather small step size, which means that the gradient steps during the backpropagation phase will be short and rather conservative. This justifies the behaviour of the learning that still seems to be descending after 1000 epochs, suggesting that, with more epochs and time available, improvements in performance could be achieved. However, it is less steep than in Figure (b), where lower loss values can be achieved. In this case, the learning rate is 1^{-3} and it is possible to notice that, already after 800 epochs, a plateau seems to be reached. Proceeding with a higher number of epochs would probably be counterproductive, leading to a larger divergence between training and validation losses and therefore running into overfitting.

Figure 6.6 displays the performance on the test set, composed of 40 images. This confirms, as anticipated by the loss behaviour, that the best performance is obtained

with a LR equal to 1^{-3}, while there is a significant drop with the higher value of LR, meaning that, for larger steps, the model risks not to converge.

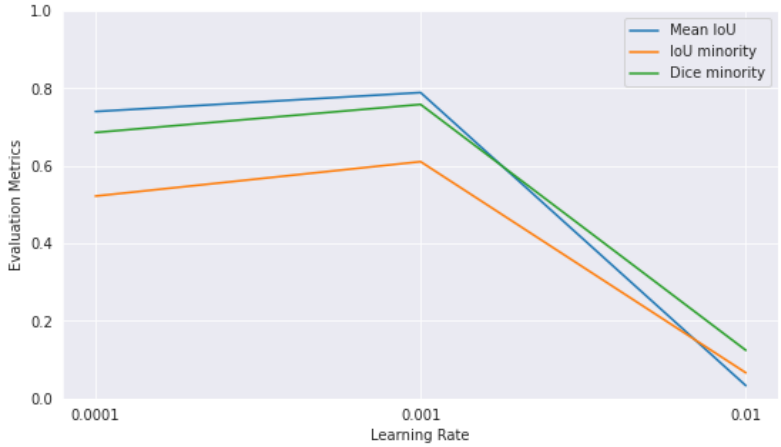
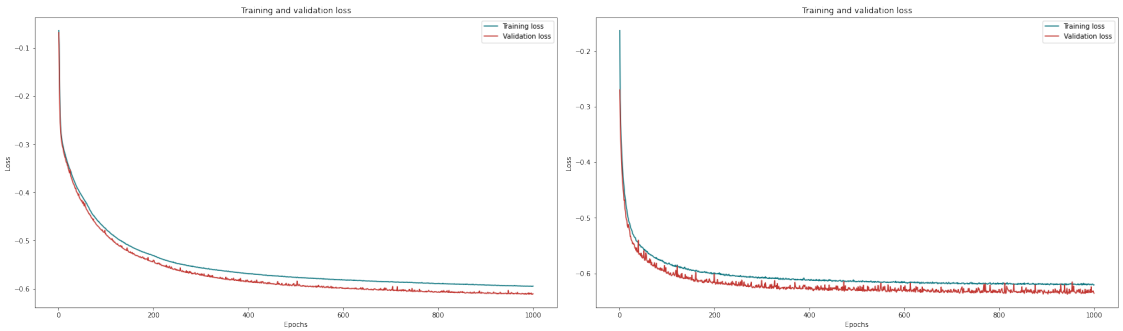


Figure 6.6: Tuning of learning rate. Metrics obtained for OCTA **original** dataset not augmented. Batch size = 8, Epochs = 1000

Augmented Dataset

Very similar performance can also be observed on the augmented dataset in Figure 6.7. In particular, with a LR of 1^{-4} (a), the updating steps are shorter and with fewer oscillations, indicating a more stable but slower model. However, by using a LR of 1^{-3} (b), the plateau is reached already after 600 epochs, thanks also to a much larger number of samples with respect to the original dataset.



(a) (b)

Figure 6.7: Tuning of learning rate. Training and validation losses from OCTA **augmented** dataset.
Batch size = 8, Epochs = 1000. (a) LR = 1^{-4}; (b) LR = 1^{-3}

Another significant difference between the two datasets can be noted in the higher training loss with respect to the validation one, which can be justified by the presence of a

noisy training set in comparison with a rather standard validation one, which is easier to be segmented. For this reason, it may be considered to further restrict the transformation activities in order to have a less robust but more effective training of the model on the domain of interest.

Figure 6.8 shows the performance on the test set, composed of 20 images.
The same conclusions can be drawn as for the original dataset, with a peak in performance at LR equal to 1^{-3} and a drop for higher values of it.

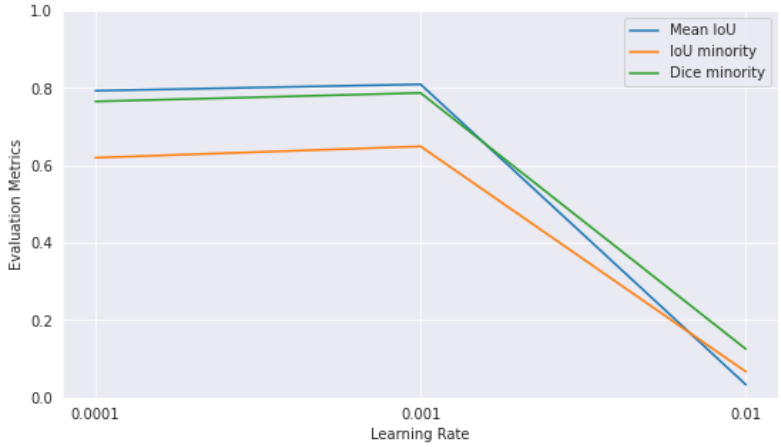


Figure 6.8: Tuning of learning rate. Metrics obtained for OCTA **augmented** dataset. Batch size = 8, Epochs = 1000

Furthermore, tuning has been performed also with respect to the batch size. Table 6.1 summarises the performance obtained with a batch size of 8, already anticipated by the previous graphs. In Table 6.2, instead, the performance for a batch size equal to 16 can be appreciated. However, by varying the batch size to 16, no significant improvements have been noticed. For this reason, a batch of 8 has been chosen as the best performing hyperparameter.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
1000	0.0001	8	0.7931	0.6199	0.7654
1000	0.001	8	0.8097	0.6493	0.7873
1000	0.01	8	0.0334	0.0669	0.1254

Table 6.1: Tuning of learning rate with batch size = 8. Metrics obtained for OCTA **augmented** dataset

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
1000	0.0001	16	0.7864	0.6069	0.7554
1000	0.001	16	0.8070	0.6449	0.7841
1000	0.01	16	0.0334	0.0669	0.1254

Table 6.2: Tuning of learning rate with batch size = 16. Metrics obtained for OCTA **augmented** dataset

Finally, by looking at the histograms of Figure 6.9, one can see the direct comparison between the original dataset and the augmented one with the same hyperparameters. A slight increase in performance due to a noisier and larger dataset is thus evident. However, despite the autoencoder is a neural network, it is overall simple and not too deep, which is the reason why the parameters to be updated during backpropagation are not as high as for a U-Net or a GAN, and thus the need for very large datasets is partially buffered by this.

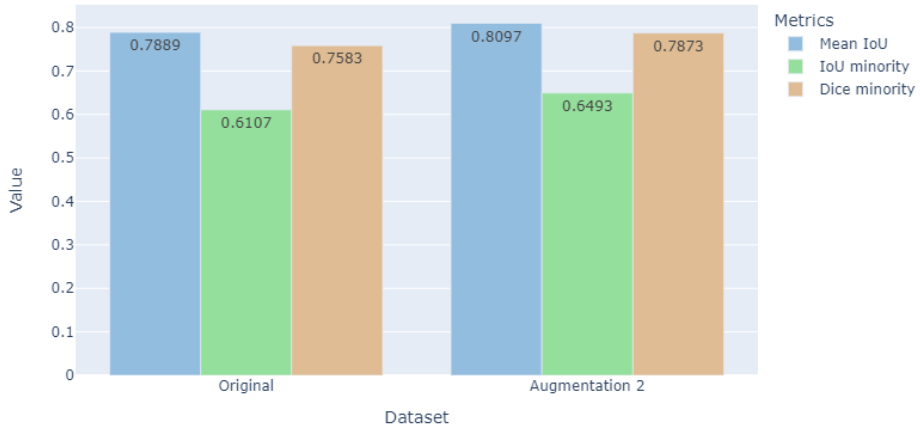


Figure 6.9: Original dataset and augmented dataset comparison.
Epochs = 1000, Batch size = 8, LR = 1^{-3}

Figure 6.10 shows the graphical result of the implemented algorithm. In particular, the images are the result of the model with epochs = 1000, batch size = 8, LR = 1^{-3} and augmented dataset. By looking at the overlapping between the ground truth mask and the predicted segmentation, it can be seen that the difference is minimal, resulting in an average IoU value of 80.97%.

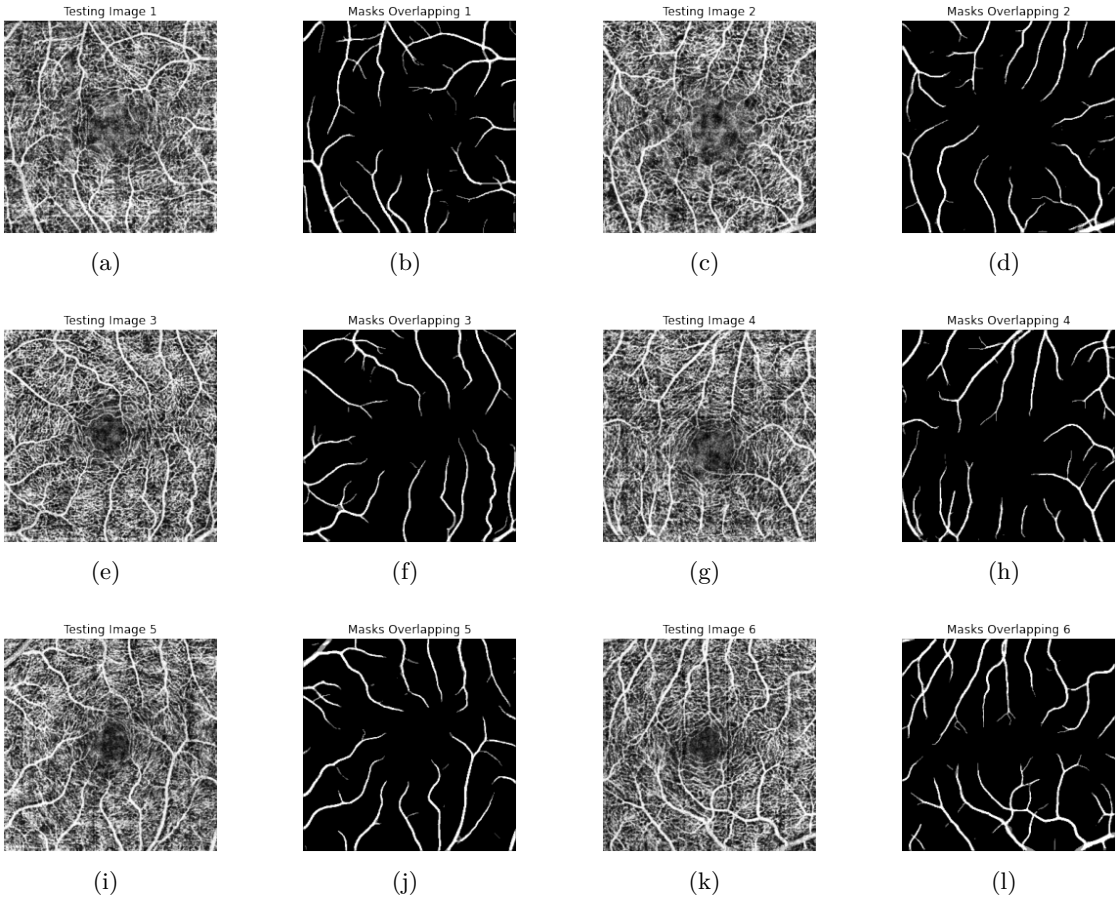


Figure 6.10: Segmentation results on 6 test images with batch size = 8, epochs = 1000, LR = 1^{-3}. Overlapping between ground truth and predicted mask

However, by zooming in the image, as shown in Figure 6.11, it is possible to notice how some parts are still incorrectly classified as background, even though they belong to the vascular tissue, justifying a minority class IoU of only 64.93%.

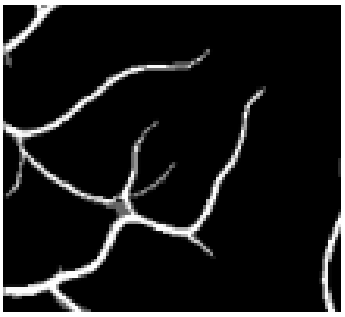


Figure 6.11: Zoom over a segmented test image

OCTA vs OCT

To conclude, one can state that the OCTA dataset performs significantly better than the OCT one in all its forms, as illustrated in Figure 6.12. This might have been easily predictable given the nature of the two imaging techniques, the former being focused on the detection of the vascular network and the latter more suited to the general analysis of the retinal structure.

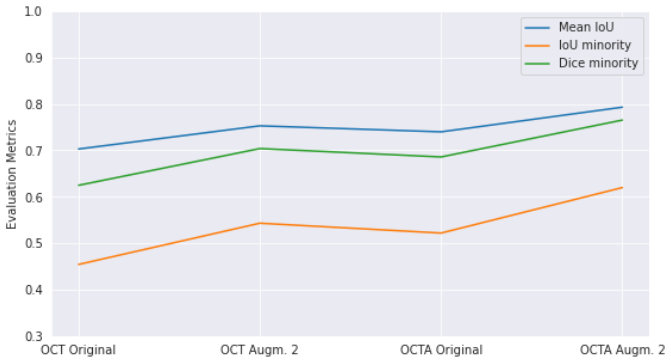


Figure 6.12: OCT vs OCTA performance comparison

Full Projection Map vs Maximum Projection Map

As mentioned in Chapter 3, it has been chosen to work mainly with full projection maps to have the opportunity to improve the results proposed by the reference paper and to deal with images that are noisier and more difficult to segment, assuming the fact that excellent results could also be obtained with maximum projection maps. Below in Figure 6.13, the best results obtained with the two projection techniques are shown.

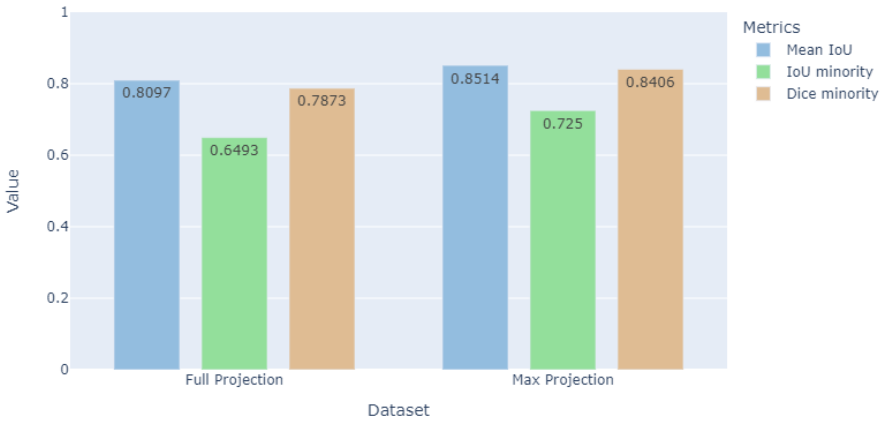


Figure 6.13: Comparison between full projection map and maximum projection map. Augmented dataset, Batch size = 8, Epochs = 1000, LR = 1^{-3}

It is possible to notice a significant improvement of 5% points in the Dice score and almost 8% points in the IoU of the minority class. This shows how precision and recall relative to the vascular part of the segmentation increase substantially. This behaviour was to be expected given the nature of the maximum projection map, which is particularly suited to highlight the retinal vascular structure.

6.2 U-Net

With the use of autoencoders there is a loss of *spatial information* which, however, is crucial for semantic segmentation since every pixel in the image should be classified. Usually in the predicted masks, there are fuzzy, soft or rounded objects due to the loss of most of the information outputted by the convolutional layers. The use of **U-Net** (Figure 6.14) satisfies to the need of retaining spatial information by providing this information from the encoder to the decoder. In detail, through the use of **skip connections**, the outputs of the convolutional layers in the encoder path are concatenated with those resulting from each upsampling phase in the decoding part, in order to avoid the loss of localization information [28].

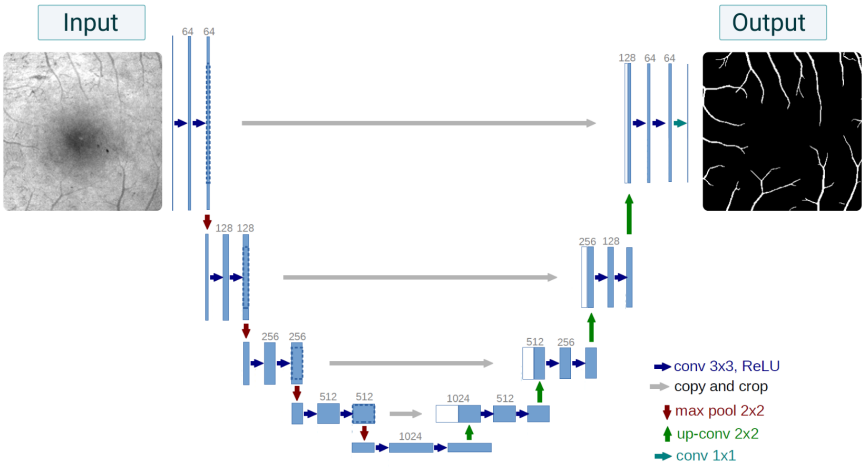


Figure 6.14: U-Net architecture. Each blue box corresponds to a multichannel feature map. The number of channels is denoted on top of the box. White boxes represent copied feature maps provided by grey skip connections. The arrows denote the different operations (adaptation of image obtained from [28])

U-Net structure is particularly suited for the segmentation of images in an end-to-end setting, meaning that, given a raw image input, it is able to provide a segmentation as output. Indeed, it is designed for semantic segmentation and is composed by two paths, as the traditional autoencoder: the path on the left (contraction or encoder path) and the path on the right (expansion or decoder path). Like all other convolutional networks, it consists of a large number of different operations. The input image is fed into the network, then the data is propagated towards the network along all possible paths and, at the end, the segmentation comes out. Each blue box in Figure 6.14 corresponds to a multichannel feature map. Most operations are convolutions followed by a non-linear activation function. Moreover, similarly to the autoencoder, the max pooling operation is used to reduce the size of the feature map, by propagating the maximum activation from each 2x2 window into the next feature map. After each max pooling operation, the number of feature channels is increased by a factor of 2. The sequence of convolutions

and max pooling operations in the encoder path results in a spatial contraction where it is possible to increase the "*what*" and at the same time decrease the "*where*". On the other hand, the decoder part of the architecture is dedicated to the creation of high-resolution segmentation maps. It consists of a sequence of up-convolutions and **concatenations** with the high-resolution features coming from the contraction path through the skip connections. The up-convolution layers use a learnt kernel to map each feature vector to the 2x2 pixel output vector, then followed by a non-linear activation function. The output segmentation map has two channels, one for the foreground and one for the background, which in this case correspond respectively to the vascular part in white and to the black background.

6.2.1 U-Net implementation

In the current implementation of U-Net, each convolution block is defined by the code (6.2).
The convolution operation is characterised by a 3x3 *kernel size* and *padding* is set as "same", meaning that it adds extra pixels to the edges, in order to force the dimension of the output image to be the same as the input one.
Then, since each feature map in output from the convolutional layers is not normalised, the *Batch Normalization* is applied to make the model easily converge. Indeed, this method helps to speed up the network, but the drawback is that it requires more memory for the calculation of mean and variance.
The activation function applied is the ReLU, as for the autoencoder model.
Moreover, it has been chosen to include the possibility of using a drop out, which is a form of regularisation that allows certain neurons in the network to be switched off. During training a tensor cell is kept active with a probability p and set to 0 otherwise. This forces the network to behave in a more robust way, reducing the problem of overfitting.

Code Listing 6.2: Convolution block code

```
def conv_block(x, filter_size, size, dropout, batch_norm=False):

    conv = layers.Conv2D(size, (filter_size, filter_size), padding="same")
                                                (x)
    if batch_norm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    conv = layers.Activation("relu")(conv)

    conv = layers.Conv2D(size, (filter_size, filter_size), padding="same")
                                                (conv)
    if batch_norm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    conv = layers.Activation("relu")(conv)

    if dropout > 0:
        conv = layers.Dropout(dropout)(conv)

    return conv
```

The convolutional block has been used to create the entire network as shown in code (6.3). It is very similar to the structure of an autoencoder, except for the concatenations characterising the skip connections.

Code Listing 6.3: Complete U-Net code

```
# Downsampling layers

conv_64 = conv_block(inputs, FILTER_S, FILTER_NUM, drop, batch_norm)
pool_64 = layers.MaxPooling2D(pool_size=(2,2))(conv_64)

conv_128 = conv_block(pool_64,FILTER_S,2*FILTER_NUM,drop,batch_norm)
pool_128 = layers.MaxPooling2D(pool_size=(2,2))(conv_128)

conv_256 = conv_block(pool_128,FILTER_S,4*FILTER_NUM,drop,batch_norm)
pool_256 = layers.MaxPooling2D(pool_size=(2,2))(conv_256)

conv_512 = conv_block(pool_256,FILTER_S,8*FILTER_NUM,drop,batch_norm)
pool_512 = layers.MaxPooling2D(pool_size=(2,2))(conv_512)

conv_1024 = conv_block(pool_512,FILTER_S,16*FILTER_NUM,drop,batch_norm)

# Upsampling layers

up_1024 = layers.Conv2D(8*FILTER_NUM, (FILTER_SIZE,FILTER_SIZE), padding =
                                        'same')(conv_1024)
up_1024 = layers.UpSampling2D(size=(UP_SAMP_SIZE, UP_SAMP_SIZE),
                                      data_format="channels_last")(up_1024)
up_1024 = layers.concatenate([up_1024, conv_1024], axis=3)
up_conv_1024 = conv_block(up_1024,FILTER_S,8*FILTER_NUM,drop,batch_norm)

up_512 = layers.Conv2D(4*FILTER_NUM, (FILTER_SIZE,FILTER_SIZE), padding =
                                      'same')(up_conv_1024)
up_512 = layers.UpSampling2D(size=(UP_SAMP_SIZE, UP_SAMP_SIZE),
                                      data_format="channels_last")(up_512)
up_512 = layers.concatenate([up_512, conv_512], axis=3)
up_conv_512 = conv_block(up_512,FILTER_S,4*FILTER_NUM,drop,batch_norm)

up_256 = layers.Conv2D(2*FILTER_NUM, (FILTER_SIZE,FILTER_SIZE), padding =
                                      'same')(up_conv_512)
up_256 = layers.UpSampling2D(size=(UP_SAMP_SIZE, UP_SAMP_SIZE),
                                      data_format="channels_last")(up_256)
up_256 = layers.concatenate([up_256, conv_256], axis=3)
up_conv_256 = conv_block(up_256,FILTER_S,2*FILTER_NUM,drop,batch_norm)

up_128 = layers.Conv2D(FILTER_NUM, (FILTER_SIZE,FILTER_SIZE), padding = '
                                      same')(up_conv_256)
up_128 = layers.UpSampling2D(size=(UP_SAMP_SIZE, UP_SAMP_SIZE),
                                      data_format="channels_last")(up_128)
up_128 = layers.Conv2D(FILTER_NUM, (FILTER_SIZE,FILTER_SIZE), padding = '
                                      same')(up_128)
up_128 = layers.concatenate([up_128, conv_128], axis=3)
up_conv_128 = conv_block(up_128,FILTER_S,FILTER_NUM,drop,batch_norm)

conv_final = layers.Conv2D(NUM_CLASSES, kernel_size=(1,1))(up_conv_128)
```

```
conv_final = layers.BatchNormalization(axis=3)(conv_final)
conv_final = layers.Activation('sigmoid')(conv_final)

model = models.Model(inputs, conv_final, name="UNet")
```

The input layer has a number of dimensions equal to 1, which typically refers to a greyscale image. In the next layer this dimension changes from 1 to 64 because there are 64 digital filters applied to the image in input (304, 304, 1). As the depth of the network grows, the number of filters increases by a factor of 2 until the last layer of the encoder, where the filters are 1024. Then, in the decoding part, the output of each *upsampling* layer is concatenated with the output of the symmetrical convolution that took place in the encoding part. As the convolutions advance, the number of filters decreases. Finally, the last *Conv2D* operation with 1x1 kernel and activation *Sigmoid* provide the output. As anticipated with the autoencoder, Sigmoid has been used for the purpose of binary segmentation, otherwise Softmax activation would have been applied for multiclass segmentation.

A summary of opening and closing blocks can be seen in detail below. In particular, Figure 6.15 shows the beginning of the U-Net with the sequence of two convolutional blocks, at the end of which, the output is passed both to the max pooling operation and to the skip connection to be concatenated to its symmetrical in the decoder part.

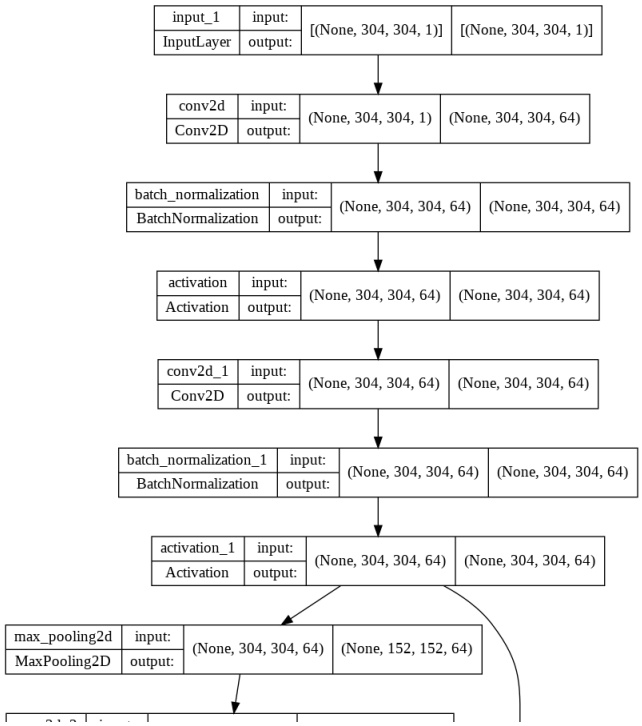


Figure 6.15: U-Net initial blocks

Figure 6.16, instead, displays the network closure where, after the upsampling operation, there is the concatenation of the feature map coming from the encoder and the

output of the upsampling in the decoder.

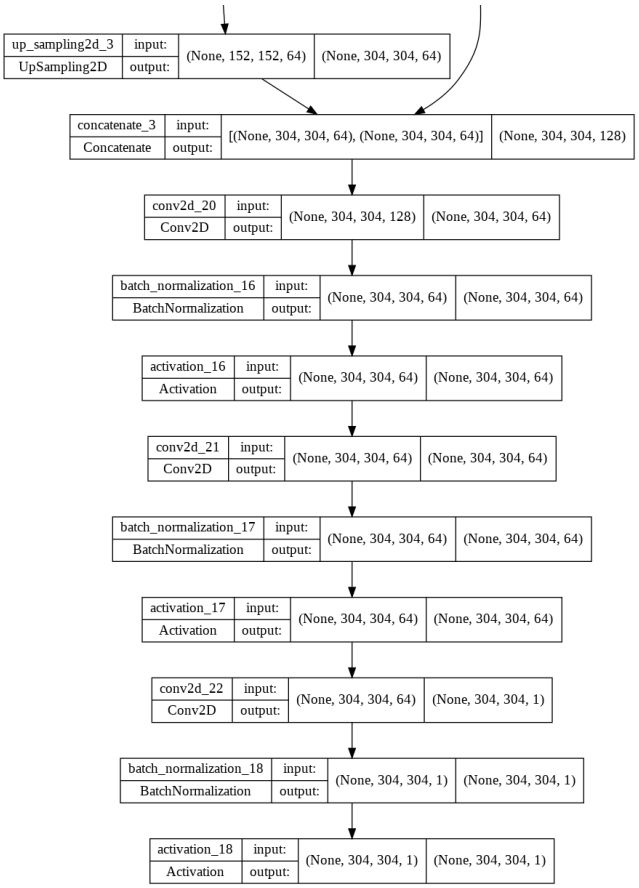


Figure 6.16: U-Net final blocks

6.3 Attention U-Net

Attention U-Net is a variation of the traditional U-Net in which only relevant activations are highlighted during training. This reduces the computational resources wasted on irrelevant activations (referred to areas that are not interesting) and provides a better generalisation of the network [29].

Attention means focusing only on the salient parts of the images which, in this specific case, are related to the blood vessels, and thus giving less importance to the background. In other words, it is a way to de-emphasise the background while, at the same time, emphasise the foreground. There are two types of attention:

Hard attention:

- Highlight relevant regions by cropping the image.
- Use of one region of an image at a time; this implies it is non-differentiable (because cropping is not a differentiable operation) and reinforcement learning is needed.
- Network can either pay attention or not, nothing in between.
- Backpropagation cannot be used.

Soft attention (attention gates):

- Weighting different parts of the image, by giving large weights to relevant crops and small weights to less important parts.
- Can be trained with backpropagation.
- During training, also the weights get trained making the model pay more attention to relevant regions.

To sum up, the main difference between hard and soft attention is that, in the latter, weights are added to pixels according to their relevance within the image.

As previously explained, traditional U-Net is able to combine spatial information from the encoder to the decoder, thanks to the use of skip connections. However, in the early stages of the encoder path, the feature representation is still quite weak because features are better learnt in deeper layers of the network. To solve the problem of poor feature representation of the initial layers, soft attention is implemented at every skip connections and actively suppresses activations at irrelevant regions, providing more weightage to the features of interest.

6.3.1 Attention U-Net implementation

Figure 6.17 shows a modified version of the traditional U-Net in which the attention block has been added. As input to the attention gate (AG) there are two arrows, the gating signal (that is basically a query) and the skip connection.

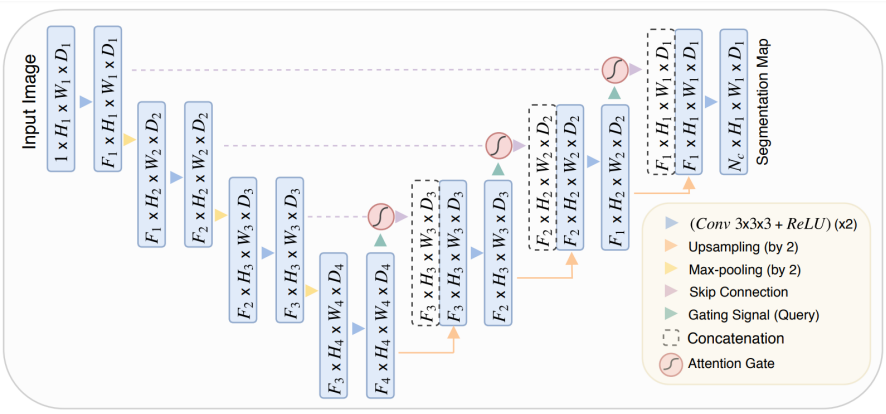


Figure 6.17: Attention U-Net segmentation model. Input image is progressively filtered and downsampled by factor of 2 at each scale in the encoding part of the network. Attention gates (AGs) filter the features propagated through the skip connections (obtained from [29])

By expanding the attention gate, visible in Figure 6.18, two inputs can be underlined:

- g, which is the *gating signal* that comes from the previous layer of the network. Since it always derives from a deeper part of the network, it has a better feature representation.
- x, which comes from the skip connections of the encoder path. Since it comes from the upper layer, it provides a better spatial information.

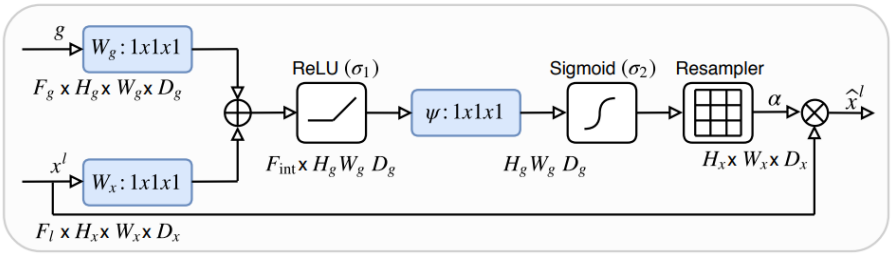


Figure 6.18: Schema of the proposed attention gate (obtained from [29])

The two inputs x and g are characterised by different dimensions because x is coming from an upper layer of the network. However, to obtain a combination of the spatial and the feature information, it is necessary that the two inputs have the same dimension. For this reason, a convolutional operation with a padding stride = (2, 2) is applied to x, in order to halve its dimension. By doing so, the two contributions x and g can be added. When adding them, aligned weights get larger, while unaligned weights get relatively smaller. They are then passed through the ReLU activation function, which allows to obtain values ranging from 0 to any positive number for the weights. Next,

it is necessary to scale them to values included in the range 0 to 1, and this is done through the application of a sigmoid function. The weights obtained are then resampled or upsampled to the original size of x. Finally, the weights are multiplied element-wise with the original vector x, in order to scale the vector itself based on relevance.

The technical implementation follows the same procedure used for the U-Net architecture, with convolutional blocks, max pooling, upsampling, batch normalisation and activation functions, but with the addition of the attention block, as evident from the (6.4) which mirrors the 6.18 structural scheme.

Code Listing 6.4: Attention U-Net Block

```
def attention_block(x, gating, inter_shape):
    shape_x = K.int_shape(x)
    shape_g = K.int_shape(gating)

# Getting the x signal to the same shape as the gating signal
    theta_x = layers.Conv2D(inter_shape, (2, 2), strides=(2, 2), padding='
                                        same')(x)  # 16
    shape_theta_x = K.int_shape(theta_x)

# Getting the gating signal to the same number of filters as the
                                    inter_shape
    phi_g = layers.Conv2D(inter_shape, (1, 1), padding='same')(gating)
    upsample_g = layers.Conv2DTranspose(inter_shape, (3, 3),strides=(
                                        shape_theta_x[1] // shape_g[1],
                                        shape_theta_x[2] // shape_g[2]),
                                        padding='same')(phi_g)  # 16

    concat_xg = layers.add([upsample_g, theta_x])
    act_xg = layers.Activation('relu')(concat_xg)
    psi = layers.Conv2D(1, (1, 1), padding='same')(act_xg)
    sigmoid_xg = layers.Activation('sigmoid')(psi)
    shape_sigmoid = K.int_shape(sigmoid_xg)
    upsample_psi = layers.UpSampling2D(size=(shape_x[1] // shape_sigmoid[1
                                        ], shape_x[2] // shape_sigmoid[2]
                                        ))(sigmoid_xg)  # 32

    upsample_psi = repeat_elem(upsample_psi, shape_x[3])
    #upsample_psi = repeat_elem(upsample_psi, 3)

    y = layers.multiply([upsample_psi, x])

    result = layers.Conv2D(shape_x[3], (1, 1), padding='same')(y)
    result_bn = layers.BatchNormalization()(result)
    return result_bn
```

6.4 Attention Residual U-Net

The Attention Residual U-Net is a neural network that employs, in addition to the concept of attention, the idea of residual, thus combining the residual structure and the channel attention mechanism.
The concept of *residual* has been firstly introduced by the inventors of ResNet architecture [30]. Deep neural networks learn a hierarchical set of representations (low, mid and high-level features); in images, this is analogous to learning edges, shapes and then objects. So, theoretically more layers should enrich the levels of the features. However, adding more and more convolutional layers on top of activations and batch normalisations, the training will eventually get worse. The authors of the paper offered a construction insight: considering a shallow architecture and its deeper counterpart with more layers, theoretically all the deeper model would need to do is to just copy the output from the shallow model with identity mappings, in a way that the deeper model should produce no higher training error than its shallow counterpart. However, the identity functions are not an easy function to learn and so the residual functions formulate the layers by having a reference to the input x through the skip connections.
Moreover, residual networks solve one of the biggest challenges in deep learning, known as **vanishing gradient**. This phenomenon often occurs when the network depth is too wide, and the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This implies that the weights do not update their values and thus, no learning is achieved. On the other hand, with residual blocks, the gradients can flow directly through the skip connections backwards from later layers to initial filters; indeed, the network incorporates identity shortcut connections which essentially skip the training of one or more layers - creating a residual block, allowing the network to memorise from previous layers. Figure 6.19 shows the structure of a residual block in ResNet.

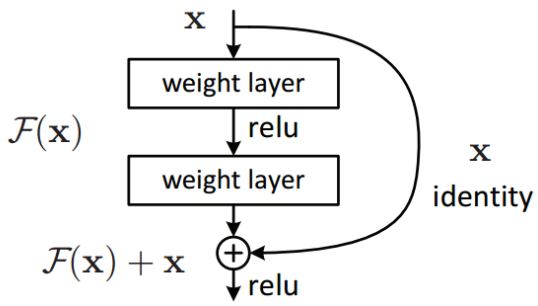


Figure 6.19: Residual learning: a building block (obtained from [30])

Considering x as the input of a neural network, and given that the general aim is to learn the true distribution $H(x)$, the difference (or the residual) is:

$$F(x) = Output - Input = H(x) - x \rightarrow H(x) = F(x) + x \tag{6.3}$$

Figure 6.20 compares a standard convolutional neural network (left side) and a residual neural network (right side). In the standard CNN, the aim is to map the input x directly to the output which is $f(x)$. On the other hand, since finding the direct mapping is not a trivial operation in deep networks, the main idea of residual blocks is to find the residual $R(x) = f(x) - x$, which is then added to the identity connection x to obtain the final output $f(x) = R(x) + x$.
To summarise, layers in a traditional network learn the true output, while layers in the residual block learn the residual.

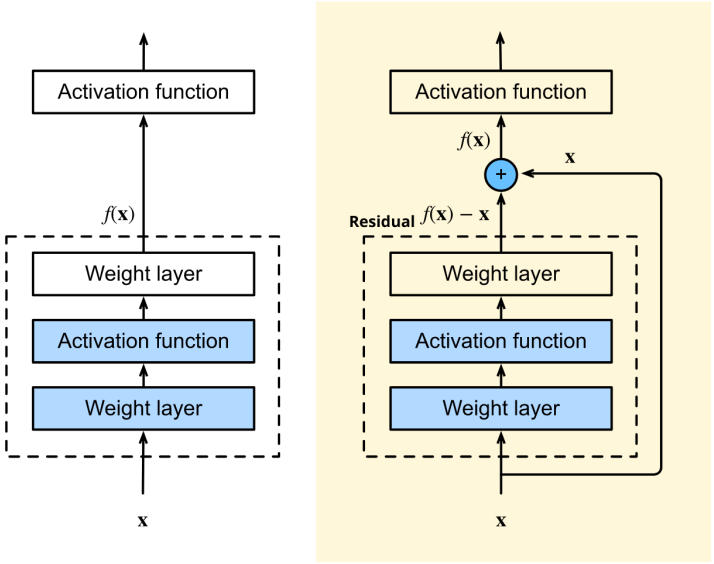


Figure 6.20: Traditional CNN vs Residual Net (adaptation of image obtained from [32])

In traditional neural networks, each layer feeds into the next one because they are connected one after the other, whereas in residual networks, each layer can feed into the next layer and directly into the layers that are two or more hops away. In this way, inputs can forward propagate faster through the residual connections across various layers. By doing so, it is possible to stack more residual blocks on top of each other, without decreasing in performance, as would happen in deep standard convolutional neural networks.

6.4.1 Attention Residual U-Net implementation

In the Attention Residual U-Net there is a typical convolution block but then the outputs are actually summed between the convolution block output and the original input itself, meaning that, when the convolution block is trained, it is actually trained for the residual part. The technical application of the structure described above can be seen in the code (6.5), where a basic Convolution Residual block is created.

Code Listing 6.5: Attention Residual U-Net - Convolution Block

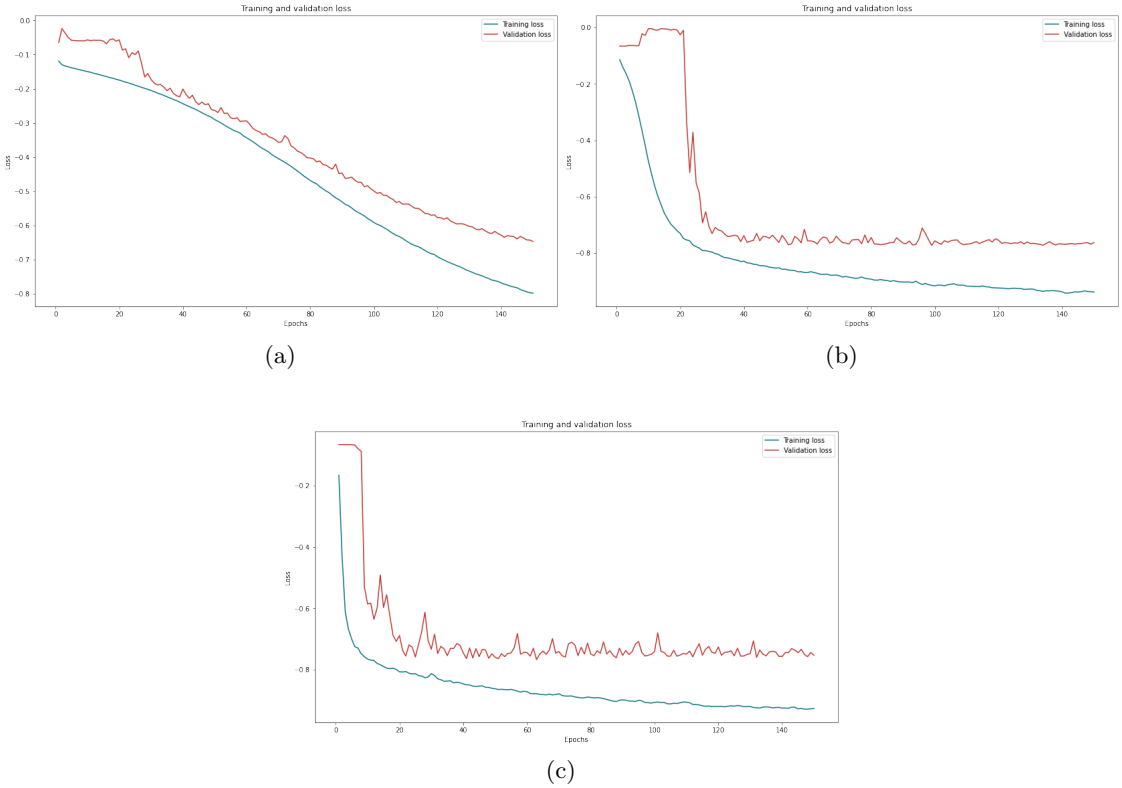
```
def res_conv_block(x, filter_size, size, dropout, batch_norm=False):
    conv = layers.Conv2D(size, (filter_size, filter_size), padding='same')
                                                (x)
    if batch_norm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    conv = layers.Activation('relu')(conv)

    conv = layers.Conv2D(size, (filter_size, filter_size), padding='same')
                                                (conv)
    if batch_norm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    if dropout > 0:
        conv = layers.Dropout(dropout)(conv)

    shortcut = layers.Conv2D(size, kernel_size=(1, 1), padding='same')(x)
    if batch_norm is True:
        shortcut = layers.BatchNormalization(axis=3)(shortcut)

    res_path = layers.add([shortcut, conv])
    res_path = layers.Activation('relu')(res_path)
    return res_path
```

6.5 U-Net structures results

As for the autoencoder model, in order to evaluate the performance various parameters have been tuned. Both the original dataset and the augmented one (with the pre-selected set of transformations) have been tested.

Traditional U-Net (Original Dataset)

The loss behaviour at the tuning of the learning rate can be observed in Figure 6.21. All the runs have been carried out with a batch size of 8 and 150 epochs.



(a) (b)

(c)

Figure 6.21: Tuning of learning rate. Training and validation losses from OCTA **original** dataset not augmented.
Batch size = 8, Epochs = 150. (a) LR = 1^{-3}; (b) LR = 1^{-2}; (c) LR = 1^{-1}

In Figure (a), the learning rate equal to 1^{-3} seems to be quite low for the U-Net structure. Indeed, its behaviour after 150 epochs is still very steep and with margins for improvement. With the autoencoder, this trend was much less evident, but it must be taken into account that the epochs were 1000, a not testable condition with the U-Net due to the temporal and computational constraints. The most promising behaviour seems to be the one of the model trained with LR = 1^{-2} (b). The validation loss seems stable and a plateau is already reached after 120 epochs. However, with a higher learning rate (LR

$= 1^{-1}$) a rather good but less stable result is achieved, characterised by more oscillations due to the choice of a wider step, as appreciable in Figure (c).

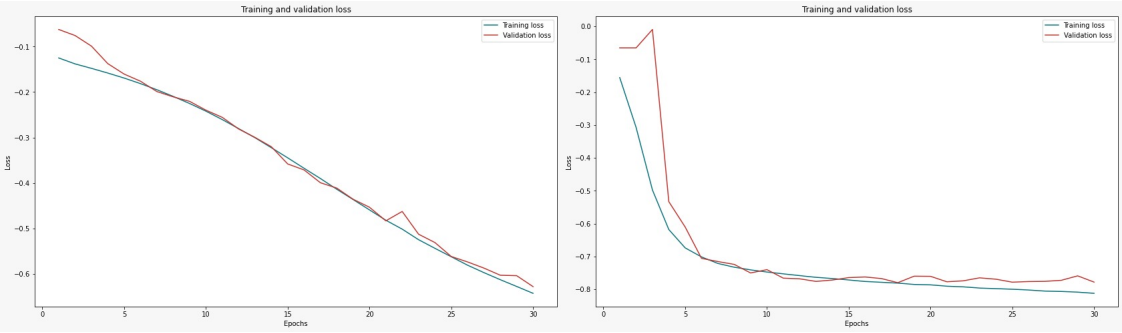
In the Table 6.3, a summary of the main metrics evaluated on the test set can be seen, confirming that a LR of 1^{-2} provides the best results.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
150	0.001	8	0.8702	0.7597	0.8635
150	0.01	8	0.8704	0.7603	0.8638
150	0.1	8	0.8653	0.7506	0.8576

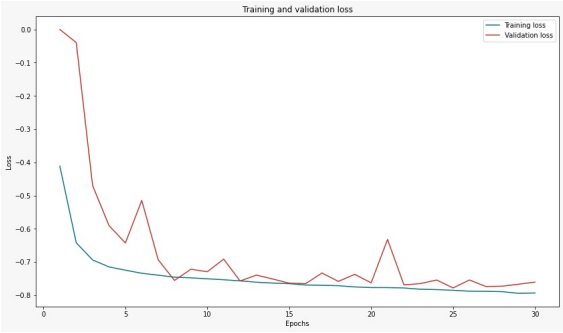
Table 6.3: Tuning of learning rate with batch size = 8 and epochs = 150. Metrics obtained for OCTA **original** dataset

Traditional U-Net (Augmented Dataset)

Similar tuning steps have been done also for the augmented dataset. In Figure 6.22 the loss trend can be appreciated.



(a) (b)



(c)

Figure 6.22: Tuning of learning rate. Train and validation losses from OCTA **augmented** dataset.
Batch size = 16, Epochs = 30. (a) LR = 1^{-3}; (b) LR = 1^{-2}; (c) LR = 1^{-1}

The number of epochs has been decreased from 150 to 30, due to GPU and time constraints. In particular, each run with the augmented dataset took about 90 minutes. For this reason, the results are not directly comparable and therefore partial. However, a very similar trend to the original dataset can be observed, but with an increase in performance, even with a limited number of epochs available.

Given the small number of epochs, as expected, the lowest LR of 1^{-3} (a) led to a very steep descending behaviour that could certainly be improved by increasing the epochs, always taking into account to avoid overfitting. The best result is obtained with a LR of $1^{-}2$ (b), while with the highest LR (c) it is noticeable an oscillatory behaviour. As in the autoencoder model, there is a significant overlapping between training and validation losses, due to the noisier and more numerous natures of the training part.

Table 6.4 summarises the different metrics obtained on the test set.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
30	0.001	16	0.8759	0.7709	0.8706
30	0.01	16	0.8809	0.7793	0.8760
30	0.1	16	0.8739	0.7665	0.8678

Table 6.4: Tuning of learning rate with batch size = 16 and epochs = 30. Metrics obtained for OCTA **augmented** dataset

A direct comparison between original and augmented dataset is provided by Figure 6.23, where a slight increase in the performance is evident with the augmented dataset.

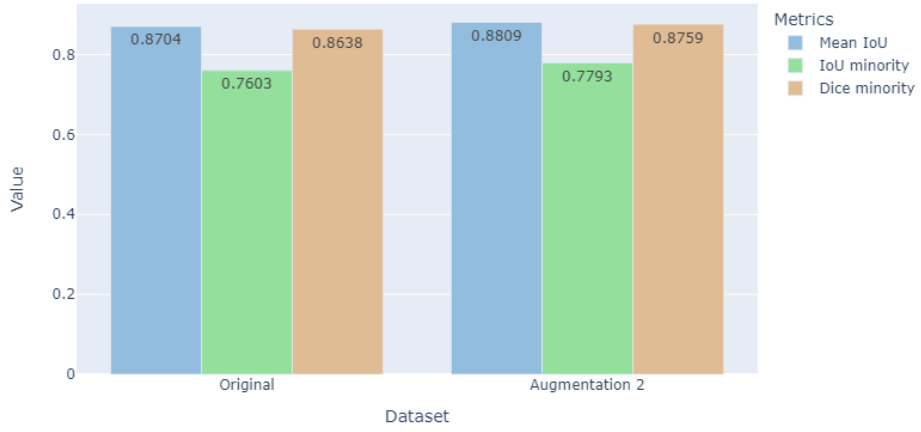


Figure 6.23: Original dataset and augmented dataset comparison.
For the **original** dataset: Epochs = 150, Batch size = 8, LR = 1^{-2}. For the **augmented** dataset: Epochs = 30, Batch size = 16, LR = 1^{-2}

However, as previously anticipated, it must be noted that a direct comparison with

the same epochs and batch sizes has not been possible due to the computational resources available.

Below is shown the graphical result of the segmented images obtained after training the model with the augmented dataset, 30 epochs, batch 16 and LR equal to 1^{-2}. The results are extremely more accurate than the ones of the autoencoder, as might be expected from a network better suited to the segmentation task. Indeed, a result of 88.09% average IoU is obtained, versus a 80.97% for the traditional autoencoder. Looking into a more detailed zoom of the overlapping between ground truth and prediction (Figure 6.25), a more accurate and detailed result can be seen, arriving at 77.93% IoU of the minority class (the vascular part) versus 64.93% of the autoencoder.

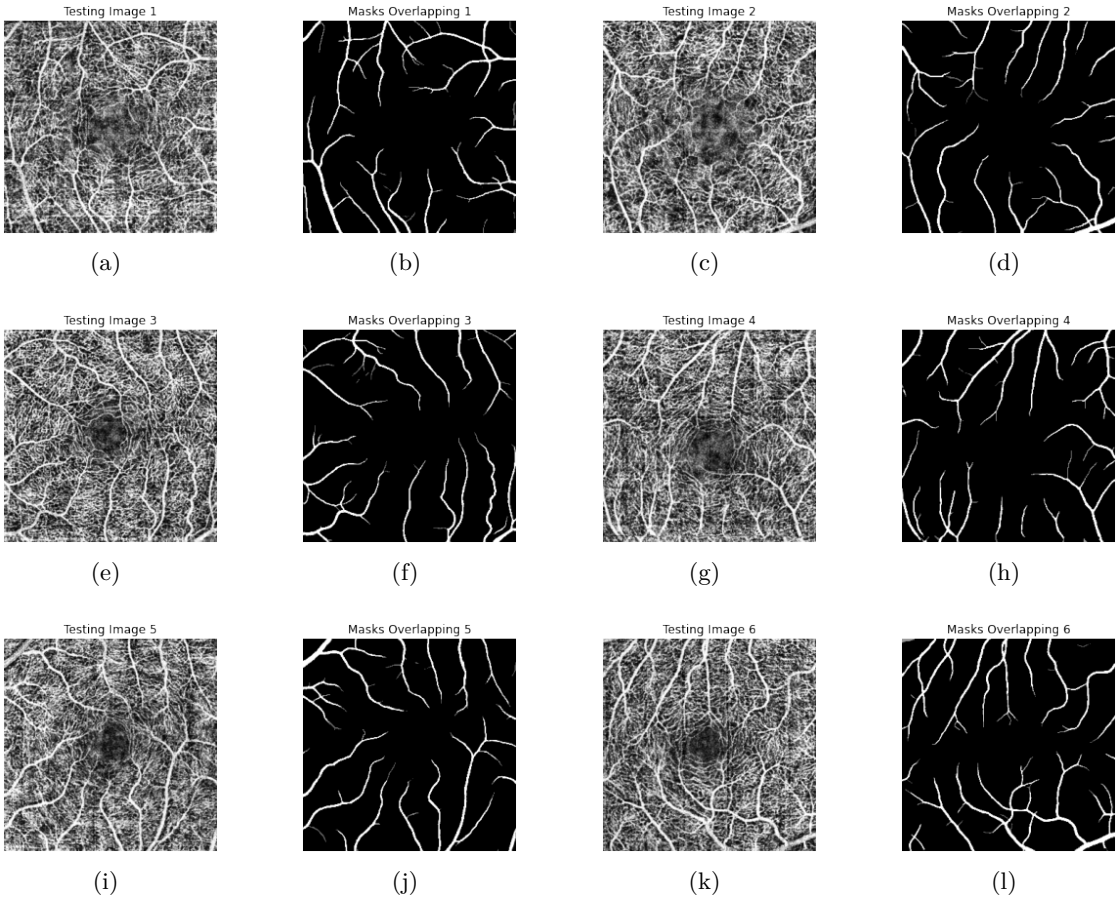


Figure 6.24: Segmentation results of traditional U-net on 6 test images with batch size = 16, epochs = 30, LR = 1^{-2} and augmented dataset. Overlapping between ground truth and predicted masks

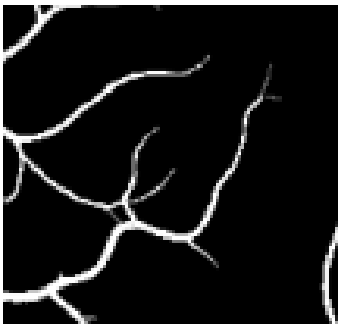


Figure 6.25: Zoom over a segmented test image

Traditional U-Net (OCT vs OCTA)

As already evident from the use of the traditional autoencoder, performance has increased significantly in the transition from OCT to OCTA, and this is an excellent sign for this extremely promising new technology in the field of vascular segmentation.

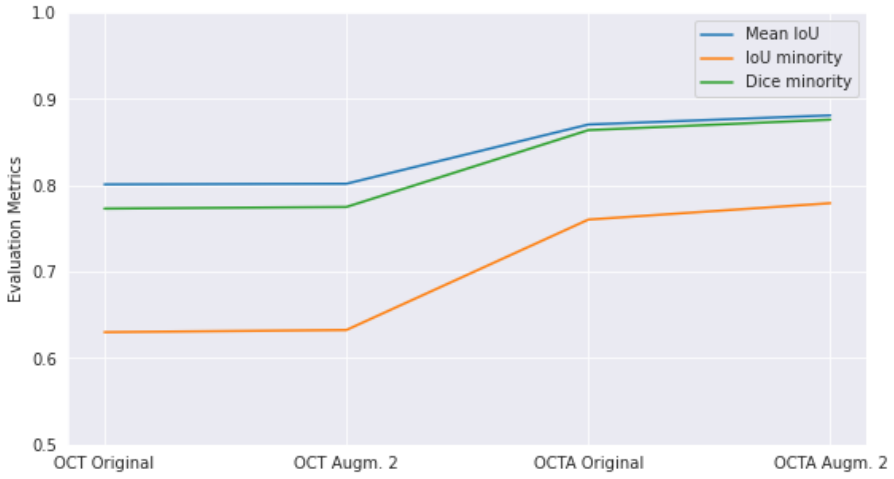


Figure 6.26: OCTA vs OCT performance comparison

Traditional U-Net (Full Projection Map vs Maximum Projection Map)

As mentioned in Chapter 3, it has been chosen to mainly work with full projection maps but also a brief comparison with maximum projection 2D images has been done. Figure 6.27 shows a comparison between the best results obtained with the two projection techniques.

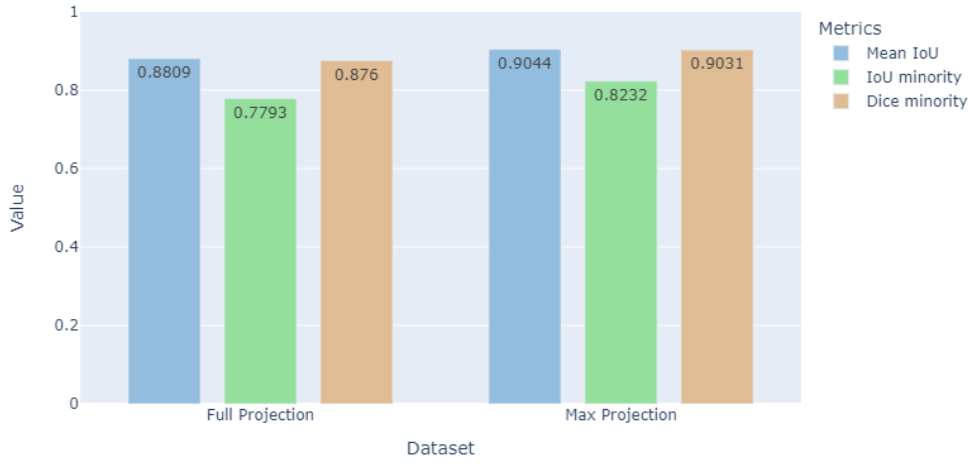
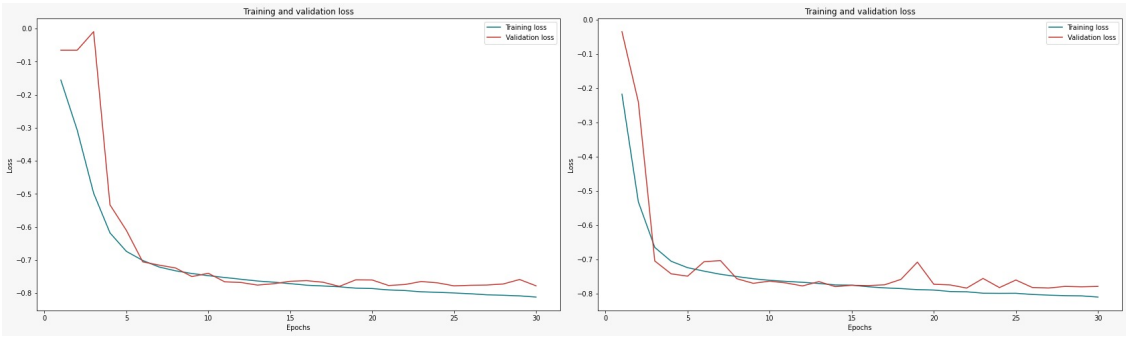


Figure 6.27: Comparison between full projection map and maximum projection map. Augmented dataset, Batch size = 16, Epochs = 30, LR = 1^{-2}

In detail, it is appreciable an improvement of almost 3% points in the Dice score and about 4.5% points in the IoU of the minority class when using the MPMs. The improvement is less significant than the one obtained with the traditional autoencoder, but the total result is very accurate, bringing this performance to be the best found so far and in line with the performance found by reference papers [7] and [8].

Attention U-Net

For the Attention U-Net, due to computational constraints related to RAM memory, a batch size of 8 has been chosen, which is rather small despite the large size of the database. In Figure 6.28 can be observed a similar trend to the traditional U-Net one, with equal LR and epochs.



(a) (b)

Figure 6.28: Training and validation losses from OCTA **augmented** dataset.
(a) Traditional U-Net: Batch size = 16, Epochs = 30, LR = 1^{-2}. (b) Attention U-Net: Batch size = 8, Epochs = 30, LR = 1^{-2}

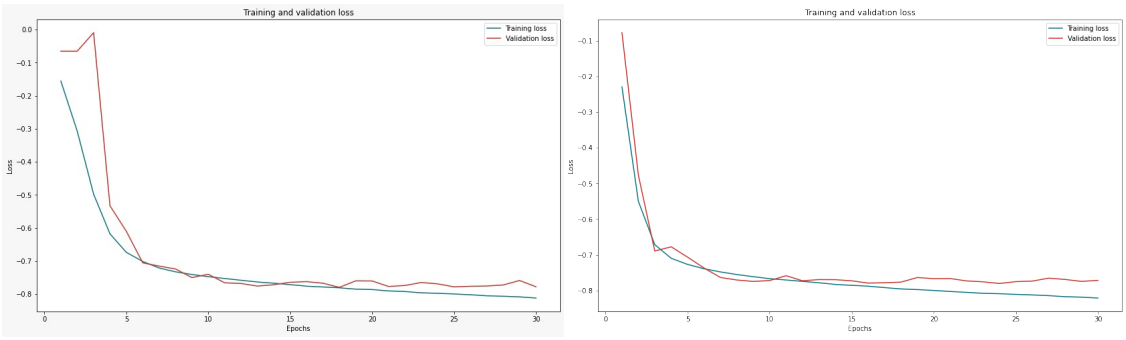
In the Table 6.5, the metrics obtained with the main changes in the learning rate are collected. The performance is very similar to the one of the traditional U-Net and therefore no significant improvement is noticeable. The reason for this could be due to several factors. Firstly, the behaviour of both models over a longer time period or epochs should be investigated. Secondly, as the Attention U-Net is a rather deep network with a higher complexity than the traditional U-Net, a larger dataset could certainly lead to an increase in performance. Finally, due to the nature of the Attention U-Net, there is a particular spatial focus that should help in isolating in the best possible way the features of the object to be segmented (which in general could be cells or mitochondria); however, in the case of a vascular segmentation and so of a network of vessels, the object to be segmented can hardly be ascribed to a precise location and thus the attention module could be less effective.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
30	0.001	8	0.8829	0.7830	0.8783
30	0.01	8	0.8807	0.7786	0.8755
30	0.1	8	0.8826	0.7825	0.8780

Table 6.5: Tuning of learning rate with batch size = 8 and epochs = 30. Metrics obtained for OCTA **augmented** dataset

Attention Residual U-Net

A very similar analysis can be made for the Residual Attention U-Net. In Figure 6.29 can be seen a similar trend in the loss functions compared to the traditional U-Net and also a slight increase in the validation loss, indicating a hint of overfitting.



(a) (b)

Figure 6.29: Training and validation losses from OCTA **augmented** dataset.
(a) Traditional U-Net: Batch size = 16, Epochs = 30, LR = 1^{-2}. (b) Attention Residual U-Net: Batch size = 8, Epochs = 30, LR = 1^{-2}

In the Table 6.6 below, it is possible to get a closer look at the main results obtained by changing the values of learning rate.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
30	0.001	8	0.8772	0.7728	0.8718
30	0.01	8	0.8811	0.7794	0.8760
30	0.1	8	0.8767	0.7716	0.8711

Table 6.6: Tuning of learning rate with batch size = 8 and epochs = 30. Metrics obtained for OCTA **augmented** dataset

However, despite the use of a more powerful network such as the Attention Residual U-Net, no significant improvement in the performance can be appreciated, compared to the previous two architectures. This may be due to the fact that, the concept of residual and the implementation of residual blocks, are useful in case of very deep neural networks to overcome problems such as degradation or vanishing gradient, which instead are not present in this specific architecture, where the number of layers is not so high.

6.6 Generative Adversarial Network - GAN

Neural networks such as autoencoders are easy to implement, stable and quite effective. Often, however, the training is based on the use of a loss function, like the MSE, which relies on the low frequency components of the images, thus creating rather blurry images.

Generative Adversarial Networks are designed for image generation and their operation is based on the use of Adversarial loss. Instead of using a mathematical function, a GAN uses a second neural network as the loss function for the first neural network. In order to perform the training, it is necessary to calculate the similarity between the predicted image and the real one, and this measure is carried out by the second neural network. Thanks to their nature and to the fact that they do not need paired data, GANs represent the state-of-the-art in image generation and image translation [33].

A basic structure of a Generative Adversarial Network is represented in Figure 6.30. It is possible to distinguish between a generator part and a discriminator one. The aim of the generator is the creation of fake images that should be as similar as possible to the real ones, whereas the discriminator model tries to distinguish the real images from the fake images created by the generator.

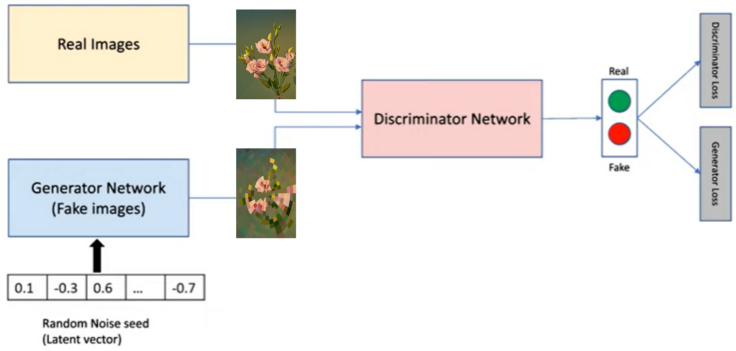


Figure 6.30: Structure of a basic Generative Adversarial Network (adaptation of image obtained from [45])

The generator receives as input a random latent vector and it has a similar role to the decoder part of an autoencoder. Indeed, starting from the random latent vector, it will be able to construct a "fake" image. The fake image is then passed through the discriminator network that receives in input both the output of the generator and the "real" image. The discriminator is usually a common convolutional network and, since its purpose is to classify an image as "real" or "fake", its output will simply be a binary 0 or 1 label. The training process results into two different kinds of losses, the generator loss that should tell to the generator how to improve his task of creating images as similar as possible to the real ones, and the discriminator loss that teaches to the discriminator

how to distinguish a real image from a fake one.

In the picture 6.31 below, the process of "fooling" the discriminator can be appreciated. An input vector is passed to the generator which generates a "fake" image that, for his fake nature, would have a label 0. The image is then passed to the discriminator but with a ground truth label 1, instead of 0. This step is for the purpose of "fooling" the discriminator which, if sufficiently trained, should realize that the image received is actually a fake image, and thus would predict a value of 0. If so, the loss between his prediction (0) and ground truth (1) would be high and would return to the generator the information that the generated image is still not similar to the real one and that the discriminator is perfectly able to classify it as fake. Conversely, when the generator is sufficiently able to generate images which are very similar to the real ones, the discriminator will be fooled and will be convinced that it is looking at a real image.

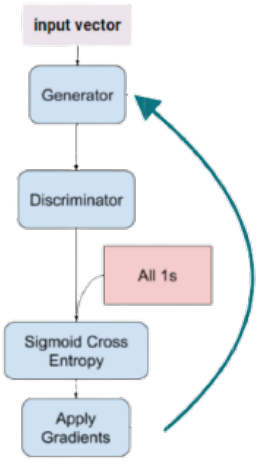


Figure 6.31: Generator loss and network updating (adaptation of image obtained from [47])

This training technique is called adversarial because the two losses of the generator and the discriminator constantly compete against each other, trying to generate images that are as real as possible while at the same time perfectly learning to distinguish fake from real ones. Indeed, when the discriminator has a lower loss because it is better at discriminating between real and fake images, the generator loss will increase and vice versa. Unfortunately, this also means that the losses are not useful to tell if the results are good or bad, as it happens instead in standard neural networks. The easiest way to evaluate a GAN is looking at the images and see if they look good or bad, but of course it is not a quantitative approach. Indeed, it is not trivial to obtain quantitative data on how a GAN performs. The most common metric is the Frechet Inception Distance which computes the distance between the feature vectors of the real and generated images using a pretrained Inception v3 classifier.

6.6.1 Pix2Pix GAN

The Pix2Pix GAN is a particular type of a conditional GAN [34]. A Generative Adversarial network is considered as conditional generative model if, in addition to a random latent vector, the class label is also provided and thus the generator is conditioned by that. The input of the generator is therefore the concatenation of the random latent vector plus the class label. The discriminator, instead, takes as input two couples of image and label, i.e., the fake pair and the real one (Figure 6.32).

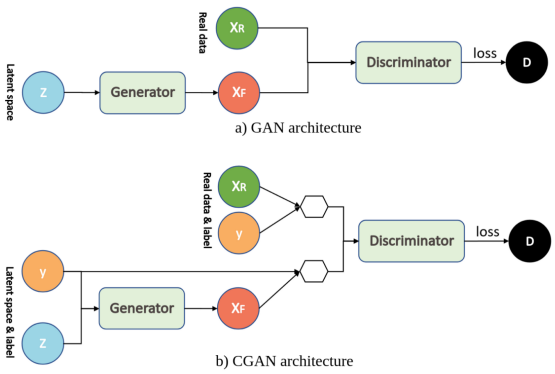


Figure 6.32: Flow Diagram representing GAN (a) and Conditional GAN (b) (obtained from [46])

Pix2Pix GAN has been conceived for the scope of image-to-image translation [35]. It is a particular architecture that takes in input pairs of images corresponding to each other. For instance, providing a segmented mask, the generator can create a realistic looking images or vice versa. It can be used for image colourisation, converting day images into night ones, or transforming areal views to maps.

As can be seen in Figure 6.33, in the Pix2Pix GAN the generator is represented by a U-Net that takes in input the original image that have to be segmented. Through the U-Net a "fake" segmented image is generated but, instead of just training the model with the Jaccard loss as already seen in U-Net chapter, in this case the updating of the model is made through the use of a discriminator structure. Indeed, the generated "fake" segmentation is combined with the real original image and this couple is given to the discriminator, which receives also the "true" couple composed by the ground truth segmentation mask concatenated with the original image itself. The role of the discriminator, as for a general GAN, is to distinguish between a real couple of image and a fake one.

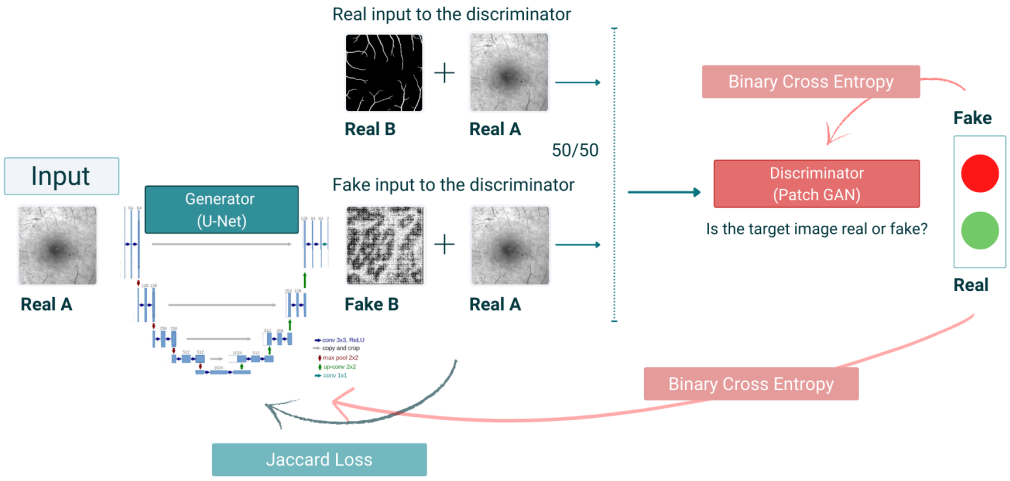


Figure 6.33: Flow Diagram representing Pix2Pix GAN

The discriminator is a binary classifier and, in particular, it is a PatchGAN [36], a simple convolutional network that tries to classify if each N×N patch in an image is real or fake (as opposed to classifying an entire image such in traditional GANs). This discriminator is run convolutionally across the image, averaging all responses to provide the final output, as can be observed in Figure 6.34. Using a 16×16 PatchGAN is sufficient to promote sharp outputs and achieve good results, but also leading to tiling artifacts.

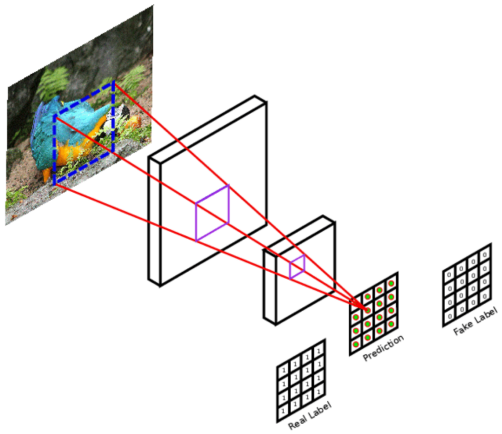


Figure 6.34: PatchGAN discriminator. Each value of the output matrix represents the probability of whether the corresponding image patch is real or artificially generated (obtained from [36])

The 70×70 PatchGAN alleviates these artifacts and achieves slightly better scores. Scaling beyond this to the full 304×304 image, does not appear to improve the visual quality of the results and this may be because the full complete image has many more parameters and greater depth than the 70×70 PatchGAN, and so it may be harder to train [35].

Adversarial training is represented by the combination of a generator loss and a discriminator loss. More specifically, in Figure 6.35 (a) the process of updating the weights of the discriminator part can be seen. As previously mentioned, the discriminator takes in input two couple of images (generated segmentation mask + input image & target segmentation mask + input image). The discriminator classifies each input as fake (0) or real (1) and, for each couple, the ground truth label is also provided (0 if the input is fake while 1 if the input is real). The loss function between the discriminator prediction and the ground truth is then calculated and the gradients are applied in order to backpropagate the errors and teach to the discriminator how to improve its classification ability.

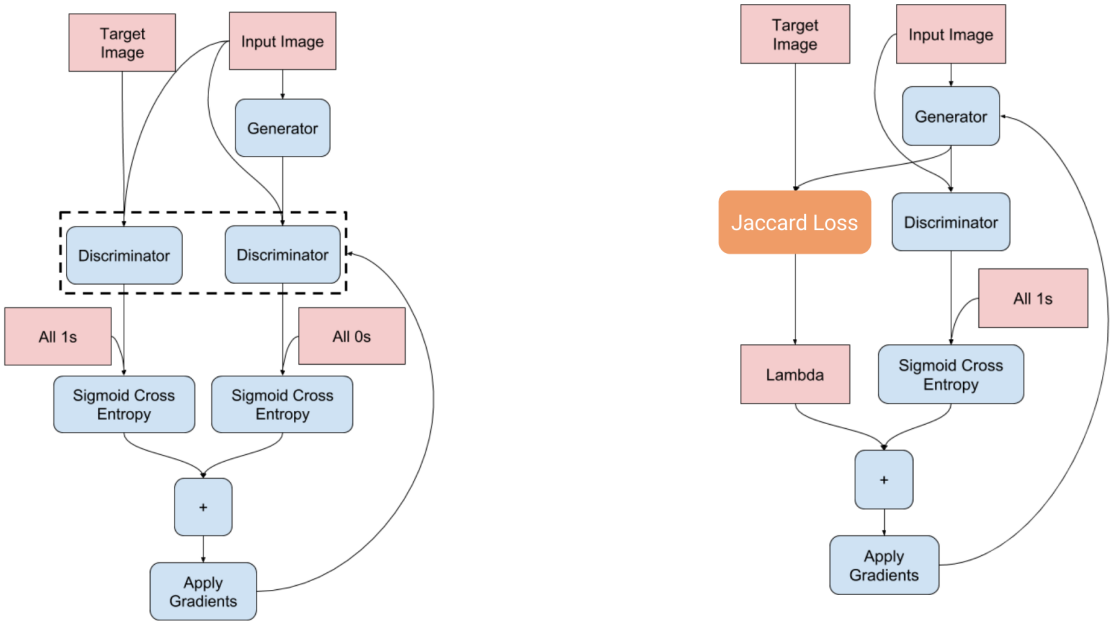


Figure 6.35: Discriminator (a) and Generator (b) updating and losses (adaptation of image obtained from [47])

In the Figure 6.35 (b), the weights updating of the generator part is displayed. In this part of the network, the U-Net generator receives in input the original image and, as said, provides as output a "fake" generated segmentation mask. This "fake" image is then compared with the target segmentation mask and the difference between these two images is computed through the Jaccard coefficient (instead of the L1 loss used in [35]).

At the same time, the couple composed by the generated mask and the original image is passed through the discriminator which, in this case, is fooled because it receives as ground truth label 1, even if the input pair of images is the fake couple. In this way, if the discriminator is able to understand that the input couple is fake, then the loss generated between his prediction (0) and the given ground truth (1) would be high, and this should tell the generator that it is not good enough in generating images similar to the real ones because the discriminator is still able to distinguish them. Vice versa, if the discriminator is fooled, it means that the generator has become very good in generating fake samples, and so it should increase its ability to distinguish a real image from a fake one. The Jaccard loss previously computed is then summed (with a $\lambda = 100$) to the Sigmoid Cross Entropy coming from the discriminator. The combination of these two losses solves the blurring problem and, at the same time, reduces the problem of visual artifacts. The gradients are then applied and the discriminator is trained.

6.6.2 Pix2Pix GAN implementation

For what concerns the implementation of the Pix2Pix GAN, at the code (6.6) the structure of the discriminator can be analysed. It can be noticed the use of the concatenation between the original image and the segmentation mask, and the application of Adam optimizer with a LR = 0.0002 as suggested from the paper [35].

Code Listing 6.6: Discriminator implementation code

```
def define_discriminator(image_shape,target_shape):
  # weight initialization
  init = RandomNormal(stddev=0.02,seed = 1305)
  in_src_image = Input(shape=image_shape)  #Original image
  in_target_image = Input(shape=target_shape)  #Segmentation mask
  # concatenate images, channel-wise
  merged = Concatenate()([in_src_image, in_target_image])

  # C64: 4x4 kernel Stride 2x2
  d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=
                                    init)(merged)
  d = LeakyReLU(alpha=0.2)(d)
  # C128: 4x4 kernel Stride 2x2
  d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer
                                    =init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # C256: 4x4 kernel Stride 2x2
  d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer
                                    =init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # C512: 4x4 kernel Stride 2x2

  d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer
                                    =init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # second last output layer : 4x4 kernel but Stride 1x1
  d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
  d = BatchNormalization()(d)
  d = LeakyReLU(alpha=0.2)(d)
  # patch output
  d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
  patch_out = Activation('sigmoid')(d)
  # define model
  model = Model([in_src_image, in_target_image], patch_out)
  # compile model

  opt = keras.optimizers.Adam(learning_rate=0.0002,beta_1=
                                    beta1_discriminator)
  model.compile(loss='binary_crossentropy',optimizer=opt,loss_weights=[0.5
                                    ])
  return model
```

The generator structure follows a U-Net model, already described in the previous

chapter. Finally, the entire GAN is assembled (6.7) and the real and fake samples are created (6.8) in order to feed them as input to the discriminator.

Code Listing 6.7: GAN implementation code (discriminator + generator)

```
def define_discriminator(image_shape,target_shape):
  def define_gan(g_model, d_model, image_shape):
  # make weights in the discriminator not trainable
  for layer in d_model.layers:
    if not isinstance(layer, BatchNormalization):
      layer.trainable = False

  # define the source image
  in_src = Input(shape=image_shape)
  # suppy the image as input to the generator
  gen_out = g_model(in_src)
  # supply the input image and generated image as inputs to the
                                        discriminator
  dis_out = d_model([in_src, gen_out])
  # src image as input, generated image and disc. output as outputs
  model = Model(in_src, [dis_out, gen_out])
  # compile model
  opt = keras.optimizers.Adam(learning_rate=lr_generator,beta_1=
                                        beta1_generator)

    #Total loss is the weighted sum of adversarial loss (BCE) and L1 loss
                                          (MAE)
    #Authors suggested weighting BCE vs L1 as 1:100.
  model.compile(loss=['binary_crossentropy', jacard_coef_loss],optimizer=
                                        opt, loss_weights=[1,100], metrics=[
                                        jacard_coef])
  return model
```

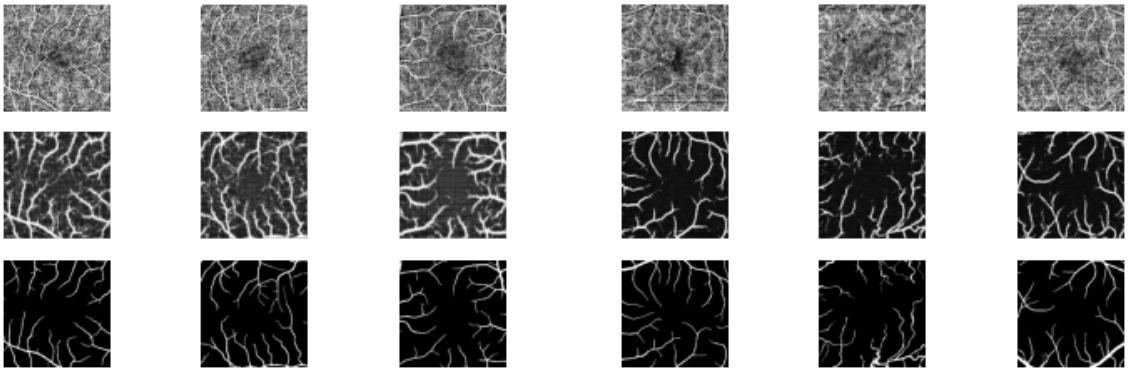
Code Listing 6.8: Sample couple generation implementation code

```
def generate_real_samples(dataset, n_samples, patch_shape):
  # unpack dataset
  trainA, trainB = dataset
  # choose random instance

  np.random.seed(1305)
  ix = np.random.randint(0, trainA.shape[0], n_samples)
  # retrieve selected images
  X1, X2 = trainA[ix], trainB[ix]
  # generate 'real' class labels (1)
  y = ones((n_samples, patch_shape, patch_shape, 1))
  return [X1, X2], y

def generate_fake_samples(g_model, samples, patch_shape):
  # generate fake instance
  X = g_model.predict(samples)
  # create 'fake' class labels (0)
  y = zeros((len(X), patch_shape, patch_shape, 1))
  return X, y
```

6.6.3 Results

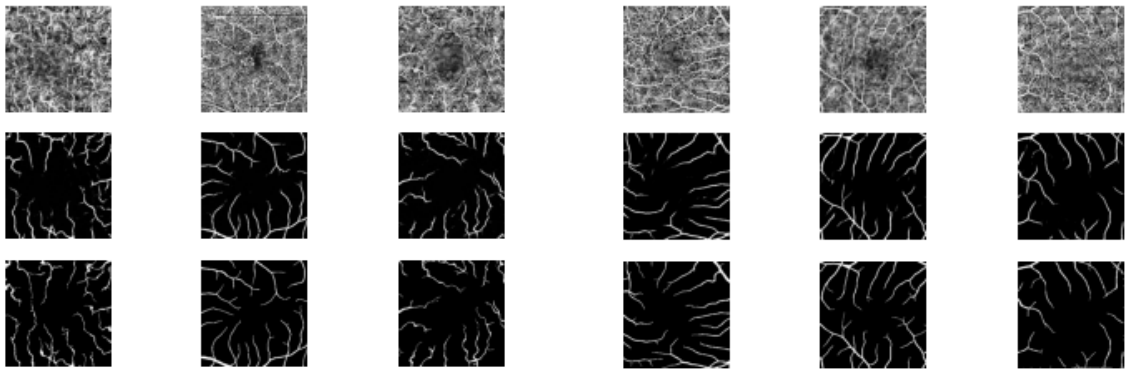
As in the previous models, a tuning of the hyperparameters has been evaluated in order to assess the performance of the Pix2Pix GAN. In particular, the tuning of the learning rate, batch size or number of epochs, is made difficult by the main characteristics of the adversarial loss, which involves a continuous oscillation between discriminator and generator improvements. Therefore, this trend makes it impossible to exclusively rely on the study of training and validation losses. For this reason, it has been decided to always carefully observe the segmentation trend in the training images and to have a look at the results on the test set, epoch by epoch, in order to avoid the phenomenon of overfitting.

Figure 6.36 provides an example of the various steps that the network performs on the training images.



(a) Epochs = 2

(b) Epochs = 4



(c) Epochs = 8

(d) Epochs = 10

Figure 6.36: Graphical evidence of training steps at different epochs. OCTA augmented dataset, batch size = 64, LR = 2^{-4}

It is noticeable that, after the first two epochs, the segmentation is still very approximate and full of noise, whereas it clearly improves at the fourth epoch, and then after 10 epochs a fairly accurate segmentation is reached, leading the model to a Mean IoU of 81.74% and a Dice score of 79.71% on the test set. This type of observation and analysis

of the results has been carried out for each of the variations proposed below.

Original Dataset

As for the choice of parameters, it has been decided to start with the ones suggested by the reference paper [35], i.e., batch size equal to 1 and LR equal to 2^{-4} for the generator. The graph in Figure 6.37 represents the trend of the three reference metrics as the epochs increase.

It can be seen that the model is able to perform a segmentation from the very first epochs. There is then a settlement of performance around the tenth epoch and, after that, the model seems to have reached a stable plateau.

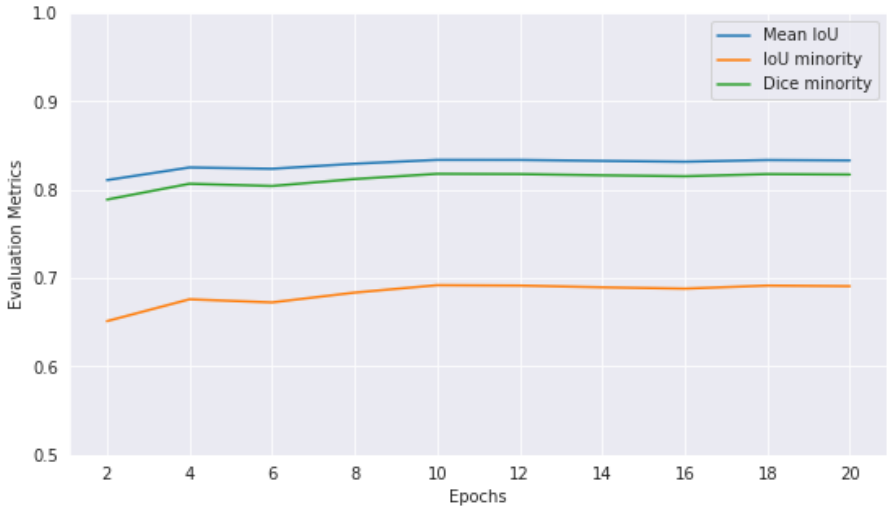


Figure 6.37: Mean IoU, Minority IoU and Dice score of the test set. Original dataset with **batch size = 1**, LR = 2^{-4}, epochs = 20

From the Table 6.7 below, it is evident that as the epochs increase, there is no substantial improvement.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
20	0.0002	1	0.8355	0.6953	0.8203
40	0.0002	1	0.8359	0.6961	0.8209
60	0.0002	1	0.8352	0.6946	0.8198
80	0.0002	1	0.8374	0.6989	0.8228
100	0.0002	1	0.8385	0.7012	0.8243

Table 6.7: Metrics over the test set. Original dataset with **batch size = 1**, LR = 2^{-4}, epochs = 100

The smaller the batch size, the sooner the peak performance is reached. For this

reason, as a second step, it has been chosen to increase the batch size value, while maintaining the same learning rate.

In the Table 6.8 below, it is possible to observe the performance recap with batch size equal to 8 and a LR of 2^{-4} as indicated by the paper. Here again, the model seems to perform quite well immediately and the increase of performance, with increasing epochs, is not very significant.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
20	0.0002	8	0.8179	0.6626	0.7971
40	0.0002	8	0.8144	0.6555	0.7919
60	0.0002	8	0.8181	0.6632	0.7975
80	0.0002	8	0.8137	0.6545	0.7912
100	0.0002	8	0.8172	0.6613	0.7962

Table 6.8: Metrics over the test set. Original dataset with **batch size = 8**, LR = 2^{-4}, epochs = 100

It has been therefore increased the batch size to 64, observing indeed that, compared to previous batches, the model appears to be slower and the peak of performance is reached after many more epochs (Figure 6.38). However, as evident in the Table 6.9, the metrics stand at lower values.

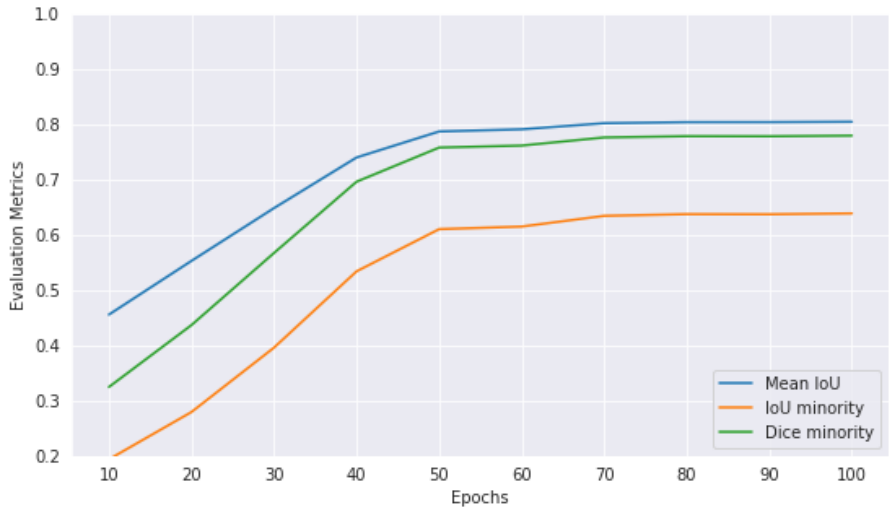


Figure 6.38: Mean IoU, Minority IoU and Dice score of the test set. Original dataset with **batch size = 64**, LR = 2^{-4}, epochs = 100

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
20	0.0002	64	0.5529	0.2793	0.4367
40	0.0002	64	0.7401	0.5340	0.6962
60	0.0002	64	0.7911	0.6151	0.7617
80	0.0002	64	0.8040	0.6375	0.7786
100	0.0002	64	0.8047	0.6389	0.7797

Table 6.9: Metrics over the test set. Original dataset with **batch size = 64**, LR = 2^{-4}, epochs = 100

Thus, as expected, it is noticeable a relationship between the chosen batch size and the speed of convergence of the model. Indeed, it has been seen that a very small batch size causes the model to be immediately stuck in the learning process, whereas a high batch size can allow the model to learn and improve for longer, but always settling on lower metrics. It has been therefore decided to investigate the potential of a high batch model, but with a larger learning rate, in order to speed up convergence and reduce the problem of getting stuck without improvement, which is more common when using a small step of the learning rate.

Graph 6.39 and Table 6.10 show the performance for the model with batch size = 64 and learning rate of the generator model equal to 1^{-2}.

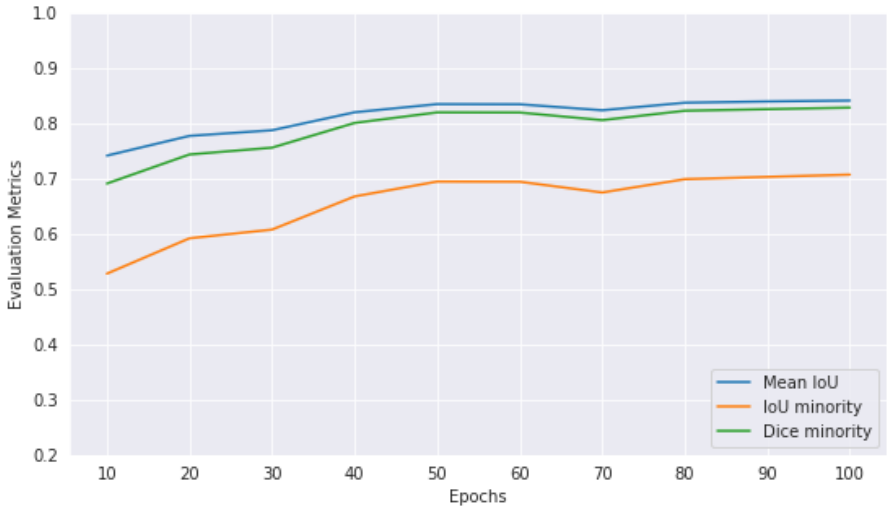


Figure 6.39: Mean IoU, Minority IoU and Dice score of the test set. Original dataset with batch size = 64, **LR** = 1^{-2}, epochs = 100

This results in a model that is able to improve, epoch after epoch, without immediately reaching a plateau phase, and settling on higher values than those seen with the previous hyperparameters.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
20	0.01	64	0.7773	0.5920	0.7437
40	0.01	64	0.8201	0.6678	0.8008
60	0.01	64	0.8346	0.6944	0.8196
80	0.01	64	0.8374	0.6990	0.8229
100	0.01	64	0.8414	0.7073	0.8285

Table 6.10: Metrics over the test set. Original database with batch size = 64, **LR** = 1^{-2}, epochs = 100

Finally, for the sake of completeness, it has also been decided to try a batch size 8, still with learning rate 1^{-2}. These parameters performed well in the use of the U-Net. In fact, also here, good results have been obtained, as observable in Table 6.11, reaching values of 84.7% of average IoU, 71.7% of minority class IoU and 83.5% of Dice score.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
20	0.01	8	0.8519	0.7256	0.8410
40	0.01	8	0.8494	0.7216	0.8383
60	0.01	8	0.8476	0.7183	0.8361
80	0.01	8	0.8476	0.7179	0.8358
100	0.01	8	0.8471	0.7185	0.8351

Table 6.11: Metrics over the test set. Original dataset with batch size = 8, **LR** = 1^{-2}, epochs = 100

Augmented Dataset

The same type of investigation has been chosen also for the hyperparameters tuning of the model trained with the augmented dataset (again with the same transformations applied as in the previous models).

However, due to the limited computational resources, it has been decided not to use the same number of epochs and to reduce them from 100 to 10. Only one training has been done on 40 epochs, with the hyperparameters suggested by the paper (batch size = 1, LR = 2^{-4}), in order to make a more direct comparison with the original dataset. The result is illustrated by the Table 6.12 and it took approximately 124 minutes. For this reason, for the subsequent experiments, it has been decided to keep the epochs to a maximum of 10.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
2	0.0002	1	0.8486	0.7194	0.8368
4	0.0002	1	0.8526	0.7268	0.8418
6	0.0002	1	0.8564	0.7339	0.8466
8	0.0002	1	0.8561	0.7334	0.8462
10	0.0002	1	0.8538	0.7289	0.8432
40	0.0002	1	0.8550	0.7313	0.8448

Table 6.12: Metrics over the test set. Augmented dataset with **batch size = 1**, LR = 2^{-4}, epochs = 40

Similar analyses to those above and with comparable results are shown below. In particular, Table 6.13 refers to a batch size of 8 and Table 6.14 to a batch size of 64.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
2	0.0002	8	0.8299	0.6850	0.8131
4	0.0002	8	0.8430	0.7101	0.8305
6	0.0002	8	0.8465	0.7163	0.8347
8	0.0002	8	0.8471	0.7168	0.8351
10	0.0002	8	0.8451	0.7128	0.8323

Table 6.13: Metrics over the test set. Augmented dataset with **batch size = 8**, LR = 2^{-4}, epochs = 40

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
2	0.0002	64	0.5624	0.2906	0.4503
4	0.0002	64	0.7328	0.5232	0.6870
6	0.0002	64	0.8103	0.6514	0.7889
8	0.0002	64	0.8197	0.6669	0.8002
10	0.0002	64	0.8174	0.6626	0.7971

Table 6.14: Metrics over the test set. Augmented dataset with **batch size = 64**, LR = 2^{-4}, epochs = 40

Finally, also for the augmented dataset, it has been decided to increase the learning rate in order to make the model converge faster. In the Figure 6.40 below, it can be seen the curve comparable with the one of the graph 6.39 of the original dataset. Both have batch size = 64 and learning rate = 1^{-2}.

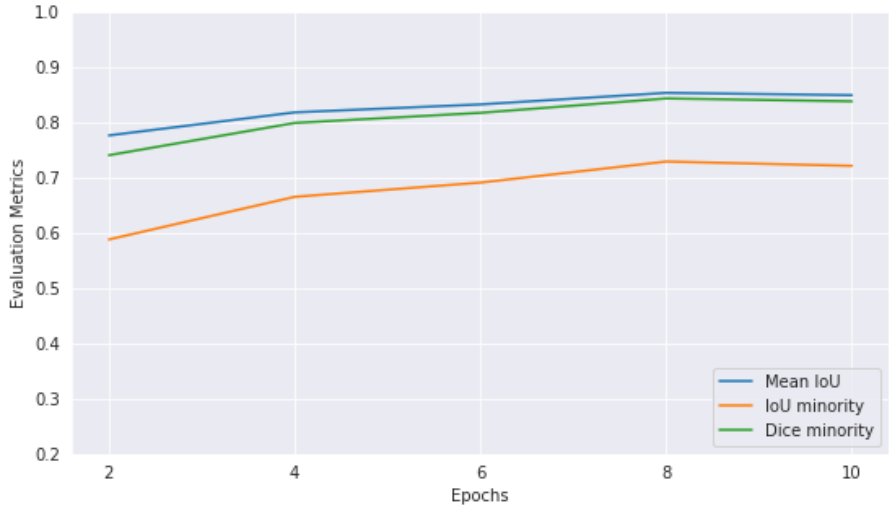


Figure 6.40: Mean IoU, Minority IoU and Dice score of the test set. Augmented dataset with batch size = 64, **LR** = 1^{-2}, epochs = 10

Details of the obtained metrics can be seen in the Table 6.15 below.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
2	0.01	64	0.7764	0.5880	0.7406
4	0.01	64	0.8181	0.6652	0.7990
6	0.01	64	0.8325	0.6910	0.8173
8	0.01	64	0.8534	0.7291	0.8433
10	0.01	64	0.8493	0.7213	0.8381

Table 6.15: Metrics over the test set. Augmented dataset with batch size = 64, **LR** = 1^{-2}, epochs = 10

Finally, the combination of batch size 8 and learning rate 1^{-2} has been tried, reaching the best results achieved so far with this model. The metrics are shown in the Table 6.16.

Epochs	Learning Rate	Batch	Mean IoU	IoU minority	Dice minority
2	0.01	8	0.8558	0.7335	0.8463
4	0.01	8	0.8618	0.7437	0.8530
6	0.01	8	0.8623	0.7447	0.8537
8	0.01	8	0.8618	0.7439	0.8531
10	0.01	8	0.8625	0.7453	0.8541

Table 6.16: Metrics over the test set. Augmented dataset with batch size = 8, **LR** = 1^{-2}, epochs = 10

From the histograms depicted in Figure 6.41, it can be seen how the choice of increasing the learning rate, compared to the hyperparameters initially suggested, led to improvements and also how, with the augmented dataset, the model consistently increased its performance by approximately two to three percentage points, especially with regard to the IoU metric of the minority class.

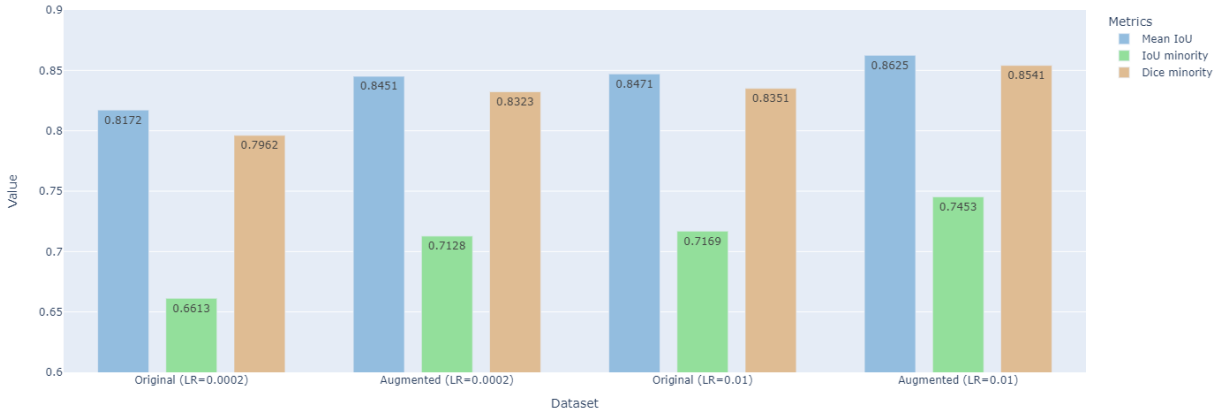


Figure 6.41: Comparison between original and augmented dataset with different learning rates. The first triplet refers to the original dataset, batch size = 8, LR = 2^{-4} and the second one refers to the augmented dataset with the same hyperparameters. The last two triplets have batch size = 8, LR = 1^{-2} and they are respectively for the original dataset and the augmented one

Figure 6.42 shows the graphical result on the best model obtained, i.e., using the augmented dataset, batch size equal to 8 and learning rate equal to 1^{-2}. An average IoU value of 86.5%, Dice score of 85.41% and minority IoU of 74.53% have been achieved. As can also be seen from the picture representing the zoom of a particular vascular part (Figure 6.43), much more precision can be seen in comparison with the basic autoencoder model, although the U-Net remains the model with the highest performance.

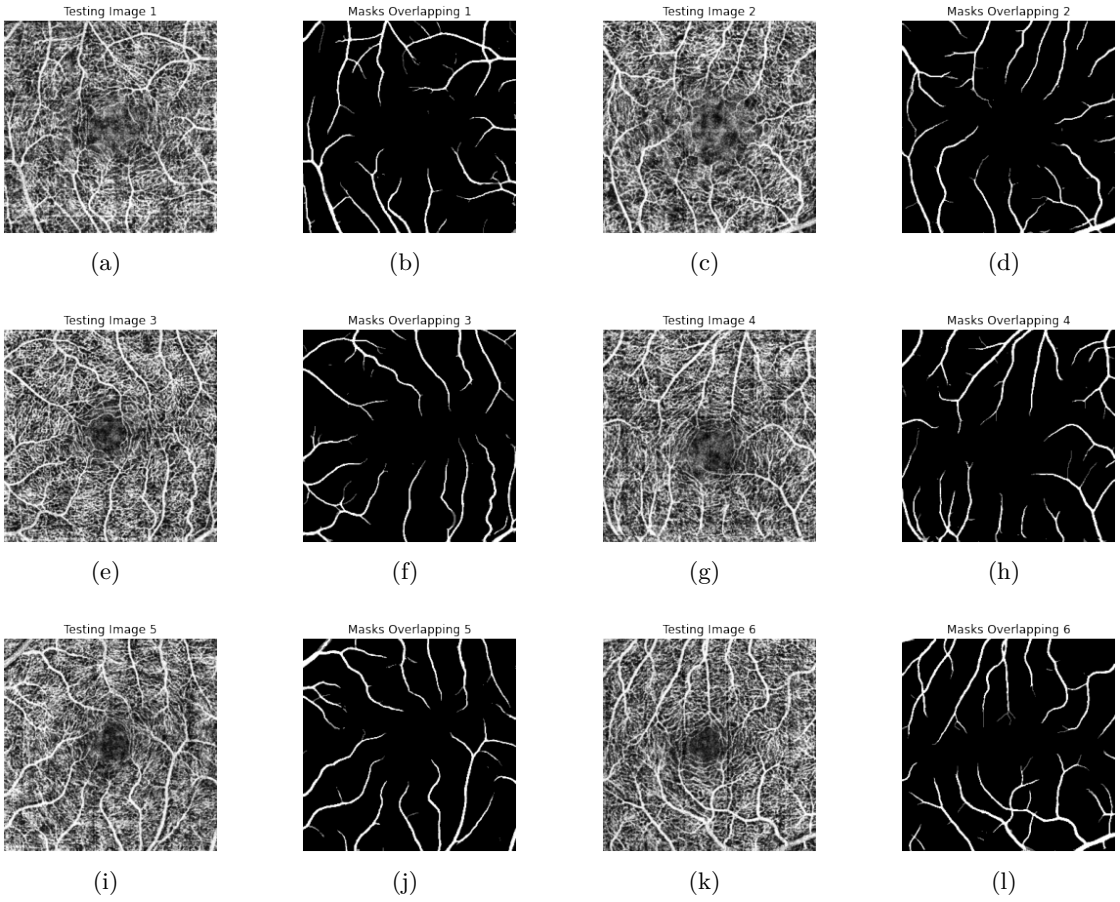


Figure 6.42: Segmentation results of Pix2Pix GAN on 6 test images with batch size = 8, LR = 1^{-2}, epochs = 10 and augmented dataset. Overlapping between ground truth and predicted masks

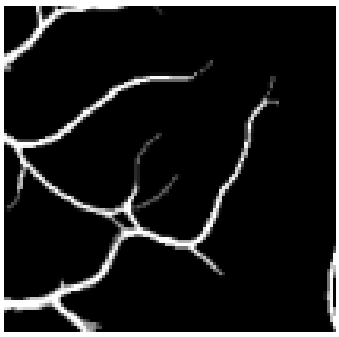


Figure 6.43: Zoom over a segmented test image

OCT vs OCTA

Finally, as already evident from the other models, performance is significantly higher for the OCTA rather than for the OCT dataset. This can be observed in Figure 6.44.

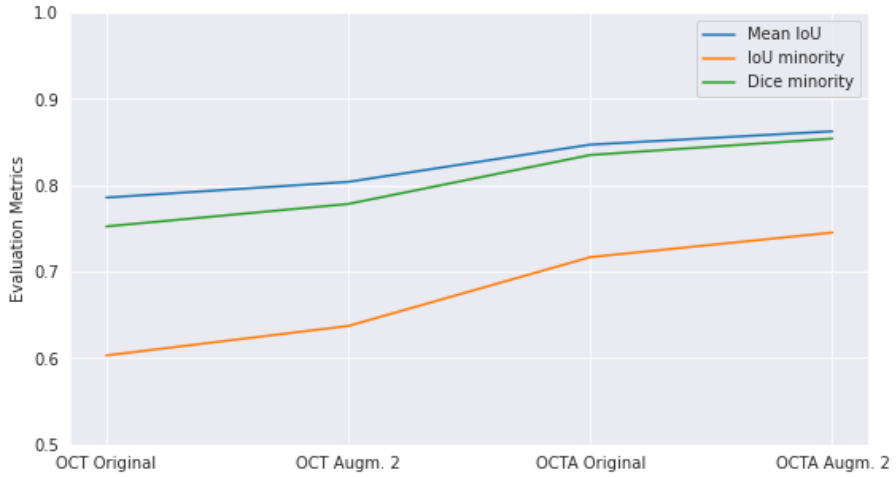


Figure 6.44: OCT vs OCTA performance comparison

Full Projection Map vs Maximum Projection Map

As mentioned in Chapter 3, it has been chosen to mainly work with full projection maps but also a brief comparison with maximum projection 2D images has been done. Below in Figure 6.45 the best results obtained with this last projection technique are shown.

In particular, with the maximum projection maps, it has been possible to obtain an improvement of almost 4.5% points in the Dice score and more than 7.5% points in the IoU of the minority class. The improvement for the vascular part of the image is very significant, leading this model to be perfectly comparable with the performance achieved with U-Net, as can be seen in chapter 7.

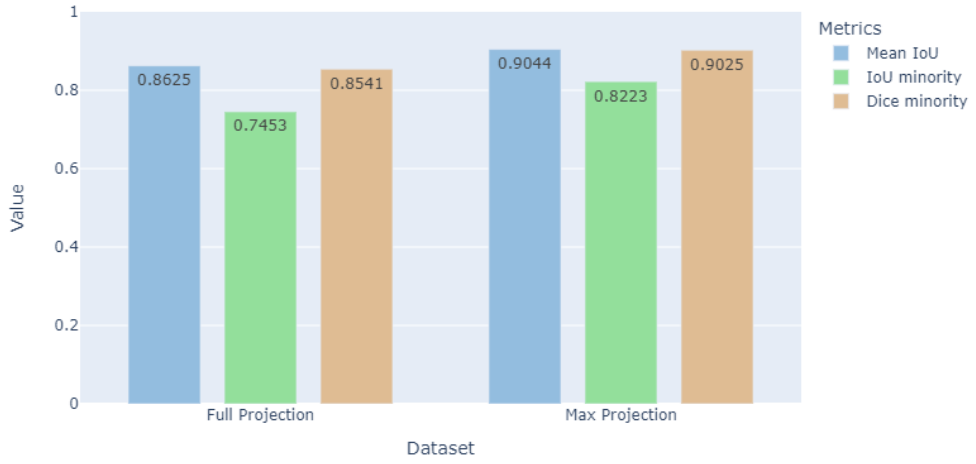


Figure 6.45: Comparison between full projection map and maximum projection map. Augmented dataset. For the full projection: Batch size = 8, Epochs = 10, LR = 1^{-2}. For the maximum projection: Batch size = 8, Epochs = 8, LR = 1^{-2}

Chapter 7

Conclusion

In this thesis work, different deep learning models have been developed in order to perform a vascular segmentation of retina. For each specific step of the analysis, different options have been evaluated through an extensive set of experiments.

Final results

In particular, one of the steps of the analysis aimed to analyse the potential of OCT and OCTA imaging techniques as starting point for performing the segmentation. From the experiments, it has been confirmed that, despite the application of various preprocessing techniques or data augmentation operations, the images derived from the full projection map of OCTA always outperforms with respect to the full projection map of OCT. This behaviour is due to the nature of the different images, OCT being more focused on structural information while OCTA is particularly suitable for extracting information on the retinal vascular network. The increase in performance is significant, with some models reaching values more than 10% higher than using OCT.

Various preprocessing techniques have been applied in order to improve the quality of input data, highlight the relevant information and increase performance. In particular, following a normalisation of the data, two types of equalisation have been chosen, histogram equalisation and CLAHE. The latter has proved to be the most performing one and therefore it has been used to train all the implemented models.

The use of data augmentation techniques, as non-invasive as possible, has allowed to have a larger dataset available on which analyses have been performed. After several experiments, a specific set of transformations has been selected to balance the need for more data and the need of not over-modifying the existing data, with the risk of creating artifacts, which would be particularly dangerous in the case of medical images. By doing so, it has been obtained an average increase in performance of up to 2.97% for the Pix2Pix GAN, a network which is quite more complex than the others, and that therefore needed more data to perform a good training.

The central part of the project involved the use of numerous segmentation models. The autoencoder represented the baseline from which to start, reaching maximum Dice value of 78.73% and IoU value of 64.93%, relative to the vascular class.
U-Net is the network that provided the highest performance, achieving an 87.59% Dice

score and 77.93% IoU, still on the minority class. The mean value of the IoU, averaging vascular and non-vascular classes, reached a value of 88.09%. A slight improvement has been obtained with the Attention U-Net architecture, with a Dice score of 87.80% and an IoU of 78.25% for the class to be segmented.
Finally, the Pix2Pix GAN scored an 85.41% Dice Score and a 74.53% IoU minority, ranking just below the U-Net performance.

It has been chosen to perform all the initial experiments on the full projection maps in order to test the model with the most difficult inputs and to perform a direct comparison between OCT and OCTA images, since the maximum projections maps were available only for the OCTA volumes. However, it has been demonstrated that, using maximum projection maps derived from OCTA, is by far the best starting point for performing retinal segmentation from 2D images, thanks to the sharpness of the blood vessels and the removal of noise from the choroid. Table 7.1 summarises the best performance obtained and a direct comparison between the two types of projection maps, for the main networks.

	Mean IoU	IoU minority	Dice minority
Autoencoder Full	80.97%	64.93%	78.73%
Autoencoder Max	85.14%	72.5%	84.06%
U-Net Full	88.09%	77.93%	87.6%
U-Net Max	90.44%	82.32%	90.31%
Pix2Pix GAN Full	86.25%	74.53%	85.41%
Pix2Pix GAN Max	90.44%	82.23%	90.25%

Table 7.1: Comparison of the best performance with OCTA full projection map and OCTA maximum projection map for each model

Comparison with reference papers

The results obtained have been constantly compared with those of the reference paper connected with the database used, [7] and [8]. Focusing on the first paper, it is possible to make a direct comparison with the results shown in Table 7.2, as they used both the OCTA full projection map and maximum projection map in respectively PRO^1+U-Net and PRO^2+U-Net methods. The U-Net implemented in the current work, allowed to obtain a value of 87.59% Dice and 77.93% IoU scores, against their results of 82.27% Dice and 70.09% IoU. Even with the use of the maximum projection map, it is appreciable an improvement of about 4% points for both metrics, as shown in Table 7.2. The increase in performance could have been achieved thanks to the use of preprocessing or data augmentation techniques that allowed more robust segmentation. Also the use of the Jaccard loss, instead of the cross entropy chosen by the paper, certainly allowed the network to be optimised in order to maximise the chosen metrics.

No.	Issue	Method	DICE (%)	JAC (%)	BACC (%)	PRE (%)	REC (%)
1	FAZ	2D-to-1D	81.74±15.67	71.63±19.38	90.59±9.82	86.21±15.38	81.33±19.62
		3D-to-2D	**88.61±11.61**	**81.23±16.35**	**94.71±6.52**	**89.92±13.54**	**89.56±13.04**
2	RV	OCT	83.02±4.20	71.17±5.68	88.70±3.16	88.37±3.29	78.57±6.41
		OCTA	87.78±2.73	78.33±4.25	91.52±2.34	92.30±2.28	83.85±4.77
		OCT+OCTA	**88.15±2.77**	**78.92±4.34**	**91.71±2.28**	**92.68±2.39**	**84.20±4.62**
	FAZ	OCT	72.25±21.52	60.28±22.60	88.00±12.70	76.20±23.04	76.36±25.45
		OCTA	83.07±19.25	74.78±23.22	90.99±10.54	89.26±18.65	82.16±21.06
		OCT+OCTA	83.92±16.62	75.10±20.08	91.71±9.15	88.80±15.45	83.58±18.33
		OCT+OCTA+D*	**88.61±11.61**	**81.23±16.35**	**94.71±6.52**	**89.92±13.54**	**89.56±13.04**
3	RV	PRO^1+FCN	76.07±4.27	61.56±5.28	85.24±3.31	80.54±3.63	72.43±6.75
		PRO^2+FCN	81.70±2.26	69.12±3.18	88.36±1.92	85.74±2.66	78.18±3.91
		PRO^1+U-Net	82.27±4.21	70.09±5.73	88.34±2.97	87.45±4.22	77.93±5.93
		PRO^2+U-Net	86.92±2.34	76.94±3.60	91.20±1.97	91.01±2.75	83.34±4.04
		IPN	**88.15±2.77**	**78.92±4.34**	**91.71±2.28**	**92.68±2.39**	**84.20±4.62**
	FAZ	Lu et al. [5]	71.73±25.30	60.87±25.82	88.60±14.10	71.84±27.25	77.48±28.09
		Díaz et al. [57]	79.70±21.25	69.88±21.15	94.64±11.39	73.53±21.67	89.62±22.58
		IPN	**88.61±11.61**	**81.23±16.35**	**94.71±6.52**	**89.92±13.54**	**89.56±13.04**
4	RV	IPN-U	78.84±2.27	65.13±3.05	87.96±1.88	79.63±2.14	78.18±3.77
		IPN-UC	87.45±2.93	77.81±4.49	91.39±2.52	91.84±2.32	83.65±5.15
		IPN	**88.15±2.77**	**78.92±4.34**	**91.71±2.28**	**92.68±2.39**	**84.20±4.62**
	FAZ	IPN-U	84.79±11.66	75.13±15.40	93.71±6.43	84.51±14.32	87.61±12.87
		IPN-UC	86.51±13.96	78.42±18.25	93.39±7.88	89.35±14.64	86.92±15.75
		IPN	**88.61±11.61**	**81.23±16.35**	**94.71±6.52**	**89.92±13.54**	**89.56±13.04**

D^* is distance map. PRO^1 is OCTA full-projection map. PRO^2 is OCTA maximum-projection map between ILM layer and OPL.

Table 7.2: Results obtained in paper [7] (table obtained from [7])

Compared to the second paper, which was more recent and had several improvements, the authors achieved results of 90.6% Dice and 82.88% IoU with the use of the maximum projection map, in line with the results obtained in the current project.

RV Segmentation					OCTA_6M (6mm × 6mm)			OCTA_3M (3mm × 3mm)		
Methods		P*	G*	M*	DICE (%)	JAC (%)	BACC (%)	DICE (%)	JAC (%)	BACC (%)
2D to 2D	Fast-FCN [44]	-	-	-	86.59 ± 3.22	76.49 ± 4.70	92.00 ± 1.88	88.45 ± 2.42	79.38 ± 3.75	93.62 ± 1.28
	Deeplab V3+ [45]	-	-	-	87.22 ± 3.14	77.47 ± 4.63	92.81 ± 1.69	88.02 ± 2.48	78.69 ± 3.82	92.71 ± 1.51
	Attention U-Net [46]	-	-	-	88.67 ± 2.76	79.75 ± 4.23	93.36 ± 1.62	90.89 ± 2.21	83.38 ± 3.56	94.71 ± 1.62
	U-Net [36]	-	-	-	88.28 ± 2.59	79.11 ± 3.97	92.98 ± 1.62	90.60 ± 2.16	82.88 ± 3.47	94.91 ± 1.38
	U-Net [36]	-	-	✓	88.44 ± 2.58	79.37 ± 3.99	92.94 ± 1.59	90.19 ± 2.21	82.20 ± 3.54	94.23 ± 1.46
	U-Net 2+ [47]	-	-	-	88.62 ± 2.61	79.66 ± 4.02	93.30 ± 1.53	90.84 ± 2.20	83.29 ± 3.55	94.96 ± 1.45
	U-Net 2+ [47]	-	-	✓	88.15 ± 2.93	78.92 ± 4.40	92.86 ± 1.74	90.31 ± 2.23	82.40 ± 3.58	94.39 ± 1.48
	U-Net 3+ [48]	-	-	-	88.49 ± 2.80	79.47 ± 4.26	93.07 ± 1.78	90.92 ± 2.08	83.41 ± 3.36	94.57 ± 1.43
	U-Net 3+ [48]	-	-	✓	**88.78 ± 2.63**	**79.92 ± 4.07**	**93.37 ± 1.57**	**90.93 ± 2.05**	**83.43 ± 3.35**	**95.12 ± 1.40**
3D to 2D	IPN [23]	-	-	-	88.64 ± 3.21	79.73 ± 4.92	93.07 ± 2.42	90.62 ± 5.96	83.25 ± 7.78	93.87 ± 4.19
	IPN [23]	-	-	✓	88.30 ± 3.36	79.21 ± 5.11	92.79 ± 2.55	91.10 ± 3.49	83.83 ± 5.29	94.63 ± 2.79
	IPN V2	✓	-	-	89.08 ± 2.73	80.41 ± 4.29	**93.52 ± 2.13**	92.46 ± 3.93	86.19 ± 5.83	**95.34 ± 3.16**
	IPN V2	✓	-	✓	88.78 ± 3.21	79.96 ± 4.89	93.40 ± 2.16	92.32 ± 3.59	85.91 ± 5.44	95.05 ± 3.03
	IPN V2+	✓	✓	-	**89.41 ± 2.74**	**80.95 ± 4.32**	93.46 ± 2.12	**92.74 ± 3.95**	**86.67 ± 5.88**	95.22 ± 3.12
	IPN V2+	✓	✓	✓	89.18 ± 3.15	80.61 ± 4.84	93.37 ± 2.16	92.63 ± 3.43	86.44 ± 5.25	95.20 ± 2.90

Table 7.3: Results obtained in paper [8] (table obtained from [8])

7.1 Future works

All experimental phases have been carried out with 12 GB of RAM and Colab GPU, which is often limited in usage time and therefore cannot be used indefinitely. Thus, it can be assumed that, by having more computational resources, results could have been improved. For instance, parameters like $20k$ iterations could have been tried, as suggested by the reference papers, however requiring more than 10 hours of running time, which is impractical for this specific case.

A first improvement might regard the training of the different models with the combination of OCT and OCTA, as proposed by [8], resulting in an augmented dataset that could be particularly useful for more complex networks such as Pix2Pix GAN, which indeed provided the highest performance gains when using more data.

Another future work may involve the use of images available in the OCTA_6M dataset, which are characterised by a larger dimension, that would allow to explore the hypothesis of implementing data augmentation by means of patches, bearing in mind, however, the problems related to cropping images containing a retinal structure and the possibility of having non-significant crops. The use of smart patch sampling techniques could be evaluated to limit the choice of crops to the object that have to be segmented [19].

In addition, the possibility of segmenting the FAZ (foveal avascular zone) could be included in the analysis to obtain a more comprehensive segmentation and use this valuable information for diagnostic purposes.

Bibliography

[1] Priyanka Malhotra, Sheifali Gupta, Deepika Koundal, Atef Zaguia, Wegayehu Enbeyle, *Deep Neural Networks for Medical Image Segmentation*

[2] Amir H Kashani, Chieh-Li Chen, Jin K Gahm, Fang Zheng, Grace M Richter, Philip J Rosenfeld, Yonggang Shi, and Ruikang K Wang, *Optical Coherence Tomography Angiography: A Comprehensive Review of Current Methods and Clinical Applications*

[3] Richard F.Spaide, James G.Fujimoto, Nadia K.Waheed, Srinivas R.Sadda, Giovanni Staurenghi, *Optical coherence tomography angiography*

[4] Maria Cristina Savastano, Marco Rispoli, Bruno Lumbroso, Luca Di Antonio, Leonardo Mastropasqua, Gianni Virgili, Alfonso Savastano, Daniela Bacherini and Stanislao Rizzo, *Fluorescein angiography versus optical coherence tomography angiography: FA vs OCTA Italian Study*

[5] Natalie Huang, W. Andrew Lee , Sean Rivera, Sandra Montezuma, *Ruptured Retinal Arterial Macroaneurysm Secondary to Toxoplasmic Kyrieleis Arteriolitis: A Case Report*

[6] Mehreen Adhi and Jay S. Duker, *Optical coherence tomography – current and future applications*

[7] Mingchao Li, Yerui Chen, Zexuan Ji, Keren Xie, Songtao Yuan, Qiang Chen, and Shuo Li, *Image Projection Network: 3D to 2D Image Segmentation in OCTA Images*

[8] Mingchao Li, Yuhan Zhang, Zexuan Ji, Keren Xie, Songtao Yuan, Qinghuai Liu and Qiang Chen, *IPN-V2 and OCTA-500: Methodology and Dataset for Retinal Image Segmentation*

[9] Shu Zheng, Yanru Bai, Zihao Xu, Pengfei Liu and Guangjian Ni, *Optical Coherence Tomography for Three-Dimensional Imaging in the Biomedical Field: A Review*

[10] Martin F. Kraus, Benjamin Potsaid, Markus A Mayer, Ruediger Bock, *Motion correction in optical coherence tomography volumes on a per A-scan basis using orthogonal scan patterns*

[11] Chenyang Xu, Dzung L. Pham, Jerry L. Prince, *Image Segmentation Using Deformable Models*

[12] Cemil Kirbas, Francis Quek, *A Review of Vessel Extraction Techniques and Algorithms*

[13] Kristen M. Meiburger, Massimo Salvi, Giulia Rotunno, Wolfgang Drexle, Mengyang Liu, *Automatic Segmentation and Classification Methods Using Optical Coherence Tomography Angiography (OCTA): A Review and Handbook*

[14] Foued Derraz, Mohamed Beladgham and M'hamed Khelif, *Application of Active Contour Models in Medical Image Segmentation*

[15] Mr.Salem Saleh Al-amri, Dr.N.V.Kalyankar, Dr.S.D.Khamitkar, *Linear and Non-linear Contrast Enhancement Image*

[16] Acharya and Ray, *Image Processing: Principles and Applications*

[17] Alexander Buslaev, Alex Parinov, Eugene Khvedchenya, Vladimir I. Iglovikov, Alexandr A. Kalinin, *Albumentations: fast and flexible image augmentations*

[18] Daniel Bar-David, Laura Bar-David, Yinon Shapira, Rina Leibu, Dalia Dori, Ronit Schneor, Anath Fischer, Shiri Soudry, *Elastic deformation of optical coherence tomography images of diabetic macular edema for deep-learning models training*

[19] Massimo Salvi, Filippo Molinari, U Rajendra Acharya, Luca Molinaro, Kristen M Meiburger, *Impact of stain normalization and patch selection on the performance of convolutional neural networks in histological breast and prostate cancer classification*

[20] Azhar Imran, Jianquiang Li, Yan Pei, Ji-Jiang Yang, adn Qing Wan, *Comparative Analysis of Vessel Segmentation Techniques in Retinal Images*

[21] Floris van Beers, *Using Intersection over Union loss to improve Binary Image Segmentation*

[22] Dor Bank, Noam Koenigstein, Raja Giryes, *Autoencoders*

[23] Sergio Saponara, Abdussalam Elhanashi, and Qinghe Zheng, *Recreating Fingerprint Images by Convolutional Neural Network Autoencoder Architecture*

[24] Krishna Dev, Zubair Ashraf, Pranab K. Muhuri, Sandeep Kumar, *Deep autoencoder based domain adaptation for transfer learning*

[25] Abien Fred M. Agarap, *Deep Learning using Rectified Linear Units (ReLU)*

[26] Hossein Gholamalinezhad , Hossein Khosravi, *Pooling Methods in Deep Neural Networks, a Review*

[27] Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization*

[28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation.*

[29] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, Daniel Rueckert, *Attention U-Net: Learning Where to Look for the Pancreas*

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*

[31] Md Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M. Taha, and Vijayan K. Asari, *Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation*

[32] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola, *Dive into Deep Learning*

[33] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative Adversarial Networks*

[34] Mehdi Mirza, Simon Osindero, *Conditional Generative Adversarial Nets*

[35] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*

[36] Uğur Demir, Gozde Unal, *Patch-Based Image Inpainting with Generative Adversarial Networks*

Additional images sources

[37] https://www.mdpi.com/2075-4418/11/8/1393
[38] https://www.retinala.com/fluorescein-angiography.html
[39] https://www.recendt.at/en/OCT.html
[40] https://moorfield-optometry-specialist-eye-care.business.site/
[41] https://www.youtube.com/watch?v=_8s2JgAgTmA&ab_channel=AdelAbdelshafik
[42] https://ieee-dataport.org/open-access/octa-500
[43] https://www.semanticscholar.org/paper/Content-Based-Image-Retrieval-Using-Machine-And-Kaur-Singh/8dcae44b8153df6b551950204e2cc4ceaea816bc
[44] https://link.springer.com/article/10.1007/s11042-021-11287-z
[45] https://medium.com/@saibharath897/generative-adversarial-networks-gans-560c5c988128
[46] https://medium.com/@ma.bagheri/a-tutorial-on-conditional-generative-adversarial-nets-keras-implementation-694dcafa6282
[47] https://github.com/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb

Acknowledgements

A special thanks goes to our thesis supervisor, Prof. Kristen Mariko Meiburger, for having given us the opportunity to study in deep topics of our interest and for her being always available to us, for her kindness and willingness in giving us ideas and useful suggestions