

Master Thesis

MASTER'S DEGREE IN INDUSTRIAL ENGINEERING

***A deep learning approach for the detection
of
intestinal lesions using endoscopic capsules***

(ESAI)

REPORT

Author: Wilhelm Auffermann
Director: Raúl Benítez
Citation date: July 2020



ETSEIB
Barcelona School of Industrial Engineering

Abstract

Today, to diagnose possible lesions in the intestinal tract, we may perform an endoscopy. There is a less-invasive technique of endoscopy which consists of an ingestible camera that records a video of all its path through the gastroenteric organs. Then, it may be reviewed by a human and/or an automation detection method. The most common lesion is one called Angiodysplasia (Referenced as AD from now on) [1] and it's the target of our analysis. This diagnosis process can be tedious when performed manually as we are talking about hours of repetitive footage.

In this master thesis, we have developed a software capable of extracting the video images from the camera, pre-processing them, and then using them as training for our deep-learning architecture approaches. Our training setups have been trained whether to detect if an image is healthy or has the lesion.

That is, only 69% of AD are detected by experts during the review of a video. Moreover, blood indicator software presents a sensitivity of 41% and a specificity of 67% [2][3]. With our technique, we have performed a sensitivity of 91% and a specificity of 94% on our validation dataset.

These findings are of the upmost interest to the medical collective. With a graphic user interface setup, we could adapt this effective model into an application with which medical staff can detect lesions.

Summary

SUMMARY	5
1.1. Tables.....	8
1.2. Figures.....	9
2. GLOSSARY	11
3. PREFACE	13
3.1. Origin of the project	13
3.2. Motivation	13
3.3. Prerequisites.....	14
3.3.1. Raw video data	14
3.3.2. Excel template	14
4. INTRODUCTION	15
4.1. Project objective	15
4.2. Project pipeline	16
4.3. Project scope.....	17
4.4. Hardware specifications	18
5. RAW DATA EXTRACTION	19
5.1. Raw data structure	19
5.1.1. General structure	19
5.1.2. Raw video origin	21
5.1.2.1. Frames technical data	23
5.1.2.2. Audio source	23
5.1.2.3. Miscellaneous	24
5.1.3. Excel	25
5.1.3.1. Raw excel origin	25
5.1.3.2. Excel technical details	25
5.1.3.2.1 Tipo	26
5.1.3.2.2 Tam	28
5.1.3.2.3 LRConsens	29
5.1.3.2.4 Loc	30

5.1.3.2.5.....	LocT	30
5.1.3.2.6.....	Sang	32
5.1.3.2.7.....	Potencial	33
5.1.3.2.8.....	Endos	33
5.1.3.2.9.....	FrameE	34
5.1.3.2.10.....	TiempoID	34
5.1.3.2.11.....	SBI	34
5.1.3.2.12.....	FrameSBI	35
5.1.3.2.13.....	NFPSBI	35
5.1.3.2.14.....	Observations	35
5.2.	Image extraction from video	36
5.2.1.	Code for image extraction	36
5.2.1.1.	Function Image_extraction	37
5.2.1.2.	Function <i>find_image</i>	39
6.	IMAGE PRE-PROCESSING	45
6.1.1.	Update of General structure	45
6.1.2.	Images technical data	46
6.1.3.	Code for image pre-processing	47
7.	DEEP LEARNING PROCESSING	50
7.1.	Introduction to the processing	50
7.1.1.	Final data structure	50
7.1.2.	Images technical data	50
7.1.3.	Strategy	51
7.2.	MNIST model with training	52
7.2.1.	Model structure	52
7.2.2.	Training	53
7.2.3.	Results	55
7.3.	VGG16 pre-trained with feature extraction and classifier	56
7.3.1.	Model introduction	56
7.3.2.	Training	61
7.3.3.	Results	63
7.4.	VGG16 pre-trained with fully connected training	65

7.4.1. Model structure	65
7.4.2. Results	68
7.5. Interpretability	70
7.5.1. False positive LIME analysis	72
7.5.2. False negative LIME analysis.....	73
8. CONCLUSIONS	76
9. PLANNING AND RESOURCES	77
10. ECONOMICAL AND MATERIAL LIST	78
11. ENVIRONMENT	79
12. REFERENCES	80
13. ANNEX	83
13.1. Template for lesion classification (in Spanish)	83
13.2. Function Image_extraction	85
13.3. Function find_image	88
13.4. Code of MNIST model applied to lesion detection	96
13.5. Code of VGG16 pre-trained with feature extraction and classifier	99
13.6. Code of VGG pre-trained with fully connected training	103

1.1. Tables

Table 1 Glossary of statistical terms about result analysis [14]	12
Table 2 Hardware detailed specifications	18
Table 3 Codification of Tipo cell values	26
Table 4 Codification of Tam cell values	28
Table 5 Codification of LRConsens cell values	29
Table 6 Codification of Loc cell values	30
Table 7 Codification of Sang cell values	33
Table 8 Codification of LRConsens cell values	33
Table 9 Codification of SBI cell values	34
Table 10 Data split	54
Table 11 Data split	61
Table 12 Confusion matrix results and abbreviations	63
Table 13 Final results of the classification of validation data	64
Table 14 Confusion matrix results and abbreviations for VGG F.C.	68
Table 15 Final results of the classification of validation data for VGG F.C.	69
Table 16 Resources allocated for this project	78

1.2. Figures

Figure 1 Project scope representation.....	17
Figure 2 Data structure of the raw data	19
Figure 3 Snapshot of the folders defining each patient.....	20
Figure 4 Snapshot of a folder belonging to a patient	20
Figure 5 Illustration of an endoscopic capsule. [9].....	21
Figure 6 Frame of a video taken by an endoscopic capsule, showing AD lesion	22
Figure 7 Properties of the video format	23
Figure 8 Audio properties	23
Figure 9 Miscellaneous technical properties of the video	24
Figure 10 Heather of the template tool to write down detections	25
Figure 11 Excel Tipo example	27
Figure 12 Excel LRConsens example	29
Figure 13 Frame of a video taken by an endoscopic capsule, showing AD lesion	31
Figure 14 Format for LocT (areas highlighted in yellow).....	32
Figure 15 High level pipeline of the project.....	36
Figure 16 Frame of a video that could be the last frame of the video	40
Figure 17 First two lines of the code in Google Colab	41
Figure 18 Running Image_Extraction	44
Figure 19 Image from the output with a lesion.....	45
Figure 20 Data structure of the extracted data	46
Figure 21 Properties of the image	46
Figure 22 Properties of the File	47

Figure 23 Cropped image that still has some letters.....	48
Figure 24 Image of a lesion totally cropped and cleared of text.....	49
Figure 25 Data structure of the final pre-processed data.....	50
Figure 26 Properties of the image	51
Figure 27 Properties of the File	51
Figure 28 Model layers and tensor shapes.....	53
Figure 29 Accuracy evolution through the epochs.....	55
Figure 30 Sample of ImageNet images [12]	56
Figure 31 Visualization of an example of kernel trick. [16].....	57
Figure 32 Layers of VGG-16 (1 of 2)	59
Figure 33 Layers of VGG-16 (2 of 2)	60
Figure 34 Confusion matrix of the SVC classifier with the test features.....	63
Figure 35 Auxiliary model for LIME interpretability	66
Figure 36 Original image of a lesion (left) and LIME zones taken into account (right).	70
Figure 37 Lime analysis of a lesion (lesion highlighted in yellow)	71
Figure 38 False positive LIME representation (1 of 2)	72
Figure 39 False positive LIME representation (2 of 2)	73
Figure 40 False negative LIME representation (1 of 2).....	73
Figure 41 False negative LIME representation (2 of 2).....	74
Figure 42 False negative LIME representation.....	75
Figure 43 Gantt chart of the project.....	77

2. Glossary

Angiodysplasia (AD)	Small vascular malformation in the digestive tract
Scikit-learn (also known as sklearn)	Machine learning library that supports various classification, regression and clustering, in our case, support vector machines.
Keras	open-source neural-network library in Python that is optimized to work with TensorFlow software
TensorFlow	library for dataflow and differentiable programming across a range of tasks, developed by Google.
Feature	The features are the elements of your input vectors. If you were using a neural network to classify people as either men or women, the features would be things like height, weight, hair length etc.
Neural network	Computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses.
pattern recognition	The act of identifying patterns within previously learned data. This can be carried out by a neural network even in the presence of noise or when some data is missing.
epoch	One complete presentation of the training set to the network during training.
input layer	Neurons whose inputs are fed from the outside world.
layer	A group of neurons that have a specific function and are processed. The most common example is in a feedforward network that has an input layer, an output layer and one or more hidden layers.
neuron	A simple computational unit that performs a weighted sum on incoming signals, adds a threshold or bias term to this value to yield a net input, and maps this last value through an activation function to compute its own activation. Some neurons, such as those found in feedback or Hopfield networks, will retain a portion of their previous activation.
output neuron	A neuron within a neural network whose outputs are the result of the network.
training set	A neural network is trained using a training set. A training set comprises information about the problem to be solved as input stimuli. In some computing systems the training set is called the "facts" file.

weight

In a neural network, the strength of a synapse (or connection) between two neurons. Weights may be positive (excitatory) or negative (inhibitory). The thresholds of a neuron are also considered weights, since they undergo adaptation by a learning algorithm.

Timestamp

Localization in time of a lesion (For example, 03:22:21) (Hour, minutes, seconds)

[7]

		True condition			
		Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition	Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR-}}$
		False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

Table 1 Glossary of statistical terms about result analysis [14]

3. Preface

3.1. Origin of the project

A medical group of Hospital del Mar led by Dr. Marco Antonio Álvarez is currently manually reviewing videos of the digestive tract. They are assisted with a simple color-detect software to diagnose AD lesions. There is an urge to attempt to automate this process improving the detection rate. A relevant part of the project is feasible today and not before due to advances in machine learning, processing power, such as Google server's public availability. The theoretical basis of this project comes from the creation of convolutional networks for image detection, miming the section in the visual cortex of the brain that is responsible for visual processing [4].

A first project has already been realized about this topic, using classic machine learning [5]. This project consisted in segmentation of the images, and then attribute extraction and classification with a supervised classifier. In our case, we have used a novel strategy: deep learning. [6]. One important topic about deep learning versus classical machine learning is that depending on the architecture chosen and the training, it can be better or worse than the traditional with segmentation and attribute extraction.

3.2. Motivation

The motivation is to achieve an added value to the sanitary system to free resources and speed-up processes, and therefore maybe save lives. A team of one senior medical doctor and two interims have been creating the annotations that have made possible this project.

3.3. Prerequisites

The main prerequisite of this project or a continuation of it consists in:

3.3.1. Raw video data

Raw video data from capsules going through the digestive tract. In our case we had 35GB of this data, providing us with a limited amount of AD lesion pictures and a great amount of “healthy” pictures.

3.3.2. Excel template

Excel template containing the parameters to be able to extract the lesions. The template can be found in section 13.1.

4. Introduction

Medicine has always embraced new technologies to deliver powerful results, such as Electro-magnetic resonance to revolutionize traumatology diagnostics, ultrasound technologies to eliminate specific cumulus in the organism, or also drug-delivery systems using nanoscale materials to bring compounds where they are needed, just to name a few. [8] In this work we present a small step forward on a specific topic.

4.1. Project objective

The objective of this project is to parameterize a set of deep learning and classification methods using python to be able to differentiate an AD lesion from healthy tissue, based on the samples extracted from raw gastrointestinal videos for training and for testing.

4.2. Project pipeline

We will be starting our project pipeline from a raw video of an endoscopic capsule and an excel sheet with times of the lesion (time is also hardcore written in the video and the time of the video is not continuous, which has increased the difficulty of the pre-processing).

The pipeline is described below:

Preprocess to gather useful data:

- 1- Invent a routine that looks for the lesions that experts have identified in the videos and takes a picture of it. The complexity is high because aside from performing several snapshots depending on the expert write-downs, it “looks” for written data on each picture.
- 2- Invent another routine to gather pictures without lesion. This is done by searching timestamps through normal distribution, considering possible conflicts with lesions, and possible duplicates.
- 3- Then, a pre-process routine has been developed to delete the time written in the image and take out some letters referring to parameters. This is deeply important to help the AI to focus on the picture and for optimization purposes.

Machine learning and classification:

- 4- After gathering and pre-processing the data, we have tried several machine learning setups that will be detailed. We used a VGG16 network pretrained on ImageNet to use its attributes to perform transfer learning into a support vector machine. We also checked what the AI was “looking at” to take the decision.

4.3. Project scope

The limits of the project are simple and are defined in a pipelined shape, in Figure 1. Upstream we have the raw data we have introduced and downstream the results and metrics from our work. Details on formats, the structure of the data, and several subsequent details will be defined later.

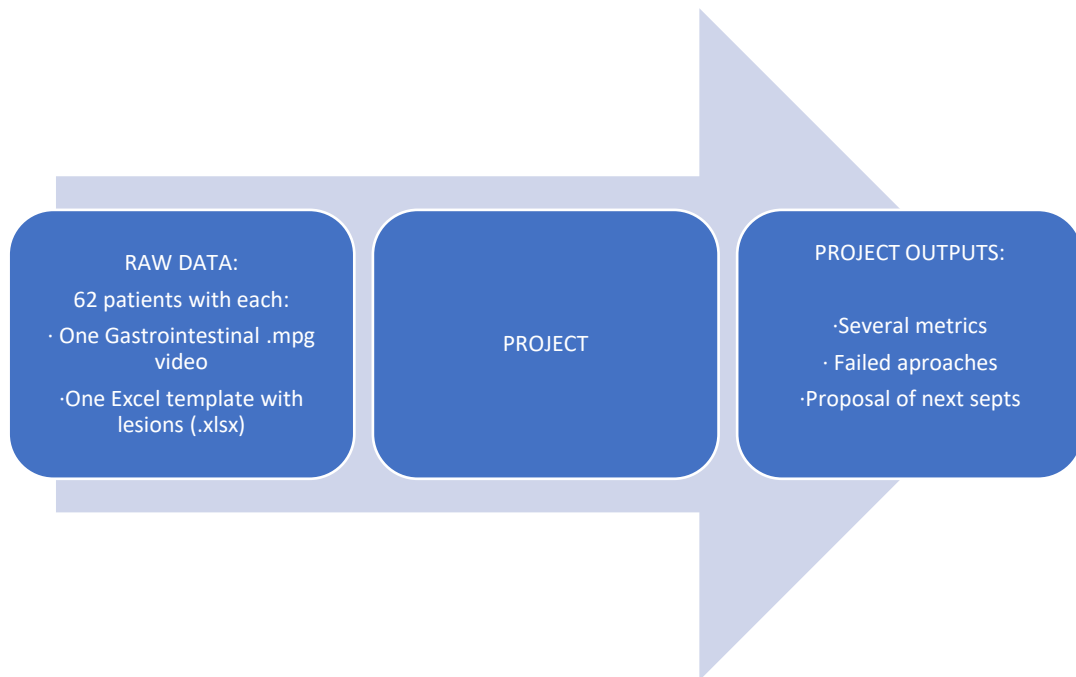


Figure 1 Project scope representation

4.4. Hardware specifications

To develop this project, we have used Google Colab servers and environment [10]. The operating system is Linux and the Colab instances are in Python 3.6. You will find the detailed specifications in the Table 2 below.

	Environment in TPU	Environment in GPU	Environment without GPU nor TPU
RAM	35,5 GB DDR4	35,5 GB DDR4	35,5 GB DDR4
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz	Intel(R) Xeon(R) CPU @ 2.20GHz	Intel(R) Xeon(R) CPU @ 2.20GHz
TPU	Variable upon need. Up to (64 GiB)	-	-
GPU	-	NVIDIA Tesla P100- PCIE-16GB	-
STORAGE	100GB	100GB	100GB
OS	Linux-x86_64 Colab	Linux-x86_64 Colab	Linux-x86_64 Colab

Table 2 Hardware detailed specifications

5. Raw Data extraction

In this section, we will be technically detailing the original structure of the data and all steps taken to pre-process it, for it to be ready for the Machine learning phase of the project.

5.1. Raw data structure

5.1.1. General structure

The data is gathered in a directory named "Videosplantilla" which contains 32.6 GB (35,012,957,629 bytes) of data. It could also be done with n folders with n from 1 to infinite, given enough hardware support.

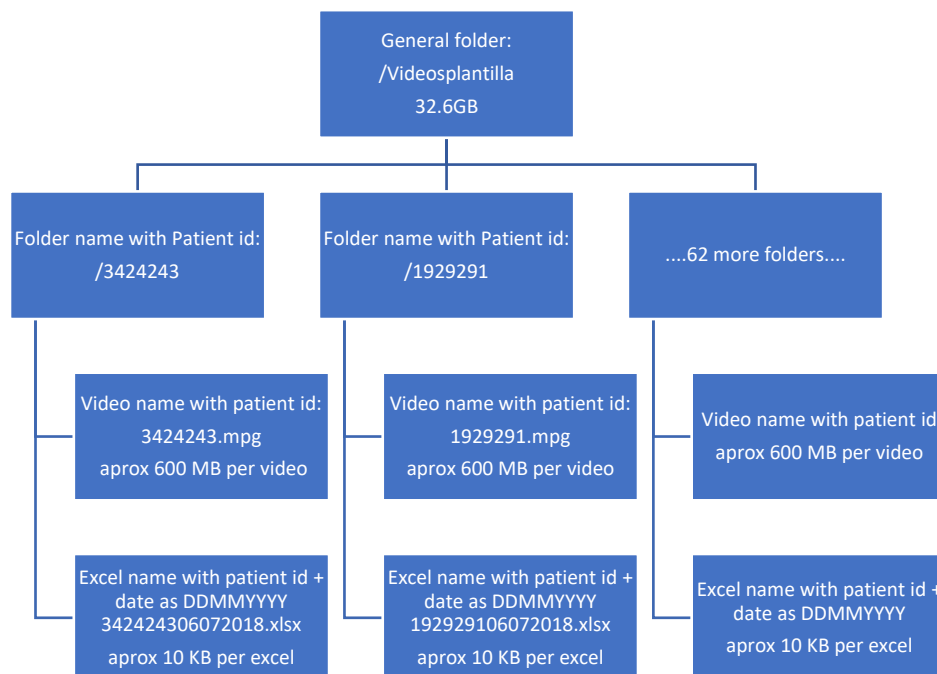


Figure 2 Data structure of the raw data

Name	Date modified	Type	Size
25340	21-Apr-20 14:19	File folder	
58324	23-Apr-20 09:24	File folder	
72318	24-Apr-20 16:04	File folder	
77762	21-Apr-20 14:19	File folder	
78721	24-Apr-20 16:04	File folder	
109040	24-Apr-20 16:04	File folder	
121788	21-Apr-20 14:19	File folder	
192734	21-Apr-20 14:19	File folder	
214619	21-Apr-20 14:19	File folder	
228088	21-Apr-20 14:19	File folder	
237206	24-Apr-20 16:04	File folder	
242259	22-Apr-20 00:12	File folder	
256661	24-Apr-20 16:04	File folder	
266096	22-Apr-20 08:47	File folder	
268610	24-Apr-20 16:04	File folder	
270198	22-Apr-20 12:37	File folder	
271838	22-Apr-20 14:22	File folder	
303130	21-Apr-20 14:19	File folder	
324430	24-Apr-20 16:04	File folder	
326581	21-Apr-20 14:19	File folder	
351565	24-Apr-20 16:04	File folder	
396041	22-Apr-20 15:22	File folder	
400380	24-Apr-20 16:04	File folder	
416200	21-Apr-20 14:19	File folder	

Figure 3 Snapshot of the folders defining each patient

Name	Date modified	Type	Size
303130.mpg	19-Nov-19 16:51	VLC media file	560,353 KB
30313014022018.xlsx	10-Feb-20 15:45	Microsoft Excel W...	9 KB

Figure 4 Snapshot of a folder belonging to a patient

5.1.2. Raw video origin

As synthesized, the raw video comes from an endoscopy capsule as shown below in Figure 5.

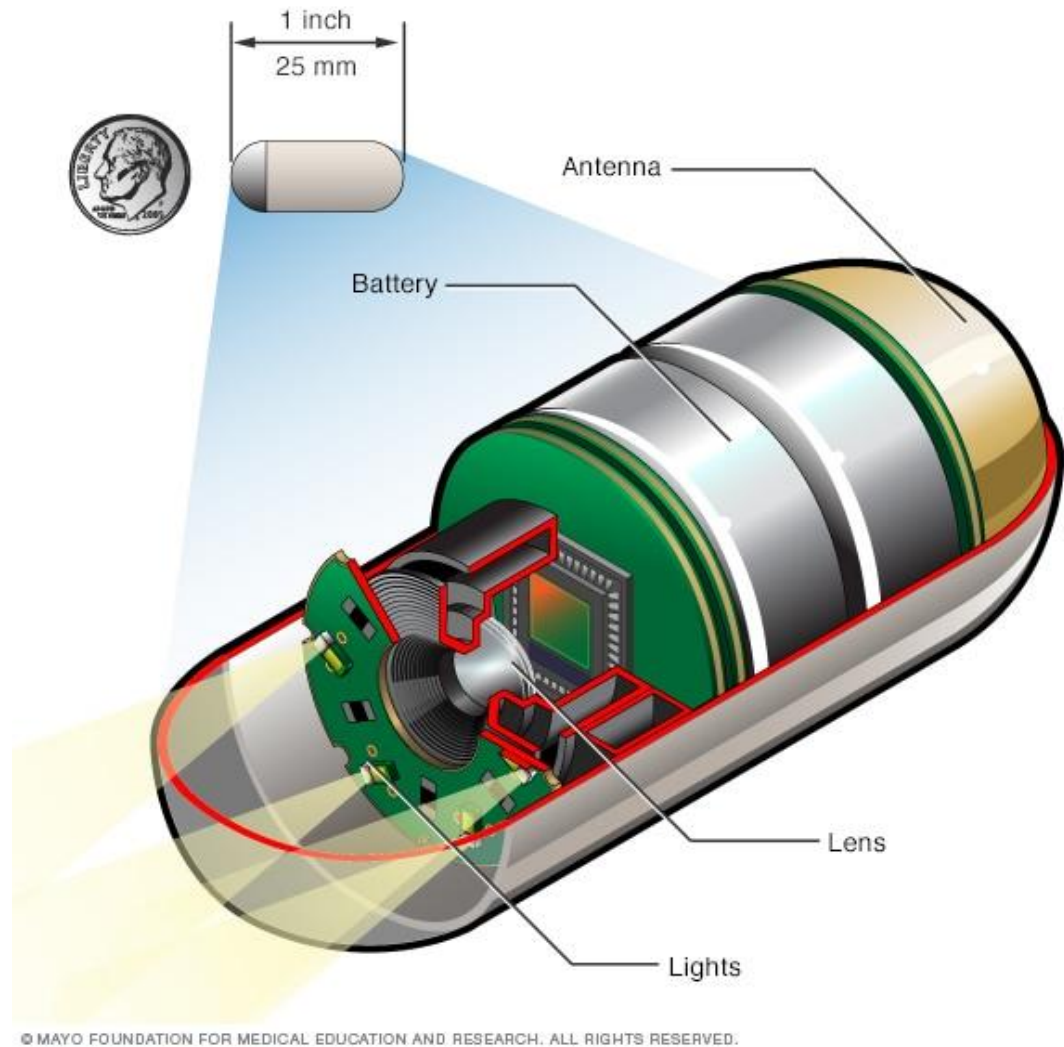


Figure 5 Illustration of an endoscopic capsule. [9]

This capsule has several subroutines that optimize the video recording with minimum resources.

For example, the camera stops saving an image if the camera gets temporarily stuck and the image is the same. This feature is interesting to save battery life. Nevertheless, this results in the complication that the video time is not real-time, but rather just an arbitrary time resulting from the addition of pictures that are not in a “stuck” situation. The real-time of a frame cannot be deducted from the time of the video.

Due to the impossibility to deduce time from the video itself, the onboard software creates a black frame from 512x512 up to 576x576 to write down the metadata.

The example in Figure 6 below shows us an example of the metadata. We can see the date (06 Feb 18), the camera model (on the bottom), the operating mode (LSR), and finally the relative time from the beginning of the video (01:00:15) (the format is "HH:MM:SS"). This last data will be the most important because we will have to compare it to the experts' information provided in the excel.



Figure 6 Frame of a video taken by an endoscopic capsule, showing AD lesion

5.1.2.1. Frames technical data

In Figure 7 below we can see detailed data. The most interesting info could be the length of the video, shy more than 47 minutes. This is a good example to understand the economy settings of the endoscopic capsule because we can see in Figure 6 above that the frame is at a timestamp of 1 hour. That could be for example minute 30 in the actual video. In 5.2 Image extraction, we will discuss this fact in depth.

Video	
Length	00:47:45
Frame width	576
Frame height	576
Data rate	1800kbps
Total bitrate	1800kbps
Frame rate	25.00 frames/second

Figure 7 Properties of the video format

5.1.2.2. Audio source

There is no audio source.

Audio	
Bit rate	
Channels	
Audio sample rate	

Figure 8 Audio properties

5.1.2.3. Miscellaneous

Finally, we can see the composition of the video in Figure 9.

File	
Name	303130.mpg
Item type	VLC media file
Folder path	
Size	547 MB
Date created	21-Apr-20 14:14
Date modified	19-Nov-19 16:51
Attributes	A

Figure 9 Miscellaneous technical properties of the video

5.1.3. Excel

As explained in section 5.1.1 General structure, each patient has one excel and one video. The excel is a tool used to write down timestamps of anomalies seen by the endoscopist in the video. Each anomaly is numbered from 1 to x from the top-left corner, as shown in Figure 10 below, and the heather contains the parameters written down by the medical team.

5.1.3.1. Raw excel origin

The filling of this file is done by the medical staff. The technician writes down any findings when reviewing the video. This is a huge time expenditure.

5.1.3.2. Excel technical details

As introduced, we will define the parameters that can be written in the excel. It will be explained after, but it is imperative for the functioning of the software, that the inputs are exactly as defined. Any deviations will automatically cause the software to work improperly (stay in blind loop, copy images not as expected, return format errors, etc.) If the fields used by our software are not filled in a specific row number (for example for $N=3$), the program will understand that there are no more lesions for this patient (it will stop iterating at $N=2$ and then jump to the next patient).

N	Tipo	Tam	LRConsens	Loc	LocT	Sang	Potencial	Endos	FrameE	TiempoID	SBI	FrameSBI	NFPSBI	QV	FrameQV	NFPQV
1																
2																
3																
4																

Figure 10 Heather of the template tool to write down detections

As shown in Figure 10 above, we can see all the parameters that can be written down by the endoscopist.

5.1.3.2.1 Tipo

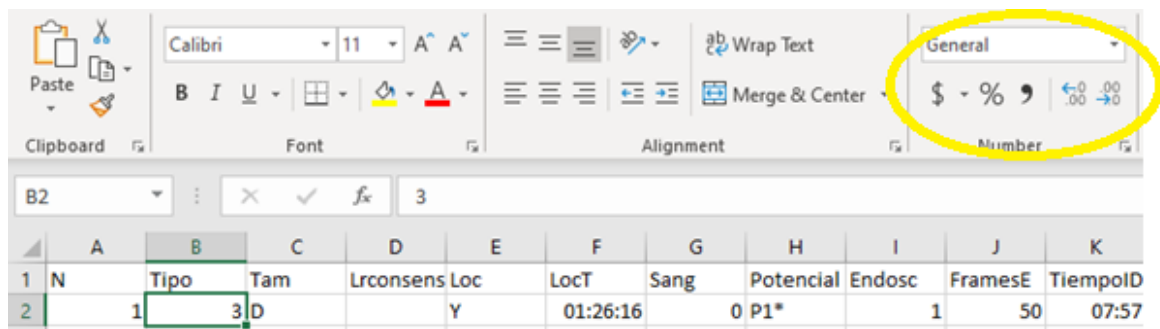
“Tipo” refers to the kind of lesion we have discovered. The range of this cell value is from 1 to 9 (both included). This can be seen in Table 3 below.

Cell value coding	<i>Original meaning in template</i>	Translated meaning
1	Lesión roja del consenso	“Consensus” red lesion
2	Úlcera	Ulcer
3	Erosión	Erosion
4	Afta	Canker sores
5	(vacio)	(empty)
6	Tumor	Tumor
7	Pólipo	Polyp
8	Sangre roja	Red blood
9	Sangre oscura	Dark blood

Table 3 Codification of Tipo cell values

We do not consider lesions different than 1 in our project. That is, the software will ignore the line with Tipo different than one. Tipo must be only an excel integer.

To understand the format needed, you can see the example shown in Figure 11 below, in with “General” cell mode and typing “1” with the keyboard. If anything is changed, it will probably not work as expected. For example, adding any decimal or a coma for thousands separation, will not work. If this field is not completed for the row, the program will understand that there are no more lesions for this patient.



	A	B	C	D	E	F	G	H	I	J	K	
1	N	Tipo	Tam	Lrconsens	Loc	LocT	Sang	Potencial	Endosc	FramesE	TiempoID	
2	1	3	D		Y	01:26:16		0 P1*		1	50	07:57

Figure 11 Excel Tipo example

5.1.3.2.2 Tam

“Tam” field represents the size of the lesion found. The range of this value is shown in Table 4 below. We do not take it into account in the software.

Cell value coding	Size [mm]	Size association
D	1-3	Minuscule
P	3-5	Small
M	5-9	Medium
G	≥ 10	Big

Table 4 Codification of Tam cell values

5.1.3.2.3 LRConsens

“LRConsens” is filled only if the “Tipo” field is at value of 1 (That is, if we have “Lesión roja del consenso”). As introduced in 4.3 Project scope, the scope of our project consists in analyzing the “Tipo1” lesions of Angiodisplasia (AD).

Cell value coding	Association
AD	Angiodisplasia
EP	Erithematous patch
RS	Red spot
PB	Phlebectasia

Table 5 Codification of LRConsens cell values

Tipo must be only written text in excel. To understand the format needed, you can see the example shown in Figure 12 below. . The cell is in “General” mode and we type “AD” (without brackets) with the keyboard.

	A	B	C	D	E	F	G	H	I	J	
1	N	Tipo	Tam	Lrconsens	Loc	LocT	Sang	Potencial	Endosc	FramesE	Tien
2		1	3 D		Y	01:26:16	0	P1*		1	50
3		2	1 M	AD	Y	05:21:07	0	P2		1	15
4		3	1 M	AD	I	07:23:53	0	P2*		1	7
5											
6											

Figure 12 Excel LRConsens example

If anything is changed, it will probably not work as expected. For example, adding any sign or equation, will not work. If this field is not completed for the row, the program will understand that there are no more lesions for this patient and the code will not work properly.

5.1.3.2.4 Loc

“Loc” field represents the position of the lesion in the digestive tract. We do not take it into account in the software.

Cell value coding	Association
D	Duodeno
Y	Yeyuno
I	Ileon

Table 6 Codification of Loc cell values

5.1.3.2.5 LocT

This field represents the localization in time shown in the video, referred to as a timestamp from now on. The format of this cell is in excel DateTime and has been manually filled. This parameter is key for the project and will be later processed into python DateTime. We can see in Figure 13 below in yellow, what the operator sees. The operator is watching the video and when he sees a lesion, he identifies the time in the “LocT” field.



Figure 13 Frame of a video taken by an endoscopic capsule, showing AD lesion

It is very important to write down time values that are between the beginning of the video and the end of the video. If a time value is outside that range, the program will not work properly.

Moreover, “LocT” must be exactly written as described in Figure 14 below. If anything is changed, it will not work as expected. For example, adding any decimal or a coma for thousand separation or changing the cell type, will result in malfunctioning. If this field is not completed for the row, the program will understand that there are no more lesions for this patient and will even go to error if only some columns are filled and others not.

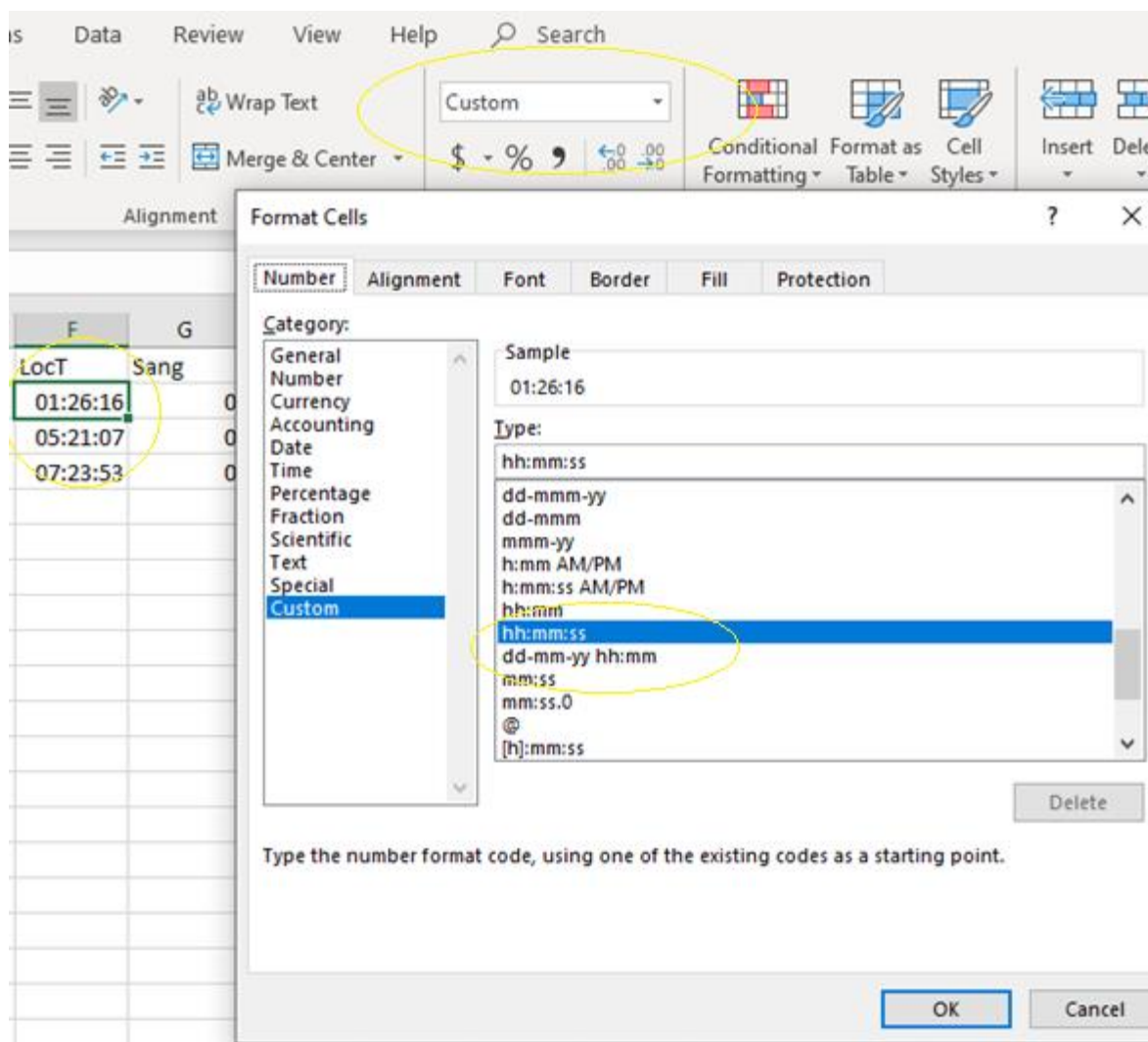


Figure 14 Format for LocT (areas highlighted in yellow)

5.1.3.2.6 Sang

This is a binary field to define the presence or absence of blood as defined in Table 7 below. We do not take it into account in the software.

Cell value coding	Association
0	Absence of blood
1	Presence of blood

Table 7 Codification of Sang cell values

5.1.3.2.7 Potencial

This is a filed currently in revision. We do not take it into account in the software.

5.1.3.2.8 Endos

This cell defines wether the endoscopist has detected or not the lesion, as defined in Table 8 below.

Cell value coding	Association
0	Not seen by endoscopist
1	Seen by endoscopist

Table 8 Codification of LRConsens cell values

5.1.3.2.9 FrameE

“FrameE” is the number of positive frames detected by the endoscopist. This parameter has high importance in the pre-processing phase because the software will collect as many different frames as the endoscopist has written down. This number is an integer.

5.1.3.2.10 TiempID

This is the time of exploration in the small intestine. We do not take it into account in the software.

5.1.3.2.11 SBI

This is a binary field whether the lesion has been detected by an existing commercial software named SBI. We do not take it into account in the software.

Cell value coding	Association
0	Not detected by SBI
1	Detected by SBI

Table 9 Codification of SBI cell values

5.1.3.2.12 FrameSBI

The number of frames detected by the commercial software (It is named FrameSBI but can contain frames from QV software). We do not take it into account in the software.

5.1.3.2.13 NFPSBI

The number of false positives of the SBI or QV software. We do not take it into account in the software.

5.1.3.2.14 Observations

There are minor inconsistencies between the template and the filled data, but only referring to existing software for detection that is out of the scope of the project. Accordingly, those minor differences do not affect the relevancy and accuracy of the data provided.

5.2. Image extraction from video

This section will technically define and guide the steps to perform the image extraction of the relevant frames of the video. As you can see in Figure 15 below, we are in the first step of our project.

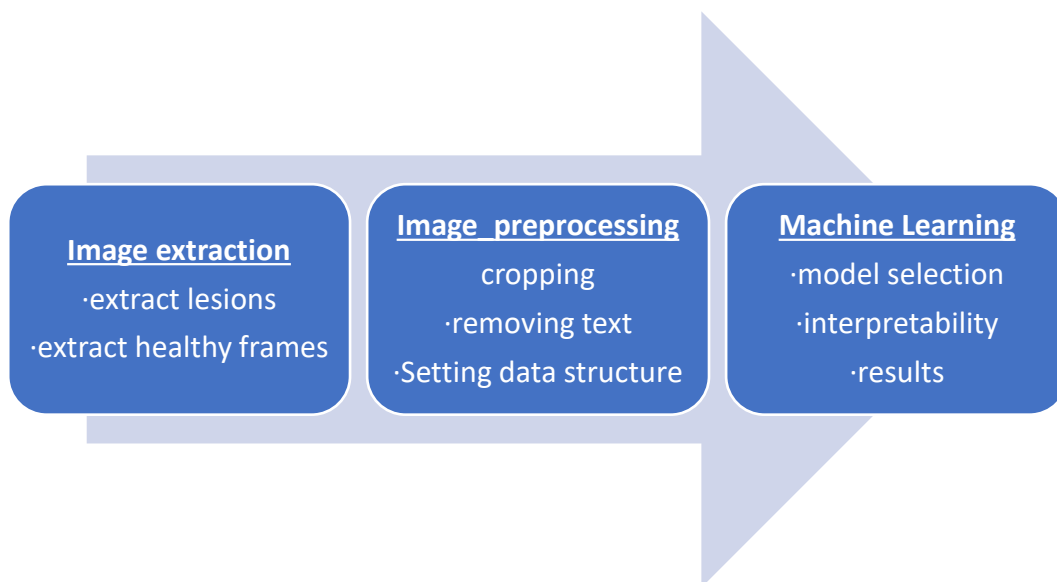


Figure 15 High level pipeline of the project

5.2.1. Code for image extraction

Note: All the code is documented in GitHub and the annex. You can find it in 13 ANNEX and for the following code, at:

[https://github.com/raulbenitez/deep_endoscopy/blob/master/Code for image extraction.ipynb](https://github.com/raulbenitez/deep_endoscopy/blob/master/Code%20for%20image%20extraction.ipynb)

The objective of this code is to Create a .jpeg database of the frames responsible for the health deficiency and a database of non-deficiencies from the same patient, taken randomly and in a balanced number. The input is the folder path of the main folder defined in 5.1.1 General structure, and the output can be described as every relevant picture signaled by the medical experts.

Hereunder we will define the two key functions that execute this task: Image_extraction and find_image.

5.2.1.1. Function Image_extraction

The function is written in annex 13 ANNEX, specifically 13.2, with extensive comments.

Image_Extraction is the main function for the extraction, and it subsequently calls the secondary function find_image. It has as sole input the main directory with all the patient folders as described in 5.1.1 General structure. You can see the call below.

```
In [0]:
general_patients_folder='/content/sample_data/Videosplantilla'
Image_Extraction(general_patients_folder)
```

It first performs a walk of every patient folder.

For every folder, it walks through its files and checks whether the filename read is an excel. When the software finds the excel from that patient, we are ready to extract the data that we will be needing. It saves the excel data in a “pandas” instance into the RAM, to be able to work with it easily. Pandas is a utility that can read excels and operate with the data in a matrix shape, that supports indexing and high-level operations (like cell-wise multiplying).

Although all the info from the excel is extracted, we will only use a few of the information. For every info we gather, we will create a verbal-friendly variable that represents the important value we are extracting. The delay on the code is acceptable given that we had only 80 patients. We then create the following variables from the important data of the patient:

Timestamps: It is a pandas column with the timestamps of the lesions

Timestamps_tipo: creates a panda's column of Booleans that are "True" if we have Type 1 lesions, the scope of our project. Otherwise, the field is "False". If the field isn't filled, we will probably have an execution error at this step.

Tipo_lesion: creates a pandas column of the type of the lesion, it is used for naming pictures once we have extracted them.

frames_detected: This variable contains a pandas column with the number of frames to take for every lesion. As sometimes this number is in the field FramesTotales and sometimes it is in FramesE, we will have the code check both and take the one that is filled. If both are filled, we will take the FramesTotales values.

Nevertheless, we are interested in having the “Tipo 1” lesions counted, not anything else. Therefore, we execute the multiplication of `Timestamps_tipo` with `frames_detected`. We then convert this result to a list to avoid errors.

Now that we have all we wanted from the excel file, we only need to find the video and then retrieve the images we need.

We walk again the files of the patient, to find this time the .mpg video file. Once we have it, we are ready to take as many pictures as the metadata in the excel tells us to. We will open a while condition, in which we will check that we are still in the index of the lesions. Each cycle, the i variable will be added 1, to symbolize the next lesion. Starting with $i=1$ (pandas has the first index as 1, and lists have first index 0, therefore you will see in some cases a -1 subtraction in the index). With $i=1$, we will be retrieving the images corresponding to the first lesion. We will assume it is type 1, otherwise, the program would pass, because of the condition just under the while (that compares the number of images to take, that would be 0 in case of a different kind of lesion).

If we have, for example, a Type 1 lesion with `FramesE=12` and `Tipo_lesion= “AD”`, we will need to retrieve 12 frames of the lesion, and 12 random healthy frames from that same patient. We will first save the 12 frames of the lesion, calling `find_image`. Once we are done, we will call again `find_image` on “random” mode, to take the 12 random healthy frames. The technical details of the operation of `find_image` will be explained in section 5.2.1.2 Function *find_image* below.

We will now finish describing our primary `Image_extraction` function. Once `find_image` has been executed for this lesion and again for the healthy pictures, we will simply add +1 to our lesion index variable i , and then continue with the next lesion written down by the medical staff. Once we are finished for this patient, all walk functions end their indexed steps, except the first one that looks for patient folders. Then, the cycle is repeated.

For every folder, it first consults the excel file to retrieve how many pictures of which lesion should we extract, and then calls our first function `find_image` every time it needs an image of a lesion or an image of healthy tissue.

5.2.1.2. Function *find_image*

Find_image is a function that seeks timestamps in a video. When it detects one, it saves the picture matching the timestamp and the subsequent next frames to a new folder in the folder of the patient.

A calling of this function could be the one below

```
def find_image(video file, full path, lesion time, Tipo lesion, frames detected, random frames,
k, rand if true)
```

```
written1=find_image(video path, full path, Timestamps,Tipo lesion.tolist(),frames detected.tolist()
[i-1],0,i,False)#for debugging, Timestamps[i],Tipo lesion[i], i)
```

video_file contains the name of the video file

full_path contains the full path of the mpg video

lesion_time is the time of a Timestamp. The main function has a pandas column containing them, and each time the function is called it takes one of them using index i-1

Tipo_lesion is the kind of lesion, from "Tipo" in the template

frames_detected is the number of frames detected by the endoscopist in case we are seeking lesions

random_frames is only used when the function is in "finding randoms" mode, (see last variable "rand_if_true"). This variable exists because we need to specify the number of random frames the program must take in case it has to take random frames. We made it different from frames_detected, to be able to take as many healthy images as we want because we have a huge pool on which to take them.

K is the index i (which lesion are we analyzing in the excel of the patient)

rand_if_true is a Boolean to activate the finding of lesions(False) or finding of random healthy images (True)

To recap our current situation and continue our example (a Type 1 lesion with FramesE=12 and Tipo_lesion= "AD",) we will now need to call this function twice. Once in lesion mode, to find the lesion and save that image and the next 11 frames, and then call it again to find 12 random healthy

images that do not match any lesion of this patient, according to medical expert data in the excel.

The function starts with two auxiliary operations. We have a huge handicap related to the location in the timeline of the video. Because this camera software stops video recording when not advancing the digestive tract, we don't know at which time the video will start or will finish, or what will be the duration of it, in seconds of video neither in real hours. We will need to retrieve some of this information, to use it in the following lines.

To tackle this issue, we will find which hour, minutes, and seconds define the beginning and the end of the video. For this function to work, the raw excel data must make sense (for example, that the lesion is inside the video!) If we input a lesion happening at 23:19:12 when the video starts at 00:21:12 and ends at 08:22:12, the code will not work.

To open the video file, we will create a video object instance, which comes from cv2, specifically the VideoCapture library. The sole input is the video path. Additionally, we have found that 3 of the videos are corrupted and cannot be opened by VideoCapture nor standard video software. In those cases, the program will not run properly.

Once we have it, we will set the video at the end to take the last picture, using the "set" attribute. We will then read the image of the end and store it in the RAM with the variable "image". Let's say our video has the last time at 01:00:15, then the picture we will have will be Figure 16 below, in a NumPy array format and shape as (572,572,3).



Figure 16 Frame of a video that could be the last frame of the video

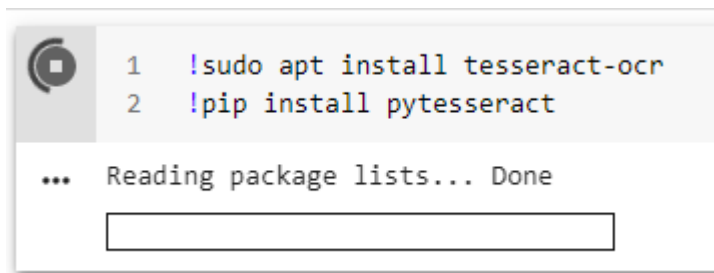
As we can see, we will need to write in RAM the time HH:MM:SS written in the yellow circle we

have drawn to illustrate. We will be doing that by first cropping the image to take only the zone emphasized in yellow. Then, we will run an optical character recognition software named Pytesseract that we will have previously installed in our machine with the following commands (in case of Colab):

In [1]:

```
!sudo apt install tesseract-ocr  
!pip install pytesseract
```

Which gives us the following execution in *Figure 17*:



```
1 !sudo apt install tesseract-ocr  
2 !pip install pytesseract  
... Reading package lists... Done
```

Figure 17 First two lines of the code in Google Colab

Finally, we get the following output:

```
Successfully installed pytesseract-0.3.4
```

Hereunder, we write down the libraries used for this process and a brief description of their need or what they are used for.

In [0]:

```
import pytesseract #This is image recognition software that we will use to  
"read" the time of a picture while we go through them all.  
import shutil #This offers a number of high-level operations on files and  
collections of files  
import os #This offers a number of operations for file management  
import random #This directory will be used to generate random data for au  
xiliary purposes  
from PIL import Image #This directory will be used to manage pictures an  
d to visualize them.  
import numpy as np #mathematical matrix utilities used for holding pictur  
es in (576, 576,3) shape  
import cv2 #Directory for file management  
import pandas as pd #Directroy for extraction of excell data  
import time # Directory to manage time data  
import math #Directory to use mathematical functions
```

```
from pytesseract import image_to_string #This OCR function converts an im
age to a string of the OCR characters detected. It will be used to read t
he date of every picture
import datetime #Format to operate the data collected from excel sheet and
from the OCR recognition.
import matplotlib.pyplot as plt #Plotting library
i=0#initialization of counter
```

Once we have read the written numbers, the OCR software stores them in a text format, in this case, a string like "HH:MM:SS".

We then convert it to DateTime type with a function that converts a string to a DateTime object.

Now that we have the DateTime object of the last frame, we will do the same to take the first frame, setting the video object to the beginning.

Once we have both ends, we are ready to start looking for lesions. We will start a while instance that will help us to walk frame by frame the video and read with OCR code in real-time. Just under the while, we have the key condition of the function. It states that if the read DateTime is equal or greater than the lesion time written down by the medical staff, we will start to save that picture to the DISK. Nevertheless, let's ignore this while and come back to it later.

In the first instances of the while loop, the code is likely not going to enter this condition, because it will be far from the lesion. Let's say that our lesion is at 00:34:02 and we start at 00:12:01. We can see that each while loop contains a read() instance of the video object. This read instance takes the next frame and writes it in the variable image.

Then again, as we did before, we crop the image and we extract the DateTime with OCR software. One additional safety we have here is that one out of tens of thousands reads, the OCR software fails in a specific frame, in that case, the safety we added is to keep trying to read it for the next frame, and it always has solved the issue.

In this loop, to safely accelerate the process, we have stated two while conditions that avoid using OCR for some frames if we are safely far from the video. This is useful because OCR is very time expensive. Once the OCR detects an image matching the lesion, the condition we introduced before is filled, and we start to execute the following steps of the conditional instance. Now, just to recap, we have found an image that is the same time that the operator wrote in the excel. What we need to do is to save as many frames as he specified.

We control how many different frames we will be taking using a simple index and a while with the condition of having the index smaller than the frames needed. We then read images and compare them with previous images to see if it is the same image. If it is a new genuine frame, it is written in a folder. The name of the file contains the ID of the patient, the lesion time, and then the time in seconds of the lesion, and finally an index of the frame number of that lesion.

Filename: patient_78721tipo1lesion_EPtime_in_secplus_frame_last2digits100144200.jpg

Code:

```
cv2.imwrite("patient_" + str(patient_folder) + 'tipo'+ '1'+ 'lesion_' + str(Tipo_lesion[k-1]) + 'time_in_sec'+ 'plus_frame_last2digits'+ '%d.jpg' % ((lesion_time[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second)*100 + j*100) , image_raw) #writes the image
```

Interpretation of the ending code of the filename:

The first digits are the number of the frame and the following ones are the lesion time in seconds. It is easy to spot because the folder will have the same time in seconds. Thus, it is easy to differentiate the number of the frame.

Once we copied to the disk all the frames of this lesion, we will continue to execute Image_extraction and this time we will be calling find_image in random mode, to save to the disk some random images.

To launch it, we specify rand_if_true=True and state the number of random images to take. We also specify the number of frames in this lesion, to have a balanced set at the end. The function then jumps to the random mode and first creates a directory to save all the frames. There is a very small probability to take 2 times the same healthy frame, we assume this case and we foresee that it will not be significant, given the number of pictures to take vs the number of frames available.

We start by finding the first and last times of the video, identically as in lesion mode. Once we have them, we create a proposal of a normal randomized distribution returning as many timestamps as needed. We will be analyzing if this proposal is indeed inside the video and also that is safely far from any lesion. Once we have checked that, we have obtained a verified set of random timestamps perfect to seek.

We walk the list of random timestamps and when we find each one we proceed to save the image

to the folder we created. The function finishes once we have walked all the proposed random images. We also added safety in case of a rare false read by the OCR software and also added speeding of the function if we are far from the timestamps (that is, avoiding using OCR for some iterations).

As we discussed, the pipeline is that the Image_Extraction function calls find_image for each lesion of each patient. Processing time could be several hours, for example between 6 and 50 hours depending on specifications.

The input we give is:

```
general_patients_folder='/content/sample_data/Videosplantilla'  
Image_Extraction(general_patients_folder)
```

And then we can see an instance of the output while running in Figure 18 below.

```
❏ Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
tesseract-ocr is already the newest version (4.00~git2288-10f4998a-2).  
0 upgraded, 0 newly installed, 0 to remove and 25 not upgraded.  
Requirement already satisfied: pytesseract in /usr/local/lib/python3.6/dist-packages (0.3.3)  
Requirement already satisfied: Pillow in /usr/local/lib/python3.6/dist-packages (from pytesseract) (7.0.0)  
668891Seeking time...00:03:47 to 03:28:58 lesion_is:02:51:37  
668891Seeking time...00:03:47 to 00:07:44 lesion_is:04:50:45  
930777Seeking time...00:13:31 to 07:30:09 lesion_is:00:51:00  
930777Seeking time...00:13:31 to 06:17:25 lesion_is:01:00:10  
930777Seeking time...00:13:31 to 08:17:35 lesion_is:04:43:53
```

Figure 18 Running Image_Extraction

A manual review of this output has been done to remove very few false pictures between good ones. This is due to the imprecision that the medical staff sometimes do. The mistake consists on creating only one timestamp where the preferent approach would have been to create more timestamps with less duration. The typical example is the case where we have 10 frames in a row, but the first 6 have lesion, the 7th has no lesion and the 8th, 9th and 10th have lesion. In this case, the right way to annotate would be to create one first timestamp of 7 frames and another of 3 frames. Instead, we see sometimes one only timestamp with 10 frames.

6. Image pre-processing

We now have extracted images such as the one shown below and we are going to tackle Image pre-processing. It is key to have a good basis for our deep learning operations. The following image visible in Figure 22 below has some serious training flaws that must be solved.



Figure 19 Image from the output with a lesion

The main flaws can be found if we think about what can distract the machine learning algorithms. The first flaw would be to show the hours minutes and seconds to the artificial intelligence, because it could easily associate this date with lesions, rather than the lesions themselves. For example, imagine that $\frac{3}{4}$ of the lesions happen in the last 20 minutes of video, due to biological reasons. Then the AI would take that into account, and we would have a hard distortion.

A similar deduction can be inferred for the date, and the mode, in this case, "RMM". Finally, the name of the camera wouldn't probably affect if they are all the same, but we will erase it in case we share pictures with other cameras. Plus, reducing the size/shape of the image is an useful optimization of resources.

6.1.1. Update of General structure

Once we have the healthy images and the ones with the lesion, we will need to re-organize

them to be pre-processed. First, we re-organize them as in Figure 20 below searching by the image in the explorer and filtering by “lesion” for the ones with the lesion or “random” for healthy ones.

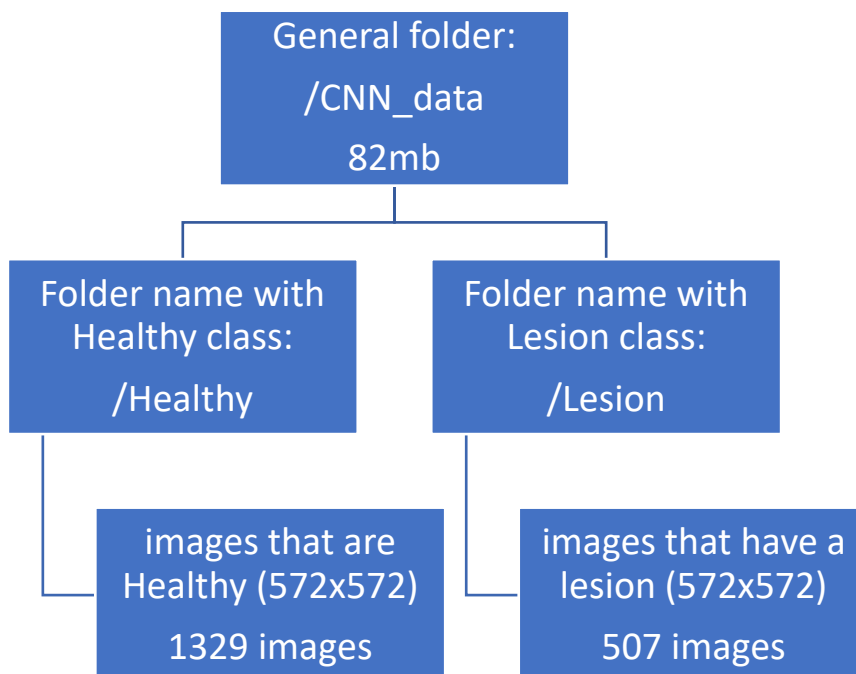


Figure 20 Data structure of the extracted data

6.1.2. Images technical data

In Figure 21 and Figure 22 below we can see detailed data. We currently have a (576,576) RGB image.

Property	Value
Dimensions	576 x 576
Width	576 pixels
Height	576 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24

Figure 21 Properties of the image

Property	Value
File	
Name	patient_78721tipo1lesion_E...
Item type	JPG File
Folder path	
Date created	28-Apr-20 12:19
Date modified	23-Apr-20 13:01
Size	48.9 KB
Attributes	A
Availability	
Offline status	
Shared with	
Owner	
Computer	

Figure 22 Properties of the File

6.1.3. Code for image pre-processing

Once we have the data organized, we will manage to perform the pre-processing we discussed in the beginning of the section.

The code snippets that will be discussed are integrally hereunder and in Github at:

https://github.com/raulbenitez/deep_endoscopy/blob/master/Image_pre_processing.ipynb

We first import the utilities already seen and start with the Lesion folder (We will later-on change the path to the healthy folder and execute again the process to those other pictures)

```
import os
import cv2
import random

path= r'/CNN_data/Lesion'
```

(Optional step) We rename the files for easy use:

```
for file in os.listdir(path):

    if file.endswith('.jpg'):
        a=str(int(random.random()*10000))
        os.rename(path+ '/' + file , path+ '/' + a + '.jpg')
```

We then crop the image from (572,572) to (512,512). This will eliminate the date, time, and brand:

```
for file in os.listdir(path):
    #%debug
    if file.endswith('.jpg'):
        img=cv2.imread(path+ '/' + file)
        left2=32
        right2=544
        top2=32
        bottom2=544
        img=img[top2:bottom2,left2:right2,:]
        image=img.copy()

        cv2.imwrite(path+'/' + file,image)
```

Now we have a good crop that we can visualize in Figure 23 below.



Figure 23 Cropped image that still has some letters

Nevertheless, some letters remain, thus the execution of the following steps. We put a black

stripe on the letters leftover and also the small white pixel on the top-right corner:

```
for file in os.listdir(path):
    #debug
    if file.endswith('jpg'):
        img=cv2.imread(path+ '/' + file)
        left2=32
        right2=544
        top2=32
        bottom2=544
        img[0:50,0:50,:]=np.full((50,50,3),0)
        img[0:57,455:512,:]=np.full((57,57,3),0)

        cv2.imwrite(path+'/' + file,img)
```

After this, we finally have a clear picture of our neural network infeed. We can see this output in Figure 24 below.



Figure 24 Image of a lesion totally cropped and cleared of text

7. Deep learning processing

Thanks to the medical staff for the confection of the video, and the first part of this project for the extraction and pre-processing of the data, we are about to use it in machine learning.

7.1. Introduction to the processing

7.1.1. Final data structure

Once we have the healthy images and the ones with lesion pre-processed, we have the final organization is shown in Figure 20Figure 25 below.

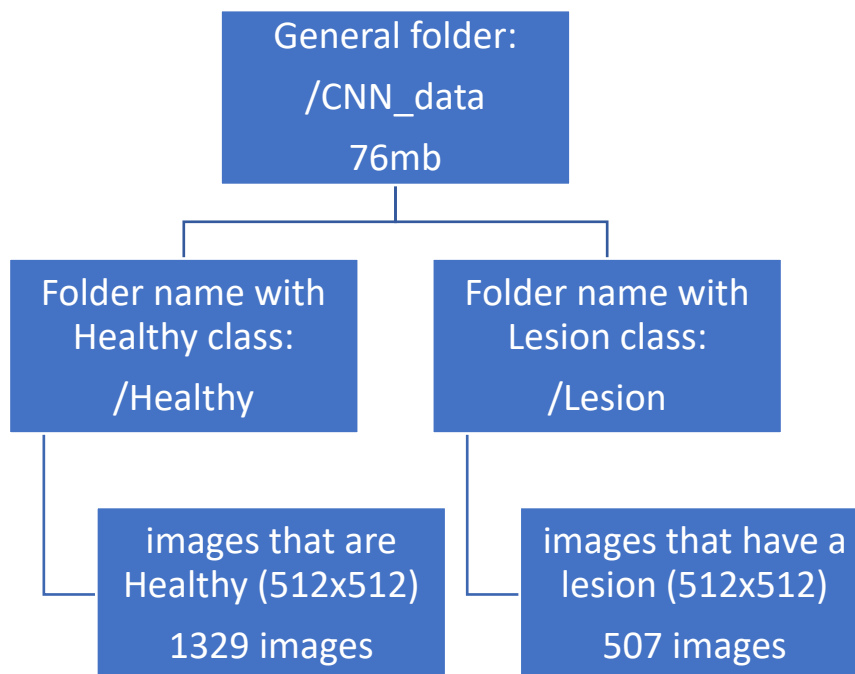


Figure 25 Data structure of the final pre-processed data

7.1.2. Images technical data

In Figure 26 and Figure 27 below we can see the final data technical details. We currently have a (512,512) RGB image. We see that the crop has also optimized the size and that will be beneficial to the learning algorithms.

Property	Value
Dimensions	512 x 512
Width	512 pixels
Height	512 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24

Figure 26 Properties of the image

File	
Name	8814.jpg
Item type	JPG File
Folder path	
Date created	07-May-20 18:32
Date modified	07-May-20 18:32
Size	35.0 KB

Figure 27 Properties of the File

7.1.3. Strategy

A very important aspect of this project is the strategy to approach the problem from wide angles to compare the results. Because there is no assurance that one strategy will be better than another, we have tried with few strategies that we thought could apply to our problem but there could be others. In this chapter, we will present the machine learning models used, but also transfer learning strategies and verification mechanisms used.

7.2. MNIST model with training

We will try a model thought for classifying the MNIST database. The MNIST database contains 60,000 training images and 10,000 testing images from integer digits from 0 to 9. We will use the model proposed to see if it can adapt to our pictures. This kind of model is not very “deep” as the following models we will be discussing. All the code is available in ANNEX 13.4 ant at :

https://github.com/raulbenitez/deep_endoscopy/blob/master/Code_of_MNIST_model_with_training.ipynb

7.2.1. Model structure

The machine learning model, which you can see in Figure 28 below. It consists of 7 layers of image-adapted machine learning. We are using a premade model which has been performing the digit recognition. This model is well known and can be consulted anytime in the GitHub link.

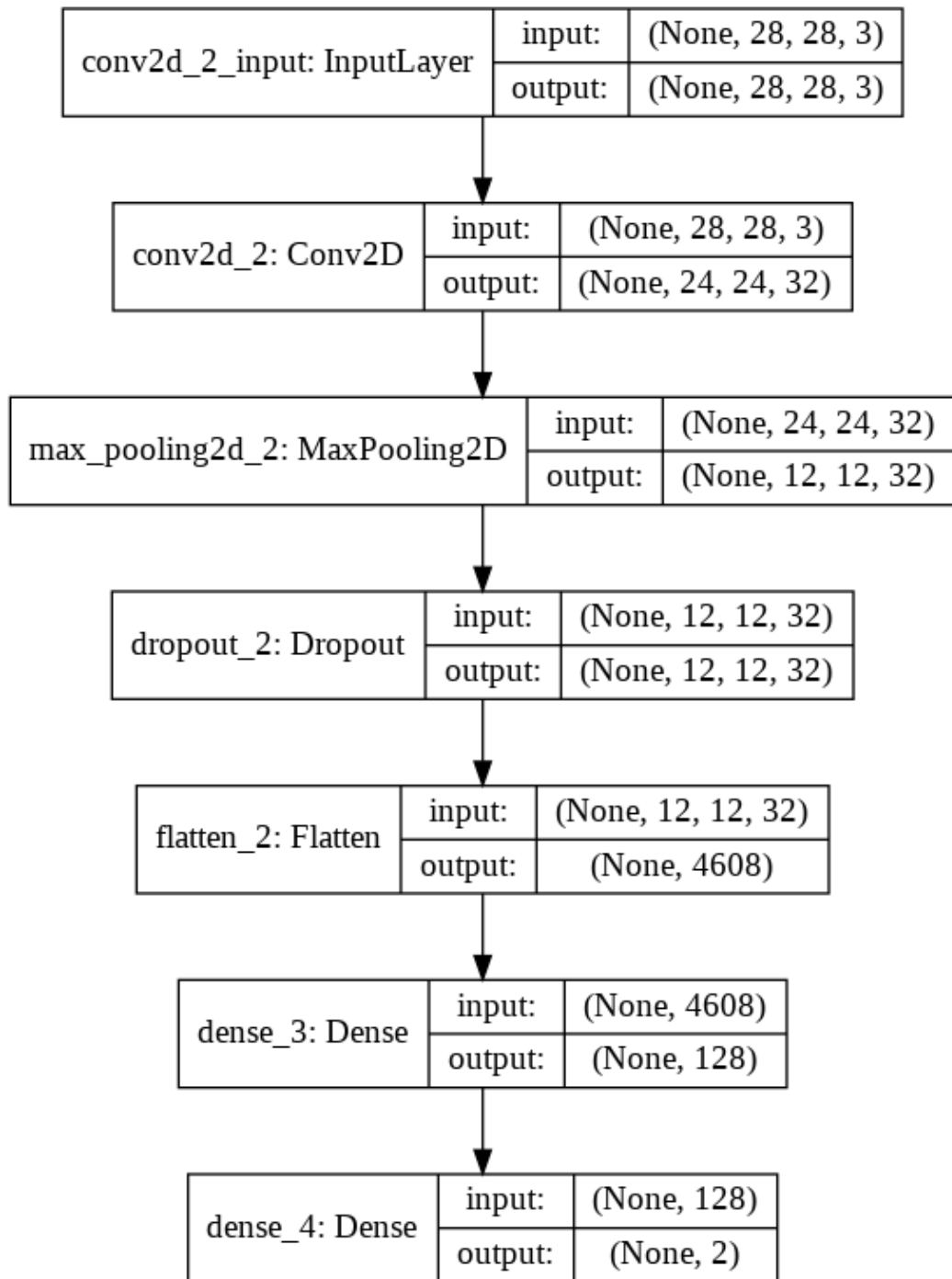


Figure 28 Model layers and tensor shapes

7.2.2. Training

We use for training the images present in 7.1.1 Final data structure. We split all our samples in 10% for verification and the rest for training, and we can see their values in the Table 1Table

10below.

	Lesion images	Healthy images
Training	447	1334
Validation	50	148

Table 10 Data split

We are going to import the data through a data generator instance. Although this instance would allow us to perform data augmentation, we are going to use it solely to load from the directory of our pictures.

7.2.3. Results

The results were not satisfactory. As we can see in Figure 29 below, the training overfitted with training data (accuracy=1 at epoch 118), and the validation stabilized at 75% well before the accuracy of the training data went up. That is, the training has been “useless” from epoch 20.

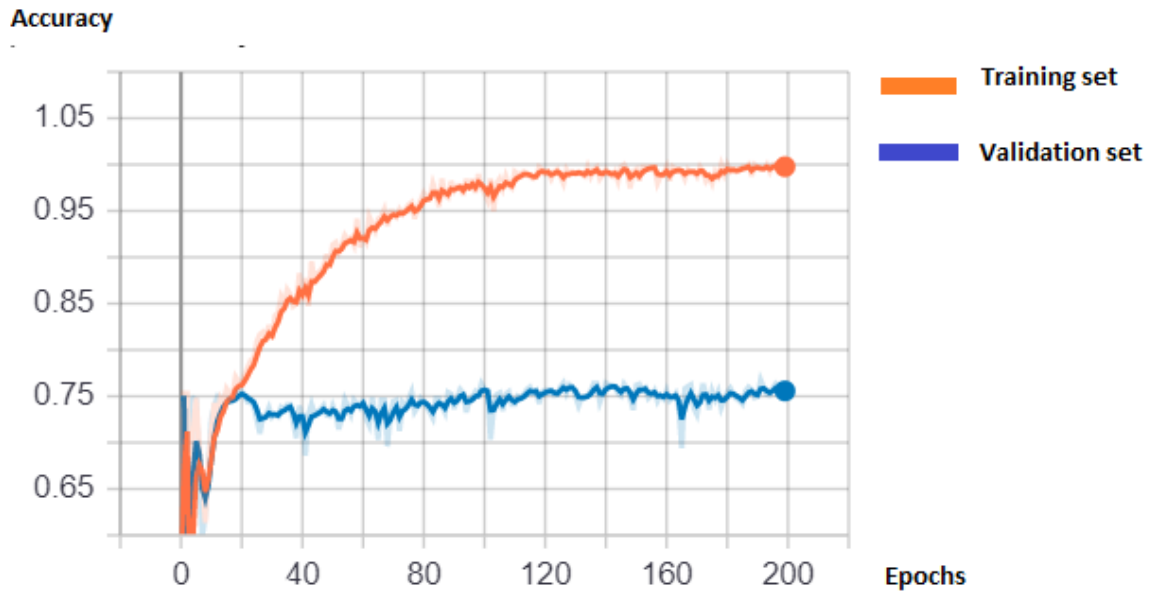


Figure 29 Accuracy evolution through the epochs

The main proposal for behavior is that the model is somewhat superficial and also looks for big parts of the image. Our images are made of very small and subtle textures, which would be both difficult to deduce with a model with few layers, and also for a model with such low resolution.

We think that in some cases, very big lesions can be detected and rightfully categorized with this network. Finally, this model has no use because 75% accuracy on a binary chance (50%) of being right is not acceptable for the purposes we are looking for.

7.3. VGG16 pre-trained with feature extraction and classifier

7.3.1. Model introduction

The VGG16 is a model conceived for large-scale image recognition. It has been trained with a dataset of over 14 million images belonging to 1000 classes, from the ImageNet image database. We show a sample of this dataset in Figure 30 below. VGG16 was trained for weeks using NVIDIA Titan Black GPU's, and the model achieves 92.7% top-5 test accuracy in ImageNet. [11]

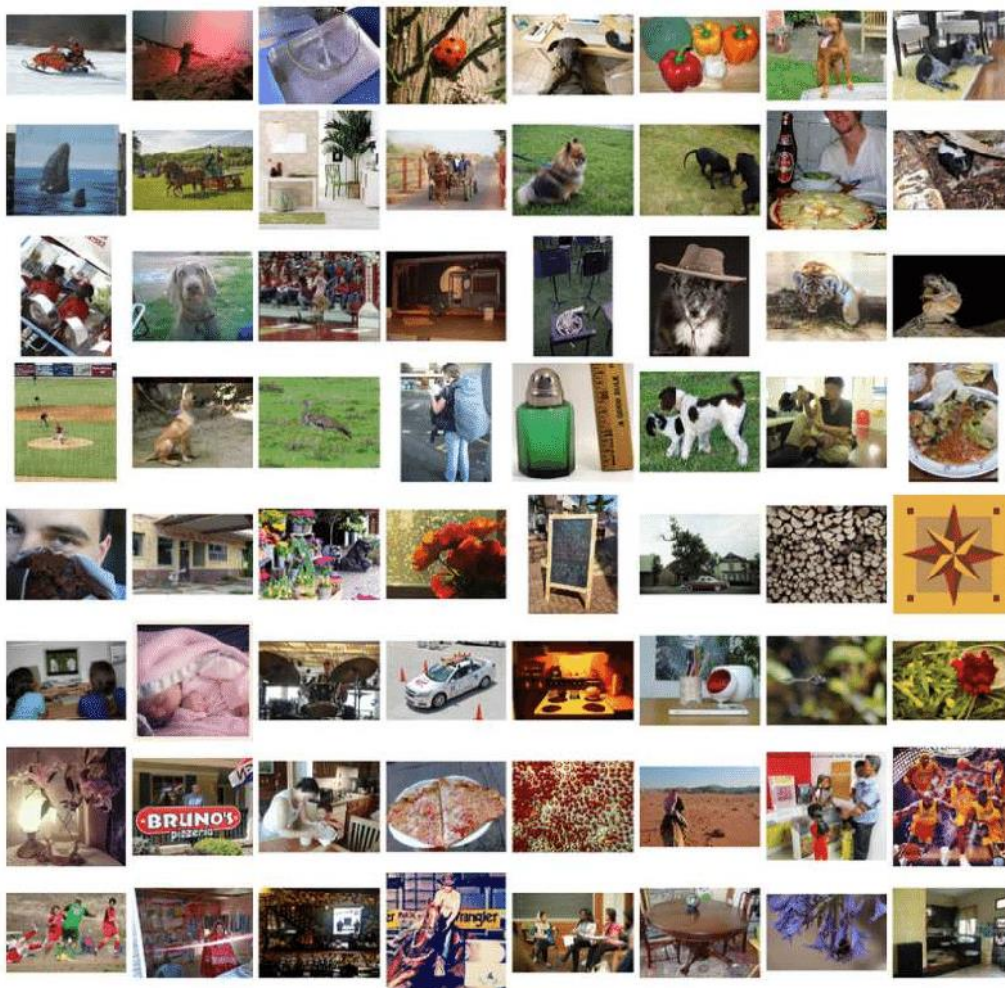


Figure 30 Sample of ImageNet images [12]

In this case, we are attempting to perform Transfer Learning in our pipeline. Transfer learning is the methodology that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. [13].

In our case, the knowledge gained while learning to recognize those 1000 classes from ImageNet could apply when trying to recognize lesions and healthy pictures.

Although cars and flowers have little in common with gastroenteric lesions from a logical point of view, they share common textures, shadows, colors, and edges. It is additionally relevant to use this technology in case we have a small set of data.

First, we will use this pre-trained image model to extract the features detected by the model for the training and validation sets. Once we have the features, we will train a classifier to classify the features between the healthy and the ones with a lesion.

The classifier will be in this case a support vector machine. The theory behind this software consists in a technique to solve a problem when it is not separable, called “kernel trick”. When a dataset is not separable in two planes (for example the left figure in

Figure 31 below), we may use a support vector machine to create an additional dimension to make our problem separable (the right figure in

Figure 31 below).

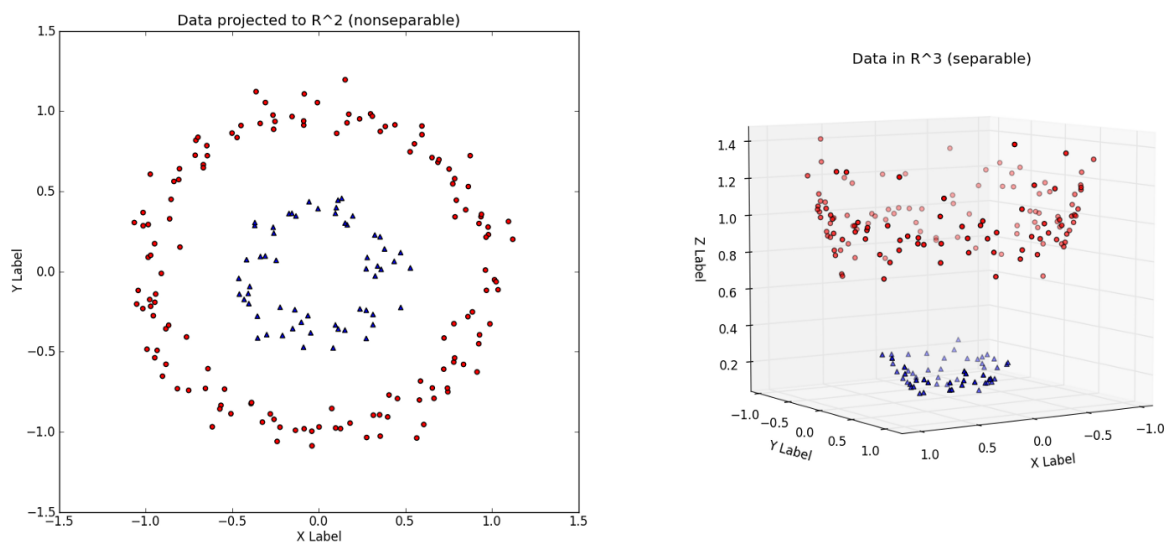


Figure 31 Visualization of an example of kernel trick. [16]

Separable is defined as being able to separate two datasets in a n-dimensional plane. Once we found the separability, we compute the result backwards to refer to the original data. The variables of the SVM we will use are there to fine-tune some aspects while performing this task. Such as ignoring one off value very far from the other values, and other statistical factors. [16]

In our case healthy frames and lesions make it hard to imagine what the kernel trick will look like. We have arrays of pictures and arrays of features, but the theory is the same. It is one of the best strategies for classification.

After introducing the “kernel trick”, we will discuss the whole setup. The process pipeline will begin with a VGG16 network pre-trained on images from ImageNet. The VGG-16 will have the structure described in Figure 32 and Figure 33 below.

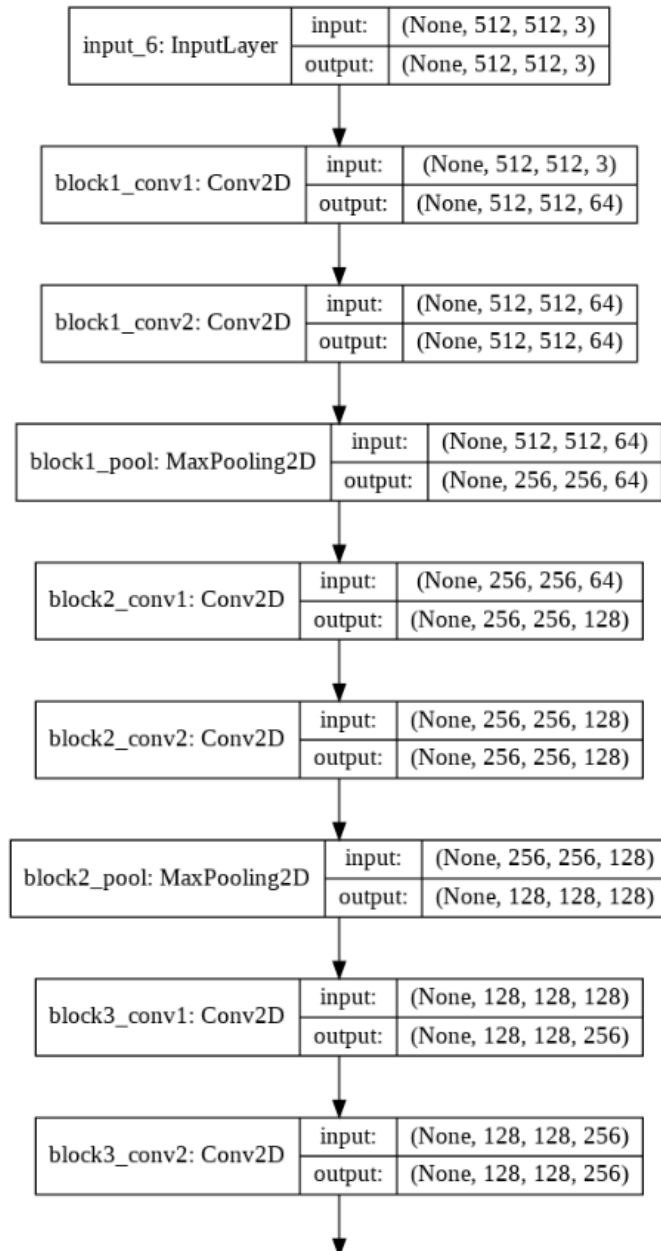


Figure 32 Layers of VGG-16 (1 of 2)

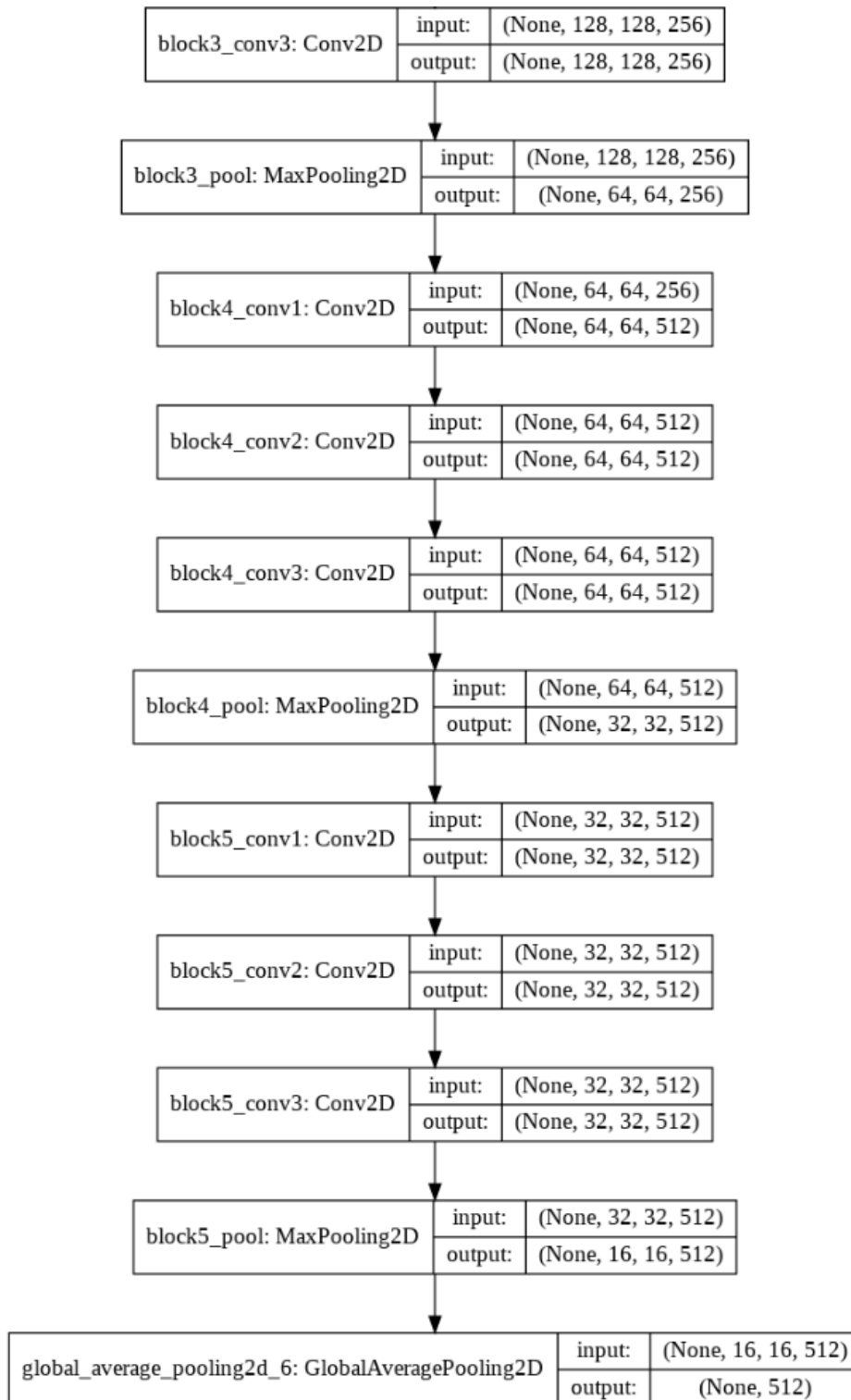


Figure 33 Layers of VGG-16 (2 of 2)

7.3.2. Training

In this section, we will go through the code creating this training. This code is available at:

https://github.com/raulbenitez/deep_endoscopy/blob/master/VGG_Pre_trained_on_imagenet_with_SVC_classification.ipynb

We will avoid writing here the inputs, but they are available and commented in section 13.5 in the annex and the GitHub link provided.

First, we are going to import the data through a data generator instance. Although this instance would allow us to perform data augmentation, we are going to use it solely to load from the directory of our pictures. The pictures are organized as shown in 7.1.1 Final data structure above.

Thanks to this utility, we will have all the pictures and their classification in x and y NumPy vectors respectively.

Then, we will use a split utility to split between train and test data. We choose for this application a 40% of test data, to have:

	Lesion images	Healthy images
Training	299	892
Validation	198	591

Table 11 Data split

After that, we will convert the class vectors to an array of integers, for example, shape (789,) for the validation vector.

Then, we will load the standard VGG model without including the top, pre-trained on imagenet, and, with an average pooling and the input shape defined as (512,512,3). The model can be seen in Figure 32 and Figure 33 above.

The fact of setting the input shape as such is the creation of this first input layer, which can be seen as input_6 in Figure 32 above.

Additionally, the setting of the average pooling creates an output of features (512 features per image) which can be seen at the bottom of Figure 33 above.

Now comes the key part of transfer learning. We are going to take these 512 features per image and feed them to a classifier, to optimize the sorting for lesions and healthy pictures.

We will now use a C-Support Vector Classification software. [15] This software will be trained to classify the features extracted and associate them with lesions or healthy classes. At the same time, we will optimize the value of algebraic parameters in this classifier with a utility software named GridSearchCV. For illustrative purposes, one of these algebraic parameters would be how far from a trend of features is one feature taken into account. Because we cannot have any direct hint on the parameter settings, we will iterate through vast ranges to find the best parameters. [16]

Once we have the best parameters for our classifier and we have the test features, we can task the classifier for its purpose. We may then plot the confusion matrix to see the results, in the next section.

7.3.3. Results

After executing the code in approximately 14 minutes in Colab servers in TPU mode, we achieve the confusion matrix of the test features, in Figure 34 below.

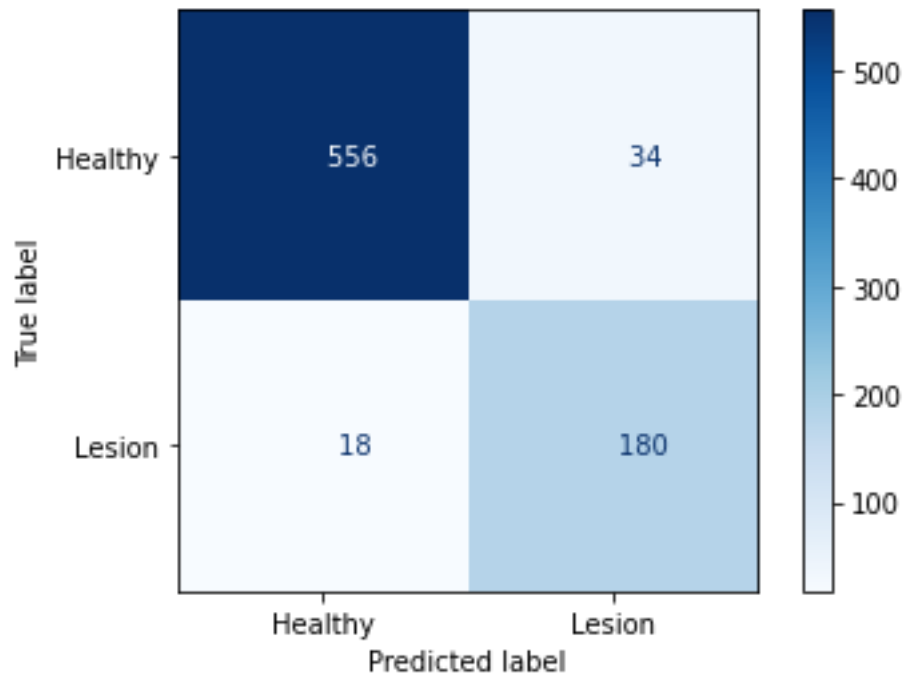


Figure 34 Confusion matrix of the SVC classifier with the test features

In this section, we will discuss the statistical variables are defined in the Glossary page 11. We are going to define the true positive, true negative, false positive, and false negative values.

Variable	Abbreviation	Value from the confusion matrix
True Positives	TP	180
True Negatives	TN	556
False Positives	FP	34
False Negatives	FN	18

Table 12 Confusion matrix results and abbreviations

Subsequently, we can see the several scoring values of our model in Table 13 below.

Variable	Formula	Value
Sensitivity, hit rate, recall, or true positive rate	$TP/(TP+FN)$ (Ec. 7.1)	0.91
Specificity or true negative rate	$TNR = TN/(TN+FP)$ (Ec. 7.2)	0.94
Precision or positive predictive value	$TP/(TP+FP)$ (Ec. 7.3)	0.84
Negative predictive value	$NPV = TN/(TN+FN)$ (Ec. 7.4)	0.97
Fall out or false positive rate	$FPR = FP/(FP+TN)$ (Ec. 7.5) (Ec. 7.6)	0.05
False negative rate	$FNR = FN/(TP+FN)$ (Ec. 7.7)	0.09
False discovery rate	$FDR = FP/(TP+FP)$ (Ec. 7.8)	0.15
Overall accuracy	$ACC = (TP+TN)/(TP+FP+FN+TN)$ (Ec. 7.9)	0.93

Table 13 Final results of the classification of validation data

These results are analyzed as good results given the small data set for training (299 lesion and 892 healthy images). Nevertheless, the validation set (198 lesions and 591 healthy) is significant enough to deliver solid results on this data, with an accuracy of 93% and a false negative rate of 9% (The probability of missing a lesion when the patient is ill). The false discovery rate (The probability of stating that there is a lesion when there is none) is fairly high (15%) but it could be assumed although it could undermine cost-effectiveness in the daily medical operations.

7.4. VGG16 pre-trained with fully connected training

To develop further the analysis of the results, we will define this third model. The objective of this model is not to create a classifier or the best classifier. The objective is rather to approach a visual representation of what is the VGG pre-trained on imagenet “looking at”, when we extracted the features from it. We will do that through software called LIME that began developing 4 years ago, and now has several applications, one of which is explaining image decisions. The simplified mechanics of LIME consist in distorting pixels of the image and classifying them with our model. If we iterate several times with different distortions of the same image, we can visually see the areas that artificial intelligence has taken into account the most when making a decision. [18]. We will see this results in section 7.5 Interpretability, after we have the results from the models.

7.4.1. Model structure

In this auxiliary model, we will use the previous VGG16 model but with a fully connected layer at the end. The code is available in the ANNEX 13.6 and in the GitHub link below:

https://github.com/raulbenitez/deep_endoscopy/blob/master/Code_of_VGG_pre_trained_with_fully_connected_training.ipynb

First, we will prepare our data as before, and split it in the same proportion as in our run-in section 7.3 above.

Then, we will import the pre-trained model from ImageNet as we did before. We will not include any top to have the same as before.

```
model_vgg16_conv = VGG16(include_top=False,  
                          weights='imagenet',  
                          input_tensor=input_layer.Input(shape=(512, 512, 3))  
                          )
```

After this, we will add 5 layers instead of a classifier. This is due to the compatibility of LIME with Transfer learning. Although we are changing the final phase of the process, the features deducted from the VGG16 will remain the same. This fact allows us to compare the outputs of this model with what parts of the image has the VGG16 detected and returned as features. For utility purposes, we then create a Flatten and 3 dense layers, which can be visualized in Figure 35 below in yellow. That way, the output will be whether we detect the lesion or we have

a healthy picture.

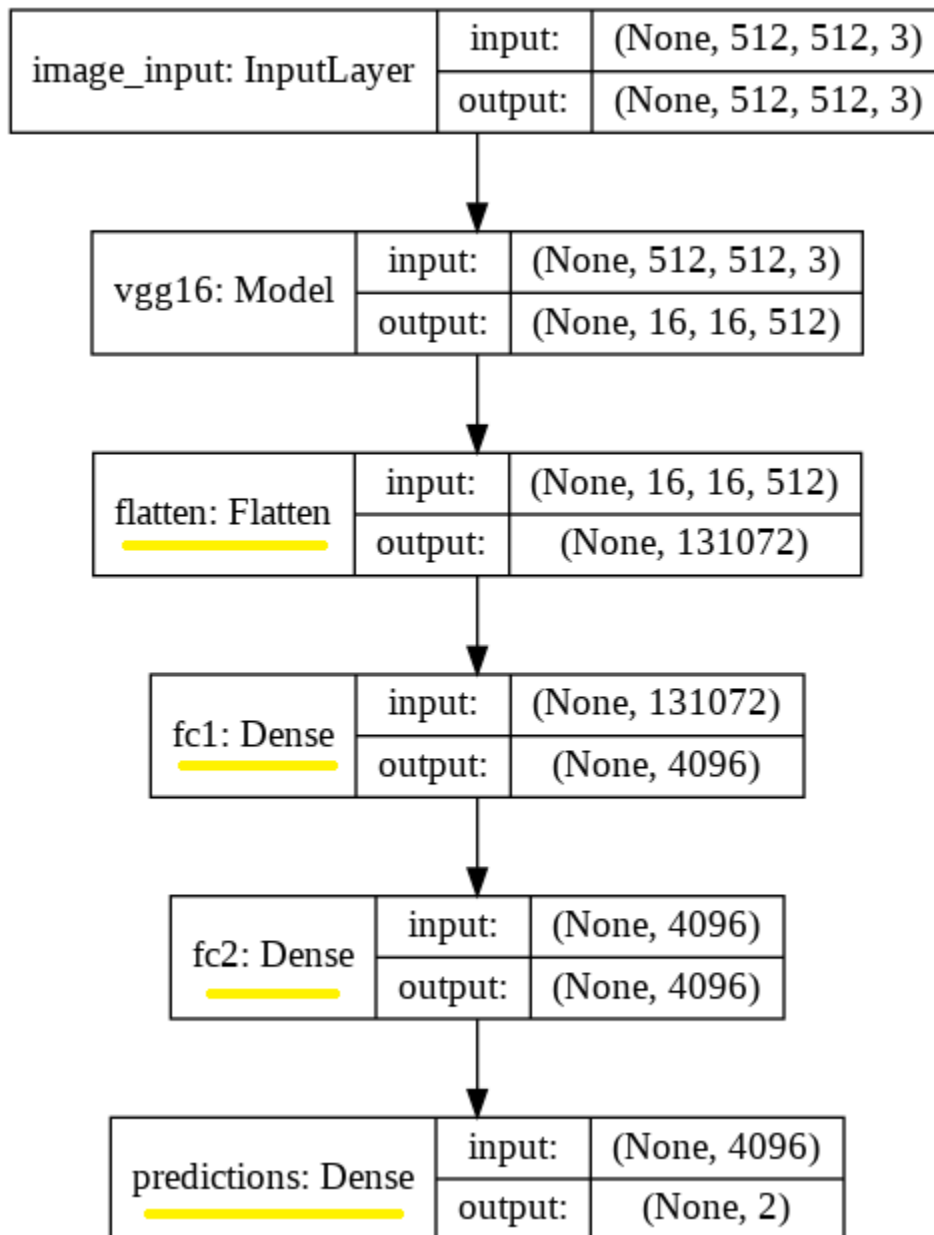


Figure 35 Auxiliary model for LIME interpretability

After this, we set the VGG16 as not trainable to preserve the original feature detection:

```
for model in model2.layers[:2]:  
    model.trainable=False
```

And we finally compile and train the model:

```
model2.compile(optimizer='rmsprop',  
               loss='categorical_crossentropy',  
               metrics=['binary_accuracy'])  
  
model2.fit(x_train, y_train,  
           batch_size=100,  
           epochs=20,  
           validation_data=(x_test, y_test))
```

After the training, we will use LIME:

```
explainer = lime_image.LimeImageExplainer()  
  
explanation = explainer.explain_instance(image_for_lime, new_model.pr  
edict, top_labels=50, hide_color=0, num_samples=1000)
```

We load to the explainer instance the image we want to analyze, our model with the *predict* attribute, and we set to show us the top 50 labels taken into account. We also state to perform 1000 iterations, and not to hide the remaining image.

7.4.2. Results

The training of the model achieves a binary accuracy of 93,4%, very close to the pre-trained with feature extraction.

In this section, we will again discuss the statistical variables are defined in the Glossary page 11, and output of our model. We are first going to define the true positive, true negative, false positive, and false negative values.

Variable	Abbreviation	Value from the confusion matrix
True Positives	TP	182
True Negatives	TN	552
False Positives	FP	32
False Negatives	FN	22

Table 14 Confusion matrix results and abbreviations for VGG F.C.

Subsequently, we can see the several scoring values of our model in Table 15 below, all specified in the Glossary page 11.

Variable	Formula	Value
Sensitivity, hit rate, recall, or true positive rate	$TP/(TP+FN)$ (Ec. 7.10)	0.89
Specificity or true negative rate	$TNR = TN/(TN+FP)$ (Ec. 7.11)	0.94
Precision or positive predictive value	$TP/(TP+FP)$ (Ec. 7.12)	0.85
Negative predictive value	$NPV = TN/(TN+FN)$ (Ec. 7.13)	0.96

Fall out or false positive rate	$FPR = FP/(FP+TN)$ (Ec. 7.14) (Ec. 7.15)	0.05
False negative rate	$FNR = FN/(TP+FN)$ (Ec. 7.16)	0.10
False discovery rate	$FDR = FP/(TP+FP)$ (Ec. 7.17)	0.15
Overall accuracy	$ACC = (TP+TN)/(TP+FP+FN+TN)$ (Ec. 7.18)	0.93

Table 15 Final results of the classification of validation data for VGG F.C.

These results are also analyzed as good results given the small data set for training (299 lesion and 892 healthy images). Nevertheless, the validation set (198 lesions and 591 healthy) is significant enough to deliver solid results on this data, with an accuracy of 93% and a false negative rate of 10%. This behavior is very similar than the previous one. The reason can be that they both share the same feature extraction. The reason for building this model remains in being able to technically build some interpretability through LIME analysis.

7.5. Interpretability

As introduced before, while analyzing the results from the model VGG with classifier, we discussed about the opportunity to analyze what the AI would be “looking at” at a feature level. Therefore, we thought about creating a fully connected layer of the VGG, in order to technically be able to apply LIME.

After executing the code with an image with a lesion rightfully detected (True Positive), we can see side by side the lesions that LIME has detected. We have associated zones with visible lesions with yellow lines, in Figure 36 below.

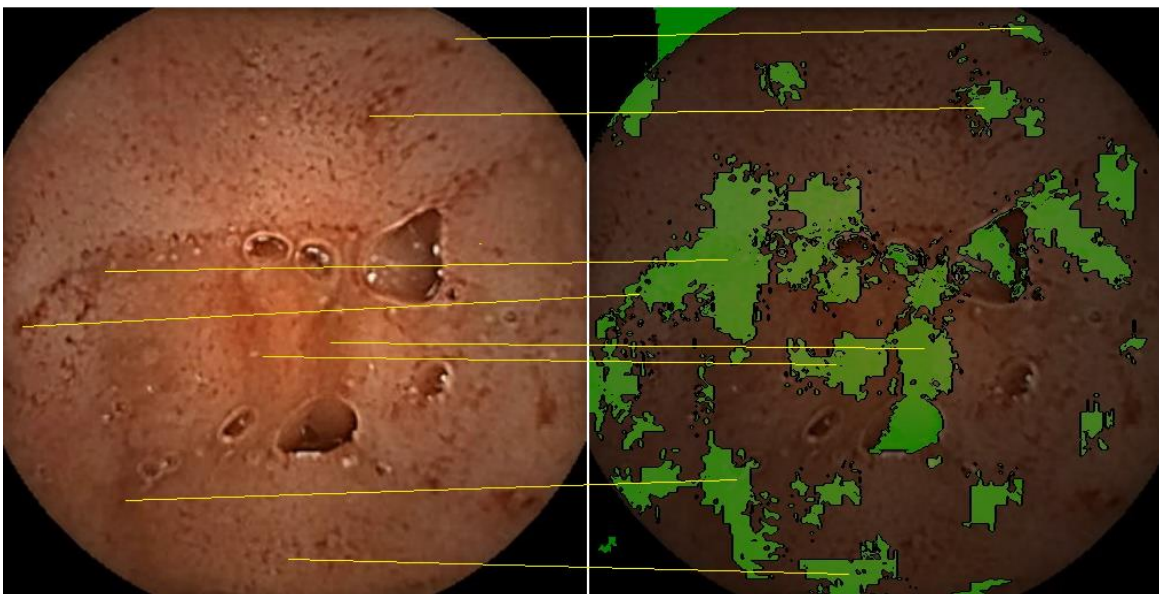


Figure 36 Original image of a lesion (left) and LIME zones taken into account (right).

Nonetheless, there are also some images with no evident explanation or complete explanation, such as Figure 37 below, also rightfully detected as a lesion.

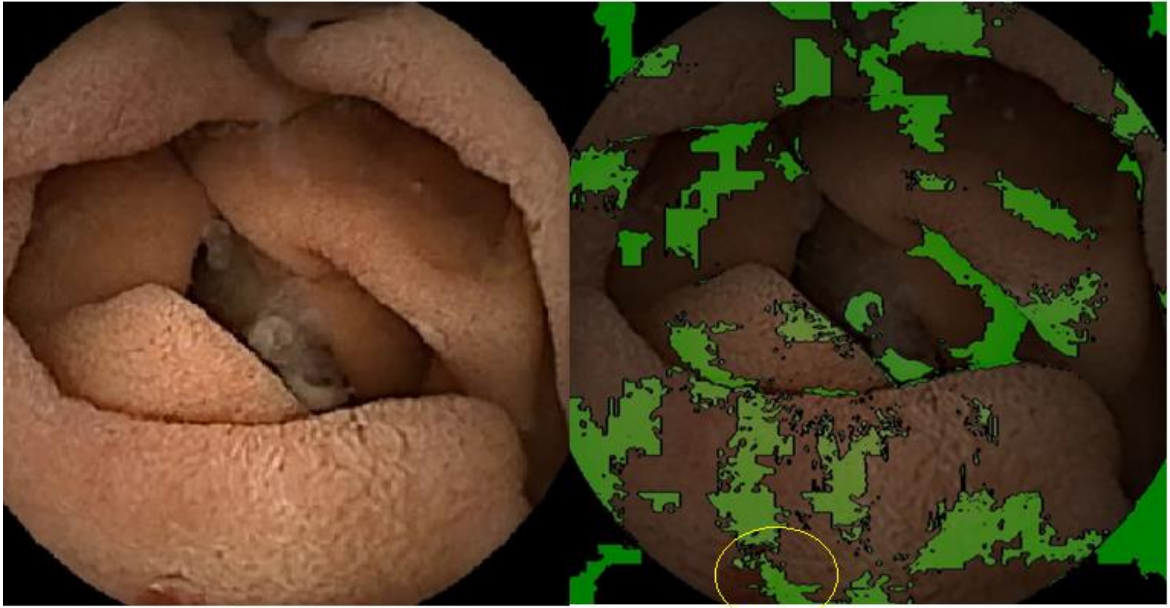


Figure 37 Lime analysis of a lesion (lesion highlighted in yellow)

We can see that LIME has taken in this case the lesion as a part of the decision (in yellow), but we also see that it has taken several parts of the picture that do not have any interest.

One possible explanation could be the weight of each of those zones in the decision. It is not the same a zone with a weight of 1% that another of 50%. Yet, with this representation, we are not able to classify the weight of the zones highlighted, they all appear in green.

Another possible explanation is LIME not being able to consider that you can have one big lesion, or one small, or several small lesions, or one big and one small, in the same picture. The model in those cases gives the right answer, but the LIME representation has no particular analysis.

Furthermore, it could also be overfitting due to a small set of data for certain lesion shape or shapes.

7.5.1. False positive LIME analysis

We have also analyzed false positive pictures (A picture that is healthy but is predicted as having lesion). We can see eight of them in Figure 38 and Figure 39 below.

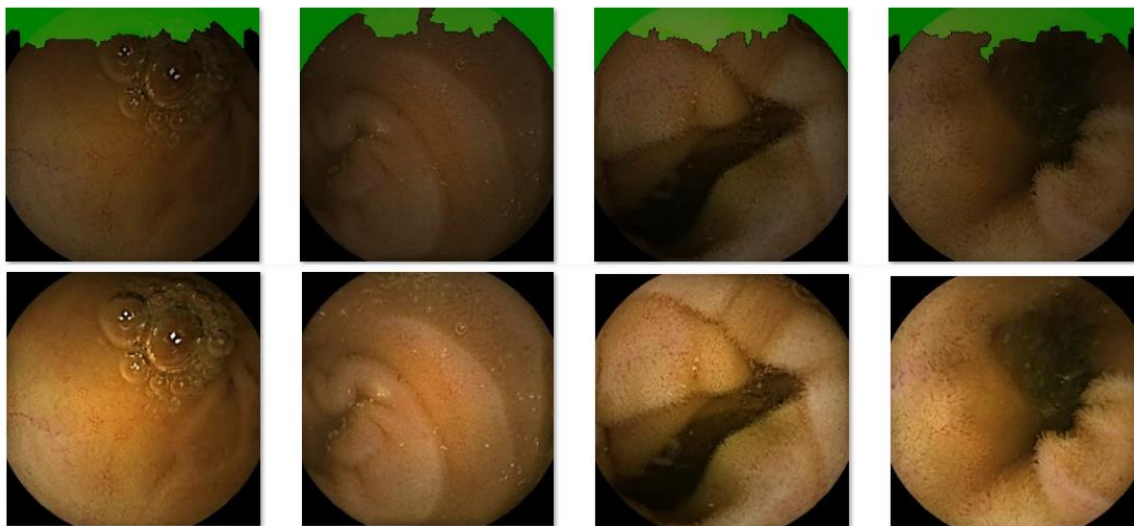


Figure 38 False positive LIME representation (1 of 2)

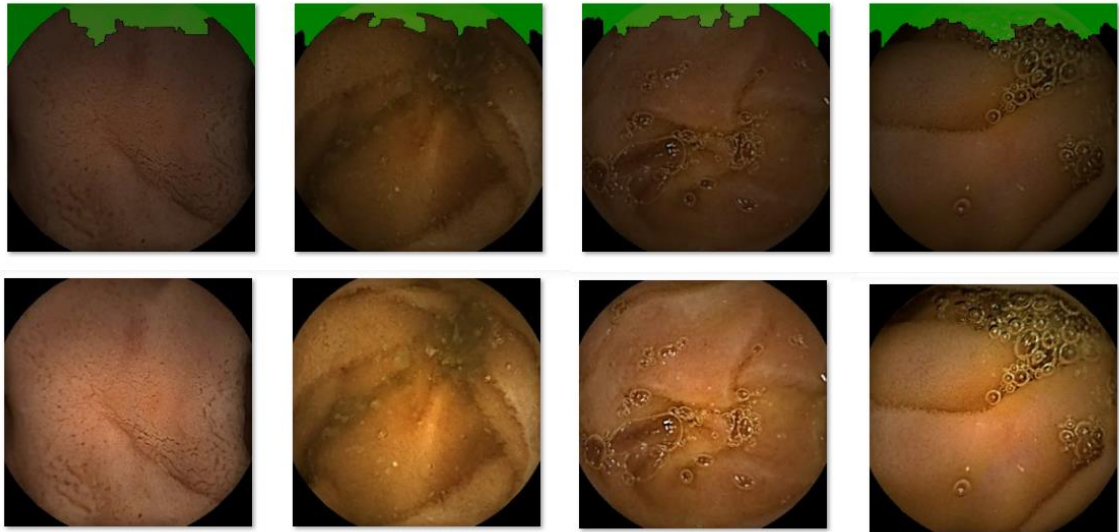


Figure 39 False positive LIME representation (2 of 2)

As we can see, we can hardly explain with LIME the reason for these healthy pictures to be classified as lesions. The artificial intelligence could seem “lost” by looking at the “beginnings” of the NumPy arrays. Therefore, we can see no useful conclusion for this category.

7.5.2. False negative LIME analysis

We have subsequently analyzed false negatives pictures (A picture that has a lesion but is predicted as being healthy). We can see eight of them in Figure 40 and Figure 41 below.

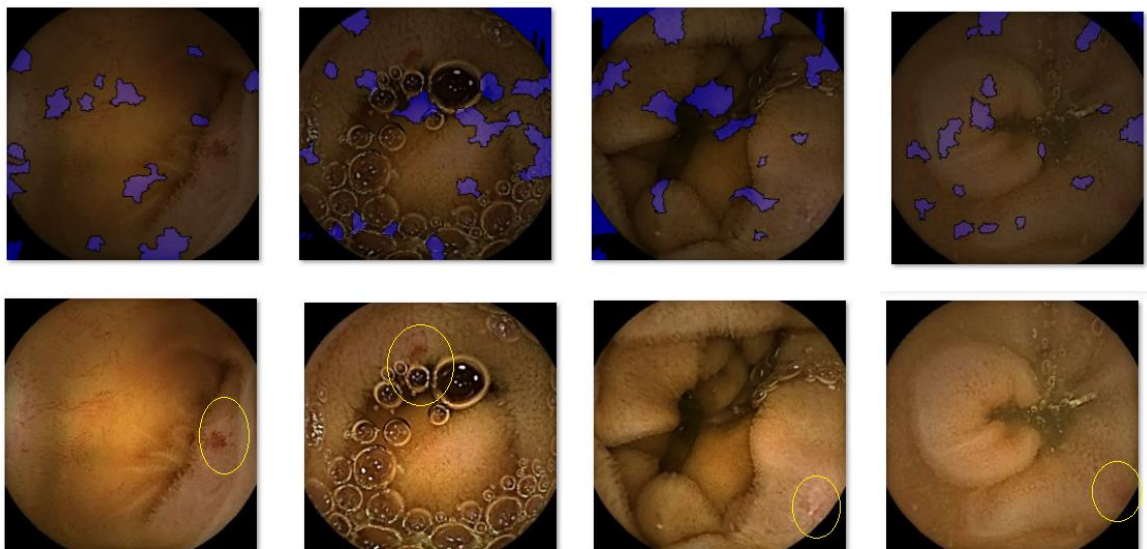


Figure 40 False negative LIME representation (1 of 2)

We can see that the lesions taken into a false negative decision have many things in common. They all are really “hard” to spot. For example, they can have the lesion in the borders of the picture (The third picture that has the lesion in the bottom, in Figure 41 below). These phenomena can happen if we have a number of pictures of a lesion in a row appearing in the video. The first frame or the last can have the lesion on the edge of the picture. We can deduce it, but it is very hard to spot as we have “half” a lesion. Nevertheless, maybe data augmentation could solve this problem (but maybe create other drawbacks).

We can also have the case of having the lesion very close to an air bubble, which makes it look different, for example in the second picture of Figure 40 above.

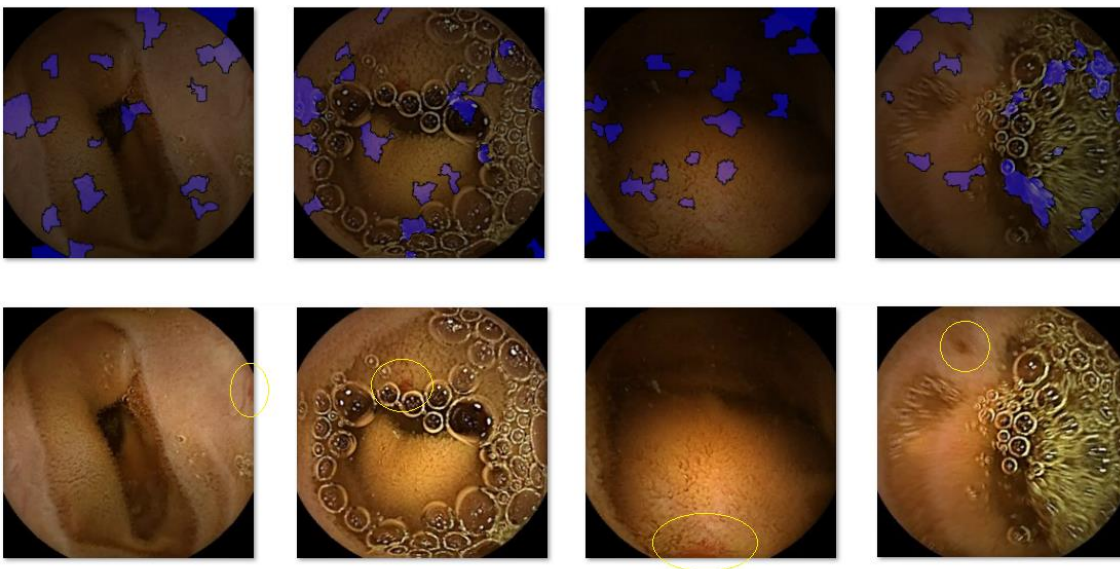


Figure 41 False negative LIME representation (2 of 2)

Finally, we can look at a final occurrence. We have a case (one in all the set) in Figure 42 below, where the AI has taken into account the lesion, but has probably found many other parameters against classifying it as a picture with lesion. This case could probably be solved with more training. It is also nonetheless another proof that the VGG16 can recognize the texture of a lesion (which we saw in Figure 36 and Figure 37 above), despite having wrongly decided.



Figure 42 False negative LIME representation

8. Conclusions

As conclusions for this master thesis, we can state that we have fulfilled “*A deep learning approach for the detection of intestinal lesions using endoscopic capsules*”.

As milestones, we propose to define 2:

On one hand, we have created and tested a tool to extract the pictures from the videos and prepare them for Artificial intelligence. This scalable tool eases the creation of wide datasets of endoscopic capsule images with their classification by medical staff.

On the other hand, we have found that the pre-trained model (VGG-16) is very good at detecting the attributes of a lesion, as it can rightfully classify with an accuracy of 93% the validation data, and a negative rate of 9% (The probability of missing a lesion when the patient is ill). We conclude that we can set it up with a fully connected network or a classifier because we esteem both results are very similar at detecting the lesions, although we have opted for the classifier.

On top of this, the LIME explanation software has shown us that our model looked at lesions, but also showed us that we might have some overfitting or distortion, as we can also see parts of the image taken that are not related to lesions. This might also be due to LIME not able to consider several sizes and combinations of lesions, although extensive research could be conducted to dive deep in the topic.

Nevertheless, it is very possible that this model struggles to detect specific types of lesion not seen before (for example a small lesion and a big lesion in the same picture, or any strange combination not present in the data). But hopefully, thanks to the first part of this project, we have a scalable software that can catch any new endoscopy video and use it to continue the training and “widen” its knowledge.

9. Planning and resources

The Project planning has been consistent with the planning defined by the university and the hours corresponding to ETCS (This project is a 12 ETCS subject). Therefore, we are looking at a dedication of 25-35 hours per credit.

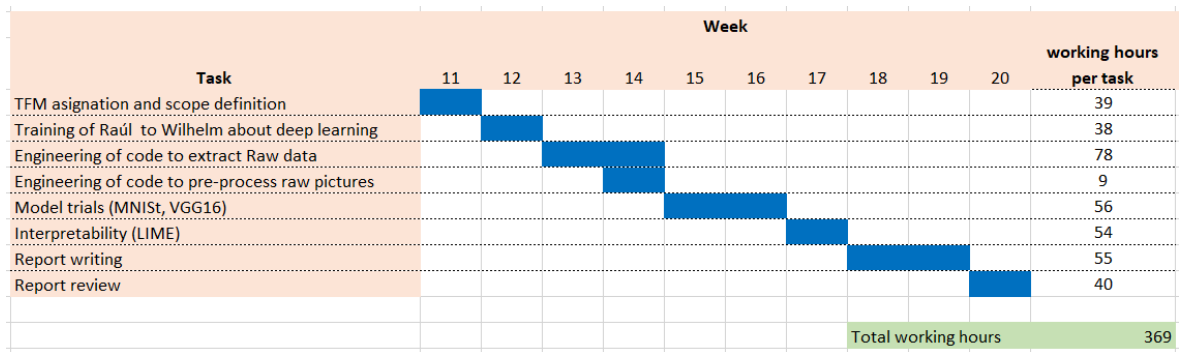


Figure 43 Gantt chart of the project

10. Economical and material list

Item	Unit cost €	Description	Unit dimension	Units	Total
Software engineer hours	210		Hour	369	77,490.00
IT tools	0	Provided by UPC	N.A.	0	-
Laptop	1542	HP EliteBook 850 G4	unit	1	1,542.00
Colab servers	0	Used in free time	hour	75	-
MS Office suit	150	Office suit for report	unit	1	150.00
Total (€):					79,182.00

Table 16 Resources allocated for this project

This Project cost would be in case of an external contractor. It has been realized with university internal resources and public-available servers.

11. Environment

The impact of this project on the environment is very narrow. Moreover, we have not bought any external hardware, that is, our carbon footprint is solely the impact of the electrical consumption, as asset depreciation is considered negligible for the duration of the project versus the utility lifespan of the equipment.

As the electricity needed from this project has come from personal source and Google servers, we do not have the data to know if the source of the electricity came from a renewable energy source but it is minimal, as the microprocessors are at 24V infeed, which is a negligible consumption at the scale of this project

12. References

- [1] P Gregory Foutch, Douglas K Rex, and David A Lieberman. Prevalence and natural history of colonic angiodysplasia among healthy asymptomatic people. *American Journal of Gastroenterology*, 90(4), 1995.
- [2] MICCAI 2017 Endoscopic Vision Challenge. MICCAI 2017 *Endoscopic Vision Challenge: Angiodysplasia Detection and Localization*
- [3] Alexey Shvets, Vladimir Iglovikov, Alexander Rakhlin, Alexandr A. Kalinin *Angiodysplasia Detection and Localization Using Deep Convolutional Neural Networks*, 2018
- [4] Lienhard, Dina A., "David H. Hubel and Torsten N. Wiesel's Research on Optical Development in Kittens". *Embryo Project Encyclopedia* (2017-10-11). ISSN: 1940-5030 <http://embryo.asu.edu/handle/10776/12995>
- [5] F. Noya, M. A. Álvarez-González, R. Benítez *Automated Angiodysplasia Detection from Wireless Capsule Endoscopy 2017*
- [6] F. Noya TFG UPC: *Sistema automàtic de detecció de lesions en imatges endoscòpiques 2015*
- [7] Neural network glossary: <https://envisat.esa.int/handbooks/meris/CNTR4-2-5.html>
- [8] Patiño T, Arqué X, Mestre R, Palacios L, Sánchez S. *Fundamental Aspects of Enzyme-Powered Micro- and Nanoswimmers*. *Acc Chem* (2018)
- [9] Endoscopic capsule example:
<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwixdTolJpAhVbSEEAHSi6DfoQFjAAegQIAxA&url=https%3A%2F%2Fwww.mayoclinic.org%2Fes-es%2Ftests-procedures%2Fcapsule-endoscopy%2Fabout%2Fpac->

[20393366&usg=AOvVaw1LS8JGxYsdRiOCbAykmwAW](#)

- [10] Virtual environment used for the project: <https://colab.research.google.com/>
- [11] Simonyan, Karen & Zisserman, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv 1409.1556. . (2014).
- [12] https://www.researchgate.net/figure/Examples-in-the-ImageNet-dataset_fig7_314646236
- [13] West, Jeremy; Ventura, Dan; Warnick, Sean "*Spring Research Presentation: A Theoretical Foundation for Inductive Transfer*". Brigham Young University, College of Physical and Mathematical Sciences. 2007-08-05.
- [14] Sensitivity and specificity: https://en.wikipedia.org/wiki/Sensitivity_and_specificity
- [15] Support vector machine used : <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- [16] Kernel trick. <https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78>
- [17] GridSearchCV: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- [18] Lime library: <https://github.com/marcotcr/lime>



13. ANNEX

13.1. Template for lesion classification (in Spanish)

N	Tipo	Tam	LRConsens	Loc	LocT	Sang	Potencial	Endos	FrameE	TiempoID	SBI	FrameSBI	NFPSBI	QV	FrameQV	NFPQV
1																
2																
3																
4																
5																
6																
7																
8																

Tipo de lesión: 1. Lesión roja del consenso; 2. Úlcera; 3. Erosión; 4. Afta; 5 (vacío); 6. Tumor; 7 Pólipo; 8 Sangre roja; 9. Sangre oscura

Tamaño: tamaño aproximado en mm. 1-3 diminuto (D); 3-5 mm pequeño (P); 5-9 mm mediano (M); >=10 mm grande (G)

LR Consenso: AD angiodisplasia; EP Erithematous patch; RS Red spot; PB Phlebectasia



Localización (si se sabe por el informe): D (duodeno); Y (Yeyuno); I (ileon)

LocT (Localización en tiempo)

Sangrado Activo: 0 No; 1 Si

REVISAR Potencial de Sangrado: P2 al potencial (angiodisplasia P,M,G; Parche eritematoso o Punto rojo con sangre en ese tramo.) P0 Flebectasia y otras lesiones.

Endos: 0 No detectada por endoscopista; 1 Detectada

FrameE: Número de frames positivos para el endoscopista

Tiempo ID: Tiempo de exploración del intestino delgado

SBI: Lesión detectada por el sistema SBI (0/1)

Frame SBI o QV: Número de frames que detecta el sistema SBI o el QV (FrameQV)

NFPSBI: Número de falsos positivos del SBI o el QV (NF

13.2.Function Image_extraction

```

def Image_Extraction(general_patients_folder): #Main that needs the directory of all the vid
eos

    for patient_folder in os.listdir(general_patients_folder): #lists all the patient director
ies and walks through them

        full_path = os.path.join(general_patients_folder, patient_folder) #enters every patient'
s directory

        for root, dirs, files in os.walk(full_path):

            for filename in files:#seeks the excel file and takes from him the parameters needed

                if (filename.endswith('.xlsx')) & ~(filename.startswith('~')) & ~(filename.start
swith('.')): #Checks that is an excel file
                    excel_path = os.path.join(full_path, filename) #Joins the path for convinien
ce

                    pandas_excel = pd.read_excel(excel_path, index_col=0,header=0)#Creates a pan
das instance in memory of the excel file

                    Timestamps=pandas_excel['LocT']# Takes from pandas the timestamps of the les
ions (where to look with the OCR)
                    Timestamps_tipo= pandas_excel['Tipo']==1 #creates a list of booleans that ar
e "True" if we have Type 1 lesions, the scope of our project
                    Tipo_lesion=pandas_excel['Lrconsenso']#creates a list of the type of the les
ion, it is for saving it with the lesion type

                    for filenamempg in files: #finds the video file and for every lesion runs find_image
to extract the images with lesion. Also runs find_image in random mode, to extract healthy
pictures from each patient

                        i=1 #Counter for the index of lesions
                        if ((filenamempg.endswith('.mpg')) & (not(filenamempg.startswith('~'))) & (not(f
ilenamempg.startswith('.')))) :

                            video_path = os.path.join(full_path, filenamempg) #Joins the video path
                            try:#We use this in case the operator has written the FramesE field
                                frames_detected=pandas_excel['FramesE']
                            except:
                                pass
                            try:#We use this in case the operator has written the FramesTotales field
                                frames_detected=pandas_excel['FramesTotales']
                            except:
                                pass

```

```

n_images_to_take=(Timestamps_tipo*frames_detected).values.tolist()#We calculate the number of images to take for each lesion (the multiplication is index-wise)

while Timestamps.size >= i and len(n_images_to_take) >=i : #While to o through the lesions

    written1=None #Variable to know if find_image was successful

    if not(math.isnan(n_images_to_take[i-1])):

        if Timestamps_tipo.tolist()[i-1] & (int(n_images_to_take[i-1])!=0): #Condition on weteher we have images to take and wether we are in range

            pass

            written1=find_image(video_path, full_path, Timestamps,Tipo_lesion.tolist(),frames_detected.tolist()[i-1],0,i,False)#for debugging, Timestamps[i],Tipo_lesion[i], i)

            os.chdir(os.path.join(full_path))

            i=i+1#Index update

            i=1#Inded reboot for the lesions

while Timestamps.size >= i and len(n_images_to_take) >=i:

    written2=None#Variable to know if find_image was successful

    if not(math.isnan(n_images_to_take[i-1])):

        if Timestamps_tipo.tolist()[i-1] & (int(n_images_to_take[i-1])!=0):#Condition on weteher we have images to take and wether we are in range

            pass

            try:

                written2=find_image(video_path, full_path, Timestamps,Tipo_lesion.tolist(),frames_detected.tolist()[i-1],n_images_to_take.tolist()[i-1], i, True)#for debugging, Timestamps[i],Tipo_lesion[i])

            except:

                try:

                    written2=find_image(video_path, full_path, Timestamps,Tipo_lesion,frames_detected[i-1],n_images_to_take[i-1], i, True)#for debugging, Timestamps[i],Tipo_lesion[i])

```

```
        except:  
            pass  
  
    i=i+1
```

13.3.Function find_image

```

def find_image(video_file, #Name of the video file
              full_path, # Full path of the mpg video
              lesion_time, #lesion_time is the time of a Timestamp. The main func-
tion has a numpy array of them, and each time the function is called it takes one of them
              Tipo_lesion, #Kind of lesion, from "Tipo" in the template
              frames_detected, #Number of frames detected by the endoscopis
              random_frames, #when random_frames is not 0, the function is in mode ran-
dom frame collection, when random_frames is 0, it collects lesions
              k, #index in the timestamps of the patient (for example if he has 3 le-
sions and k=2, that is we are in the last lesion)
              rand_if_true #Boolean to activate finding of lesions(False) or find-
ing of randoms (True)
):
    left, right, top, bottom=0,133,0,33#Pixels index to perform the OCR on
    time_read_in_picture=datetime.datetime.strptime(
time('00:00:00', '%H:%M:%S') #We set to 0 the OCR re-
turn that will guide us through the video
        if rand_if_true==False:#Enters the mode of seeking lesions (and not seeking healthy pic-
tures)
            #We seek the time of the last frame
            vidObj = cv2.VideoCapture(video_path) #This opens a videocaptura ob-
ject, to read the video
            vidObj.set(2, 1)#This sets the video at the end
            success, image = vidObj.read() #Success is a boolean to see if the read-
ing was good. image is a numpy array (572,572,3)
            height, width, channel=image.shape#As said, the fomrat is channel-last
            image=image[top:bottom,left:right,:] #We perform a selection of the upper-left cor-
ner (where the time is)
            text =image_to_string(image,config ='--psm 6')#We execute the OCR software
            time_of_last_frame=datetime.datetime.strptime(text, '%H:%M:%S')# We con-
vert it to datetime format

            #Finds the first time of the video
            vidObj = cv2.VideoCapture(video_path) #We reset the object
            vidObj.set(1, 1) #We set it at the beggining
            success, image = vidObj.read() #Success is a boolean to see if the read-
ing was good. image is a numpy array (572,572,3)
            height, width, channel=image.shape#As said, the fomrat is channel-last
            image=image[top:bottom,left:right:]#We perform a selection of the upper-left cor-
ner (where the time is)
            text =image_to_string(image,config ='--psm 6')#We execute the OCR software
            time_of_first_frame=datetime.datetime.strptime(text, '%H:%M:%S')# We con-
vert it to datetime format
            vidObj.release()#We release the video

```



```

        while ( (lesion_time[k] < time_of_last_frame.time()) (success):#if success was false, that would mean of a compromised video file or the end of the video
            #This while could be the key part of the function, it looks for a lesion time in OCR that matches the one that the expert has written.

            if (time_read_in_picture.time()>= lesion_time[k] ):# When there is a True value, we have found the beginning of a series of positive images.
                try:#We make a directory for every lesion detected
                    os.mkdir(full_path + '\\frames of lesion_time[k]_in_seconds%d'% ((lesion_time[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second) )
                    os.chdir(full_path + '\\frames of lesion_time[k]_in_seconds%d'% ((lesion_time[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second) )
                except:#If the directory is already made, only enters it and then copies the remaining pictures
                    os.chdir(full_path + '\\frames of lesion_time[k]_in_seconds%d'% ((lesion_time[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second) )
                    j=0#counter initiation for frames detected by the endoscopist
                    text_register=[]#This register writes down every new datetime, for example "01:11:12". This is to avoid repeating frames in the same second, because they are identical.

                    while (j < frames_detected): #takes as much frames as the endoscopist has seen
                        success, image = vidObj.read() #Success is a boolean to see if the reading was good. image is a numpy array (572,572,3)
                        image_raw=image.copy()# We will use this variable to store a copy of the image if we may want to save it later.
                        image=image[top:bottom,left:right,: ]#We perform a selection of the upper-left corner (where the time is)
                        text =image_to_string(image,config ='--psm 6')#We execute the OCR software
                        if not((text in text_register)):#We verify that it is indeed a new image (that the time has passed)
                            try:#We try in case we already have it from an aborted execution
                                cv2.imwrite("patient_" + str(patient_folder) + 'tipo'+ '1'+ 'lesion_' +str(Tipo_lesion[k-1])+'time_in_sec'+ 'plus_frame_last2digits'+ '%d.jpg' % (((lesion_time[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second)*100 + j*100) , image_raw)#writes the image
                                print('Writing....' + text + '    frame:    ' + str(j) + ' of ' + str(int(frames_detected))+ '    and lesion type    ' + str(Tipo_lesion[k-1]) )
                            except:
                                print('Not copied bcs of imwrite, maybe already there...' + text
                                )

                            pass

                        text_register.append(text)#Appends the newly seen image

```

```

        j=j+1#Updates the count because we have one image less to find

        vidObj.release()#Releases the video object
        return True #Returns True as output
        good_read=False#This boolean is used to be sure that we have succesfully read an
        image
        while not(good_read):#waits for a good read. We have a bad read aproximately 1 o
n 10.000 times, but we perform a lot of reads, hence this variable is useful
            success, image = vidObj.read() #Success is a boolean to see if the reading
was good. image is a numpy array (572,572,3)
            image_raw = image.copy()# We will use this variable to store a copy of the i
mage if we may want to save it later.
            text=image_to_string(image[top:bottom,left:right,:],config = '--
psm 6')#We execute the OCR software
            good_read=False#This boolean is used to be sure that we have succesfully rea
d an image
            try: # We have a bad read aproximately 1 on 10.000 times,
                time_read_in_picture=datetime.datetime.strptime(text, '%H:%M:%S')
                good_read=True#This boolean is used to be sure that we have succesfully
read an image
            except:
                good_read=False#This boolean is used to be sure that we have succesfully
read an image

            print('Seeking....' + text + ' to lesion time: ' + str((lesion_time[k])) + '
the ' +str(k)+ ' item of ' +str(len(lesion_time)))#Verbosity
            if (time_read_in_picture.time().hour*3600+time_read_in_picture.time().minute*60+
time_read_in_picture.time().second) < ((lesion_time[k].hour * 3600 + lesion_time[k].minute *
60 + lesion_time[k].second) *0.90):
                #This function accelerates the reading by bypassing the OCR step, in order to s
peed up far from the lesion_time
                m=0
                while m <800:
                    success, image = vidObj.read() #Success is a boolean to see if the read
ing was good. image is a numpy array (572,572,3)
                    m=m+1

            if (time_read_in_picture.time().hour*3600+time_read_in_picture.time().minute*60+
time_read_in_picture.time().second) < ((lesion_time[k].hour * 3600 + lesion_time[k].minute *
60 + lesion_time[k].second) *0.97):
                #This function accelerates the reading by bypassing the OCR step, in order to sp
eed up far from the timestamp, and is the one that remains when we approach the lesion_time
                m=0
                while m <80:

```

```

        success, image = vidObj.read() #Success is a boolean to see if the re
ading was good. image is a numpy array (572,572,3)
        m=m+1

    elif rand_if_true==True:#Enters the mode of seeking healthy pictures
        try:#We make a directory for every healthy set needed
            os.mkdir(full_path + '\\random_frames_lesions_excluded %d' % int(((lesion_time[k
].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second)))
            os.chdir(full_path + '\\random_frames_lesions_excluded %d' % int(((lesion_time[k
].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second)))
        except:
            try:# or use the existent if already there
                os.chdir(full_path + '\\random_frames_lesions_excluded %d' % int(((lesion_ti
me[k].hour * 60 + lesion_time[k].minute) * 60 + lesion_time[k].second)))
            except:
                pass

        #We seek the time of the last frame
        vidObj = cv2.VideoCapture(video_path) #This opens a videocaptura object, to read the
video
        vidObj.set(2, 1)#This sets the video at the end
        success, image = vidObj.read() #Success is a boolean to see if the reading was good.
image is a numpy array (572,572,3)
        height, width, channel=image.shape#As said, the fomrat is channel-last
        image=image[top:bottom,left:right,:] #We perform a selection of the upper-
left corner (where the time is)
        text =image_to_string(image,config ='--psm 6')#We execute the OCR software
        time_of_last_frame=datetime.datetime.strptime(text, '%H:%M:%S')# We convert it to d
atetime format

        #Finds the first time of the video
        vidObj = cv2.VideoCapture(video_path) #We reset the object
        vidObj.set(1, 1) #We set it at the beggining
        success, image = vidObj.read() #Success is a boolean to see if the reading was good.
image is a numpy array (572,572,3)
        height, width, channel=image.shape#As said, the fomrat is channel-last
        image=image[top:bottom,left:right,:]#We perform a selection of the upper-
left corner (where the time is)
        text =image_to_string(image,config ='--psm 6')#We execute the OCR software
        time_of_first_frame=datetime.datetime.strptime(text, '%H:%M:%S')# We convert it to d
atetime format
        vidObj.release()#We release the video

```

```

    image_raw=[] # We will use this variable to store a copy of the image if we may want
to save it later.

    max_video_time= (time_of_last_frame.time().hour*3600+time_of_last_frame.time().minut
e*60+time_of_last_frame.time().second) #We define the max video time in integrer
    min_video_time=(time_of_first_frame.time().hour*3600+time_of_first_frame.time().minu
te*60+time_of_first_frame.time().second)#We define the min video time in integrer

    mu, sigma = (max_video_time-min_video_time)/2 , (max_video_time-
min_video_time)/4 # we create normal parameters for randomizing
    random_times=np.random.normal(mu, sigma, int(random_frames))#we create a random set,
but we have to test that it is inside the video and also not in lesions. We do this whole f
unction for every lesion index.
    random_times=random_times.tolist()#We change the format to list
    is_in_range=False #This boolean controls that the set of random data is inside the t
imes of the video and not in lesions

    while not(is_in_range): #Looks for a normal distribution that fits the real timestam
ps of the video and not lesions
        is_good_random=[]#This will be a list of booleans, each boolean will be True if
the generated time is OK with all the conditions.
        for random_time in random_times: #Walks through the random times to analyse if t
hey are OK
            is_good_random.append(random_time>min_video_time) #Checks that the times cre
ated randomly are not before the beggining of the video
        for random_time in random_times:#Walks through the random times to analyse if th
ey are OK
            is_good_random.append(random_time<max_video_time)#Checks that the times crea
ted randomly are not after the end of the video

        if all(is_good_random) :#If nothing is outside this first filter of being inside
the video, it continues with this set
            for random_time_0 in random_times:#It reads all the random times
                random_time=datetime.datetime.strptime((str(datetime.timedelta(seconds=i
nt(random_time_0)))), '%H:%M:%S')#Changes the random time to proper format for comparison wit
h the OCR software
                random_time_to_sec= random_time.time().hour*3600+random_time.time().minu
te*60+random_time.time().second #Changes the random time to proper format for comparison wit
h the OCR software
                for lesion_to_avoid in lesion_time: #Now walks through all the lesions
                    min_lesion_margin= int(lesion_to_avoid.hour * 3600 + lesion_to_avoid
.minute * 60 + lesion_to_avoid.second *0.98)#Creates a safety margin of 2% from the lesion t
imestamp

```

```

        random_time_to_sec=random_time.time().hour*3600+random_time.time().minute*60+random_time.time().second #Changes the random time to proper format for comparison with the OCR software

        max_lesion_margin=lesion_to_avoid.hour * 3600 + lesion_to_avoid.minute * 60 + lesion_to_avoid.second +frames_detected*20 #This is done to avoid taking images that are not in the initial timestamp of the lesion but closely after, it depends on the number of images detected by the endoscopist

        is_good_random.append((random_time_to_sec < min_lesion_margin) or (random_time_to_sec >max_lesion_margin))#Saves the boolean condition in our verifier list of booleans

    if all(is_good_random) :#If our random set of data has passed all the conditions

        is_in_range=True #We have an optimal set of data
    else: #In case we failed along the way, we just give it another chance resetting it all

        is_good_random=[]#This will be a list of booleans, each boolean will be True if the generated time is OK with all the conditions.
        random_times=np.random.normal(mu, sigma,int( random_frames))#We reset and generate a new issuing of random data
        random_times=random_times.tolist() #We change the format to list

    else: #In case we failed along the way, we just give it another chance resetting it all

        is_good_random=[]#This will be a list of booleans, each boolean will be True if the generated time is OK with all the conditions.
        random_times=np.random.normal(mu, sigma, int(random_frames))#We reset and generate a new issuing of random data
        random_times=random_times.tolist() #We change the format to list

    u=0 #counts the number of frames
    for random_time_0 in random_times: #walks through the good randomized times we have
        u=u+1#counts the number of frames
        is_found=False#reset the life bit that looks for a match
        time_read_in_picture=datetime.datetime.strptime('00:00:00', '%H:%M:%S')#resets the OCR reading

        random_time=datetime.datetime.strptime((str(datetime.timedelta(seconds=int(random_time_0))), '%H:%M:%S')#modifies random time to proper format

        vidObj.release() #Releases the video
        vidObj = cv2.VideoCapture(video_path) #Creates again the video object

        while (not(is_found)):#We walk the video until we find the random time we have created, in the video (i.e. Finding the image)

```

```

        if time_read_in_picture >= random_time: #True when we arrive at the image t
hat we were looking for

            try:#We write to the drive the image found
                cv2.imwrite(patient_folder +str(int(random_time))+ '.jpg' , image_ra
w)#writes the image
                print('Writing....' + text + '    frame:    ' + str(u) + ' in ' +
str(int(len(random_times))) )

            except:#Or we verbalize that there is probably already an image there
                print('Not printed because probably already there')

            is_found=True #We change this variable to stop the while and trigger the
next random time we need

            good_read=False #We reset the good read before the next read
            while not(good_read):#waits for a good read on the OCR software, happens 1 o
n 10.000 times that we don't have a good read
                success, image = vidObj.read() #Success is a boolean to see if the readi
ng was good. image is a numpy array (572,572,3)
                image_raw = image.copy() #We save a copy for preserving the original pic
ture

                text=image_to_string(image[top:bottom,left:right,:],config = '--
psm 6') ##We execute the OCR software and perform a selection of the upper-
left corner (where the time is)
                good_read=False#We reset the good read before the next read
                try:
                    time_read_in_picture=datetime.datetime.strptime(text, '%H:%M:%S')#We
convert it to datetime for useful comparison

                    good_read=True#We state that we did a good read
                except:
                    good_read=False#If the read failed

                print(str(patient_folder) +'Seeking time....'+ text + '    to ' +str(random
_time.time())+ '    lesion_is:' +str(lesion_time[k]))#Verbosity
                text_register.append(text) #Registers the date read (string text)

            if (time_read_in_picture.time().hour*3600+time_read_in_picture.time().minute
*60+time_read_in_picture.time().second) < ((random_time.hour * 3600 + random_time.minute * 6
0 + random_time.second) *0.96):

```

```
#This function accelerates the reading by bypassing the OCR step, in order
to speed up far from the random_time
m=0
while m <800:
    success, image = vidObj.read() #Success is a boolean to see if the
reading was good. image is a numpy array (572,572,3)

    m=m+1
    continue
    if (time_read_in_picture.time().hour*3600+time_read_in_picture.time().minute
*60+time_read_in_picture.time().second) <((random_time.hour * 3600 + random_time.minute * 60
+ random_time.second) *0.99):
        #This function accelerates the reading by bypassing the OCR step, in order
to speed up far from the lesion_time
        m=0
        while m <40:
            success, image = vidObj.read() #Success is a boolean to see if the
reading was good. image is a numpy array (572,572,3)

            m=m+1

return
```

13.4. Code of MNIST model applied to lesion detection

```

import keras
from keras.models import Sequential
from tensorflow.python.keras.callbacks import TensorBoard
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy
import datetime # in order to save logs with date
%load_ext tensorboard
from sklearn.model_selection import train_test_split # to split the data
between train and validation
from PIL import Image # in order to reshape images
from keras.models import Sequential#imported in order to build the model
el
from keras.layers import Dense#imported in order to build the model
from keras.layers import Dropout#imported in order to build the model
from keras.layers import Flatten#imported in order to build the model
from keras.layers.convolutional import Conv2D#imported in order to build
the model
from keras.layers.convolutional import MaxPooling2D#imported in order
to build the model
from keras.utils import np_utils#imported in order to build the model
from keras import backend as K#imported in order to build the model
from tensorboard import notebook#imported in order to analyse the model
l
from keras.layers import Dense#imported in order to build the model
from keras.utils.vis_utils import plot_model #imported in order to plot
the model

# create generator, which we are not going to use for anything but fetch
the data from the directory
datagen = ImageDataGenerator()
# prepare the iterator for gathering all the data in one fetch with all
the data
data = datagen.flow_from_directory('/content/drive/My Drive/CNN_data_M
AY_with extra_false_close', class_mode='binary',target_size=(512, 512
), batch_size=1968)
# we create x and y as each a numpy array of lesions and of healthy

x, y = data.next()#This two operations will create our data appropriately
y for the model. We split train and validation with 10% of data for testing
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.1) # This is the percentage of test pictures we
want

```



```
new_shape=(28,28,3) #This is the final input shape we need
x_train_new = numpy.empty(shape=(x_train.shape[0],)+new_shape) #we use
this variable to copy there the new resized variable
for idx in range(x_train.shape[0]): #We walk through all the pictures
in the tensor
    x_train_new[idx] = numpy.array(Image.fromarray(x_train[idx], 'RGB')
.resize((28,28))) #we resize it

x_test_new = numpy.empty(shape=(x_test.shape[0],)+new_shape)#we use th
is variable to copy there the new resized variable
for idx in range(x_test.shape[0]):#We walk through all the pictures in
the tensor
    x_test_new[idx] = numpy.array(Image.fromarray(x_test[idx], 'RGB').r
esize((28,28))) #we resize it

del x_train #we delete variables to save RAM memory
del x_test #we delete variables to save RAM memory
del x #we delete variables to save RAM memory
del y #we delete variables to save RAM memory
x_train=x_train_new
x_test=x_test_new

num_classes = 2 # We have 2 classes: with lesion or healthy
#We code the labels of class in binary categorical format with 2 posit
ions:
y_train2 = keras.utils.to_categorical(y_train, num_classes)
y_test2 = keras.utils.to_categorical(y_test, num_classes)

K.set_image_data_format('channels_last') # We set as channels last the
image data format(batch, height, width, channels)

model = Sequential() #We start the model

# Function in which we define the model:
def baseline_model():

# Entry layer:
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=( 28, 28, 3), activation=
'relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
```

```
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
# Output layer(softmax):
    model.add(Dense(2, activation='softmax'))
# Compilation of the model:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

# Call to the model:
model = baseline_model()

#log_dir = "/content/drive/My Drive/Tensorflow_logs/logs" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") # Uncomment if we are interested in differentiating log runs
log_dir = "/content/drive/My Drive/Tensorflow_logs/logs"
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1, update_freq='batch', profile_batch=0)
logs=log_dir

#We now fit the model with our training data:
model.fit(x_train, y_train2, validation_data=(x_test, y_test2), epochs=200, batch_size=180, verbose=2, callbacks=[tensorboard_callback])

#We evaluate with the test data:
scores = model.evaluate(x_test, y_test2, verbose=1)
print("Exactitud del modelo: %.2f%%" % (100*scores[1]))

plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

%tensorboard --logdir '/content/drive/My Drive/Tensorflow_logs/logs'
```

13.5. Code of VGG16 pre-trained with feature extraction and classifier

```
import keras#imported in order to build the model
from keras.preprocessing.image import ImageDataGenerator#imported in order to build the data in RAM
from keras.models import Sequential#imported in order to build the model
from keras.layers import Dense, Dropout, Activation, Flatten#imported in order to build the model
from keras.layers import Conv2D, MaxPooling2D#imported in order to build the model
from keras import optimizers
import os # for saving pictures
from sklearn.model_selection import train_test_split# to split the data between train and validation
from sklearn import datasets
from sklearn import svm
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.engine import input_layer
from tensorflow.python.keras.callbacks import TensorBoard
from keras.preprocessing.image import ImageDataGenerator
import numpy #For picture and tensor operation
import datetime # in order to save logs with date
%load_ext tensorboard
from PIL import Image # in order to reshape images if needed
from keras.utils import np_utils#imported in order to build the model
from keras import backend as K#imported in order to build the model
from tensorboard import notebook#imported in order to analyse the model
from keras.utils.vis_utils import plot_model #imported in order to plot the model
from sklearn.model_selection import GridSearchCV #library to search the best parameters of the classifier
from keras.models import clone_model
from keras.utils.vis_utils import plot_model#imported in order to view results
from sklearn.externals import joblib# to save the model
from sklearn.metrics import classification_report #imported in order to view results
from sklearn.metrics import confusion_matrix #imported in order to view results
from sklearn.metrics import plot_confusion_matrix#imported in order to view results
import matplotlib.pyplot as plt#imported in order to view results
```

```
#We create a generator
datagen = ImageDataGenerator()
# prepare an iterators for each dataset
data = datagen.flow_from_directory('/content/drive/My Drive/CNN_data_M
AY_with_extra_false_close', class_mode='binary',target_size=(512, 512
), batch_size=1987)
#test = datagen.flow_from_directory('/content/drive/My Drive/paracnn/T
est', class_mode='binary',target_size=(512, 512))
# confirm the iterator works
x, y = data.next()
```

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.4)
num_classes = 2
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model.h5'

print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Have it as an array of integers:
y_test1 = numpy.argmax(y_test,axis=1)
y_train1 = numpy.argmax(y_train,axis=1)
model2 = VGG16(include_top=False, weights='imagenet', input_tensor=inp
ut_layer.Input(shape=(512, 512, 3)),pooling='avg')
plot_model(model2, to_file='model_plot.png', show_shapes=True, show_la
yer_names=True)
```

```
feat1_test = model2.predict(x_test)
feat1_train = model2.predict(x_train)
```

```
parameters = {'kernel': ('linear', 'rbf', 'poly', 'sigmoid'), 'C': numpy.l
inspace(0.1, 10000000000, 50).tolist(), 'class_weight': ['balanced'], 'gam
ma': ['scale', 'auto']}
svc = svm.SVC(probability=True)

clf_search=GridSearchCV(svc,
                        parameters,
                        scoring='balanced_accuracy',
                        n_jobs=-1,
                        refit=True,
                        cv=None, #None, to use the default 5-fold cross validatio
n,
                        verbose=10,
                        pre_dispatch='2*n_jobs',
                        return_train_score=True,
                        )

clf_search = clf_search.fit(feat1_train, y_train1)

sorted(clf_search.cv_results_.keys())
print(clf_search.best_estimator_)
```

```
joblib.dump(clf_search, '/content/drive/My Drive/best_tfidf.pkl')

# Load
clf_search = joblib.load('/content/drive/My Drive/best_tfidf.pkl')

clf2=clf_search

feat1_test = np.squeeze(feat1_test)

y_pred2 = clf2.predict(feat1_test)

a=plot_confusion_matrix(clf2,feat1_test, y_test1,display_labels=data.c
lass_indices,
                        cmap=plt.cm.Blues,
                        normalize=None,
                        values_format="10g"
                        )

print(classification_report(y_test1, y_pred2, target_names=data.class_
indices))
print(confusion_matrix(y_test1, y_pred2),)

i=0
import cv2
path_negative=r'/content/drive/My Drive/False_negative'
path_positive=r'/content/drive/My Drive/False_positive'
for image in x_test:
    if y_test[i]==1 and y_pred2=0:
        cv2.imwrite(path_negative+'/' + str(i) + '.jpg',x_test[i])
    i=i+1
i=0
for image in x_test:
    if y_test[i]==0 and y_pred2=1:
        cv2.imwrite(path_positive+'/' + str(i) + '.jpg',x_test[i])
```

13.6. Code of VGG pre-trained with fully connected training

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import optimizers
import numpy as np
import os

import numpy as np from sklearn.model_selection import
train_test_split from sklearn import datasets from sklearn import svm
# example of progressively loading images from file from
keras.preprocessing.image import ImageDataGenerato
```

```
# create generator
datagen = ImageDataGenerator()
# prepare an iterators for each dataset
#data = datagen.flow_from_directory('/content/drive/My Drive/CNN_data_28_april_500vs1300', class_mode='binary',target_size=(512, 512), batch_size=1825) THIS ONE IS GOOD
data = datagen.flow_from_directory('/content/drive/My Drive/CNN_data_MAY_with_extra_false_close', class_mode='binary',target_size=(512, 512), batch_size=1968)
#test = datagen.flow_from_directory('/content/drive/My Drive/paracnn/Test', class_mode='binary',target_size=(512, 512))
# confirm the iterator works
x, y = data.next()
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
num_classes = 2
num_predictions = 20

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
#y_validate=keras.utils.to_categorical(y_validate, num_classes)

# Have it as an array of integers:
y_test1 = np.argmax(y_test,axis=1)
y_train1 = np.argmax(y_train,axis=1)
#y_validate1=np.argmax(y_validate,axis=1)
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np
from keras.engine import input_layer
import keras
```



```

for model in model2.layers[:2]:
    model.trainable=False
model2.summary()
odel2.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['binary_accuracy'])

model2.fit(x_train, y_train,
          batch_size=100, #109 y 10 epochs
          epochs=20,
          # We pass some validation for
          # monitoring validation loss and metrics
          # at the end of each epoch
          validation_data=(x_test, y_test))

```

```

from keras.models import Sequential from keras.models import
clone_model from keras.layers import Dense from keras.utils.vis_utils
import plot_model plot_model(model2, to_file='model_plot.png',
show_shapes=True, show_layer_names=True)

```

```

model2.save('/content/drive/My
Drive/vgg16_fully_connected_for_lime.h5')

```

```

import tensorflow as tf new_model =
tf.keras.models.load_model('/content/drive/My
Drive/vgg16_fully_connected_for_lime.h5')

new_model=model2

```

```

!pip install lime

```

```

import keras
import lime
from lime import lime_image
image_number=2
#image_for_lime=x_train[image_number]
#class_output=y_train1[image_number]

image_for_lime=x_test[image_number]
class_output=y_test1[image_number]

```

```

explainer = lime_image.LimeImageExplainer()

```

```

explanation = explainer.explain_instance(image_for_lime, new_model.predict,
dict, top_labels=50, hide_color=0, num_samples=1000)

```

```
from keras.models import Sequential
from keras.layers import Dense, Activation
#Get back the convolutional part of a VGG network trained on ImageNet
model_vgg16_conv = VGG16(include_top=False,
                        weights='imagenet',
                        input_tensor=input_layer.Input(shape=(512, 512, 3))
                        #pooling='max'
                        )
# as first layer in a Sequential model

from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.layers import Input, Flatten, Dense
from keras.models import Model
import numpy as np

#Create the input format
input = Input(shape=(512, 512, 3), name = 'image_input')

#Use the generated model
output_vgg16_conv = model_vgg16_conv(input)

#Add the fully-connected layers
u = Flatten(name='flatten')(output_vgg16_conv)
u = Dense(4096, activation='relu', name='fc1')(u)
u = Dense(4096, activation='relu', name='fc2')(u)
u = Dense(2, activation='softmax', name='predictions')(u)

#Create your own model
model2 = Model(input=input, output=u)

#In the summary, weights and layers from VGG part will be hidden, but
they will be fit during the training
model2.summary()

#Then training with your data !
```

```

for model in model2.layers[:2]:
    model.trainable=False
model2.summary()
model2.compile(optimizer='rmsprop',
               loss='categorical_crossentropy',
               metrics=['binary_accuracy'])

model2.fit(x_train, y_train,
          batch_size=100, #109 y 10 epochs
          epochs=20,
          # We pass some validation for
          # monitoring validation loss and metrics
          # at the end of each epoch
          validation_data=(x_test, y_test))

from keras.models import Sequential from keras.models import
clone_model from keras.layers import Dense from keras.utils.vis_utils
import plot_model plot_model(model2, to_file='model_plot.png',
show_shapes=True, show_layer_names=True)

model2.save('/content/drive/My Drive/vgg16_fully_connected_for_lime.h5
')
import tensorflow as tf
new_model = tf.keras.models.load_model('/content/drive/My Drive/vgg16_
fully_connected_for_lime.h5')

# Check its architecture
new_model.summary()

new_model=model2

!pip install lime
import keras
import lime
from lime import lime_image
image_number=2
#image_for_lime=x_train[image_number]
#class_output=y_train1[image_number]

image_for_lime=x_test[image_number]
class_output=y_test1[image_number]
explainer = lime_image.LimeImageExplainer()

explanation = explainer.explain_instance(image_for_lime, new_model.pre
dict, top_labels=50, hide_color=0, num_samples=1000)

```

```
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt
```

```
temp, mask = explanation.get_image_and_mask(class_output, positive_only=True, num_features=2000, hide_rest=False)
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask)/255)
```

```
temp, mask = explanation.get_image_and_mask(class_output, positive_only=False, num_features=60, hide_rest=False) #feature 60 is OK at v3! #touching the number of features is the key
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask)/255)
```

```
path= r'/content/drive/My Drive/Lime_results/v3'
```

```
import os
import cv2
import random
from skimage.segmentation import mark_boundaries
i=0
seed=random.randint(1,10000)
for indexed in y_test1:
    print(indexed)
    if indexed ==1:
        image_for_lime=x_test[i]
        class_output=y_test1[i]
        explainer = lime.lime_image.LimeImageExplainer()

        explanation = explainer.explain_instance(image_for_lime, new_model.predict, top_labels=50, hide_color=0, num_samples=1000)
        #temp, mask = explanation.get_image_and_mask(class_output, positive_only=False, num_features=2000, hide_rest=False,min_weight=0.1)
        temp, mask = explanation.get_image_and_mask(class_output, positive_only=False, num_features=60, hide_rest=False) #feature 60 is OK at v3! #touching the number of features is the key

        saved_image=mark_boundaries(temp/2 +0.5,mask)
        saved_image = cv2.cvtColor(saved_image, cv2.COLOR_BGR2RGB)
        image_for_lime = cv2.cvtColor(image_for_lime, cv2.COLOR_BGR2RGB)
        cv2.imwrite(path+'/' +str(seed) + '_number__' +str(i)+ '.jpg', saved_image)
        cv2.imwrite(path+'/' + str(seed) + 'onumber__' +str(i) + '.jpg', image_for_lime)
        i=i+1
```