Automatic segmentation of mononucleated cells in peripheral blood images for features comparison between cells

Universitat Oberta de Catalunya

Miguel Ángel Calafat Torrens

Master in Bioinformatics and Biostatistics

Advisor: Edwin Santiago Alférez Baquero

UNIVERSITAT DE BARCELONA

15th of January of 2023

This work is subject to an Attribution license

https://creativecommons.org/licenses/by/3.0/

Title:	*Automatic segmentation of mononucleated cells in peripheral blood images for features comparison between cells*
Author:	*Miguel Ángel Calafat Torrens*
Tutor:	*Edwin Santiago Alférez Baquero*
SRP:	*Edwin Santiago Alférez Baquero*
Date of delivery:	*31th of January 2023*
***Studies*:**	*Master in Bioinformatics and Biostatistics*
Area:	*Machine Learning*
Language:	*English*
Keywords:	*Deep learning, biomedical images, cells segmentation, leukemia, lymphoma*

Abstract

In recent years, with the rise of artificial intelligence and its strong entry into the world of biomedical imaging, the need to exchange knowledge not only from humans to machines, but also from machines to humans has become more and more relevant.

We want a machine to be capable of performing tasks related to data processing in health sciences, and that these tasks become increasingly complex, even reaching the point of equaling or exceeding the capabilities of the best experts in the field. However, the flow of information does not always have to go in the same direction (from human expert knowledge to machines) but it can also circulate in the opposite direction, increasingly helping experts to understand non-intuitive patterns (or that are not immediately apparent), and to describe these patterns based on reasonably familiar metrics.

One of the most criticized aspects of deep neural networks is that the knowledge created is not easily interpretable, but their capacity is far from any doubt.

Thus, this work develops an automatic process of semantic segmentation of mononuclear cells present in medical images of peripheral blood taken under a microscope. Once the segmentation is done, a series of image features corresponding to the segmented areas are extracted. This features are displayed in an easy-to-use app, so that the pathologist themselves can find relationships and patterns with morphological parameters, and thus understand more clearly aspects of the information with which they work.

Automatic segmentation of mononucleated cells in peripheral blood images for features comparison between cells

Contents

1 Summary

In recent years, with the rise of artificial intelligence and its strong entry into the world of biomedical imaging, the need to exchange knowledge not only from humans to machines, but also from machines to humans has become more and more relevant.

We want a machine to be capable of performing tasks related to data processing in health sciences, and that these tasks become increasingly complex, even reaching the point of equaling or exceeding the capabilities of the best experts in the field. However, the flow of information does not always have to go in the same direction (from human expert knowledge to machines) but it can also circulate in the opposite direction, increasingly helping experts to understand non-intuitive patterns (or that are not immediately apparent), and to describe these patterns based on reasonably familiar metrics.

One of the most criticized aspects of deep neural networks is that the knowledge created is not easily interpretable. There have been and continue to be many attempts to bring deep neural networks closer to the general public in order to make them more understandable (ref: Tsukasa Ueno 2018); but deep down there lies an indisputable reality, and that is that machines store and handle information in a different way than humans do. What for a human is a simple diffuse shadow on an image, for a machine it is a precise representation of values. If necessary, the machine can easily distinguish between hundreds or thousands of different shades, while human acuity does not go that far.

This capacity, together with the speed of learning, and current analysis techniques, allow machines to extract a series of graphic features that can easily be converted into understandable information for humans. We refer to shape features such as major and minor diameter of the nucleus of a cell, eccentricity of the cytoplasm, ratio between area and perimeter of the nucleus, etc. This information can be important so that the pathologist themselves can find relationships and patterns with morphological parameters, and thus understand more clearly aspects of the information with which they work.

For the reasons mentioned above, this work proposes an automatic process of semantic segmentation of mononuclear cells present in medical images of peripheral blood taken under a microscope. This process is done with deep neural networks that follow an encoder-decoder scheme. Once all the pixels in the image have been identified and it is known to which area they belong (nucleus, cytoplasm or others), a series of image features corresponding to these areas are extracted (for example, the number of pixels in a class is proportional to the area of that zone).

Finally, all these features are displayed together in graphs and compared with the population features (those extracted from cells from the sample database). The comparison of the results with the averages by cell type allows us, at a first glance, to realize the features that have the greatest difference.

For example, it could be very interesting to know what is the area of the nucleus divided by the area of the entire cell (nucleus plus cytoplasm). Data like this, with a value very close to unity, would be indicating that practically the entire cell is occupied by the nucleus, which can be a good indicator of some ailments.

In contrast, there could be other less intuitive metrics, such as the entropy or kurtosis, which can lead the clinitian to pay attention to patterns where previously there was only general knowledge.

2 Introduction

This work focuses on the semantic segmentation of certain mononuclear cells and on the extraction of graphic features from those segmentations. Semantic segmentation is performed from medical images of cells present in peripheral blood. Specifically, images of 4 different types are used:

- Monocytes: Healthy monocyte cells
- Lymphocytes: Healthy lymphocyte cells
- HCL (hairy cell leukemia): Strictly speaking, these cells are also lymphocytes; but they have their own characteristics that appear in the images taken from the microscope, and that make them distinguishable from healthy lymphocytes.
- APL (acute promyelocyte leukemia): In this case, this type of cell, the myelocyte, which comes from the myeloblast, is a blast, which under normal conditions should not be present in the peripheral blood.

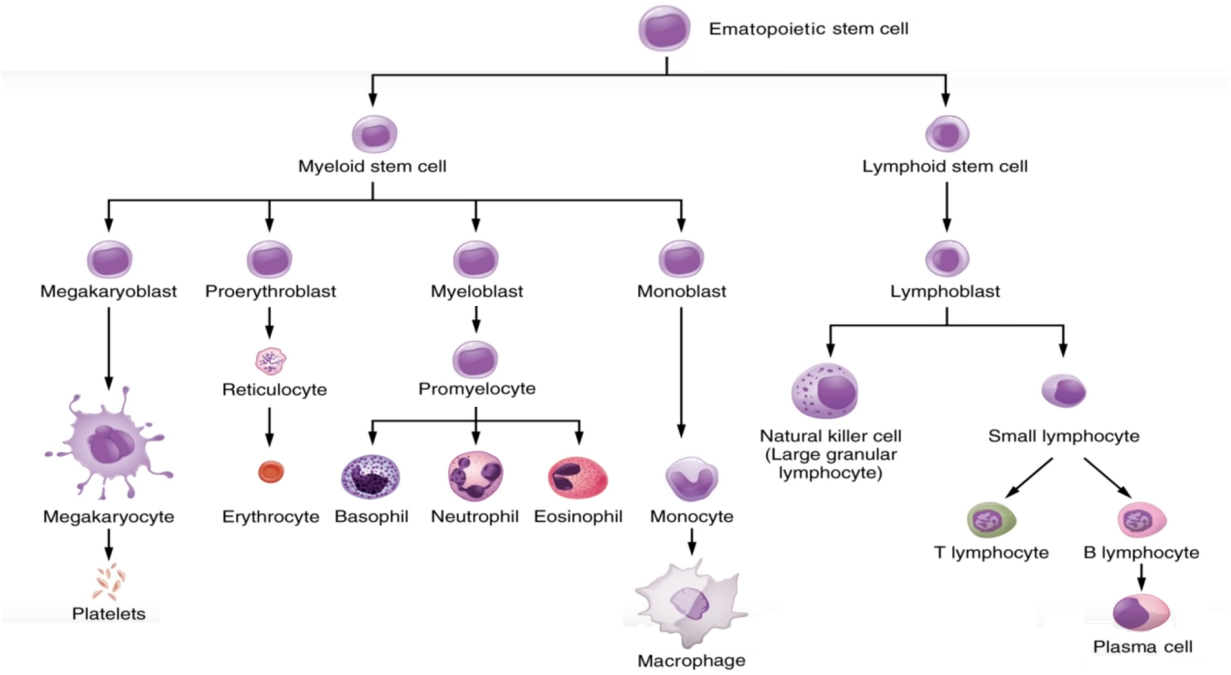

Figure 1: Ematopoietic stem cell. Original image form: ProfessorDaveExplains 2021

Although the main milestone of this work is semantic segmentation, the motivation for it is the possibility of extracting graphic features from these segmentations. The objective of this extraction is the analysis of the features to generate knowledge from deep learning systems to humans.

2.1 Objectives of the thesis

The final objective is to provide a platform that allows the analysis of graphic features of images of cells present in peripheral blood, based on the unequivocal distinction between its different parts. For this, as explained in the previous section, it is necessary to automate a semantic segmentation process that identifies the different areas of interest.

For this reason, it can also be said that one of the main objectives is the training of a neural network model that is capable of carrying out the aforementioned segmentations successfully.

In summary, the objectives cuold be summarized as follows:

- Implement a pipeline comparison of different models to be used for semantic segmentation.
- Evaluate the performance of the different models through appropriate metrics.
- Once the model is selected, implement a pipeline to train and fine tune the model.
- Since the output of the model will be masks (matrices labeling each pixel of the original image), another objective is to program a system to automatically extract graphical features from the couples image – mask.
- Develop a user interface for ease of use.

2.2 Ethical and global behavior

The Ethical and Global Engagement Competence (EGEC) is defined at the Master's level as follows: "*Act in an honest, ethical, sustainable, socially responsible and respectful manner with respect to human rights and diversity, both in academic practice and in the professional, and design solutions to improve these practices.*"

In this sense, three major dimensions are addressed:

- Sustainability
- Ethical behavior and social responsibility (SR)
- Diversity (gender, among others) and human rights

In terms of sustainability, this work cannot contribute much, since as it is software its impact could be said to be nil, both in positive and negative terms.

In the section on ethical behavior and social responsibility, obviously, this work does have a significant impact. It should not be forgotten that this work tries to transfer knowledge from machines to people, regardless of any consideration of gender or any other type, so that they can help people with life-threatening illnesses (leukaemias).

In this sense, we could indeed speak of a clear social orientation, which could be encompassed both in the section on social responsibility, and in that of diversity and human rights.

Specifically in this last section, that of diversity and human rights, it is worth saying something about privacy. The state of health of a person is personal data, and in this sense it should be noted that during the project no data that could identify people has been accessed at any time. Furthermore, the trained neural network model stores only the numerical information to be able to calculate the output from the input pixel values, so it is completely impossible to deduce or infer any personal data from it.

Finally, it should be noted that the work carried out, both in the design stage and in the final result, will be suitable for use by anyone belonging to any group (obviously people interested in the identification of mononuclear cells in peripheral blood samples); but that no bias is produced or reinforced in any way in favor of or against any group that can be identified.

2.3 Methodology

The strategy to follow begins by obtaining a dataset, which can be small, to use in the development of the first pipeline. This pipeline is the one used to select the best model for this kind of work, based on appropriate metrics.

Once a small test dataset is available, the pipeline will be developed and several models will be tested with the aim of leaving them tuned for this specific use.

Once the previous steps have been achieved, a different pipeline will be developed in order to train the best model and adjust it. At this point, it would be appropriate to already have the final dataset.

At the same time, the various options for extracting graphic parameters will be studied. Finally, with the previous points achieved, a friendly graphical environment as a user interface will be developed.

2.4 Work planning

The work planning has been carried out based on a capacity of 5 days per week, but considering only 5 hours per day, which adds up to 25 hours per week. This is consistent with the number of credits for this subject, and the rest of the subjects that make up the master's studies during the current semester.

2.4.1 Tasks

The tables 1, 2, 3 and 4 show the main tasks and sub-tasks that have been identified and its estimated duration in days.

Table 1: Definition planning

Items	Days
Thesis proposal and work planning	7
Literature review and analysis	6
Follow-up document delivery (PEC1)	1
TOTAL	14

Table 2: Development stage 1

Items	Days
Colection and adaptation of models	15
Development of selection and training pipelines	9
Follow-up document delivery (PEC2)	1
TOTAL	25

Table 3: Development stage 2

Items	Days
Tests and evaluation of models	15
Feature extraction and app	9
Follow-up document delivery (PEC3)	1
TOTAL	25

Table 4: Documentation and presentation

Items	Days
Documentation	14
Memory and presentation closing proposal	0
Public defense	1
TOTAL	15

2.4.2 Calendar

The official start of the course is on september 28th. The project must be publicly defended one day between January 23rd and February 3rd of 2023. The exact date will be defined in December 2022 and will depend on the availability of the university.

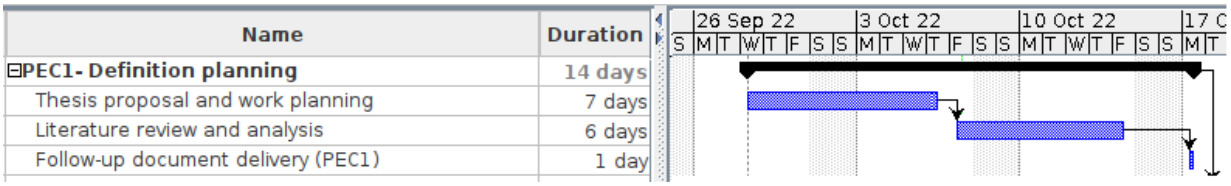

Figure 2: Gant chart of first deliverable of the project

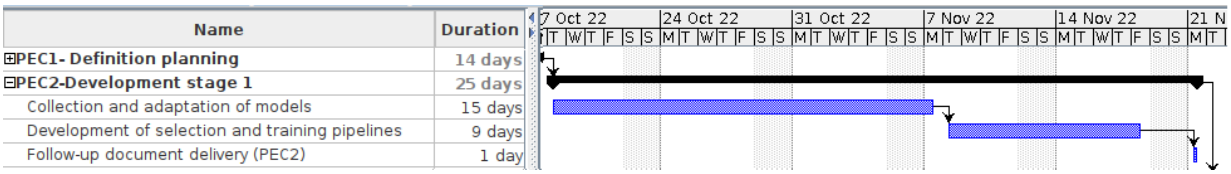

Figure 3: Gant chart of second deliverable of the project

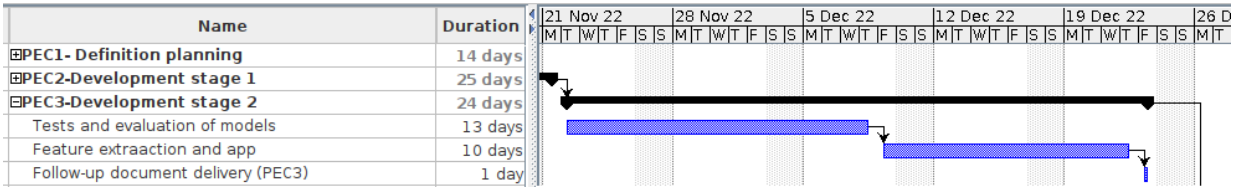

Figure 4: Gant chart of third deliverable of the project

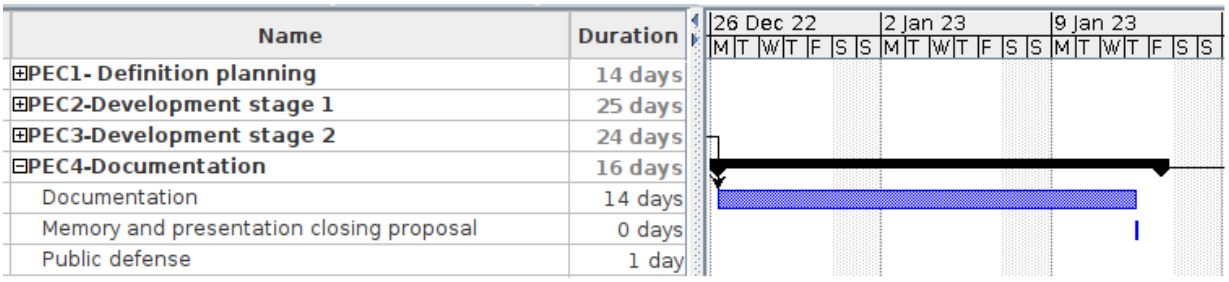

Figure 5: Gant chart of final deliverable of the project

2.4.3 Risk analysis

The risk that any project of this type must assume is the possibility that the time reserved for each task wasn't enough. Keep in mind that this project requires training that can be computationally very expensive. To try to mitigate this risk, training will be carried out using GPUs whenever possible.

Another obvious risk is the one related to obtaining a suitable dataset for the task to be developed. Taking into account that part of this milestone depends on third parties, this could cause delays, or the training had to be done with a less suitable dataset.

On the other hand, it is highly likely that most of the models to be tested are developed on some platform in python; however, nothing ensures that they can all be used on the same platform (there could easily be models in TensorFlow that are not in Pytorch, or that are not sufficiently tested). This could lead to spending more time than expected on adaptations on the pipeline, or to testing different models on different platforms. Both options are totally valid; but they imply investing more time.

2.4.4 The framework - Pytorch

The framework selected to carry out the work has been Pytorch (Paszke et al. 2019). It is a widely known open source machine learning framework for Python, which has a wealth of documentation, examples, plugins and compatibility with other libraries, and which makes development reasonably affordable.

The training of the models have been developed on a local platform, in which a virtual environment has been installed with all the necessary libraries. At this point it should be noted that some of the training, as will be seen in later chapters, can be computationally expensive. This means that they can take many hours, even with the use of a mid-range GPU.

The initial intention was to leave all the scripts ready to be executed on Google Colab, but this option has finally been discarded, since the free version does not allow the use of the GPU for more than 12 hours at a time. This limitation makes it impossible to execute the training loop on all the models tested with a Kfold configuration like the one that will be seen in the following sections.

3 Segmentation models - state of the art

The neural network model that implements semantic segmentation is an important decision. There are many models and each year new ones may appear, or new implementations of the old ones with some improvements. However, so far they are all based on a basic configuration known as U-Net, which first appeared in 2015 as a result of the excellent work of Ronneberger, Fischer, and Brox (2015).

The U-Net model has become a de facto standard for semantic segmentation, and a base level for testing other models. In the case at hand, the selection of the model must respond to some specific criteria.

In this work a selection of a certain configuration of most of the current models has been done. All the selected models (except the U²-Net model (Qin et al. 2020)) are found in the "*segmentation_models_pytorch*" (Iakubovskii 2019) library, which, as its name suggests, contains neural network models that were used in semantic segmentation.

Below is a brief description of each of the models evaluated.

U-Net (Ronneberger, Fischer, and Brox 2015)

As said before, the U-Net model has become a base level for testing other models. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. This means that the network has an encoder-decoder configuration (contractive path – expansive path).

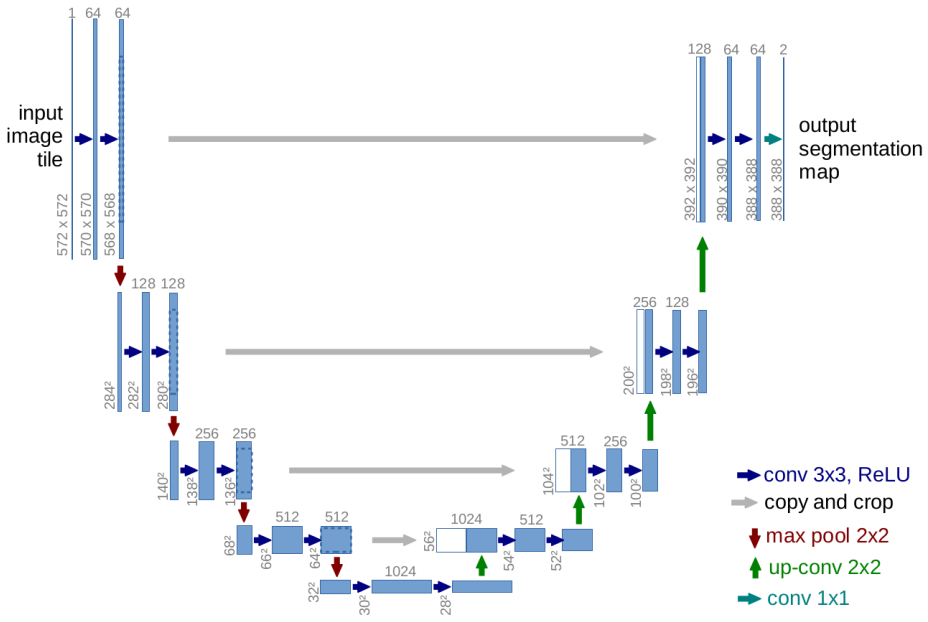

Figure 6: Original U-Net. Source: Ronneberger et al. (2015)

The encoder part is typical in convolutional networks used for classifying. In this case it consists of convolutional layers of size 3×3 activated by ReLU (rectified linear unit), and followed by max pooling layers with stride 2. In each convolution block the number of feature channels is doubled; this is why in Figure 6 we see that it starts with 64, and continues with 128, 256, 512 until it reaches a depth of 1024.

ReLU is a non-linear activation function that responds to the formula:

$$f(x) = max(0, x) \tag{1}$$

Max pool is a common type of layer in neural networks that filters an input matrix such that it is left with the maximum value in each window. Below is an example of such a filtering in each 2×2 window.

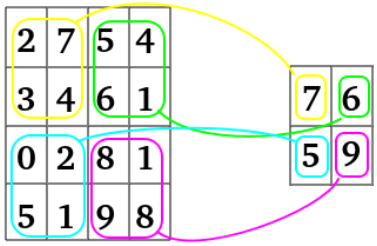

Figure 7: Example of max pooling calculation whith stride 2

The intermediate point between encoder and decoder is called bottleneck. If it were to be classified, the output of the bottleneck would be connected to a fully connected layers classifier.

The decoder part is roughly symmetrical to the encoder part. Every step consists of an upsampling of the feature map followed by a convolution (up-convolution) that halves the number of feature channels, and two 3×3 convolutions, followed by a ReLU activation. Also, this encoder part (expansive path) is supplemented with connections from the decoder that help maintain the overall context. This structure allows to have all the accuracy of the fully convolutional layers without loss of context.

Pyramid Attention Network (PAN) (Li et al. 2019)

Attention (refs: Vaswani et al. (2017) and Luong, Pham, and Manning (2015)) is a mechanism in neural networks that allows focusing on parts of the source instead of treating it as a whole. This allows you to eliminate recursion and, depending on how it's implemented, can complement or even eliminate the need for convolutions.

Spatial pyramid pooling (He et al. 2015) is a strategy that eliminates the need for the input images of convolutional networks to have fixed dimensions, thus generating a fixed-length representation regardless of image size/scale.

The Pyramid Attention Network (PAN) combines attention mechanism and spatial pyramid to extract precise dense features for pixel labeling instead of complicated dilated convolution and artificially designed decoder networks.

The proposed approach achieved state-of-the-art performance in 2019 on PASCAL VOC 2012 and Cityscapes benchmarks with a new record of mIoU accuracy 84.0% on PASCAL VOC 2012, while training without COCO dataset.

Feature Pyramid Network (FPN) (Kirillov et al. 2019)

Feature pyramids are a component in recognition systems for detecting objects at different scales, but at the same time they can be expensive both computationally and in terms of memory. In this

case, a network is designed that, taking advantage of the multiscale pyramidal hierarchy typical of deep convolutional networks, build feature pyramids with very little extra cost.

This has become a popular instance segmentation method, with a semantic segmentation branch using a shared Feature Pyramid Network (FPN) backbone. This simple baseline not only remains effective for instance segmentation, but also yields a lightweight, top-performing method for semantic segmentation.

Pyramid Scene Parsing Network (PSPNet) (Zhao et al. 2017)

Scene parsing is to segment and parse an image into different image regions associated with semantic categories, such as sky, road, tree and mountain.

This network exploits the capability of global context information by different-regionbased context aggregation and a pyramid pooling module together with it.

DeepLabV3+ (Liang-Chieh Chen et al. 2018)

Spatial pyramid pooling modules and U-shape structure are used for semantic segmentation task. The former networks are able to encode multi-scale contextual information by probing the incoming features with filters or pooling operations at multiple rates and multiple effective fields-of-view, while the latter networks can capture sharper object boundaries by gradually recovering the spatial information.

In DeepLabV3+ the advantatges of both methods above mentioned are combined. DeepLabV3+ extends DeepLabv3 (Liang Chieh Chen et al. 2018) by adding a simple yet effective decoder module to refine the segmentation results especially along object boundaries.

DeepLabV3 already meant the inclusion of several innovations, such as the inclusion of convolution with upsampled filters (atrous convolution) that allows to control the resolution at which feature responses are computed within Deep Convolutional Neural Networks. It also incorporated the atrous spatial pyramid pooling (ASPP) proposal to robustly segment objects at multiple scales, and improved the localization of object boundaries by combining methods from DCNNs and probabilistic graphical models.

Multi-scale Attention Network (Ma-Net) (Fan et al. 2020)

Multi-scale self-attention to adaptively integrate local features with their global dependencies.

There have been a large number of variants of U-Net based on Multi-scale feature fusion to improve the segmentation performance for medical image segmentation. These proposals extract the context information of medical image via applying the multi-scale feature fusion. In Ma-Net self-Attention mechanism is introduced to adaptively integrate local features with their global dependencies. Thus, the MA-Net can capture rich contextual dependencies based on the attention mechanism.

U-Net++ (Zhou et al. 2018)

As its name suggests, this model is an extension of the U-Net model. This is a deeply-supervised encoder-decoder network where the encoder and decoder sub-networks are connected through a series of nested, dense skip pathways. These re-designed skip pathways aim at reducing the semantic gap between the feature maps of the encoder and decoder sub-networks, and constitute the main improvement of this model. These skip pathways, shown in green and blue, can be overviewed in Figure 8, taken from the original referenced paper from Zhou et al. (2018).

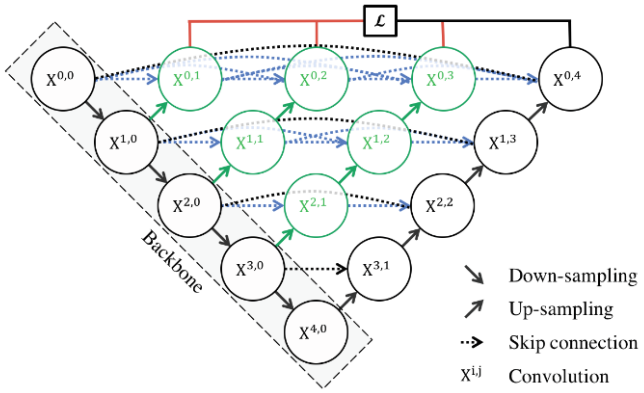

Figure 8: Skip pathways (coloured) on U-Net++ as improvement over original U-Net

U^2~Net (Qin et al. 2020)

U^2-Net is a two-level nested U-structure that is able to capture more contextual information from different scales thanks to the mixture of receptive fields of different sizes, and also increases the depth of the whole architecture without significantly increasing the computational cost. This architecture enables to train a deep network from scratch without using backbones from image classification tasks.

LinkNet (Chaurasia and Culurciello 2018)

A novel deep neural network architecture specifically designed for semantic segmentation. In this network the first aim is to make efficient use of scarce resources availables on embedded platforms, compared to deep learning worksatations. Thus, this network is not only committed to precision, but also to efficiency. That means that it learns without any significant increase in number of parameters.

4 The datasets

An absolutely critical part of supervised deep learning processes is the dataset. The original idea of this work was to be able to train the model with an extensive collection of real medical images; however, the limitations imposed by the protection of personal data have meant that by the moment this option has been ruled out.

In this work, two datasets have been used, as described in the following sections. One may think that there is no need to have two datasets and that one would be enough. In fact, it is so; what happened is that the final dataset was obtained more than halfway through the development period, which made it necessary to have a basic dataset to move forward.

There has been a provisional dataset with a set of cells that do not exactly correspond to the ones that are the object of the work, and with few images. This dataset has been used to develop the training pipeline and to carry out all the software tests and training of the different models, except for the final model.

On the other hand, there has been a definitive dataset with images of exactly the cells that are the object of this work.

4.1 The basic dataset

In the first place, a public dataset with all the necessary ground truth masks has been used to train a semantic segmentation model. This dataset has been published by "*the CellaVision blog*" and can be found at reference: Zheng et al. (2018). It is a dataset that has only 100 good quality images of dimensions 300×300. The cell types in this dataset do not exactly fit the purpose of this work; but it may be enough to make a first approach to the problem. Later, with the final dataset available, the hyperparameters are adjusted. This dataset consists of the cell types shown in Figure 9, while the intention of the final work will focus on monocytes, lymphocytes, HCL cells (Hairy Cell Leukemia.) and APL cells (Acute Promyelocyte Leukemia).

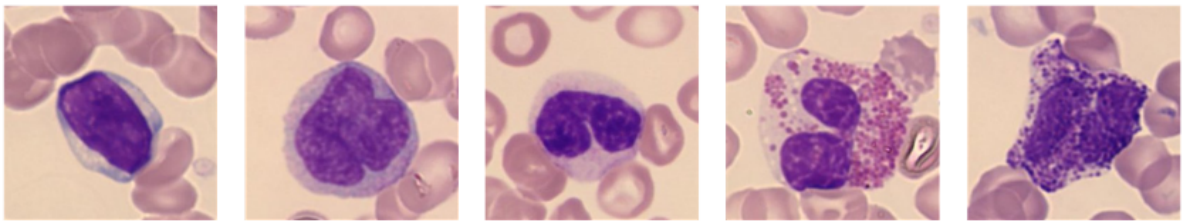

Figure 9: Example of cell from the basic dataset used. From left to right: Lymphocyte, monocyte, neutrophil, eosinophil and basophil

The masks are made in such a way that they are stored in the disc as grayscale images. They have the maximum value (255) for the nucleus, the minimum value (0) for the background, and an intermediate value (128) for the cytoplasm. Thus, therefore, these are masks that should be used to identify 2 classes of elements without counting the background: nucleus and cytoplasm.

4.2 The final dataset

The final dataset has been generated by a GAN network; that is, they are not real images, but artificial images generated by an artificial intelligence process that is outside the scope of this

project. These images have been generated by K. Barrera with the GAN developed in Barrera et al. (2023).

In this case, as indicated in previous sections, we have exactly the cell types that had been planned to be the objective of this work:

- Monocyte
- Lymphocyte
- HCL (Hairy cell leukemia.)
- APL (Acute Promyelocyte Leukemia)

There are exactly 566 monocyte images, 552 lymphocyte images, 516 LCH images, and 426 APL images. Therefore, we have a reasonably balanced dataset of 2,060 images.

All images have their mask. The masks have been obtained through a complex process that uses various techniques, and which is not the subject of this project either. Specifically, the masks have been obtained through a set of techniques described in Santiago Alférez et al. (2015), and that have been applied by the author of the referenced work. In summary, there is a fairly balanced dataset of 2,060 images of mononuclear cells and their respective masks. Let's see a brief sample.

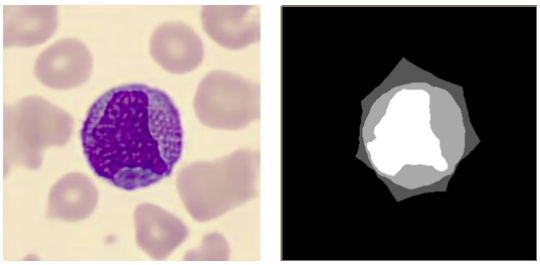

Figure 10: Example of monocyte image and mask from the final dataset

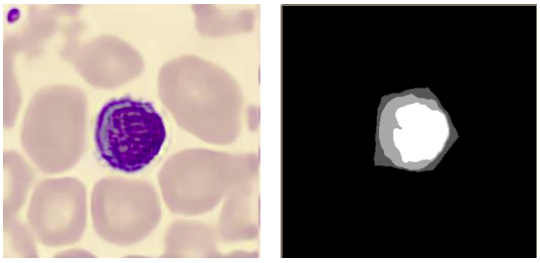

Figure 11: Example of lymphocyte image and mask from the final dataset

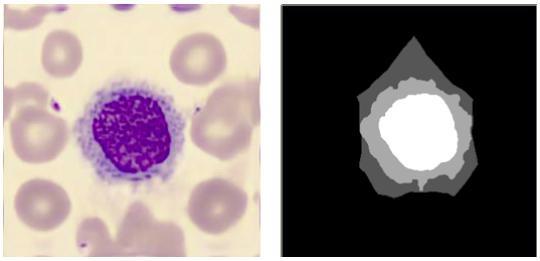

Figure 12: Example of HCL image and mask from the final dataset

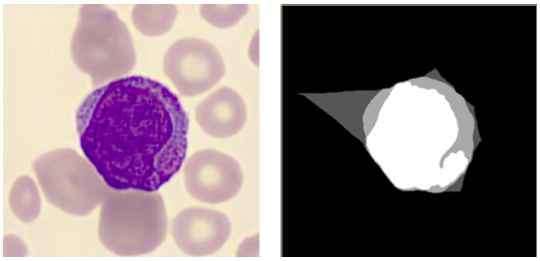

Figure 13: Example of APL image and mask from the final dataset

A very important aspect to highlight from these images is that the mask has, in addition to the background, 3 different gray values. It has a first area with a value of 255, which is the kernel, just like in the reduced dataset. It also has a second area, with a value of 170, which corresponds to the cytoplasm. In addition, it has a third zone of value 85 that corresponds to a zone around the cytoplasm. This zone, which we will call the environment zone, is around the cell and does not correspond to any physical element, but to a conceptual one. This is an environment that is chosen so that no other cells appear; this environment could be interesting since villi may appear on some cells, which would be indicative of some ailment. For this reason it may be interesting to have identified this area for future studies.

In summary, the final dataset has 3 classes or levels of gray, in addition to the background. These gray levels are only visible in images saved to disk. When the masks are loaded to train the model, they are converted to binary masks, as will be seen in the chapter 5.1.2.

5 The segmentation pipelines

5.1 The pipeline for model selection

For the selection of the model, a pipeline has been set up that consists of a series of stages, some of which are different from those that we will find in the final training pipeline.

5.1.1 Augmentation

First of all, it must be taken into account that this step has been carried out on the basic dataset which, as indicated above, only has 100 images of up to 6 different types of cells. This number of images is insufficient to perform a fine adjustment of the models; but not to make an estimate of performance.

Even so, in order to improve the reliability of the results as much as possible, the basic dataset has been subjected to an augmentation process that differs significantly from the usual procedure.

Normally, the augmentation is done on the fly; that is, at the same moment of the training, when the images are loaded, a series of transformations are applied to them in such a way that each image can be different in each epoch. For example, it may be common to randomly apply the flip transformation (reverses the image across one axis). In this way, on one load the image would be oriented in one direction and on the next load it could be oriented in the opposite direction.

It is important to realize that cell images are rotation invariant, and from this point of view it can be interesting to do many rotations, since each one of them gives us a different, but perfectly correct image of the cell. Keeping this in mind it has been decided to carry out an aggressive augmentation in this sense. From each image, 15 more images have been obtained through rotations and flips. Specifically, seven 45-degree rotations have been made to each image. Then it has been flipped and then seven 45-rotations of the flipped image have been made.

Each new image has been saved to disk, so that from an original dataset of 100 images we have ended up with a dataset of 1600 images. In this way, although it is true that it is not possible to have more examples of cells than those originally available, certain variability in the images is achieved. In addition, it is a realistic variability, since it is understood that the images of cells do not really have an orientation as such, therefore they can be in any position (any rotation is equally valid). See an example at Figure 14

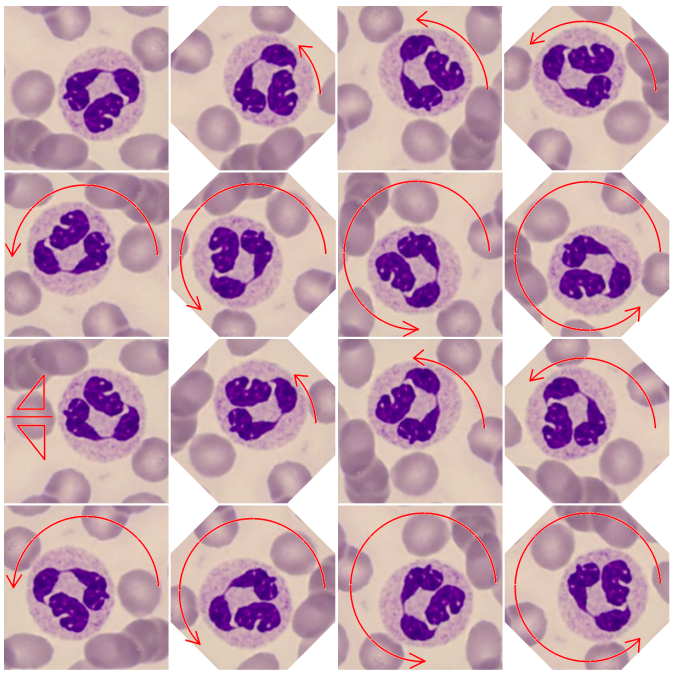

Figure 14: Augmentation in basic dataset. One image to sixteen

5.1.2 Implementation of datasets and dataloaders objects

The implementation of the dataset and dataloader objects is fundamental for a correct functioning of the pipeline. Depending on the type of algorithm being built, there may be useful default settings of these objects; however, in this case, the Dataset object has to be user defined to adapt it to the characteristics of the segmentation and of the source data.

A semantic segmentation requires that the images go in pairs. On the one hand, you will have the original image and on the other hand, you will have the corresponding mask. It is necessary to make sure that each image has its mask and vice versa, and that when an image is loaded its corresponding mask is loaded and not another.

Furthermore, each geometrical transformation that is done to an image will necessarily have to be applied to its mask. For example, if a rotation was applied to the image, but this rotation was not applied to the mask, then the network would not be able to recognize patterns properly. The same would happen with a crop or, in general, any transformation that is applicable to both images.

In this case, as has been said before, no transformation will be applied on the fly, since the augmentation process has been carried out previously and has been recorded on disk; however, the dataset and dataloaders objects are identical in this case and in the case of the final pipeline, in which on the fly transformations will be applied. For this reason, emphasis is placed on this aspect and the definition of the Dataset objects contemplates the possibility of transformations by image-mask pairs, as will be seen shortly.

Another aspect to take into account that has to be resolved at this point is the fact that multiclass semantic segmentation requires labeling (some masks) in one-hot encoding format. This means that, for each class, there will be a matrix of numbers mapping each pixel of the image. This matrix will have a 1 in the pixels that are part of the evaluated class and a 0 in all the others. In the case at hand (with the basic dataset) we have 2 classes, and with the final dataset we will have 3 classes, it will be necessary to implement a generic functionality that can transform the masks

as they arrive (flat and in grayscale) into masks suitable for training (three-dimensional, with as many channels as classes, and in binary)

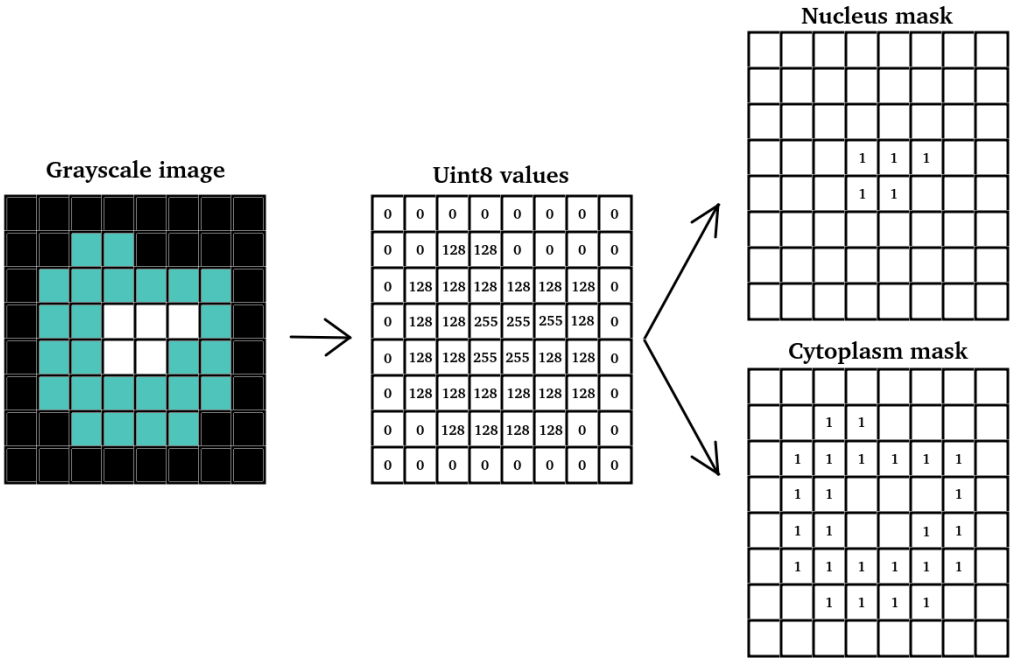

Figure 15: Building masks from grayscale (unsigned integer) to binary one-hot encoding

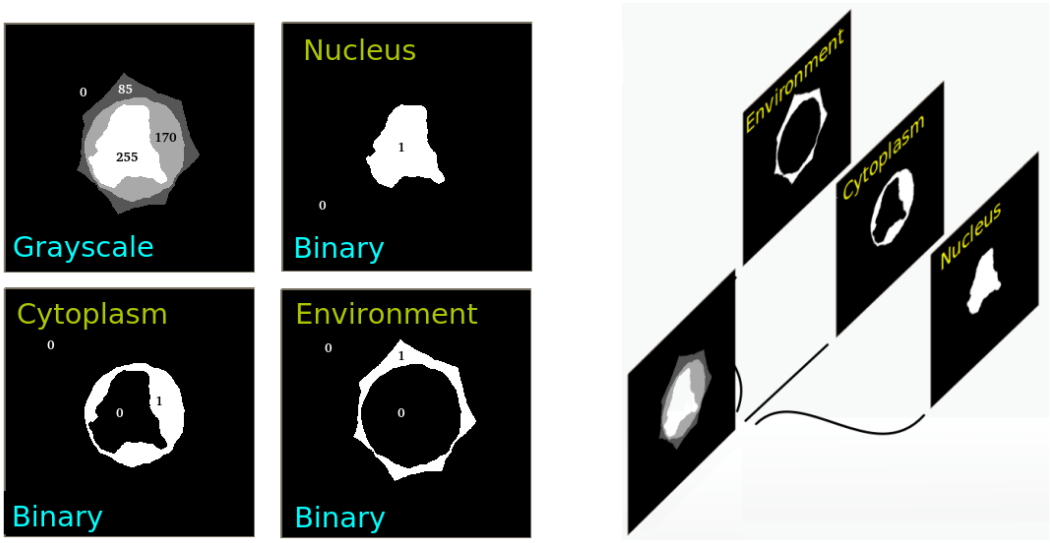

Figure 16: Example of masks from grayscale (unsigned integer) to stacked tensors in binary one-hot encoding

All of this, both the necessary transformations for augmentation and the transformation from flat mask to mask with volume, is done in the custom class called SegDataset, which is a class that inherits attributes from the default class in pytorch, Dataset. The full code that describes the SegDataset class can be consulted in Appendix A, as well as in reference Calafat (2023). Just keep in mind that image loading and augmentation take place in the `__getitem__()` method, while the transformation of the mask from the original state to the one needed for training is done by calling the internal method `__deep_mask__()`.

```
    def __getitem__(self, idx):
        # Grab the image path from the current index
        im_path = self.image_paths[idx]
        # Load the image with PIL as numpy array
        im = np.array(Image.open(im_path))
        # Read mask with PIL as numpy array
        ms_path = self.mask_paths[idx]
        mk = np.array(Image.open((ms_path)))

        # Apply albumentations transforms
        augmented = self.transforms(image=im, mask=mk)
        image = augmented['image']
        mask = augmented['mask']

        # Convert flat mask to deep mask (one channel per class)
        mask = self.__deep_mask__(mask)

        # return a tuple of the image and its mask as tensors
        return (to_tensor(image), to_tensor(mask))
```

Figure 17: Code that provides a new item from the dataset and applies transformations

```
    def __deep_mask__(self, mask):
        # Convert a flat grayscale mask into a channeled grayscale mask

        # Convert a np mask with a grayscale format (H x W) with value range
        # 0-255 (equally spaced; i.e. 2 classes + background would be 0, 128,
        # 255) to a numpy mask (H x W x C) with C equal to number of channels
        # (classes) with 255 in the active pixels and 0 otherwise (uint8
        # format)

        # Number of different levels
        nlevels = self.num_class

        # Mask arrives with 255 gray levels equaly spaced
        mask = np.round(mask / (255 / nlevels))

        # Leave each level in a channel
        masks = []
        for k in range(nlevels):
            base = np.zeros_like(mask, dtype=np.uint8)
            ix = np.argwhere(mask == (k + 1))
            base[ix[:, 0], ix[:, 1]] = 255
            masks.append(base)

        # Stack all channels
        return np.stack(masks, axis=2)
```

Figure 18: Code that transforms a flat grayscale level into 2-level stacked tensor of matrices with as many channels as classes

On the other hand, the dataloaders are defined in a standard way, since they just call the *SegDataset* objects.

5.1.3 Loss function and performance metrics

In all training of a model there must be a loss function so that the model can gradually minimize the error and gradually adjust to what is desired. In the case at hand, given that finally the masks (targets) are made up of 0's and 1's, the immediate loss function would be the Binary Cross Entropy (BCE) loss.

$$BCELoss = \frac{-1}{N}\sum_{n=1}^{N}[y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n)] \tag{2}$$

where y_n is 0 or 1 depending on the mask pixel value, and $\hat{y}_n$ is the output of the neural network.

Traditionally, a sigmoid function is applied to the output of the neural network, which converts the scores into probabilities. In this case we are not going to do this, and for this reason the loss function used will be the Binary Cross Entropy Loss with Logits that is available in Pytorch (2018), which refers precisely to the fact that the output is scores (also called 'logits') and not probabilities; that is, it is the output without applying the sigmoid function to it.

However, we are going to combine this loss function with another one. By itself the BCE loss would be enough to carry out the training, but it has been preferred to combine it with a modification of

the Dice coefficient as done in Naoto (2018) because the pertinent simulations have been carried out and the results obtained are better with the combined loss (by "better" it is meant that the Dice coefficient is higher).

The Dice coefficient is a well-known metric for evaluating the correct identification of areas; that is, it perfectly fits the task of semantic segmentation.

The Dice coefficient equals twice the number of pixels common to two areas divided by the sum of the number of pixels in each area.

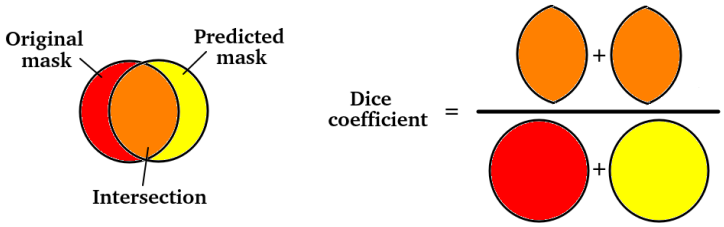

Figure 19: Graphical explanation of Dice coefficient

As can be deduced from the previous explanation, the Dice coefficient will be a value between 0 and 1, with 1 being the best value. A Dice value equal to 1 would indicate that the predicted area is equal to the target area, so the result would be unbeatable.

If we try to combine the Dice coefficient in the loss function, then it is necessary that both parts (the BCE part and the Dice part) behave in the same direction. For this reason, in the evaluation of the loss function we will incorporate losses equal to 1 minus the Dice coefficient.

In this case, a 50% weighting of each loss has been chosen ($w = 0.5$) in equation (3), so that the loss function takes both aspects equally into account. The choice of the proportion has been based on several test simulations with different ratios.

$$loss = w \cdot loss_{BCE} + (1 - w) loss_{DICE} \tag{3}$$

On the other hand, it should also be noted that the performance metric used will be directly the Dice coefficient, since it is calculated for the loss function and it is a data that is intuitively better understood than the loss.

5.1.4 The pipeline with Kfold cross validation

Finally, a pipeline has been developed that contains an algorithm with K-Fold cross validation, as is usually done with small datasets.

This pipeline has been run 3 times with k=10, k=8 and k=6 respectively. In each run all 9 models seen in the section 3 have been trained, which implies that finally 24 complete trainings have been carried out on each model. After each K-Fold training round, the average metric (Dice coefficient) of the K iterations has been calculated on each validation set. Finally, the three averages have been averaged. The final result is shown in table 5.

Table 5: Mean Dice coefficients of the models (in percentage)

Model	Dice coefficient %
DeepLabV3+ EfficientNet-b6	94.90
Manet timm-EfficientNet-b6	94.80
Unet++ EfficientNet-b6	95.15
Unet timm-Efficientnet-b6	95.12
U2net	94.46
PAN EfficientNet-b6	94.89
FPN timm-EfficientNet-b6	95.04
PSPNet timm-EfficientNet-b6	94.28
Linknet timm-EfficientNet-b6	90.64

With the existing dataset, all the models work reasonably well. Finally, the one that obtains the best result is chosen, which is the Unet++ model with a pre-trained encoder according to the EfficientNet-b6 network; although it is clear that any other could be selected and also good results would be achieved.

5.2 The pipeline for train, validation and test

As previously indicated, a second pipeline has been developed and is also available at reference Calafat (2023). This second pipeline is to be used with the final dataset, and is intended to fine-tune the model selected in the previous pipeline. A summary of its content is shown in Figure 20.

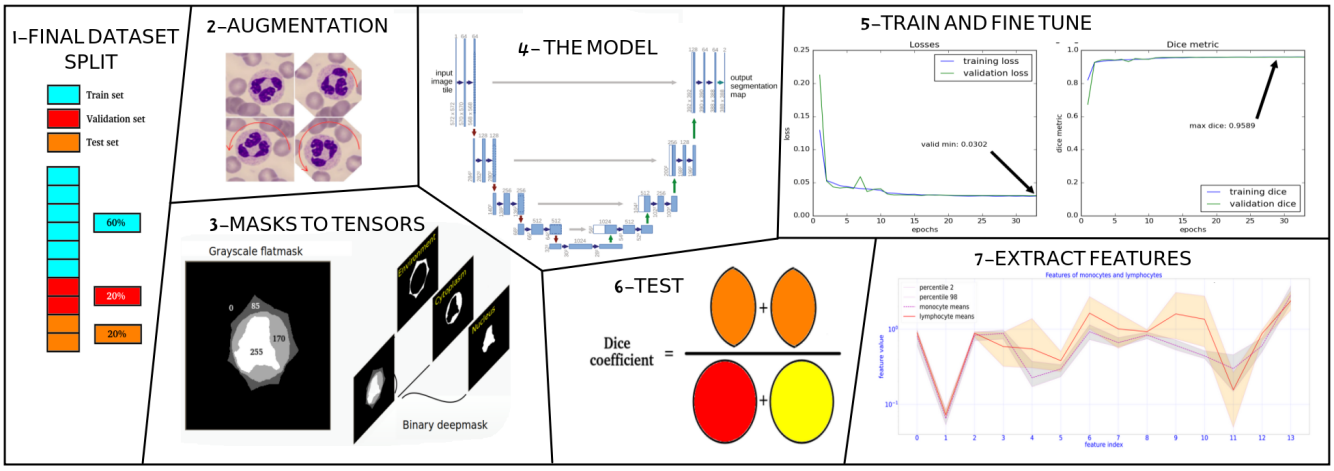

Figure 20: Summary of the pipeline contents used to fine tuning the segmentation model

In this case, the dataset and daloader objects are the same as those described in the previous sections. The only thing to highlight is that, now, an augmentation method is used consisting of random crops, rotations and flips that will be implemented on the fly. In other words, unlike in the previous case, in this case the augmented dataset is not saved to disk, but rather it is implemented as the images are loaded.

Another important point regarding the training process for model selection is a different use of the Dice coefficient. In the first case there were only two classes (nucleus and cytoplasm); however, now, with the final dataset, there are 3 classes (nucleus, cytoplasm and environment). The surrounding zone can become quite large and does not correspond to a physical object such as the nucleus or cytoplasm. From this point of view, it is not recommended to use it in the calculation of the Dice coefficient, since it could falsely improve the result. That is to say, an improvement in the Dice coefficient based on the improvement of the prediction of the environment zone would not be really beneficial if it implies that the nucleus or cytoplasm zone is not being sufficiently improved. For this reason, the calculation of the Dice coefficient in this training is only performed on the classes of nucleus and cytoplasm, leaving the active environment zone for the calculation of the BCE (Binary Cross Entropy).

On the other hand, this time the typical method of train, validation and test is used, with a percentage of images of 60%, 20% and 20% respectively. The images that make up each dataset are randomly selected on the fly.

The training graphs are shown in Figure 21. We see the loss function and the Dice coefficient as a function of the epoch number.

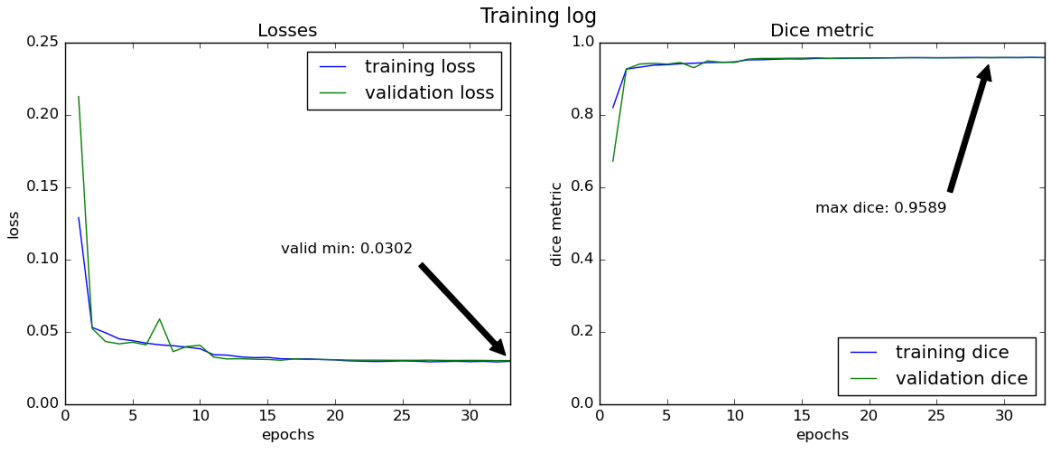

Figure 21: Training log of the tuned model

In the same procedure, for completeness, the model is tested and an example of parameter extraction is shown. These two aspects will be dealt with in sections 5.3 and 6.3 and can be viewed in the full code at reference Calafat (2023).

5.3 Results of segmentation model

Previously, Figure 21 has shown that the maximum Dice coefficient achieved with the adjusted model has been 95.89. This is a fairly good result, although it must be taken into account that it has been obtained with the validation data.

To test the model reliably, it must be done with the test dataset, which is a set that the model has not seen before. So after the training part, the algorithm is executed with the entire test data set and the Dice coefficient is calculated. The result has been a Dice coefficient of (95.7± 0.4)%, which confirms that the model generalizes quite well, since in the test it has achieved an equivalent score than that obtained with the validation dataset.

On the other hand, Figure 22 show an example of segmentation obtained with the trained model.

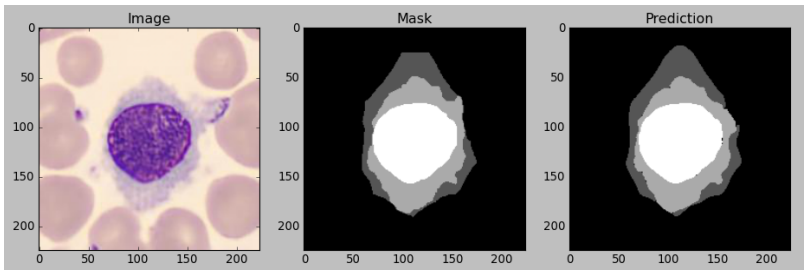

Figure 22: Example of image, original mask and segmentation predicted by the trained model

In this case we see that the model generates a very precise mask, even in difficult areas such as the cytoplasm in the upper right. Without being bad, there is less precision in the environment area, but this, as said before, is unimportant. See another test example in Figure 23.

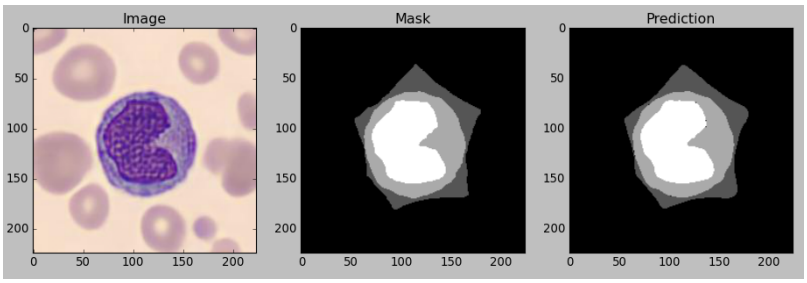

Figure 23: Example of image, original mask and segmentation predicted by the trained model

Obviously the algorithm is not expected to be able to segment perfectly. Next we see some segmentations that, without being bad, have not been so precise.

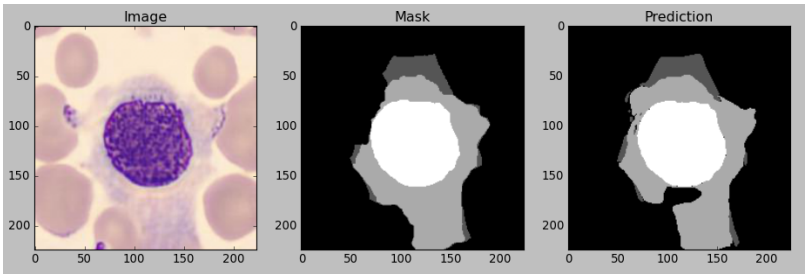

Figure 24: Example of image, original mask and segmentation predicted by the trained model. There's a hole in the prediction of the cytoplasm class

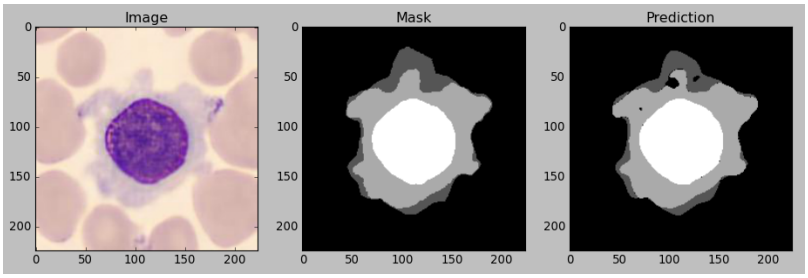

Figure 25: Example of image, original mask and segmentation predicted by the trained model. There's a hole in the prediction of the cytoplasm class

Although in general the results are quite good. Some more examples are shown in Figure 26.

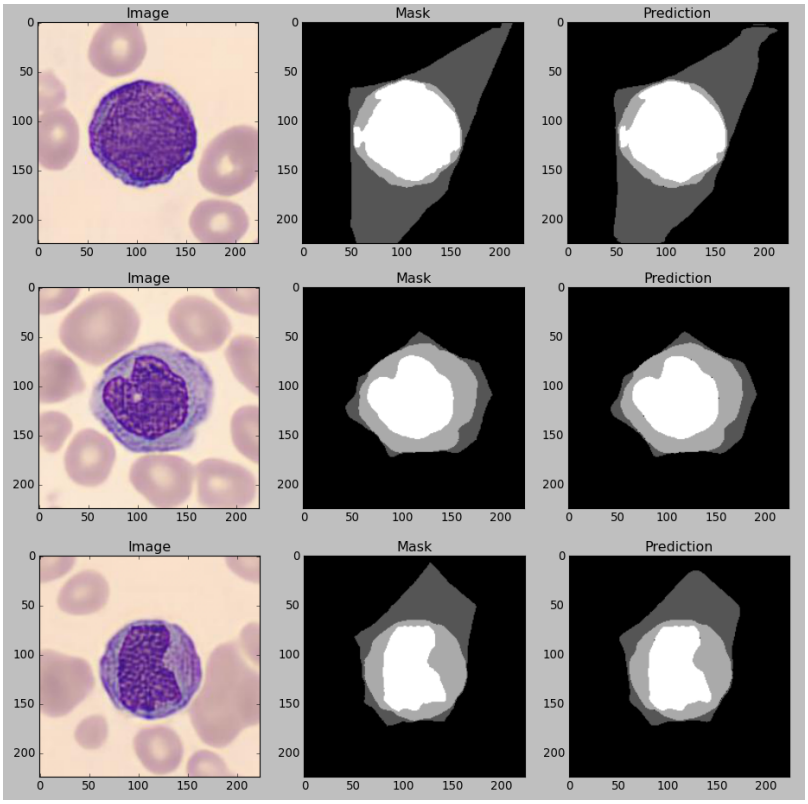

Figure 26: Example of images, original masks and good predictions

6 Graphic feature extraction

A first objective of this work was the implementation of a semantic segmentation model, for this reason the results have been presented in the previous section. However, a second objective was the extraction of graphic parameters from the generated masks and establish a base to be able to go deeper into the comparison of different parameters between the different cell types.

Two types of parameters have been extracted, shape parameters and pixel-value (or first order) parameters. Note that the first ones are extracted directly on the masks, since the shape parameters, such as area, or largest diameter, have to be calculated on the layer that contains the object to be analyzed (that is, on the mask). On the other hand, the pixel-value parameters (such as mean, median, skewness or kurtosis) are calculated on the original image transformed to grayscale, but only considering the pixels that remain activated by the corresponding mask. So, for example, to calculate the mean value of the nucleus pixels, the nucleus mask is applied first, and then the mean is calculated over the selected pixel values.

Each of the extracted parameters is briefly explained below. All these features have been extracted using the Pyradiomics (2023) library, which incorporates them by default. A more detailed explanation, and even the source code of each feature can be consulted in the documentation of the aforementioned library.

6.1 Shape features

- Elongation: Elongation shows the relationship between the two largest principal components in the ROI (Region Of Interest) shape. For computational reasons, this feature is defined as the inverse of true elongation.

- MajorAxisLength: This feature yield the largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component.

- MaximumDiameter: Maximum diameter is defined as the largest pairwise Euclidean distance between ROI ((Region Of Interest)) surface mesh vertices.

- MinorAxisLength: This feature yield the second-largest axis length of the ROI-enclosing ellipsoid and is calculated using the largest principal component.

- Perimeter: Perimeter of ROI (Region Of Interest).

- PerimeterSurfaceRatio: Ratio between perimeter and area. Here, a lower value indicates a more compact (circle-like) shape.

- PixelSurface: The surface area of the ROI (Region Of Interest).

- Sphericity: Sphericity is the ratio of the perimeter of the ROI to the perimeter of a circle with the same surface area as the ROI (Region Of Interest).

From this point on the features are combinations of the shape features of the nucleus and the cytoplasm.

- Combined elongation: Nucleus elongation divided by cytoplasm elongation.
- Combined MajorAxisLength: Nucleus major axis length divided by cytoplasm major axis length.
- Combined MaximumDiameter: Nucleus maximum diameter divided by cytoplasm maximum diameter.
- Combined MinorAxisLength: Nucleus minor axis length divided by cytoplasm minor axis length.
- Combined Perimeter: Nucleus perimeter divided by cytoplasm perimeter.
- Combined PerimeterSurfaceRatio: Nucleus perimeter surface ratio divided by cytoplasm perimeter surface ratio.
- Combined PixelSurface: Nucleus pixel surface divided by the sum of nucleus and cytoplasm pixel surfaces.
- Combined Sphericity: Nucleus sphericity divided by cytoplasm sphericity.

6.2 Pixel-value features

- 10Percentile: 10th percentile of pixel values.
- 90Percentile:: 90th percentile of pixel values.
- Energy: Energy is a measure of the magnitude of pixel values in an image. A larger values implies a greater sum of the squares of these values. The values showed are divided by $1 \cdot 106$ to avoid the difference of order of magnitude with the rest of features.
- Entropy: Entropy specifies the uncertainty/randomness in the image values. It measures the average amount of information required to encode the image values.
- InterquartileRange: Interquartile range between 25th and 75th percentiles.
- Kurtosis: Kurtosis is a measure of the 'peakedness' of the distribution of values in the image ROI.
- Maximum: The maximum gray level intensity within the ROI (Region Of Interest).
- MeanAbsoluteDeviation: Mean Absolute Deviation is the mean distance of all intensity values from the Mean Value of the image array.
- Mean: The average gray level intensity within the ROI (Region Of Interest).
- Median: The median gray level intensity within the ROI (Region Of Interest).

- Minimum: The minimum gray level intensity within the ROI (Region Of Interest).
- Range: The range of gray values in the ROI (Region Of Interest).
- RobustMeanAbsoluteDeviation: Robust Mean Absolute Deviation is the mean distance of all intensity values from the Mean Value calculated on the subset of image array with gray levels in between, or equal to the 10th and 90th percentile.
- RootMeanSquared: RMS is the square-root of the mean of all the squared intensity values. It is another measure of the magnitude of the image values.
- Skewness: Skewness measures the asymmetry of the distribution of values about the Mean value. Depending on where the tail is elongated and the mass of the distribution is concentrated, this value can be positive or negative. The value represented in the plot, as it is in logarithmic scale, is the real value plus one, to avoid that negative values become minus infinite.
- Uniformity: Uniformity is a measure of the sum of the squares of each intensity value. This is a measure of the homogeneity of the image array, where a greater uniformity implies a greater homogeneity or a smaller range of discrete intensity values.
- Variance: Variance is the the mean of the squared distances of each intensity value from the Mean value. This is a measure of the spread of the distribution about the mean.

6.3 Result of the comparison of features in the complete dataset

Some about feature extraction are shown in Figure 27 as a comparison between cell types. In the following section the software developed that contains this analysis and many more will be referenced (and linked), so that in order for this document not to be excessively long and cumbersome, only a part of the analysis of shape features between monocytes and lymphocytes is reflected in this document.

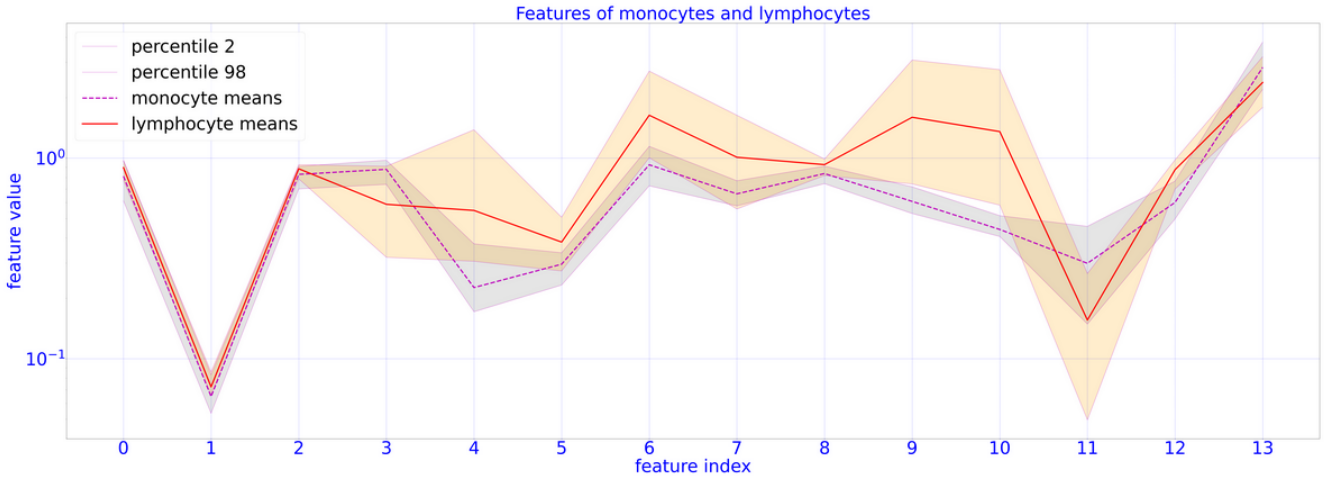

Figure 27: Graphical comparison of monocyte – lymphocyte shape features

On the abscissa axis we have a feature index, which is nothing more than an indicator to be able to identify the corresponding feature in the following table. On the ordinate axis we have the value of the corresponding feature in logarithmic scale.

The graph clearly shows how the range in which some features move is very similar between the two types of cells; while in other cases the ranges are quite different, and it is these features that are the most interesting from a practical point of view.

Table 6 shows the features of the graph for the case of monocytes.

Table 6: Monocyte features

Index	Features	Percentile.2	Percentile.50	Percentile.98	Mean	Std.Deviation
0	Nucleus Elongation	0.6120	0.8233	0.9753	0.8116	0.1069
1	Nucleus PerimeterSurfaceRatio	0.0535	0.0631	0.0831	0.0648	0.0081
2	Nucleus Sphericity	0.7016	0.8442	0.9090	0.8290	0.0588
3	Cytoplasm Elongation	0.7393	0.8856	0.9753	0.8784	0.0592
4	Cytoplasm PerimeterSurfaceRatio	0.1714	0.2137	0.3737	0.2262	0.0454
5	Cytoplasm Sphericity	0.2329	0.2992	0.3372	0.2959	0.0240
6	Combined Elongation	0.7270	0.9343	1.1433	0.9253	0.1177
7	Combined MajorAxisLength	0.5775	0.6564	0.7703	0.6631	0.0556
8	Combined MaximumDiameter	0.7453	0.8398	0.9113	0.8369	0.0427
9	Combined MinorAxisLength	0.5293	0.6040	0.7186	0.6081	0.0445
10	Combined Perimeter	0.4065	0.4336	0.5161	0.4407	0.0266
11	Combined PerimeterSurfaceRatio	0.1488	0.2977	0.4564	0.2988	0.0743
12	Combined PixelSurface	0.4968	0.5910	0.7669	0.6010	0.0648
13	Combined Sphericity	2.2023	2.7870	3.7828	2.8286	0.3780

While the same data corresponding to lymphocytes are shown in Table 7.

Table 7: Lymphocyte features

Index	Features	Percentile.2	Percentile.50	Percentile.98	Mean	Std.Deviation
0	Nucleus Elongation	0.7750	0.9021	0.9686	0.8956	0.0450
1	Nucleus PerimeterSurfaceRatio	0.0661	0.0709	0.0861	0.0724	0.0053
2	Nucleus Sphericity	0.7876	0.8926	0.9270	0.8841	0.0345
3	Cytoplasm Elongation	0.3203	0.5745	0.9101	0.5881	0.1538
4	Cytoplasm PerimeterSurfaceRatio	0.3060	0.4699	1.3824	0.5481	0.4309
5	Cytoplasm Sphericity	0.2737	0.3755	0.5070	0.3812	0.0641
6	Combined Elongation	1.0022	1.5395	2.7106	1.6339	0.4597
7	Combined MajorAxisLength	0.5574	0.8972	1.6331	1.0088	0.3563
8	Combined MaximumDiameter	0.8182	0.9318	0.9881	0.9271	0.0492
9	Combined MinorAxisLength	0.7429	1.5137	3.0773	1.5958	0.6157
10	Combined Perimeter	0.5826	1.3168	2.7563	1.3544	0.6044
11	Combined PerimeterSurfaceRatio	0.0496	0.1530	0.2659	0.1558	0.0515
12	Combined PixelSurface	0.7004	0.8992	0.9796	0.8748	0.0711
13	Combined Sphericity	1.7873	2.3498	3.1995	2.3797	0.3893

For example, we can see in the graph that the ranges of the first feature (Nucleus elongation) are reasonably similar, which does not provide us with much information. This can be verified later in the table, seeing that the range between the 2nd percentile and the 98th percentile for this feature is 0.612 – 0.9753 in the case of monocytes, while it is 0.775 – 0.9686 in the case of lymphocytes.

Another more revealing feature could be number 10 (the combined perimeter, which we remember is calculated as the division between the perimeter of the cytoplasm and the perimeter of the nucleus). Here we see that the ranges between the 2nd and 98th percentiles do not intersect at any point, so this feature might indeed be relevant and suitable for further observation. The citing range in the case of monocytes is 0.4065 – 0.5161, while in the case of lymphocytes it is 0.5826 – 2.7563.

This and many other analyzes can be done with the developed tool that is explained in the next section.

7 The feature comparison app

An application has been developed for the rapid extraction of features from cell images, and for a later comparison. This application shows the comparison seen in the previous section and many more. The application is in the repository referenced at Calafat (2023) under the name “app.py”.

This application allows you to upload an image, automatically calculates the mask with the developed model, and then shows the results of the parameter extraction in both graphic and numerical format. In addition, in graphical format, it superimposes the graph of the features of the loaded cell on any of the other groups analyzed (monocytes, lymphocytes, HCL and APL).

Its use is extremely simple, since after loading the image you only have to make a single click to extract and compare all the features.

As an example, some screenshots of the application are shown, although probably the best way to understand it is to try it or watch a video.

This tool has been developed in Gradio, which is a python library for graphical environments. Once the application is launched, you have to open the browser at the address `localhost:7860`. The following screen will appear:

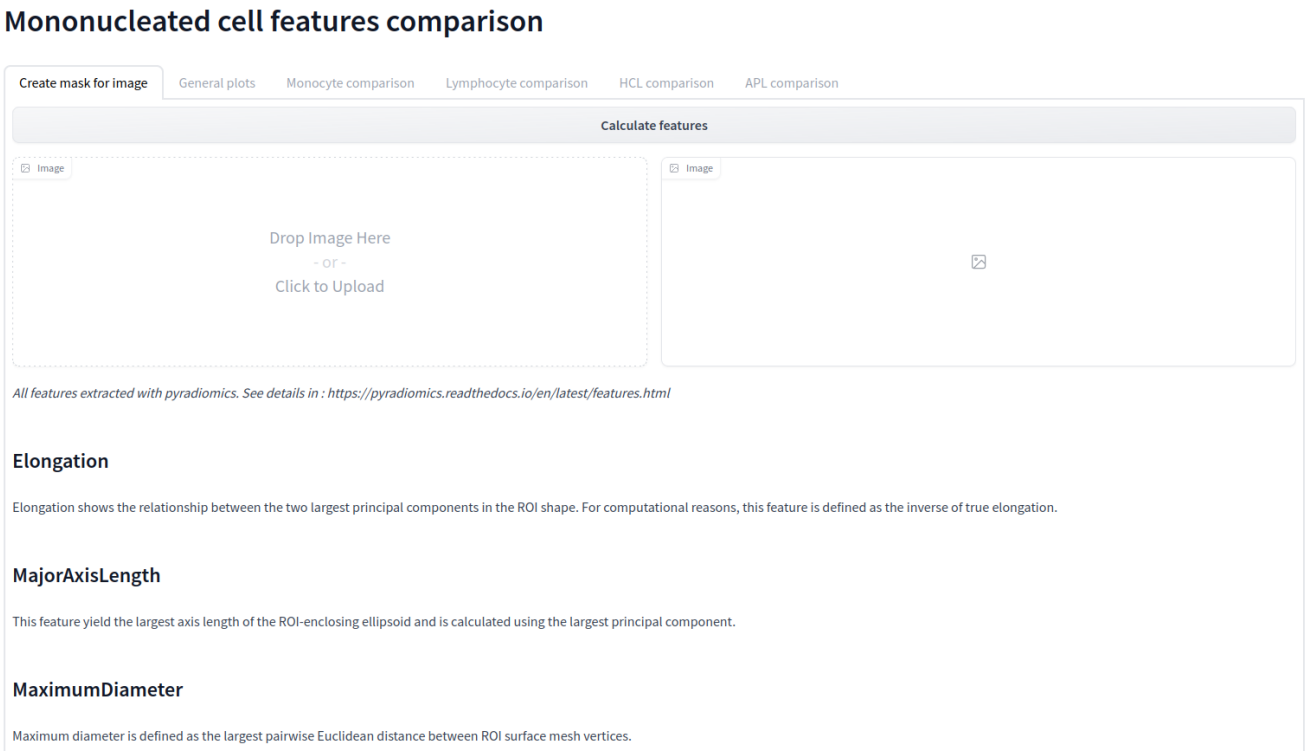

Figure 28: App home screen

All you have to do here is drag the cell image over the area (where it says “drop image here”) and click “calculate features”. The calculated mask will appear in the box on the right, and all the graphs and tables will appear in the rest of the tabs. In addition, a brief description of each feature appears at the bottom of the home screen in case it might be of interest.

There is an example in Figure 29. The loaded cell is from the HCL (Hairy Cell Leukemia.) group:

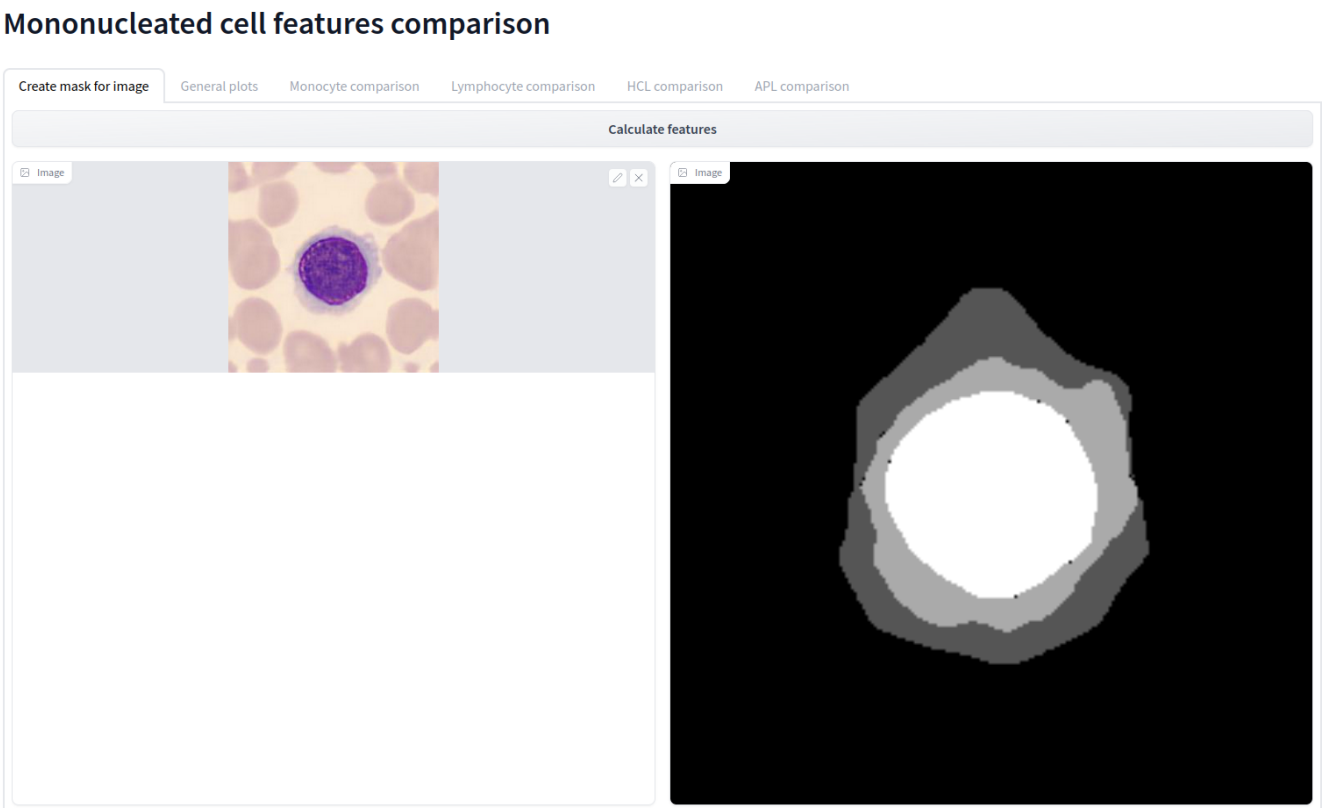

Figure 29: App home screen with a segmentation done

The graphic result of the comparison with the HCL group is as follows:

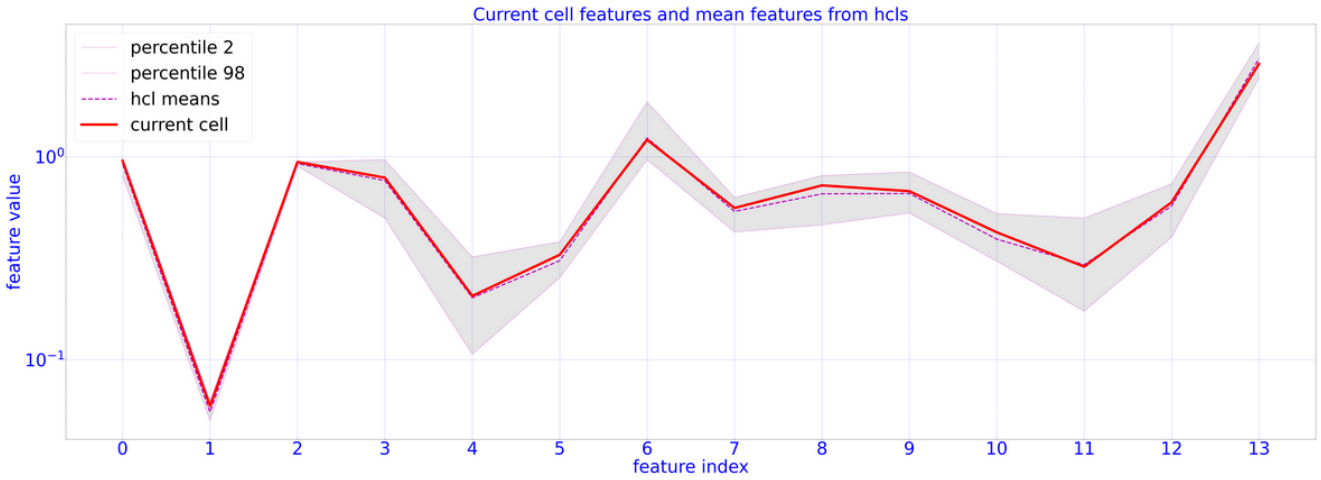

Figure 30: Application screenshot. Shape features 1 from current cell overlaped with HCL means

While if we compare them, for example, with the features of lymphocytes, the result is very different.

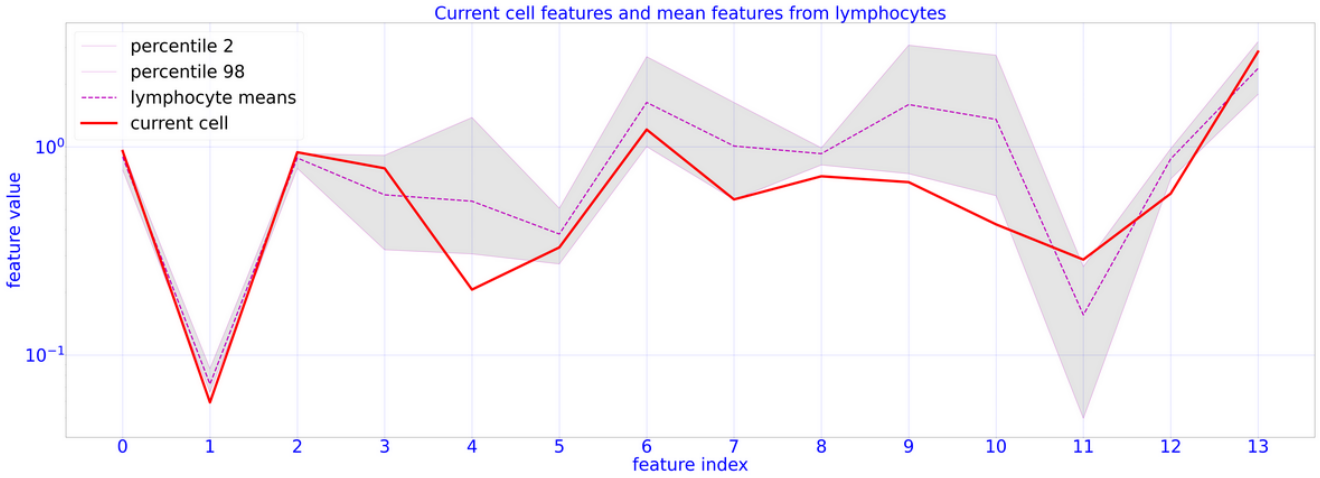

Figure 31: Application screenshot. Shape features 1 from current cell overlaped with lymphocyte means

8 Conclussion

In conclusion, two pipelines have been developed for training models in segmentation task. The first pipeline, with K-fold cross validation, has been done as a comparison between models, and with the aim to get the best one with a cells dataset.

The second pipeline has been used to train and fine tune the selected model in our specific segmentation task with the final dataset.

The trained model has been tested and its performance, based on well known metric such as Dice coefficient, has been very good with a result of (95.7± 0.4)%.

An application has been developed as a comparison tool between graphic features. This tool loads the trained model and when a new image is loaded, automatically does the segmentation and extracts several graphical features with a single click. Besides, facilitates the numerical and graphical comparison with the features of those cells that have been considered in this work (monocytes, lymphocytes, LCH and APL).

All this has been done with the utmost respect for personal data, in such a way that no data that could identify any person has been accessed or stored. This is so, not only due to the fact that dataset 1 is perfectly anonymized and dataset 2 is synthetic, but also due to the very nature of neural network models, that stores only numerical information about weights so it is completely impossible to deduce or infer any personal data from it.

On the other hand, the following aspects could be considered as future work:

On the one hand, other types of features related to textures could be extracted. The library used for feature extraction Pyradiomics (2023) itself incorporates an endless number of them.

It is possible, if the appropriate datasets are available, to extend the study to many other cell types.

In those features that seem to be most relevant, a detailed study could be carried out, which would make it possible to have a set of specific and revealing features for each type of cell.

9 Bibliography

Acevedo, Andrea, Anna Merino, Santiago Alférez, Ángel Molina, Laura Boldú, and José Rodellar. 2020. "A Dataset of Microscopic Peripheral Blood Cell Images for Development of Automatic Recognition Systems." *Data in Brief* 30. https://doi.org/10.1016/j.dib.2020.105474.

Alférez, S. 2015. "390. Methodology for Automatic Classification of Atypical Lymphoid Cells from Peripheral Blood Cell Images."

Alférez, Santiago, Anna Merino, Andrea Acevedo, Laura Puigví, and José Rodellar. 2019. "Color Clustering Segmentation Framework for Image Analysis of Malignant Lymphoid Cells in Peripheral Blood." *Medical and Biological Engineering and Computing* 57. https://doi.org/10.1007/s11517-019-01954-7.

Alférez, Santiago, Anna Merino, Laura Bigorra, Luis Mujica, Magda Ruiz, and Jose Rodellar. 2015. "Automatic Recognition of Atypical Lymphoid Cells from Peripheral Blood by Digital Image Analysis." *American Journal of Clinical Pathology* 143. https://doi.org/10.1309/AJCP78IFSTOGZZJN.

Barrera, Kevin, Anna Merino, Angel Molina, and José Rodellar. 2023. "Automatic Generation of Artificial Images of Leukocytes and Leukemic Cells Using Generative Adversarial Networks (Syntheticcellgan)." *Computer Methods and Programs in Biomedicine* 229: 107314. https://doi.org/https://doi.org/10.1016/j.cmpb.2022.107314.

Calafat, M. Á. 2023. *GitHub - Mikweeh/Cell-Segmentation-and-Feature-Extraction.* Github. Retrieved January 15, 2023. https://drive.google.com/drive/folders/1lA0Gzn-_45JNec9BKCwFKz-_H47QHehB?usp=sharing.

Chaurasia, Abhishek, and Eugenio Culurciello. 2018. "LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation." In *2017 IEEE Visual Communications and Image Processing, VCIP 2017*. Vol. 2018–January. https://doi.org/10.1109/VCIP.2017.8305148.

Chen, Liang Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2018. "Rethinking Atrous Convolution for Semantic Image Segmentation Liang-Chieh." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.

Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. 2018. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 11211 LNCS. https://doi.org/10.1007/978-3-030-01234-2_49.

Fan, Tongle, Guanglei Wang, Yan Li, and Hongrui Wang. 2020. "Ma-Net: A Multi-Scale Attention Network for Liver and Tumor Segmentation." *IEEE Access* 8. https://doi.org/10.1109/ACCESS.2020.3025372.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37. https://doi.org/10.1109/TPAMI.2015.2389824.

Iakubovskii, Pavel. 2019. "Segmentation Models Pytorch." *GitHub Repository.* GitHub.

Kirillov, Alexander, Ross Girshick, Kaiming He, and Piotr Dollar. 2019. "Panoptic Feature Pyramid Networks." In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019-June:6392–6401. IEEE. https://doi.org/10.1109/CVPR.2019.00656.

Li, Hanchao, Pengfei Xiong, Jie An, and Lingxue Wang. 2019. "Pyramid Attention Network for Semantic Segmentation." In *British Machine Vision Conference 2018, BMVC 2018*.

Luong, Minh Thang, Hieu Pham, and Christopher D. Manning. 2015. "Effective Approaches to Attention-Based Neural Machine Translation." In *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*. https://doi.org/10.18653/v1/d15-1166.

Mo, Yujian, Yan Wu, Xinneng Yang, Feilin Liu, and Yujun Liao. 2022. "Review the State-of-the-Art Technologies of Semantic Segmentation Based on Deep Learning." *Neurocomputing* 493. https://doi.org/10.1016/j.neucom.2022.01.005.

Naoto, U. 2018. *GitHub - Usuyama/Pytorch-Unet: Simple PyTorch Implementations of u-Net/FullyConvNet (FCN) for Image Segmentation.* Github. Retrieved January 15, 2023. https://github.com/usuyama/pytorch-unet.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library."

ProfessorDaveExplains. 2021. *Youtube - Introduction to Inmunology.* Youtube. Retrieved January 15, 2023. https://www.youtube.com/watch?v=xQiF2ZwI2uo&feature=youtu.be.

Pyradiomics. 2023. "Pyradiomics Documentation." pyradiomics v3.0.1.post15+g2791e23 documentation. https://pyradiomics.readthedocs.io/en/latest/.

Pytorch. 2018. "Torch.nn.functional.binary_cross_entropy_with_logits." PyTorch 1.13 documentation. https://pytorch.org/docs/stable/generated/torch.nn.functional.binary_cross_entropy_with_logits.html.

Qin, Xuebin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R. Zaiane, and Martin Jagersand. 2020. "U2-Net: Going Deeper with Nested u-Structure for Salient Object Detection." *Pattern Recognition* 106. https://doi.org/10.1016/j.patcog.2020.107404.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 9351. https://doi.org/10.1007/978-3-319-24574-4_28.

Tsukasa Ueno, Qiangfu Zhao. 2018. "Interpretation of Deep Neural Networks Based on Decision Trees." In *Proceedings - IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, IEEE 16th International Conference on Pervasive Intelligence and Computing, IEEE 4th International Conference on Big Data Intelligence and Computing and IEEE 3rd Cyber Science and Technology Congress, DASC-PICom-DataCom-CyberSciTec 2018.* https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00052.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." In *Advances in Neural Information Processing Systems.* Vol. 2017–December.

Zhao, Hengshuang, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. 2017. "Pyramid Scene Parsing Network." In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017.* Vol. 2017–January. https://doi.org/10.1109/CVPR.2017.660.

Zheng, Xin. 2018. *GitHub - Zxaoyou/Segmentation_WBC: White Blood Cell (WBC) Image Datasets.* Github. Retrieved January 15, 2023. https://github.com/zxaoyou/segmentation_WBC.

Zheng, Xin, Yong Wang, Guoyou Wang, and Jianguo Liu. 2018. "Fast and Robust Segmentation of White Blood Cell Images by Self-Supervised Learning." *Micron* 107. https://doi.org/10.1016/j.micron.2018.01.010.

Zhou, Zongwei, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. 2018. *UNet++: A Nested u-Net Architecture for Medical Image Segmentation. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 11045 LNCS. https://doi.org/10.1007/978-3-030-00889-5_1.

A Appendix

```
# Custom class for Dataset
class SegDataset(Dataset):

    def __init__(self, image_paths, mask_paths, transformations=None,
                 num_class=2):
        # Store the image and mask filepaths, and augmentation transforms
        self.image_paths = image_paths
        self.mask_paths = mask_paths
        self.transforms = transformations
        self.num_class = num_class

    def __len__(self):
        # Return the number of total samples contained in the dataset
        return len(self.image_paths)

    def __getitem__(self, idx):
        # Grab the image path from the current index
        im_path = self.image_paths[idx]

        # Load the image with PIL as numpy array
        im = np.array(Image.open(im_path))

        # Read mask with PIL as numpy array
        ms_path = self.mask_paths[idx]
        mk = np.array(Image.open((ms_path)))

        # Apply albumentations transforms
        augmented = self.transforms(image=im, mask=mk)
        image = augmented['image']
        mask = augmented['mask']

        # Convert flat mask to deep mask (one channel per class)
        mask = self.__deep_mask__(mask)

        # return a tuple of the image and its mask as tensors
        return (to_tensor(image), to_tensor(mask))

    def __deep_mask__(self, mask):
        # Convert a flat grayscale mask into a channeled grayscale mask

        # Convert a np mask with a grayscale format (H x W) with value range
        # 0-255 (equally spaced; i.e. 2 classes + background would be 0, 127,
        # 255) to a numpy mask (H x W x C) with C equal to number of channels
        # (classes) with 255 in the active pixels and 0 otherwise (uint8 format)

        # Number of different levels
        nlevels = self.num_class

        # Mask arrives with 255 gray levels equaly spaced
        mask = np.round(mask / (255 / nlevels))

        # Leave each level in a channel
        masks = []
```

```
        for k in range(nlevels):
            base = np.zeros_like(mask, dtype=np.uint8)
            ix = np.argwhere(mask == (k + 1))
            base[ix[:, 0], ix[:, 1]] = 255
            masks.append(base)

        # Stack all channels
        return np.stack(masks, axis=2)
```