



**Kaunas University of Technology**  
Faculty of Electrical and Electronics Engineering

**Technical University of Cartagena**  
School of Industrial Engineering

# **Automatic Diabetic Foot Wound Detection Based on Computer Vision Methods**

Bachelor's Final Degree Project

---

**Alberto Hita Esquer**

Project author

**Prof. Dr. Vidas Raudonis**  
**Assoc Prof. Dr. Miguel Almonacid Kroeger**

Supervisors

---

**Cartagena, Kaunas 2022**



**Kaunas University of Technology**  
Faculty of Electrical and Electronics Engineering

**Technical University of Cartagena**  
School of Industrial Engineering

# **Automatic Diabetic Foot Wound Detection Based on Computer Vision Methods**

Bachelor's Final Degree Project  
Industrial Electronics and Automation Engineering (5071)  
Intelligent Robotics Systems (6121EXO013)

---

**Alberto Hita Esquer**

Project author

**Prof. Dr. Vidas Raudonis**  
**Assoc Prof. Dr. Miguel Almonacid**  
**Kroeger**

Supervisors

**Lect Dr. Rimvydas Simutis**

Reviewer

---

**Cartagena, Kaunas 2022**



**Kaunas University of Technology**  
Faculty of Electrical and Electronics Engineering  
**Technical University of Cartagena**  
School of Industrial Engineering  
Alberto Hita Esquer

# **Automatic Diabetic Foot Wound Detection Based on Computer Vision Methods**

Declaration of Academic Integrity

I confirm the following:

1. I have prepared the final degree project independently and honestly without any violations of the copyrights or other rights of others, following the provisions of the Law on Copyrights and Related Rights of the Republic of Lithuania, the Regulations on the Management and Transfer of Intellectual Property of Kaunas University of Technology (hereinafter – University) and the ethical requirements stipulated by the Code of Academic Ethics of the University;
2. All the data and research results provided in the final degree project are correct and obtained legally; none of the parts of this project are plagiarised from any printed or electronic sources; all the quotations and references provided in the text of the final degree project are indicated in the list of references;
3. I have not paid anyone any monetary funds for the final degree project or the parts thereof unless required by the law;
4. I understand that in the case of any discovery of the fact of dishonesty or violation of any rights of others, the academic penalties will be imposed on me under the procedure applied at the University; I will be expelled from the university and my final degree project can be submitted to the Office of the Ombudsperson for Academic Ethics and Procedures in the examination of a possible violation of academic ethics.

Alberto Hita Esquer

**APPROVED BY:**

KTU Faculty of Electrical and Electronics Engineering

Head of the Department of Automation

Assoc. prof. dr. Gintaras Dervinis

2022 05 04

**TASK OF FINAL PROJECT OF UNDERGRADUATE (BACHELOR) STUDIES**

<b>Issued to the Student:</b>	<i>Alberto Hita Esquer</i>	Group ERB8/2
<b>1. Project Subject:</b>		
Lithuanian Language:	Automatinis diabetinės pėdos žaizdų aptikimas taikant kompiuterinės regos metodus	
English Language:	Automatic Diabetic Foot Wound Detection Based on Computer Vision Methods	

Approved 2022 April . 29d. Decree of Dean Nr. *V25-03-10*

**2. Goal of the Project:** To develop and investigate the classification method that can be used for automatic diabetic foot classification

**3. Specification of Final Project:** The work must meet the methodological requirements for the preparation of final projects for the KTU Faculty of Electrical and Electronics Engineering.

**4. Project's Structure.** *The content is concretized together with supervisor, considering the format of the final project, which is listed in 14 and 15 points of Combined Description of Preparation , Defence and Keeping of Final Projects Methodical Requirements*

- 4.1 Analyze the state-of-the-art literature about classification algorithms.
- 4.2 Provide detailed information about research object, i.e., diabetic foot.
- 4.3 Apply selected classification methods to foot images of known database.
- 4.4 Develop diabetic foot classification algorithm and test it.

**5. Economical Part.** *If economical substantiation is needed; content and scope is concretized together with supervisor during preparation of final projects*

none

**6. Graphic Part.** *If necessary, the following schemes, algorithms and assembly drawings; content and scope is concretized together with supervisor during preparation of final projects*

Show algorithms and mathematical reasoning

**5. This Task is Integral Part of Final Project of Undergraduate (Bachelor) Studies**

**6. The Term of Final Project Submission to Defense Work at a Public Session of Qualification Commission.** *until 2020-06-02*  
(date)

I received this task:	<i>Alberto Hita Esquer</i>	<i>2021-03-01</i>
	(student's name, surname, signature)	(date)
Supervisors:	<b>Assoc. Prof. Dr. Miguel Almonacid Kroeger</b> <b>Prof. Dr. Vidas Raudonis</b>	<i>2021-03-01</i>
	(position, name, surname, signature)	(date)

Alberto Hita Esquer. Automatic Diabetic Foot Wound Detection Based on Computer Vision Methods. Bachelor's Final Degree Project. Supervisors Prof. Dr. Vidas Raudonis Assoc and Prof. Dr. Miguel Almonacid Kroeger; Faculty of Electrical and Electronics Engineering, Kaunas University of Technology; School of Industrial Engineering, Technical University of Cartagena.

Study field and area (study field group): Electronics and automation engineering, technological science, robotic engineering

Keywords: Diabetic foot disease, ischemia, ulcer, Diabetic foot wound, recognition, detection, convolutional neural networks, computer vision.

Kaunas, Cartagena, 2022. P.80

### **Summary**

The objective of this final degree project is to develop a model able to detect automatically the presence or not of ischemia and infection in the wound produced by diabetic foot disease. The work consists of an introduction of the mentioned disease as well as some statistical data that motivates this work. Next, the reader is introduced to all the theoretical concepts used to carry out the mentioned task. The model based on computer vision methods will be deduced after analyzing different studies focused on diabetic foot ulcer detection. This method is exposed paying special attention to all stages from data collection to the results exposition which are later suitably analyzed. At the end of the project, the obtained conclusions and possible improvements are presented.

Alberto Hita Esquer. Detección automática de heridas en el pie diabético basada en métodos de visión artificial. Trabajo de Fin de Grado. Supervisores Prof. Dr. Vidas Raudonis Assoc y Prof. Dr. Miguel Almonacid Kroeger; Facultad de Ingeniería Eléctrica y Electrónica, Universidad Tecnológica de Kaunas; Escuela de Ingeniería Industrial, Universidad Politécnica de Cartagena.

Campo y área de estudio (grupo de campo de estudio): ingeniería electrónica y de automatización, ciencia tecnológica, ingeniería robótica.

Palabras clave: Enfermedad del pie diabético, isquemia, úlcera, herida del pie diabético, reconocimiento, detección, redes neuronales convolucionales, visión artificial.

Kaunas, Cartagena, 2022. P.80

### **Sumario**

El objetivo de este trabajo final de grado es desarrollar un modelo capaz de detectar de forma automática la presencia o no de isquemia e infección en la herida producida por la enfermedad del pie diabético. El trabajo consta de una introducción de la mencionada enfermedad así como algunos datos estadísticos que motivan este trabajo. A continuación, se introduce al lector en todos los conceptos teóricos utilizados para llevar a cabo la tarea mencionada. El modelo a desarrollar, basado en técnicas de visión artificial, se deducirá del análisis de diferentes estudios centrados en la detección de úlceras de pie diabético. Este método se expone prestando especial atención a todas las etapas desde la recogida de datos hasta la exposición de los resultados, que son debidamente analizados. Al final del proyecto se presentan las conclusiones obtenidas y las posibles mejoras.

Alberto Hita Esquer. Automatinis diabetinės pėdos žaizdų aptikimas taikant kompiuterinės regos metodus. Bakalauro studijų baigiamasis projektas. Vadovai Prof. Dr. Vidas Raudonis ir Prof. Dr. Miguel Almonacid Kroeger; Elektros ir elektronikos fakultetas, Kauno technologijos universitetas, Pramonės inžinerijos mokykla, Kartegenos technikos universitetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Automatikos ir elektronikos inžinerija, technologiniai mokslai, robotikos inžinerija.

Reikšminiai žodžiai: Diabetinė pėda, išemija, diabetinės pėdos žaizdos, atpažinimas, aptikimas, konvoliuciniai neuronų tinklai, kompiuterinė rega.

Kaunas, Kartagena, 2022. P.80

## **Santrauka**

Bakalauro baigiamojo projekto tikslas – sukurti kompiuterinės regos algoritmą skirtą automatiškai atpažinti žaizdas diabetinėje pėdoje ir nustatyti žaizdos atsiradimo priežastį (išemija arba infekcija). Projektą sudaro įvadas apie diabetinę pėdą, yra pateikiami statistiniai duomenys, apibrėžiantys šio projekto aktualumą ir svarbą. Toliau pristatomas siūlomas kompiuterinės regos algoritmas, jo realizavimo metodika bei eksperimentinio testavimo rezultatai. Projekto pabaigoje yra pateikiamo modeliavimo ir tyrimo rezultatus apibendrinančios išvados, naudotos literatūros sąrašas ir pagrindiniai priedai.

## Table of contents

List of figures.....	10
List of tables .....	12
List of abbreviations and terms .....	13
Introduction. ....	14
1. Diabetic foot disease.....	15
1.1. Causes.....	15
1.2. Complications.....	15
1.3. Classification .....	15
1.4. Diabetic foot disease over the world .....	16
1.5. Objective.....	18
2. State of the art.....	19
2.1. What is artificial intelligence?.....	19
2.1.1. Machine learning.....	19
2.1.2. Deep learning.....	19
2.2. Neural networks.....	20
2.2.1. Neurons and the brain.....	20
2.2.2. Model representation .....	21
2.2.3. How do neural networks work?.....	21
2.3. Computer vision .....	23
2.3.1. Digital images.....	23
2.3.2. Colour spaces.....	23
2.3.3. Image filtering .....	25
2.4. Convolutional neural networks.....	27
2.4.1. Convolutional process .....	29
2.5. Training process .....	30
2.5.1. Backpropagation algorithm .....	30
2.5.2. Epochs and batches.....	31
2.5.3. Optimizer and loss parameters .....	31
2.6. Metric parameters .....	32
2.7. Transfer Learning and fine-tuning.....	32
2.8. CNN architectures .....	33
2.8.1. ResNet .....	33
2.8.2. MobileNet.....	34
2.8.3. NasNet .....	35
2.8.4. Inception Net .....	36
2.8.5. Inception-ResNet V2 .....	36
2.8.6. EfficientNet .....	37
2.9. Diabetic Foot Ulcer (DFU) database.....	37
2.9.1. “Deep learning in diabetic foot ulcers detection: A comprehensive evaluation” by Moi Hoon Yap [1].....	38
2.9.2. “DFUC 2020: Analysis Towards Diabetic Foot Ulcer Detection” by Bill Cassidy [2] .....	39
2.9.3. DFUNet: Convolutional Neural Networks for Diabetic Foot Ulcer Classification by M. Goyal, N. D. Reeves, A. K. Davison, S. Rajbhandari, J. Spragg and M. H. Yap.[3].....	39



2.9.4. Recognition of ischaemia and infection in diabetic foot ulcers: Dataset and techniques by Manu Goyal, Neil D.Reeves, Satyan Rajbhandari, Naseer Ahmad, Chuan Wang and Moi Hoon Yap [4]	40
2.9.5. Summary.....	41
3. Working environment.....	43
3.1. Python.....	43
3.1.1. Tensorflow.....	43
3.1.2. NumPy.....	43
3.1.3. OpenCV.....	43
3.1.4. Scikit-learn.....	43
3.2. Google Colaboratory.....	43
4. Binary classification for infection and ischemia recognition.....	44
4.1. Dataset preparation.....	44
4.1.1. Infection imbalance solving.....	44
4.1.2. Ishcemia imbalance solving.....	45
4.1.3. Python code.....	46
4.2. Splitting the data.....	48
4.3. Data augmentation.....	48
4.4. Proposed network.....	50
4.5. Networks' results.....	53
4.5.1. Results for infection recognition.....	53
4.5.2. Results for ischemia recognition.....	58
4.6. Ensemble method for binary classification.....	63
4.6.1. Final metrics for infection recognition.....	63
4.6.2. Final metrics for ischemia recognition.....	64
4.7. Results overview.....	64
5. Multilabel classification for infection and ischemia recognition.....	67
5.1. Dataset preparation.....	67
5.1.1. Imbalance solving.....	67
5.1.2. Python code.....	68
5.2. Splitting the data.....	69
5.3. Data augmentation.....	69
5.4. Proposed networks.....	70
5.5. Network's results.....	72
5.6. Ensemble method.....	75
5.7. Results overview.....	76
Conclusions and results.....	78
Improvements.....	78
List of references.....	79
Information sources.....	81

## List of figures

<b>Fig. 1</b> University of Texas Wound Classification System. ....	16
<b>Fig. 2</b> Diabetic patients distribution. ....	16
<b>Fig. 3</b> Diabetic foot patients distribution. ....	17
<b>Fig. 4</b> Global causes of disability comparison. ....	17
<b>Fig. 5</b> AI hierarchy. ....	19
<b>Fig. 6</b> Schematic of a human neuron. ....	20
<b>Fig. 7</b> Human neuron vs artificial one. ....	21
<b>Fig. 8</b> Neural network representation. ....	22
<b>Fig. 9</b> Sigmoid function representation. ....	22
<b>Fig. 10</b> Left picture shows numeric representation of an image and the right one includes visual one. ....	23
<b>Fig. 11</b> RGB colour space. ....	24
<b>Fig. 12</b> Typical representation of HSV colour space. ....	25
<b>Fig. 13</b> Convolution operation on images. ....	25
<b>Fig. 14</b> Padding example ....	26
<b>Fig. 15</b> A: Normal image, B: Mean filter applied, C: Gaussian filter applied ....	26
<b>Fig. 16</b> Edge detection example ....	27
<b>Fig. 17</b> Computer vision solution implementation. ....	28
<b>Fig. 18</b> . Neural network limitations example. ....	28
<b>Fig. 19</b> Convolutional neural network representation. ....	29
<b>Fig. 20</b> Convolutional neural network representation. ....	30
<b>Fig. 21</b> Some optimizers comparison ....	31
<b>Fig. 22</b> ResNet sizes. ....	33
<b>Fig. 23</b> ResNet34 representation ....	34
<b>Fig. 24</b> . ResNet50 example block ....	34
<b>Fig. 25</b> MobileNet block. ....	35
<b>Fig. 26</b> Inception block. ....	36
<b>Fig. 27</b> . Database examples. ....	37
<b>Fig. 28</b> . Results obtained by the research. ....	38
<b>Fig. 29</b> . Performance of the benchmark algorithms. FRCNN represents Faster R-CNN. ....	39
<b>Fig. 30</b> . DFUNet and GoogLenet results. ....	40
<b>Fig. 31</b> . DFUNet architecture. ....	40
<b>Fig. 32</b> Performance of binary classification of infection. ....	41
<b>Fig. 33</b> Performance of binary classification of ischemia. ....	41
<b>Fig. 34</b> Dataset division for infection recognition. ....	44
<b>Fig. 35</b> Dataset distribution for ischemia recognition. ....	45
<b>Fig. 36</b> Data augmentation transformations. ....	49
<b>Fig. 37</b> Final CNN summary based on ResNet 101. ....	51
<b>Fig. 38</b> ResNet101 accuracy and loss. ....	53
<b>Fig. 39</b> . Testing data metrics and confusion matrix for ResNet101 ....	53
<b>Fig. 40</b> MobileNet accuracy and loss. ....	54
<b>Fig. 41</b> MobileNet Metrics obtained on testing set and confusion matrix ....	54
<b>Fig. 42</b> Inception-ResNet V2 accuracy and loss. ....	55
<b>Fig. 43</b> Inception-ResNet V2 metrics obtained on testing set and confusion matrix. ....	55

<b>Fig. 44</b> NasNet accuracy and loss. ....	56
<b>Fig. 45</b> NasNet metrics obtained on testing set and confusion matrix. ....	56
<b>Fig. 46</b> Inception accuracy and loss .....	57
<b>Fig. 47</b> Inception net metrics obtained on testing set and confusion matrix. ....	57
<b>Fig. 48</b> ResNet101 accuracy and loss.....	58
<b>Fig. 49</b> ResNet101 metrics obtained on testing set and confusion matrix .....	58
<b>Fig. 50</b> MobileNet accuracy and loss. ....	59
<b>Fig. 51</b> MobileNet metrics on testing set and confusion matrix. ....	59
<b>Fig. 52</b> Inception-ResNet V2 accuracy and loss .....	60
<b>Fig. 53</b> Inception-ResNet V2 metrics obtained on testing set and confusion matrix .....	60
<b>Fig. 54</b> NasNet accuracy and loss. ....	61
<b>Fig. 55</b> NasNet metrics obtained on testing set and confusion matrix. ....	61
<b>Fig. 56</b> Inception accuracy and loss. ....	62
<b>Fig. 57</b> Inception net metrics obtained on testing set.....	62
<b>Fig. 58</b> Ensemble method metrics obtained in testing set and confusion matrix. ....	63
<b>Fig. 59.</b> Ensemble method metrics obtained in testing set and confusion matrix. ....	64
<b>Fig. 60</b> Final metrics for hosted solution.....	64
<b>Fig. 61</b> Prediction examples for ischemia recognition. ....	65
<b>Fig. 62</b> Prediction examples for infection recognition. ....	66
<b>Fig. 63.</b> Dataset distribution for multilabel classification. ....	67
<b>Fig. 64</b> Final CNN summary based on MobileNetV2.....	71
<b>Fig. 65</b> MobileNet metrics for multilabel classification. ....	72
<b>Fig. 66</b> Confusion matrix for infection recognition on the left and for ischemia recognition on the right using MobileNetV2.....	72
<b>Fig. 67</b> NASNet Mobile metrics for multilabel classification.....	73
<b>Fig. 68</b> Confusion matrix for infection recognition on the left and for ischemia recognition on the right using NASNet Mobile.....	73
<b>Fig. 69.</b> NASNet Mobile metrics for multilabel classification.....	74
<b>Fig. 70</b> Confusion matrix for infection recognition on the left and for ischemia recognition on the right using EfficientNet. ....	74
<b>Fig. 71</b> Ensemble method metrics for multilabel classification. ....	76
<b>Fig. 72</b> Confusion matrix for infection recognition on the left and for ischemia recognition on the right using ensemble method.....	76
<b>Fig. 73</b> Results for multilabel classification solution. ....	77
<b>Fig. 74</b> Examples for multilabel recognition task. ....	77
<b>Fig. 75</b> Results compare with the model size.....	78

## List of tables

<b>Table 1.</b> Importing input images .....	46
<b>Table 2</b> Downloading Y labels .....	46
<b>Table 3</b> Separating input images for infection recognition.....	46
<b>Table 4</b> Separating input images for ischemia recognition.....	47
<b>Table 5.</b> Converting into a NumPy array for infection recognition.....	47
<b>Table 6</b> Converting into NumPy array for ischemia recognition.....	47
<b>Table 7.</b> Creating characteristic vector for infection recognition. ....	47
<b>Table 8</b> Creating characteristic vector for ischemia recognition .....	48
<b>Table 9</b> Data splitting.....	48
<b>Table 10</b> Creating data generator.....	50
<b>Table 11</b> Creating data augmentation iterator.....	50
<b>Table 12</b> Downloading the CNN. ....	50
<b>Table 13</b> Creating the complete net. ....	51
<b>Table 14</b> Model training.....	52
<b>Table 15</b> Ensemble method for binary classification algorithm .....	63
<b>Table 16</b> Importing data.....	68
<b>Table 17</b> Separating input into four groups .....	68
<b>Table 18</b> Creating input array for multilabel classification. ....	69
<b>Table 19</b> Creating characteristic vector for multilabel classification. ....	69
<b>Table 20</b> Data splitting for multilabel classification.....	69
<b>Table 21</b> Creating data generator for multilabel classification.....	70
<b>Table 22</b> Creating data augmentation iterator for multilabel classification.....	70
<b>Table 23</b> Downloading the CNN .....	70
<b>Table 24</b> Creating the complete net .....	70
<b>Table 25</b> Model training.....	71
<b>Table 26</b> Ensemble method for multilabel classification.....	75

## List of abbreviations and terms

### Abbreviations:

Assoc. prof. – associate professor;

Lect. – lecturer;

Prof. – professor.

CNN. – Convolutional neural networks

DFU – Diabetic Foot Ulcer

TP – True Positives

FP – False positives

TN – True negatives

FN – False negatives

GPU – Graphics Processing Unit

RAM – Random-access memory

AI – Artificial Intelligence

CPU- Central Processing Unit

TPU – Tensor processing unit

## **Introduction.**

Computer vision is becoming more and more important nowadays, and algorithms used are increasing their complexity and their performance. In this context, it is acceptable to think about challenging tasks carried out by humans which could be automatized by applying machine learning algorithms.

In this paper, computer vision methods will be used to extract valuable results in the recognition of the wound produced by diabetic foot disease. This syndrome produces ulcers and ischemia which are one of the most important causes of disability in the world, due to its severity and the lack of specialists to attend to all diabetic foot patients in the world.

This work will be based on creating a high-performance model able to detect diabetic foot wound stages. In addition, a small-size solution able to be used in any mobile device will be developed. Finally, both results will be compared.

## **1. Diabetic foot disease**

Diabetic foot is a disease that causes severe wounds mainly in feet and legs which could lead to the amputation of a foot, leg, or part of them.

### **1.1. Causes**

Diabetic foot is mainly caused because of two pathologies that are commonly suffered by diabetic patients. These two pathologies are:

- Neuropathy. This pathology most often appears in lower extremities but can affect any part of the body. It causes damage to foot muscles' innervations which leads to a lack of movement and consequently to a deformation of the affected muscles. This fact creates lumps or deformities which are likely to develop skin breakdown and ulceration. Due to the innervations damage patients usually don't feel any kind of pain so this wound deteriorates without being noticed.
- Peripheral vascular disease. This pathology is similar in diabetic and non-diabetic patients, however, in diabetic patients, it appears before and it develops quicker so at the end this disease could be quite more severe at advanced ages. It causes ischemia (deficient blood flow due to narrow and blocked arteries) and consequently wounds healing due to a lack of nutrients and oxygen carried by the blood.

### **1.2. Complications**

Injury infection is one of the most important things to avoid due to the patient impossibility to overcome it as the defense mechanisms are damaged due to neuropathy and ischemia. As blood is not blowing correctly, there is a lack of cells which are in charge of defeating bacteria which cause infection, and it develops until amputation is required in order to avoid infection growth.

### **1.3. Classification**

There are two major groups in which diabetic foot disease can be separated into. Making this classification is quite important because of the difference in therapeutic process as each type produces different kinds of risks for our body. The types are the following ones:

- The Neuropathic Foot where neuropathy dominates. It causes complications such as fissures, bullae, neuropathic (Charcot) joint, neuropathic edema, and digital necrosis.
- The Neuroischemic foot where the blood difficulty to flow is the main factor although neuropathy is present. It causes complications such as pain at rest, ulceration on foot margins, digital necrosis, and gangrene.

One of the most worldwide accepted classifications is the one shown in Fig.1.

### Wound classification system[11]

Stages	Description
Stage A	No infection or ischemia
Stage B	Infection present
Stage C	Ischemia present
Stage D	Infection and ischemia present

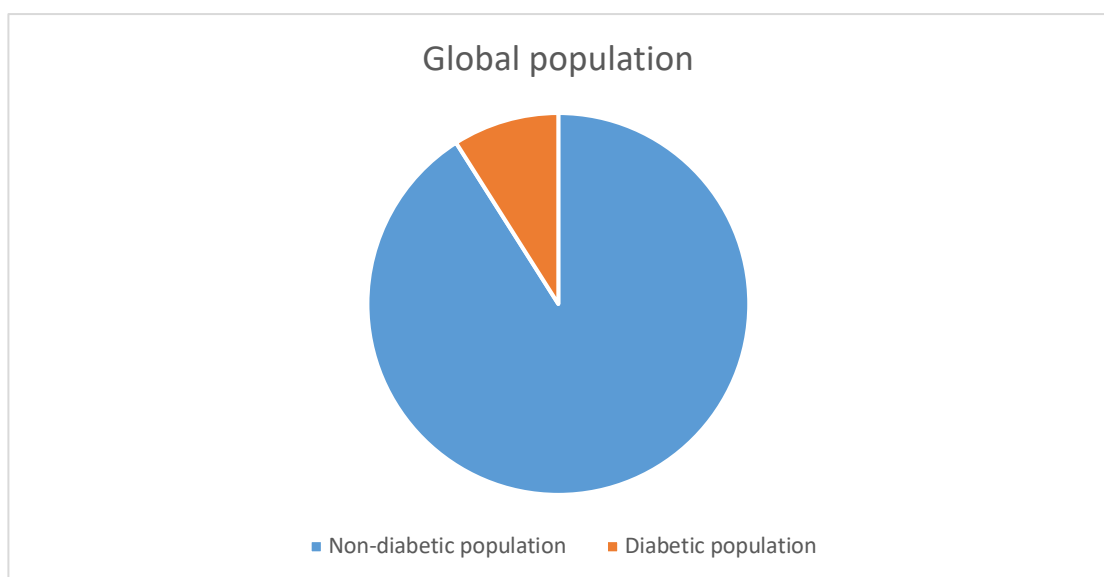
  

Grading	Description
Grade 0	Epithelialized wound
Grade 1	Superficial wound
Grade 2	Wound penetrates to tendon or capsule
Grade 3	Wound penetrates to bone or joint

**Fig. 1** University of Texas Wound Classification System.

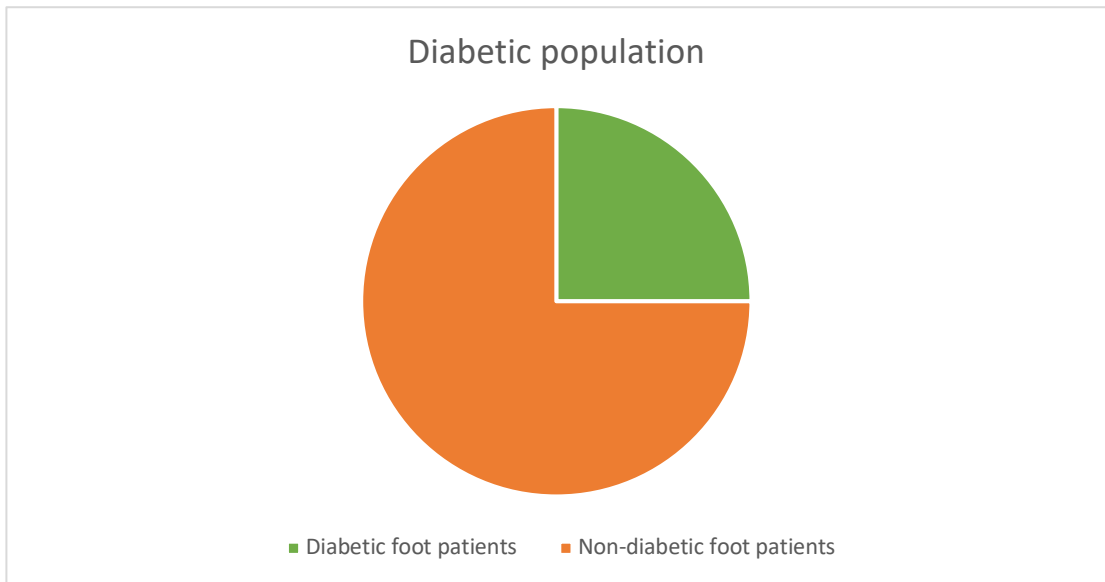
#### 1.4. Diabetic foot disease over the world

Diabetic foot disease is a quite common syndrome in the diabetic people. It is estimated that around 9% of the global population suffer from diabetes and around 25% of those patients are likely to suffer from diabetic foot syndrome. Fig. 2 and Fig. 3 are a graphical view of this fact.



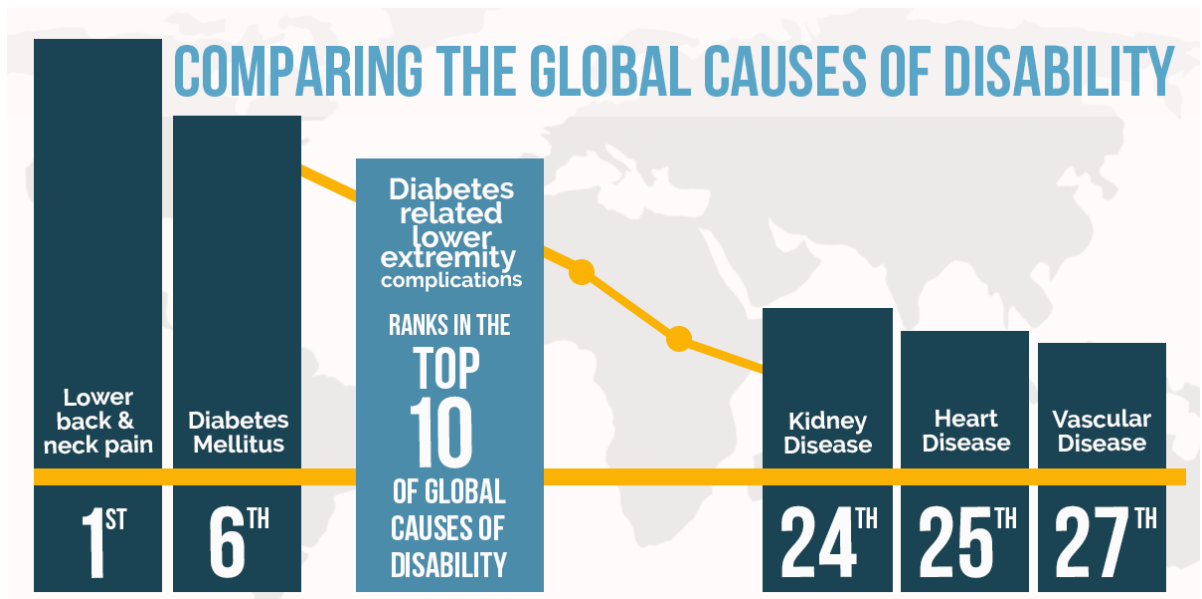
**Fig. 2** Diabetic patients distribution.





**Fig. 3** Diabetic foot patients distribution.

All data above, makes diabetic foot one of the most common disability causes over the world. Fig.4 shows another example of important causes of disabilities over the world.



**Fig. 4** Global causes of disability comparison.

In Africa, the standard of living is not as good as in other continents, this fact directly affects specialist doctor accessibility which is a significant fact in patients' recovery. To land this fact in numbers, in South Africa there are more than 4 million diabetic foot patients with only 200 specialists so each specialist should take care of 20,000 each. In addition, South Africa is one of the richest countries in Africa (top 3 GDP in Africa) with a huge social inequality so for the lower class this lack of specialists is even worse.

In more developed continents such as Europe or Asia, many specialist teams for ulcer and ischemia care have been created however this is not something entirely spread out and there are still many countries in these continents with signs of underdevelopment in this issue.

## **1.5. Objective**

It is a fact that there are many places in the world where diabetic foot specialists are not easily reachable. This supposes an additional risk factor for patients who suffer from this syndrome. Developing a tool that would be able to detect infection and ischemia and based on that given stage of the disease to the non-specialist doctors could make the difference in avoiding amputation. The purpose of the work is the development and comparison of different classification algorithms able to recognize the stage of the diabetic foot syndrome, that can be later used as a part of whole diabetic foot analysis tool able to recognize every stage and grade of the injury by detecting each part of the wound.

## 2. State of the art

In this section, the theoretical framework, in which this work is based, will be exposed. In addition, some papers about this issue will be reviewed.

### 2.1. What is artificial intelligence?

In 1950, Alan Turing, who is considered the father of computer sciences, came up with the following question: "Can machines think? ". It is considered the germ of artificial intelligence as its answer construction has created this huge scientific field.

One of the most worldwide spread artificial intelligence definitions is the one given by John McCarthy: "It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable".

#### 2.1.1. Machine learning

Machine learning is a huge field inside artificial intelligence, its goal is to imitate human way of thinking using large amounts of data, learning from them in a process called "training". Typical algorithms used in this field are regression models and decision trees. Machine learning models usually need the human intervention to learn as it is necessary to set input features to the algorithm so it can provide accurate predictions.

#### 2.1.2. Deep learning

By the year 2010, the improvement in computational performance let the first deep learning algorithms to be developed. Deep Learning aim is similar to machine learning but its algorithms have more complex structures which works well without pretraining human intervention, in other words, it can be assumed as a "scalable machine learning" as Lex Fridman propose in his notes. In this work we will focus on this type of algorithms, specifically in neural networks, that will be explained in the following section. In Fig. 5, hierarchy inside artificial intelligence can be appreciated.

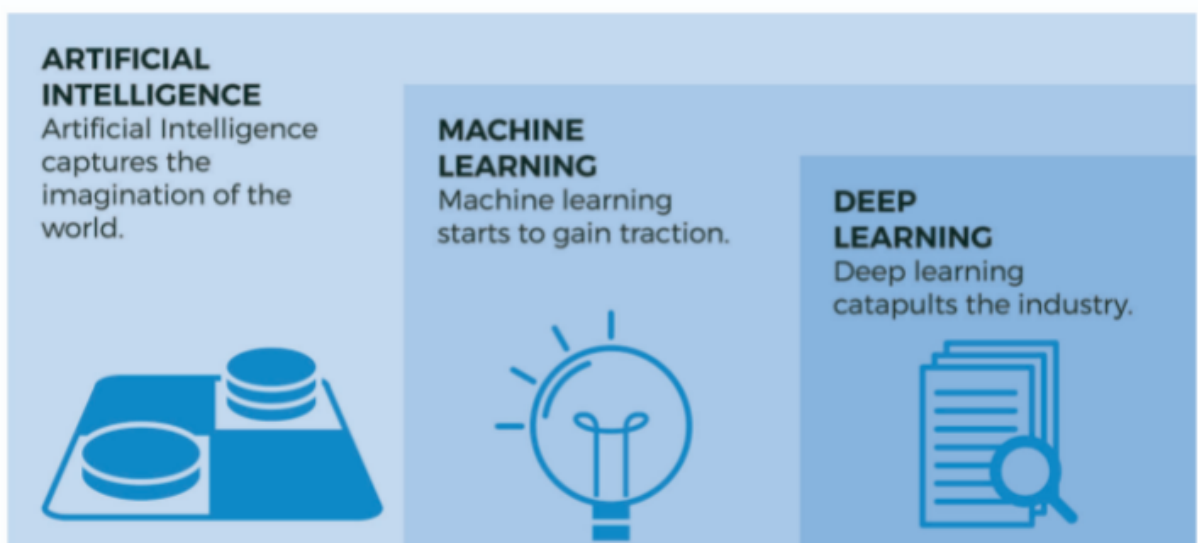


Fig. 5 AI hierarchy.

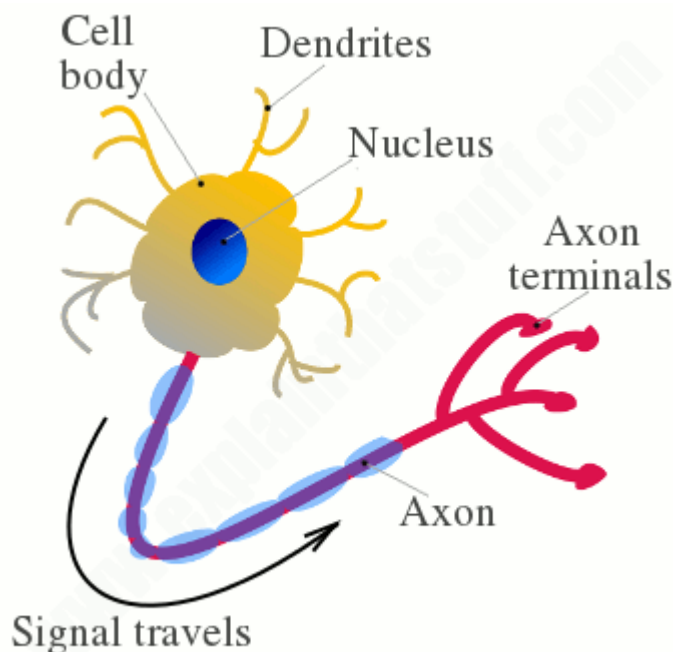
## 2.2. Neural networks

Neural networks are artificial imitations of how our brain works, nowadays they are one of the most useful AI algorithms because of computational performance improvements. They have become an important part inside machine learning algorithms and the main pillar in deep learning ones.

In human beings, brains are made up billions of neurons which are connected forming networks. This neural network can learn from different inputs and the same learning structure is used for every simple knowledge acquired. It has been proved that if we reconnect the optical nerve to the auditory cortex, this part of the brain would learn to see instead of listening. So, it is not about the algorithm is about the input data. Extrapolating this into the current field of study, we could create a unique algorithm which could be completely scalable to a huge range of tasks.

### 2.2.1. Neurons and the brain

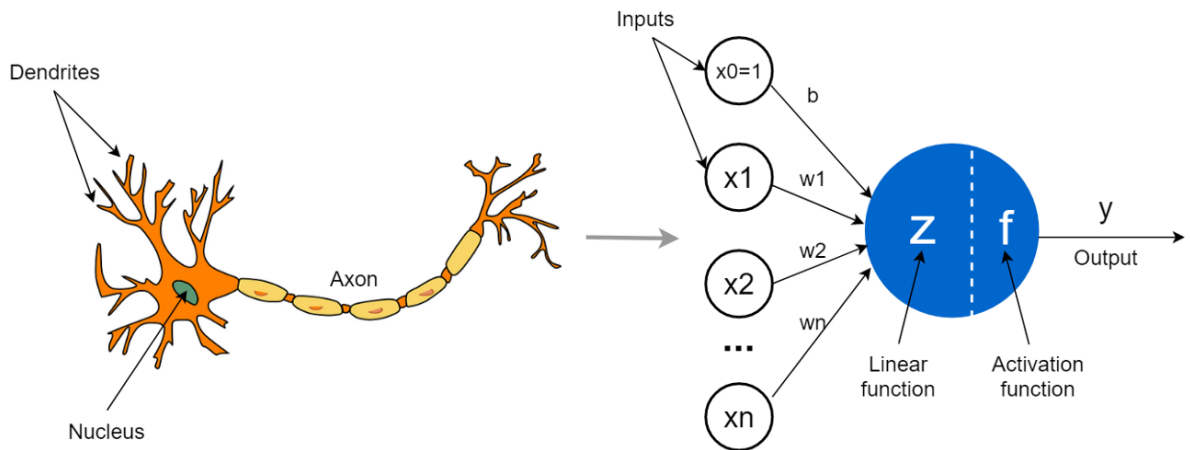
In a very basic way, brain neural networks are made up of billions of neurons, each neuron has three main parts: dendrites, cell body and the axon. The dendrites are the part which collect information from other neurons as electrical inputs. Based on that information, the axon will transmit an answer to other neurons. This answer can be active or inactive. In Fig. 6 the structure of a human neuron can be appreciated.



**Fig. 6** Schematic of a human neuron.

### 2.2.2. Model representation

The fundamental unit of the model is the artificial neuron, and it is an equivalent representation of the human one. These basic nodes join themselves forming layers, and several layers form the neural network. We will learn more about its structure in Fig. 7.



**Fig. 7** Human neuron vs artificial one.

As we can see we have four different parts:

- Inputs. The inputs are just the value of each node in the previous layer, it is important to say that every neuron receives information of every node in the predecessor layer.
- Linear function. Each node has associated a “weight” which is number that multiplies its value before sending the information to the next layer. The linear function is just the addition of every input multiply by its weight.
- Activation function. The activation function is what defines the output of the neuron depending on the value obtained from the linear function. There are multiple types of activation functions, even you can find different activation functions among the several layers of a neural network.
- Output. It is just the output value of the neuron; which could be sent to the next layer of the net (previously multiplied by the neuron weight) or it could be the final output of the network.

### 2.2.3. How do neural networks work?

In order to understand how neural networks work it is important to know the mathematical expression of linear and activation functions.

The following calculations will be referred to one node of the net and it is completely applicable to the rest of the artificial neurons.

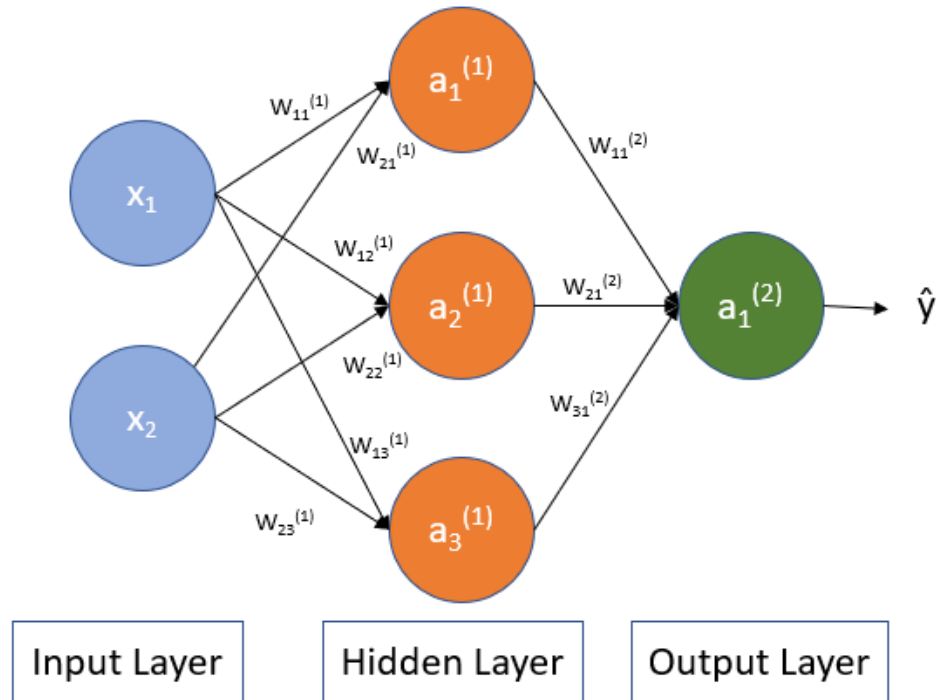
#### 2.2.3.1. Linear function

The linear function can be expressed as:

$$f(x) = \sum_{i=0}^n w_i x_i + w_{bias} * bias$$

On the one hand, as we can see there is a term called bias, it is a 1 value node and there is one per layer. It acts as input in the linear function and it has its own weight.

On the other hand, it is essential to notice that weights are not something characteristic of neurons, as a neuron has different weights for each node of the following layer as can be appreciated in Fig. 8.

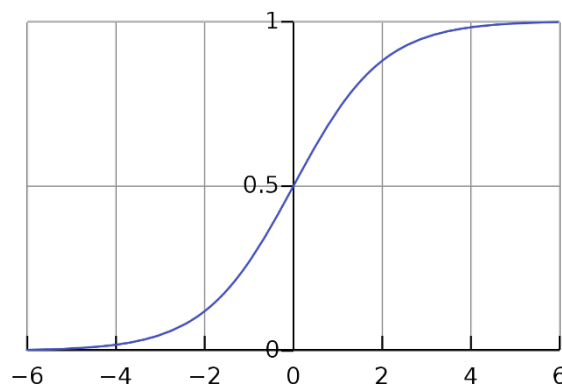


**Fig. 8** Neural network representation.

### 2.2.3.2. Activation function

As it is understood by its name, this function determines the level of activation of the neuron. Its dependent variable is defined by the linear function. There are many types of activation functions, even in a neural network more than one activation function can be applied. In this introduction to neural networks, sigmoid activation function intuition will be introduced as an example. It is very commonly applied to output neuron when a binary output is needed as it is very easy to convert the function output into 0 or 1 by a simple threshold setting as we will see in Fig. 9.

The function representation is the following one:



**Fig. 9** Sigmoid function representation.

Looking at the representation, we realise that the function values are represented by all real numbers between 0 and 1, so by setting a threshold of 0.5 ( it depends on the level of confidence we are looking for) we will obtain:

$$g(x) = \begin{cases} 1, & f(x) > 0.5 \\ 0, & f(x) < 0.5 \end{cases}$$

### 2.3. Computer vision

Computer vision is a scientific discipline (subfield of AI) based on applying machine learning techniques to digital images in order to obtain valuable information and predictions.

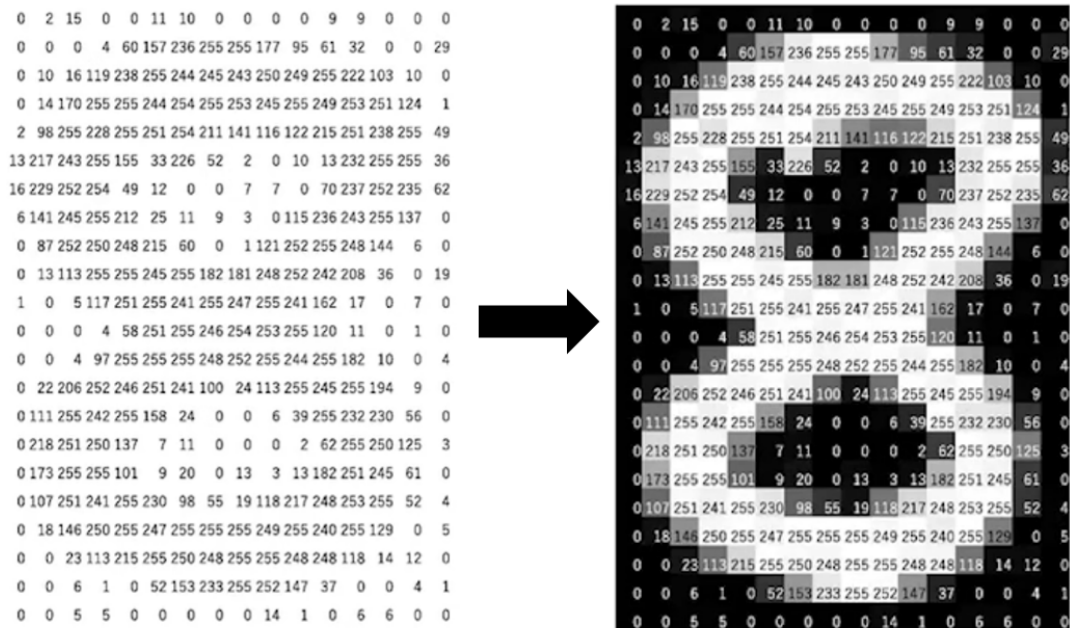
#### 2.3.1. Digital images

A digital image is numeric representation of an image based on one or more bidimensional matrices, whose values are called pixels. Usually, these values go from 0 to 255 ( $2^8$  levels of intensity), that is to say, images usually have a “color depth” of 8 bits. There are several ways to make a numerical representation of an image, attending to the number of bidimensional matrices and what each matrix represents. Each representation system is called “colour space”.

#### 2.3.2. Colour spaces

The following colour spaces are explained assuming that we are in front of 8 bits colour depth images, that is, values go from 0 to 255. This colour depth is the most common one, however we can find other ones, such as 10 bits color depth in which values go from 0 to 1024.

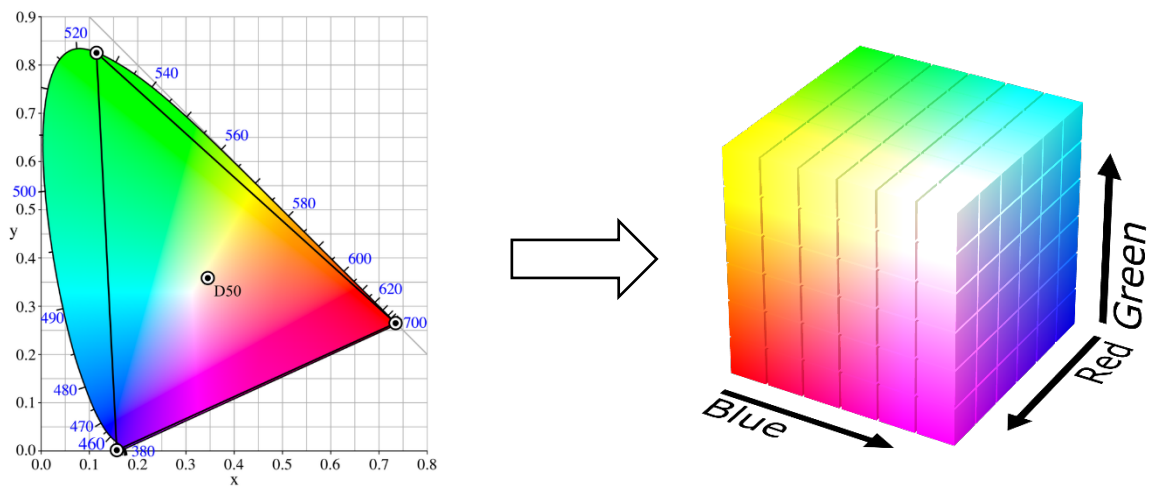
The simplest and the most ancient one is the “gray scale”. It is a one bidimensional matrix-based representation in which each pixel represents a grey value from 0 to 255, being 0 black and 255 white. An example of this it is shown in Fig. 10.



**Fig. 10** Left picture shows numeric representation of an image and the right one includes visual one.

At the beginning of computer vision, colour was not something that algorithms considered because of the computational costs that it would add. Nowadays, due to all the technological advances in CPUs, GPUs and TPUs, colour has achieved the position it deserves when it comes to represent image features. The two main colour spaces, when we are representing colour in digital images are RGB and HSV:

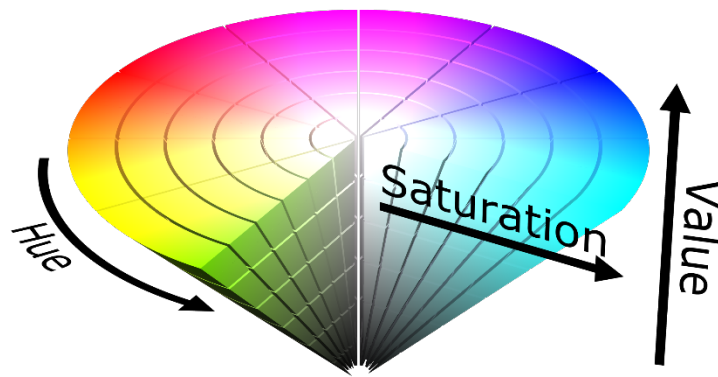
- RGB uses three bidimensional matrices to numerically represent images and their colour. These three matrices represent three color channels Red Green and Blue (RGB). By the combination of these three primary colours, it is possible to obtain any chromacity of the triangle defined by these colours, according Grassmann's law of light additivity. As we can see in Fig. 11, white colour is represented by the value (255,255,255).



**Fig. 11** RGB colour space.

- HSV also uses three bidimensional matrices, but these channels represent Hue, Saturation and Value (HSV). Hue represents the color inside the chromatic circle by the degree value which this colour occupies inside it. Usually, hue is represented from 0 to 180 so that the 360° of the chromatic circle can be represented by multiplying the value by two. Saturation is defined as the amount of gray which can be found inside a colour, so a value of 0 represents the gray colour and a 255 value represents the colour defined by the Hue information. Value defines the brightness of the colour, and it works together with the saturation parameter. Below, we can observe how colour changes according to mentioned parameters.



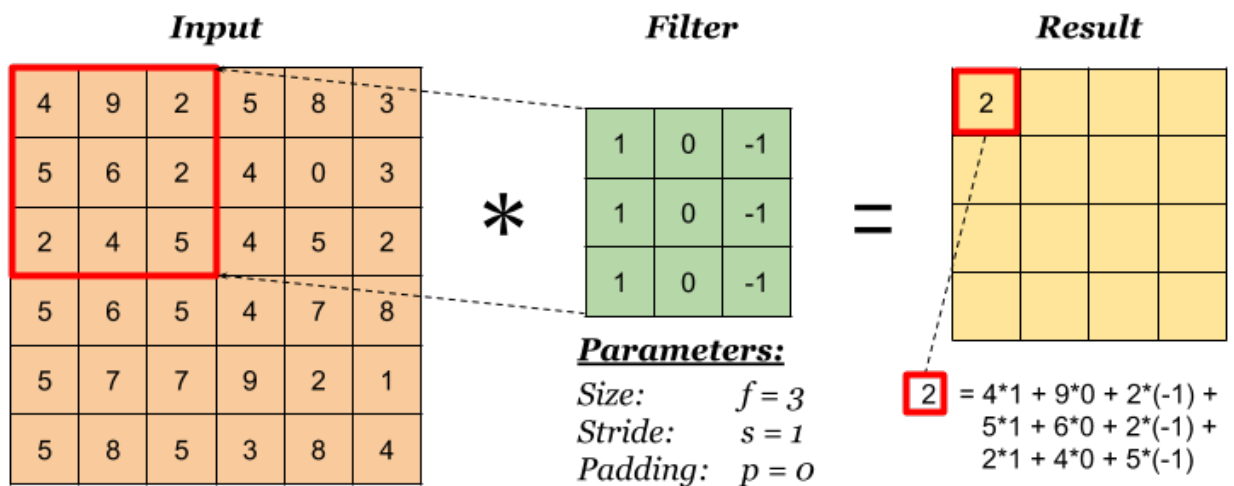


**Fig. 12** Typical representation of HSV colour space.

### 2.3.3. Image filtering

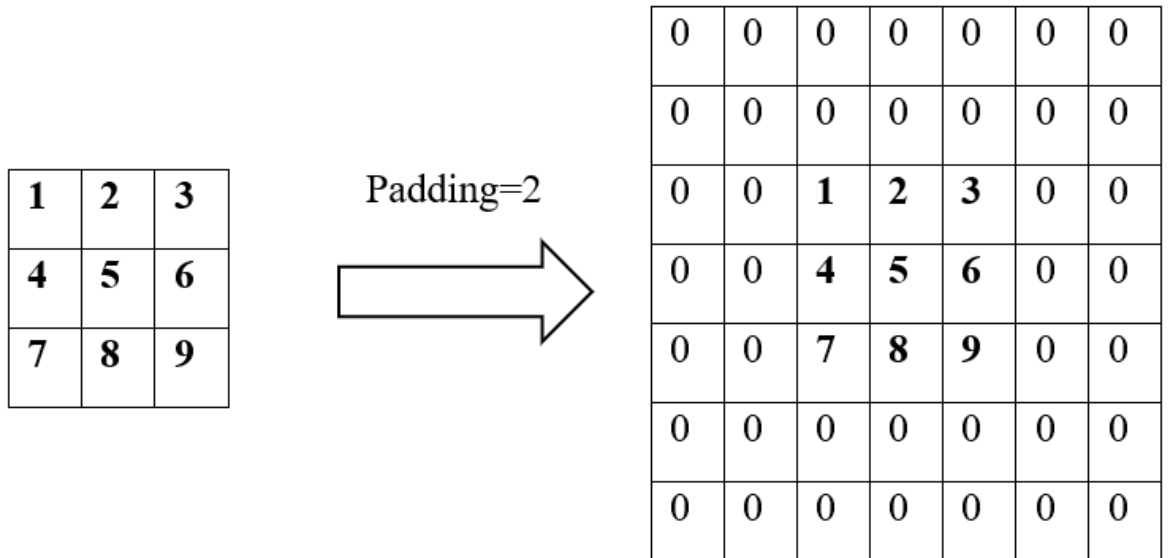
Image filtering is the base of modern image processing as it lets us to extract visual information of the image, such as textures or shapes, and to improve image quality by correcting imperfections. There are two types of image filters: linear and non-linear ones.

Linear filtering of an image is based on applying a convolution operation to the input image in order to obtain a new one. This operation is based on overlapping the input image over another matrix called filter or kernel, multiplying each cell by the inferior and adding the products. The symbol of convolution operation is (\*). Below, a graphic explanation of convolution on images is presented.



**Fig. 13** Convolution operation on images.

Filtering parameters are size, stride and padding. Size is usually 1x1, 3x3 or 5x5, but it can be of completely different dimensions. Stride refers to the “size of the jump of the filter”, that is, how many cells the filter will move forward for the next set of operations. Padding is a feature that adds empty pixels to the frame of the input image, as we can see in Fig.14, in order to control the output size.

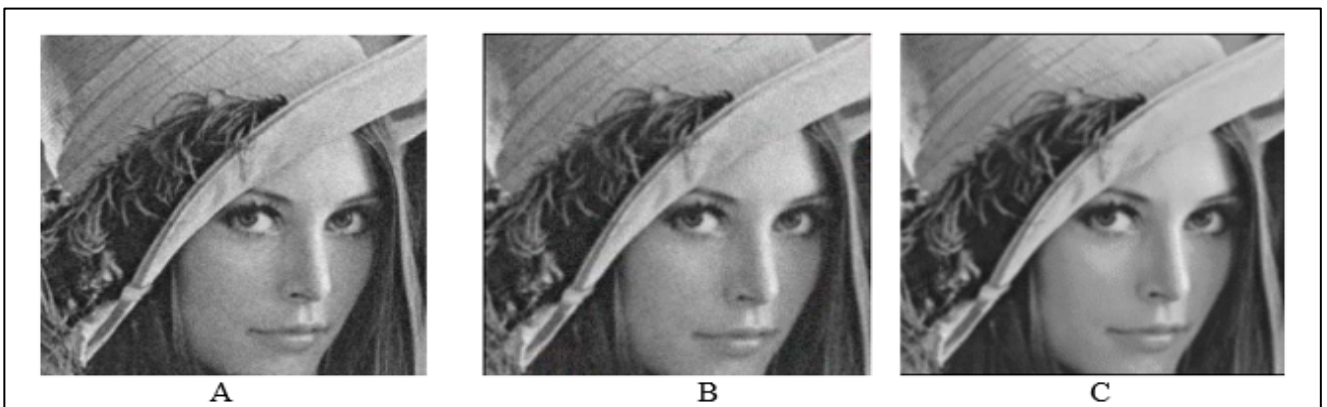


**Fig. 14** Padding example

Just to come up with an idea of what linear filters do, some examples will be shown. There are two important filters when it is necessary to reduce the noise in an image, which is the random intensity variation of the pixels. These two famous filters are the mean and the gaussian filter.

<table style="border: none;"> <tr> <td style="padding-right: 10px;">Mean filter:</td> <td style="padding-right: 10px;">1/9</td> <td style="padding-right: 10px;">1/9</td> <td style="padding-right: 10px;">1/9</td> <td style="padding-left: 20px;">(mean of the neighboring pixels)</td> </tr> <tr> <td></td> <td>1/9</td> <td>1/9</td> <td>1/9</td> <td></td> </tr> <tr> <td></td> <td>1/9</td> <td>1/9</td> <td>1/9</td> <td></td> </tr> </table>	Mean filter:	1/9	1/9	1/9	(mean of the neighboring pixels)		1/9	1/9	1/9			1/9	1/9	1/9	
Mean filter:	1/9	1/9	1/9	(mean of the neighboring pixels)											
	1/9	1/9	1/9												
	1/9	1/9	1/9												

<table style="border: none;"> <tr> <td style="padding-right: 10px;">Gaussian filter:</td> <td style="padding-right: 10px;">1</td> <td style="padding-right: 10px;">2</td> <td style="padding-right: 10px;">1</td> <td style="padding-left: 20px;">(approximation to gaussian distribution)</td> </tr> <tr> <td></td> <td>2</td> <td>4</td> <td>2</td> <td></td> </tr> <tr> <td></td> <td>1</td> <td>2</td> <td>1</td> <td></td> </tr> </table>	Gaussian filter:	1	2	1	(approximation to gaussian distribution)		2	4	2			1	2	1	
Gaussian filter:	1	2	1	(approximation to gaussian distribution)											
	2	4	2												
	1	2	1												



**Fig. 15** A: Normal image, B: Mean filter applied, C: Gaussian filter applied

Another important group of linear filters are the edge detection ones, which are filters able to detect changes in the intensity of neighbouring pixels. It is based on the derivative principle, as it is an operation which calculates the instantaneous rate of variation of a function.

$$\text{Derivative definition: } \lim_{h \rightarrow 0} \left( \frac{f(x+h) - f(x)}{h} \right)$$

$$\text{Discrete approximation: } \frac{f[n+1] - f[n]}{N}$$

Edge detection is about introducing filters inspired by discrete approximation of derivative since we can define digital images as discrete functions. One of the simplest kernels is:

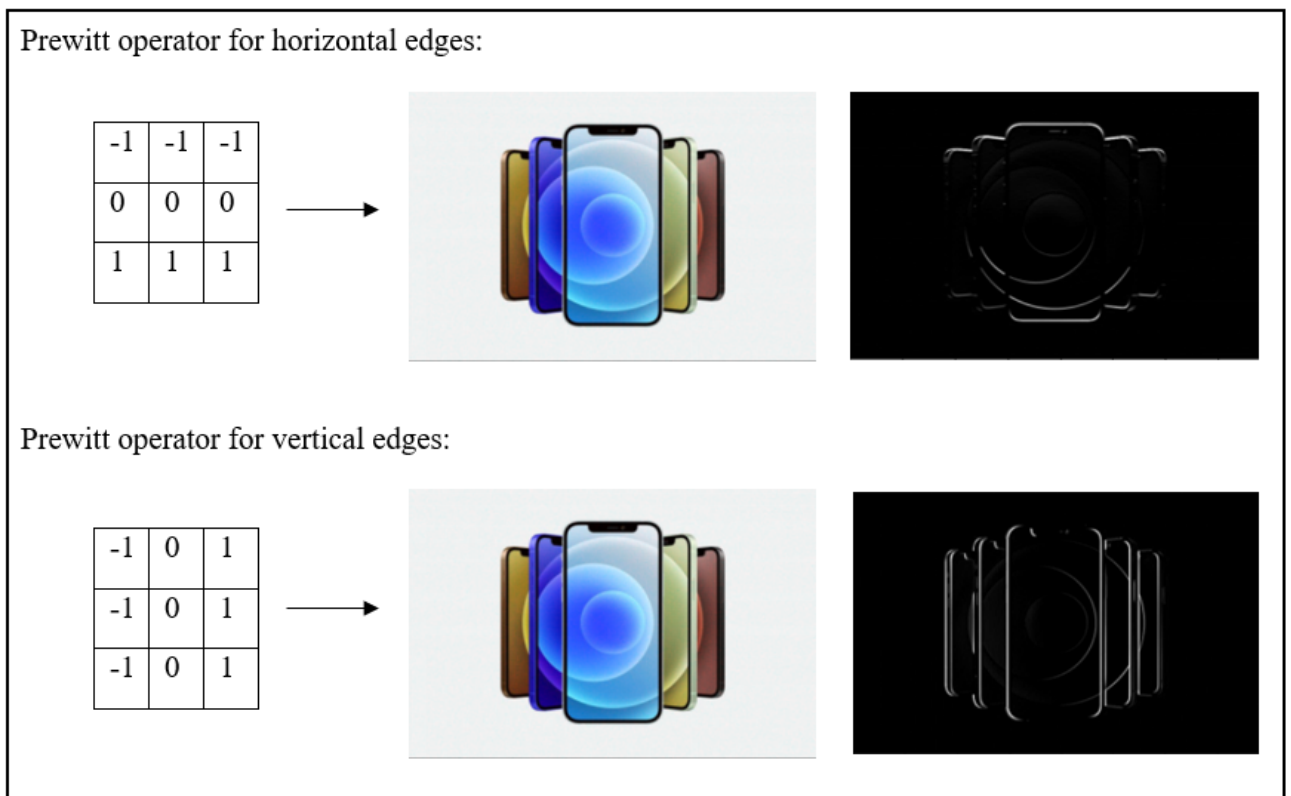
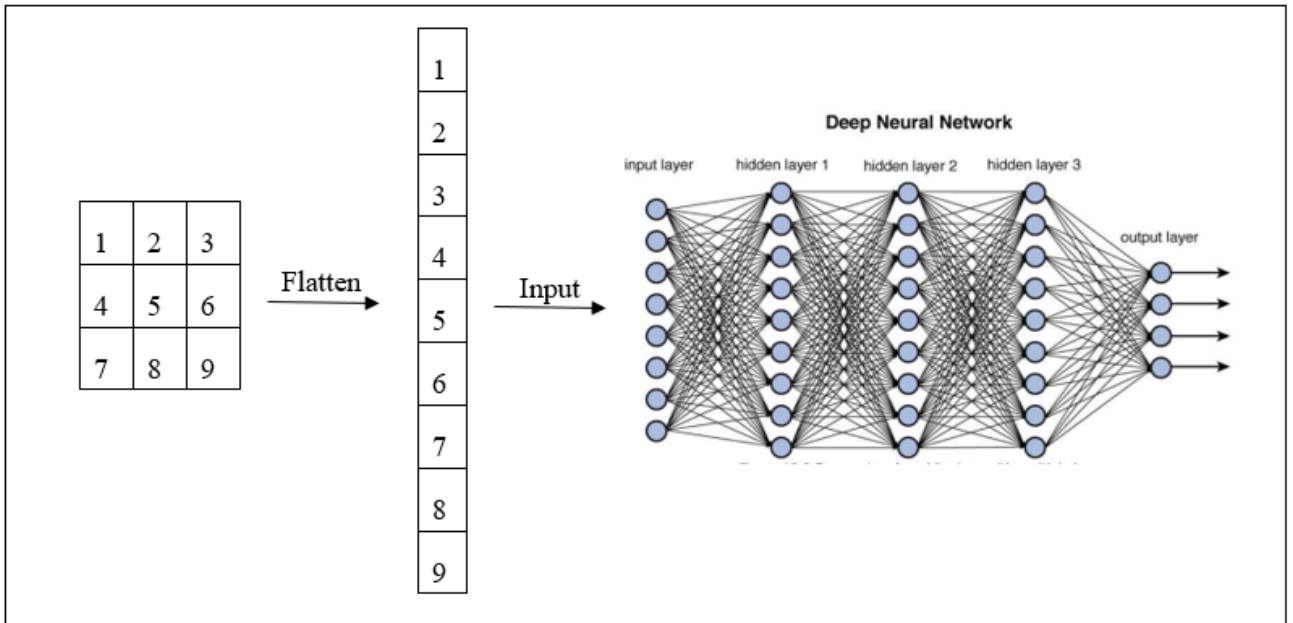


Fig. 16 Edge detection example

## 2.4. Convolutional neural networks

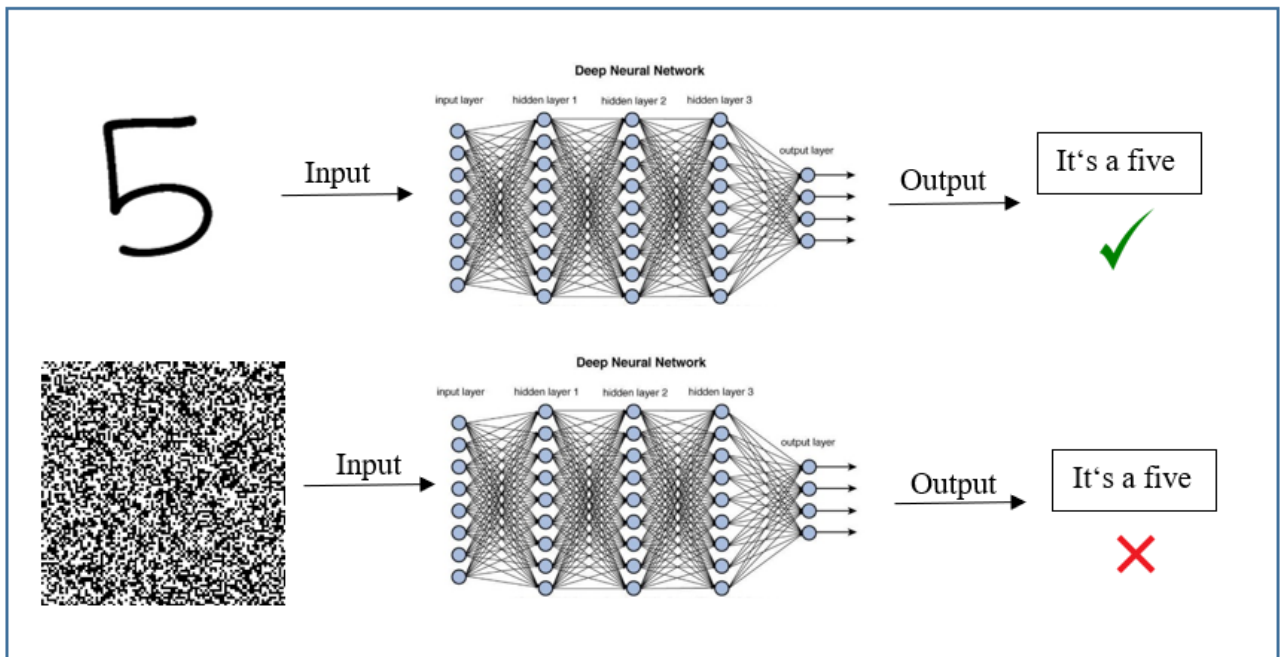
Convolutional neural networks constitute the key tool for deep learning algorithms and they are the base of computer vision, as they let us to “extract visual information of an image”. This last sentence seems to be trivial, but it is not. Using conventional neural networks, we have two ways of proceeding when we face a computer vision problem:

On the one hand, we can make a flatten operation in order to convert the matrix representation of an image into a flat vector and then introduce this vector into the neural network:



**Fig. 17** Computer vision solution implementation

This way of solving the problem is absolutely correct and it may work for simple situations. Unfortunately, is not a powerful solution since the neural network receives “random” pixels which only contribute its intensity value to the neural network, and it does not receive any visual information about the image. As an example, if we think about the typical character recognition problem solved by the proposed method, we could face the following problem:



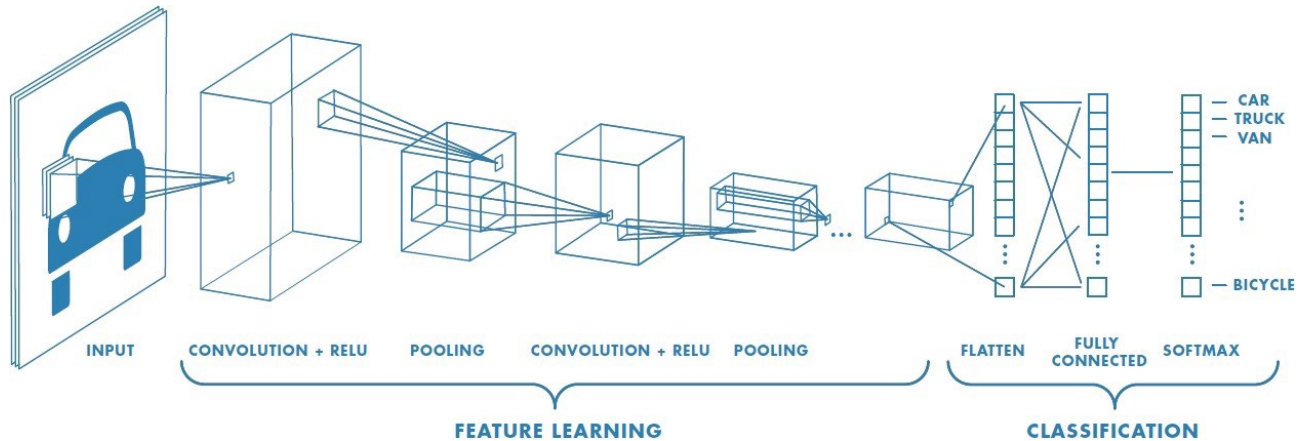
**Fig. 18 .** Neural network limitations example.

This happens because neural networks do not receive visual information, so, something that for a human beign is a completely random image for the neural network could represent a correct pattern of pixels which corresponds to number “5”.

On the other hand, we can face the computer vision problem by extracting manually the visual information using different filters and introducing this information into the neural network. This could

be a good solution for simple problems where you know exactly what the most representative parameters of each class are. However, this is not a scalable solution, as it needs human intervention to work.

Convolutional neural networks brought a solution for both problems. It consists of adding a convolutional block before the conventional neural network, which is called as “dense layer” in this context. This added block is composed of several filters in which kernel values are defined during the training process. This makes possible to extract visual information of the image while the algorithm remains scalable.



**Fig. 19** Convolutional neural network representation.

### 2.4.1. Convolutional process

#### Convolutional layer

It is the core block of this type of networks and where almost all the computation occurs. It is made up of an input which is an image represented in whatever colour space (grey scale, RGB, HSV, etc.), a variable number of undefined filters and an output whose depth is defined by each filter’s output.

The unsetting kernels/filters will carry out a convolutional operation as the one explained in the section before, and their values will be calculated during the training, minimizing the error of the neural networks output.

#### Pooling layer

It leads to the dimensionality reduction as it is important for the last stage of the convolutional process. There are two main types of pooling “max pooling” and “average pooling”.

- Max pooling uses a kernel that moves across the convolutional layer output picking the maximum value of all covered pixels by the kernel at the same time.
- Average pooling makes a convolutional operation over the convolutional layer output in order to get the average of all the values covered by the kernel.

As we can see in Fig.19, is it possible to concatenate different processes of “convolutional layer + pooling” in order to get a “deeper learning” of the input image.

## Fully-Connected layer

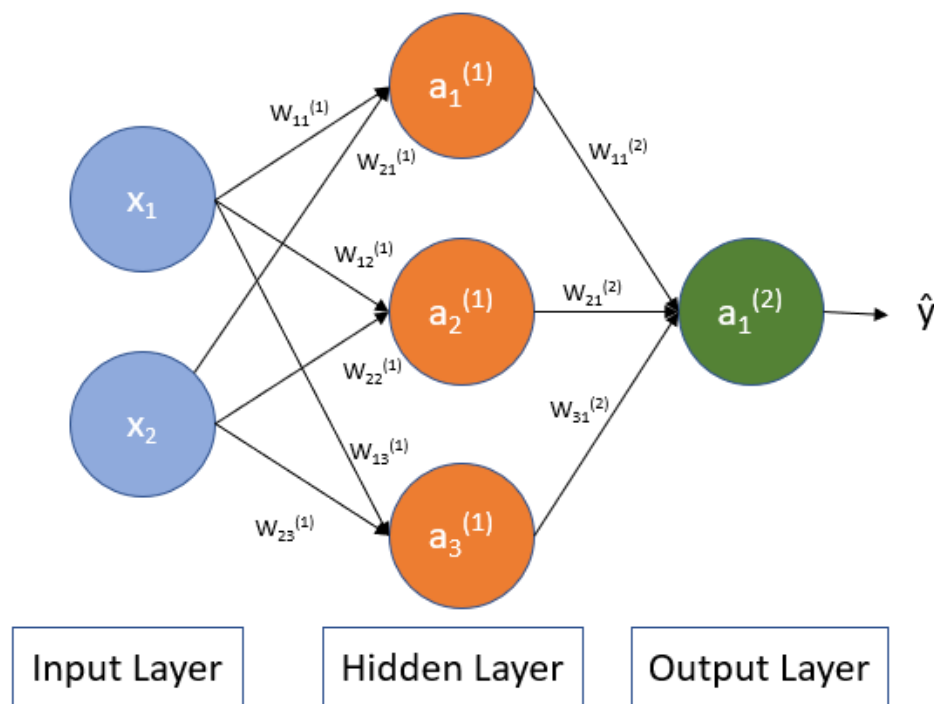
A flatten operation it is applied to the output of the last pooling operation. This one dimension array is “fully-connected” to the dense layer. Now we understand the importance of the pooling operation since reducing the dimension of convolution outputs is essential to connect matrices to the dense layer.

## 2.5. Training process

### 2.5.1. Backpropagation algorithm

How is the training carried out? This is an essential question which must be solved in order to understand all parameters that affect our training process and that we will have to declare.

Backpropagation algorithm is what makes possible training process. It became important in 1989 thanks to the paper “Learning representations by back-propagating errors” written by Rumelhart, Hinton and Williams. To understand this algorithm, it is important to keep in mind Fig. 20.



**Fig. 20** Convolutional neural network representation.

The process starts with a forward propagation process, this is what is explained in section 2.2.3. Inputs values are sent to hidden layer where linear and activation function are applied in each neuron, the obtained values go through the net following the same process until output layer value is calculated, in that moment, a hole forward propagation process has been carried out.

Next step is to evaluate the predicted output against the exected one. What measures the error between both outputs is called cost function and it could be a simple mean squared error or more complex functions such as cross-entropy one.

As mentioned in 1989 paper, backpropagation: “repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector”.

So, backpropagation is the process of going back through the net adjusting its weights in order to reduce the cost function. The level of adjustment of each parameter is given by the gradient of the cost function respect that parameter. When we are working with CNNs, backpropagation algorithm also adjusts kernels’ values in order to reduce cost function.

### 2.5.2. Epochs and batches

In the training process, the model doesn’t carry out the backpropagation algorithm either with a unique input data or all the data available. The data set is divided into batches, so, only when all inputs which constituted a batch have gone forward the net, the backpropagation algorithm will be carried out and the model weights will be actualized. Processing in batches helps to save memory and improve training speed.

Batch size is a parameter, defined by the model designer, which defines how many data inputs will constitute a batch. Typical batch sizes are 16, 32 o 64.

An epoch is when all dataset has gone forward and backward through the model. That is to say, when N batches of inputs have been processed, being:

$$N = \frac{\text{Total number of inputs}}{\text{batch size}}$$

### 2.5.3. Optimizer and loss parameters

An optimizer is the function used to adjust weights in backpropagation algorithm. There are a lot of optimizers such us gradient descent, stochastic gradient descent, Adam, etc. In this paper, Adam optimizer will be used as it is one of the most effective ones as it can be seen in Fig. 21.

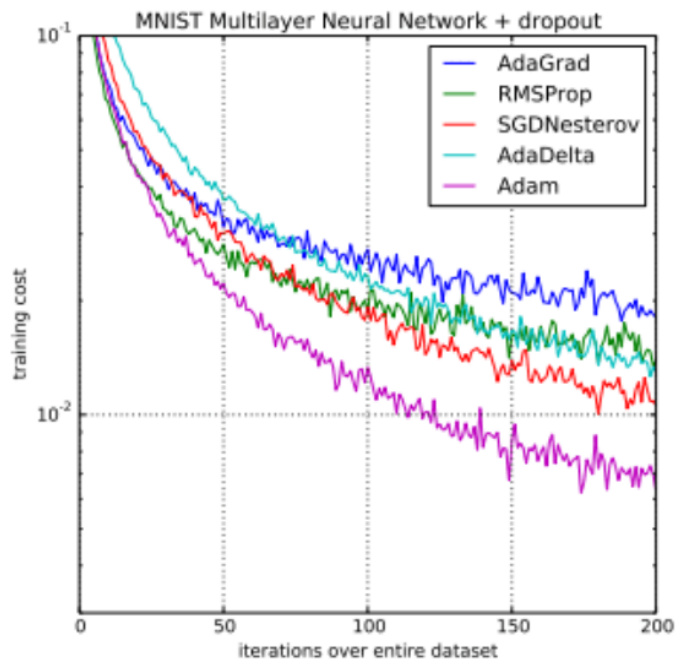


Fig. 21 Some optimizers comparison

Adam optimizer's particularity is that it updates the learning rate for each individual weight in the net.

Loss parameter defines the function used to express the cost (error) of the model. In this paper binary cross entropy will be used. On the one hand, binary cross entropy is the negative mean of the logarithms of corrected probabilities ( $p_c$ ). Being corrected probabilities, the probability of a prediction to belong to the expected output.

Binary cross entropy:

$$Cost(p_c) = -\frac{1}{N} \sum_{i=0}^N \log(p_c)$$

## 2.6. Metric parameters

To evaluate how good is a machine learning algorithm we can use different parameters to come up with a global idea of how the model is working on testing set.

- Accuracy: It is a measure of how many predictions our model had right.

$$Accuracy = \frac{TP + TN}{Total\ predictions}$$

- Recall: It is the number of true positives (TP) divided by the total number of elements which should belong to the positive class.

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

- Precision: It measures how many positive cases were predicted correctly.

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

- F-score. It is the harmonic mean between Precision and Recall

$$F - score = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

## 2.7. Transfer Learning and fine-tuning

Transfer learning is a deep learning method which is based on the use of a pre-trained CNN, with preserved weights from a previous training with a large and general dataset. This method follows the intuition that if a model is trained on a general enough dataset, it will be a good tool to obtain characteristics of the visual world and it will be able to solve different kind of problems by using these visual characteristics. There are two main ways of applying transfer learning:

- Feature extraction: Use the pretrained model to obtain characteristics from new examples, then a classifier is added at the top and it will be trained from scratch. So, the pretrained



weights of the CNN will be locked during the training process and only the classificatory weights will change their value.

- Fine tuning: This method unfreezes some layers of the pretrained model in order to train their weights at the same time of the classifier ones. This is computationally more expensive, but it obtains quite good results.

## 2.8. CNN architectures

When is time to face a computer vision problem it is not worth trying to create a specific CNN that fits the problem. There are many CNNs created by research teams which are tested and approved by all scientific community and will fit the problem to be faced.

### 2.8.1. ResNet

In CNN it is commonly proposed that “the deeper the better”, this makes sense as the models have more parameters to adapt to different visual situations. However, at a certain depth, it makes no sense in continuing increasing the number of layers because generalization becomes a complex task for the model.

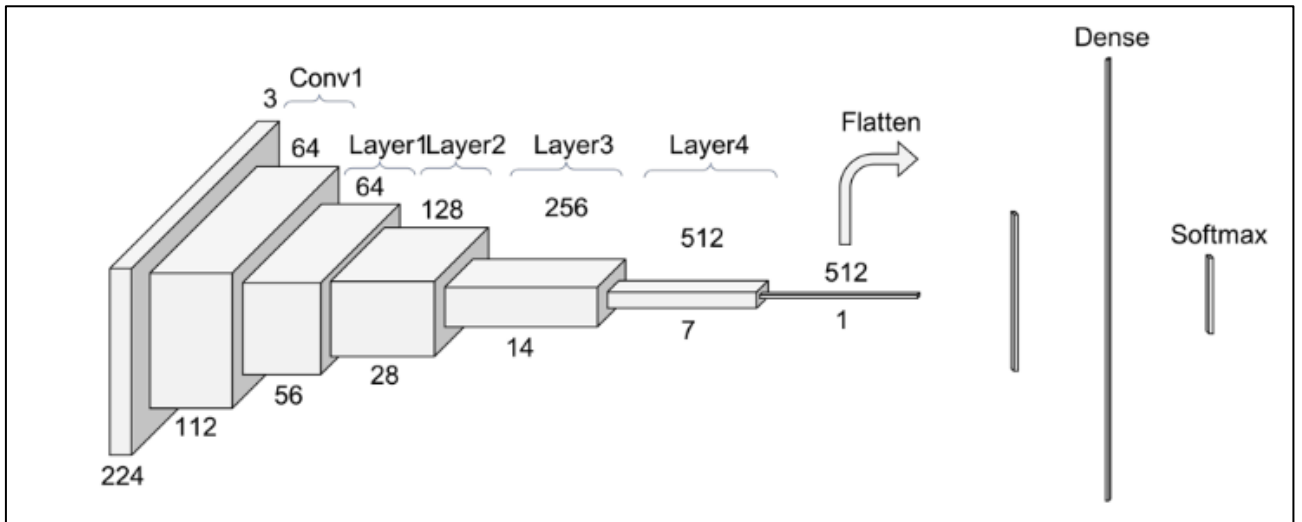
In this context, ResNet solve the problem of vanishing gradient, which make the gradients go to 0 when nets are too deep. These 0 gradients cause the parameters to be static, stopping the learning process.

There are different sizes of ResNets as we can see in Fig. 22.

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

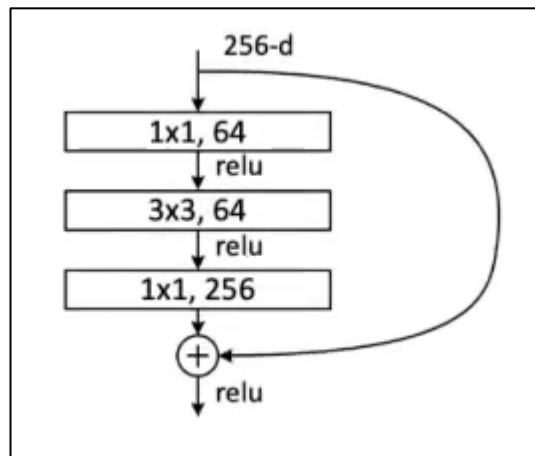
**Fig. 22** ResNet sizes.

ResNet network starts with one standard convolutional block, but what make different to this network are the following convolutional layers which are based in the repetition of blocks as the one shown in Fig. 23.



**Fig. 23** ResNet34 representation

As we can see in Fig. 24, by applying this process ResNet ensure that the number of layers remain constant in each layer, at the same time, it increases the number of operations. Adding the input tensor to the output one in each repetition is essential for the mentioned purpose.

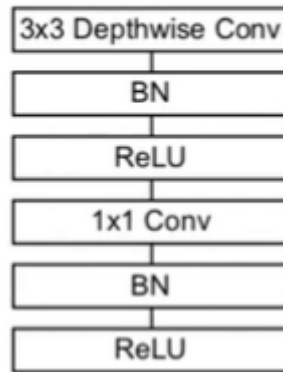


**Fig. 24.** ResNet50 example block

### 2.8.2. MobileNet

MobileNet is a lightweight net created in order to get the best possible efficiency between size and accuracy. This net was created for tasks which must be run in embeded or mobile systems such as robotics, self-driving, augmented reality, etc.

Mobile net is constituted by the repetition of a type of block called the MobileNet block, this repetition is preceded by a standard convolution.



**Fig. 25** MobileNet block.

This block is made up of the following operations:

- Depthwise convolution: It is a type of convolution in which a different filter is applied for each input channel while standard convolution applies multiple layer filters with the same number of layers as channels.
- In MobileNet V2, the 1x1 convolution (called pointwise convolution) reduces the number of channels.
- Batch normalization: It normalizes its inputs with some trainable parameters and with their mean and standard deviation.
- ReLU: activation function which converts into 0 all negative numbers

Another important feature of MobileNet is the residual connections between the bottleneck layers, which are the layers made up of few layers compared with their predecessors.

### 2.8.3. NasNet

Neural Search Architecture (NAS) is the new paradigm of neural networks, its goal is to use AI to design AI. One of the most challenging tasks by the time of facing a machine learning problem is to choose the appropriate network architecture; this is what NAS is solving.

NasNet is not a predefined network, it is able to change its architecture and its block structure using machine learning algorithms in order to achieve the highest performance.

Generally, Nas involves three components:

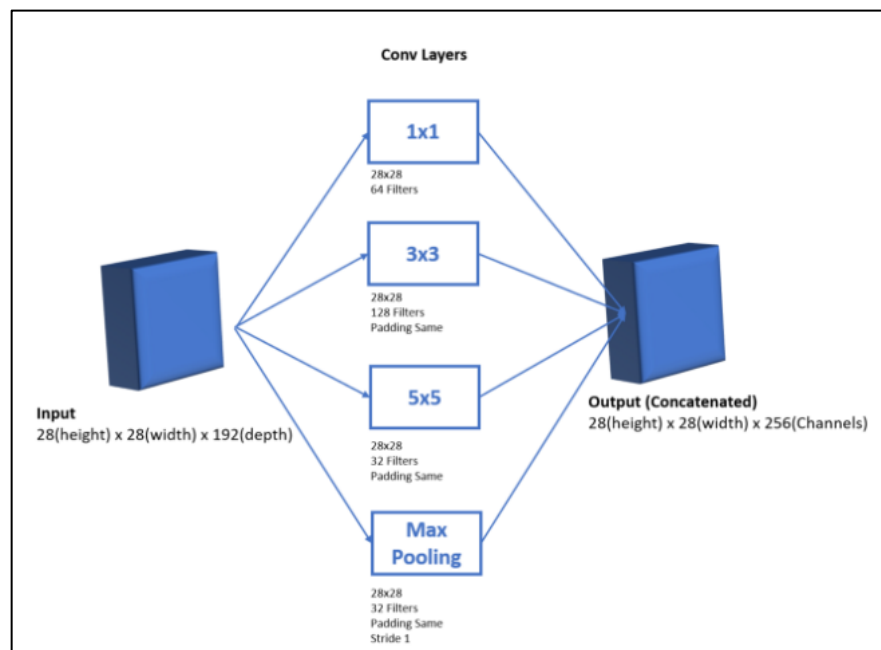
- Search Space. It defines the possible architectures and block structures that the net is going to try. It is divided into Global Search Space (architecture-based search) and Cell-based Search Space (which defines the block structure).

- Performance Estimation. It measures the performance of the model proposed by the Search Space.
- Search Strategy. Using different algorithms, it finds the best strategy to find the most appropriate architecture.

#### 2.8.4. Inception Net

Power CNNs are called to be large; this is the principle followed by Inception Net. However, by the time of developing a net, researchers must be cautious because just increasing net depth would lead to overfitting.

Inception Net is based on the repetition of the characteristic Inception blocks preceded by a standard convolution block. This first convolution operations help to reduce the dimensions of the input by increasing the number of channels.



**Fig. 26** Inception block

Inception block applies parallel operations to the input, these operations are different types of convolutions and a max pooling. After this operation process, all the results generated will be concatenated to create a final block output. This concatenation is possible because the dimension of the input is constant by adding to it the necessary padding in each convolution or pooling. This block lets the model to extract “different kind” of information without going deeper in the net.

#### 2.8.5. Inception-ResNet V2

This type of network is based on joining both philosophies: Inception and ResNet, so the functional blocks include parallel operations observed in Inception network and the input summatory at the end of this operations.

### 2.8.6. EfficientNet

EfficientNet is able to scale its dimensions (width, depth and resolution) according some fixed scalling coefficients. EfficientNetB0, which is the simplest one, is based on MobileNet structure with some squeeze and excitation blocks. From B0 to B7, there are different complexity networks. In this paper EfficientNetB1 will be used.

### 2.9. Diabetic Foot Ulcer (DFU) database

DFU database is a public database for the purpose of research under the approval from the UK National Health Service (NHS) Re-search Ethics Committee (REC). Due to the application made by Professor Vidas Raudonis, the dataset was shared with me in order to use it for research purposes. This dataset consists of 5955 images of diabetic foot patients where we can find healthy skin and unhealthy skin due to ulcers or ischemia produced by diabetic foot disease. These pictures were taken in Lancashire Teaching Hospital making use of three different cameras: Kodak DX4530, Nikon D3300and Nikon COOLPIX P100. The images were acquired at around 30-40cm using parallel orientation to the skin plan and avoiding the use of flash as main source of light. An image example of each group will be shown in following figure.

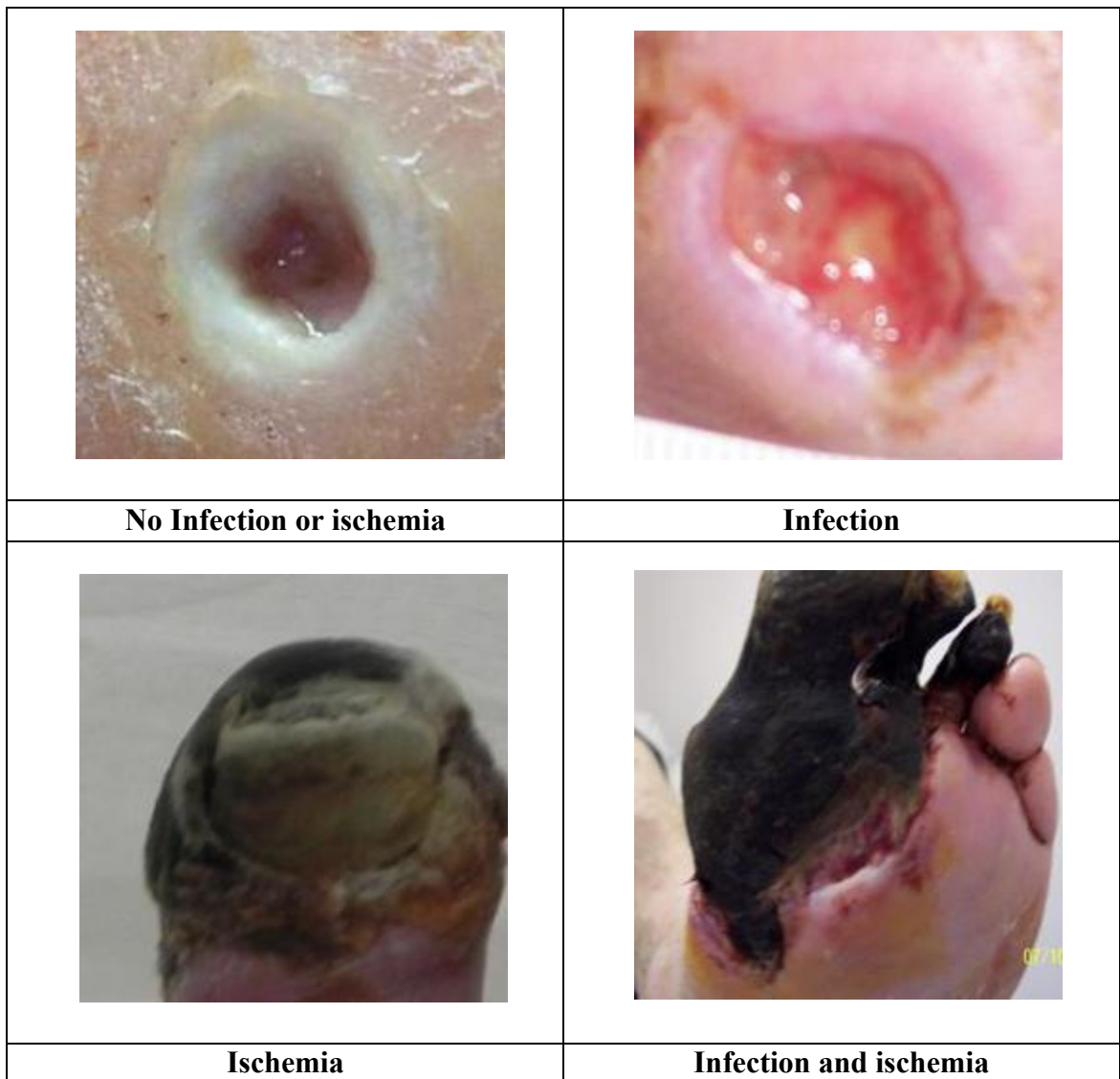


Fig. 27.Database examples.

This database has been used for several research papers, some of the most representative ones will be described in following sections.

### 2.9.1. “Deep learning in diabetic foot ulcers detection: A comprehensive evaluation” by Moi Hoon Yap [1]

This paper is based in the application of different object detection algorithms to the DFU database in order to detect the wound position in case it is present in the picture. The deep learning algorithms applied are: Faster R–CNN, YOLOv3, YOLOv5, and EfficientNet. Descriptions of an ensemble method and a new Cascade Attention DetNet were implemented. The working model is always the same during the hole paper, being described in the following steps:

- Data augmentation. Due to the heterogeneity of the taken pictures and the small amount of them, some data augmentation techniques were used such us blurring, brightness scaling or rotation transformations.
- Model training and implementation. The model is trained by the dataset images and the ones produced using data augmentation techniques.
- Post-processing. Some post processing techniques such as test-time augmentation is used to improve accuracy, added to the application of an ensemble method which is able to combine predictions from different models.

In Fig. 28. we can see: “A comparison of ensemble methods with different combinations of object detection frameworks, where FRCNN is Faster R–CNN, DetNet is CA-DetNet, EffDet is EfficientDet and ‘ALL methods’ represents an ensemble method based on Faster R–CNN, CA-DetNet, EfficientDet, YOLOv3 and YOLOv5”.

Methods	TP	FP	Recall	Precision	F1-Score	mAP
FRCNN + DetNet	1510	426	0.7201	0.7800	0.7488	0.6619
FRCNN + EffDet	1502	345	0.7163	0.8132	0.7617	0.6425
FRCNN + YOLOv3	1423	310	0.6786	0.8211	0.7431	0.6205
FRCNN + YOLOv5	1453	350	0.6929	0.8059	0.7451	0.6421
FRCNN + YOLOv5+EffDet	1396	252	0.6657	0.8471	0.7455	0.6109
FRCNN + YOLOv5+DetNet	1384	295	0.6600	0.8243	0.7331	0.6132
FRCNN + DetNet + EffDet	1435	270	0.6843	0.8416	0.7549	0.6229
ALL methods	1277	198	0.6090	0.8658	0.7150	0.5642

Fig. 28.Results obtained by the research.

As we can see the best result has been obtained by making used of post-processing stages detecting skin lesions but no differencing between ischemia or ulcer and neither between its grades or stages.

### 2.9.2. “DFUC 2020: Analysis Towards Diabetic Foot Ulcer Detection” by Bill Cassidy [2]

This article is very similar to the previous one, it makes use of several deep learning algorithms in order to detect ulcers in human skin using DFU dataset. In this case results are more limited in terms of stats (compared with previous paper), as post-processing methods are not applied to improve performance. In this case the following algorithms have been used: Faster R-CNN, YOLOv5 and EfficientDet. From this work we can get the results which are shown in Fig. 29.

Benchmark Algorithm	Recall	Precision	F1-Score	mAP
FRCNN R-FCN	0.7511	0.6186	0.6784	<b>0.6596</b>
FRCNN ResNet101	0.7396	0.5995	0.6623	0.6518
FRCNN Inception-v2-ResNet101	<b>0.7554</b>	0.6046	0.6716	0.6462
YOLOv5	0.7244	0.6081	0.6612	0.6304
EfficientDet	0.6939	<b>0.6919</b>	<b>0.6929</b>	0.6216

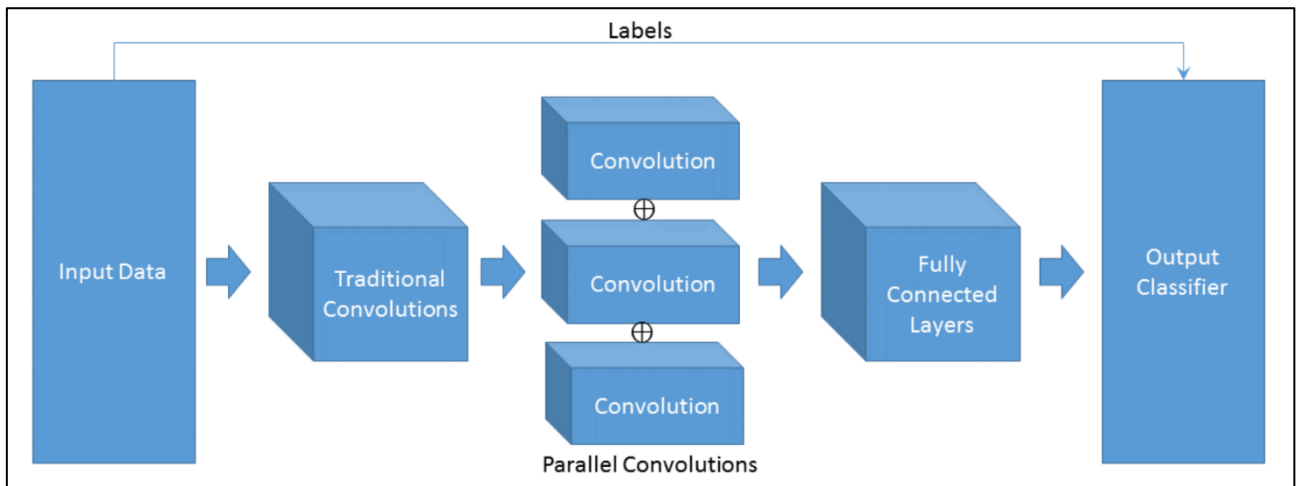
**Fig. 29.** Performance of the benchmark algorithms. FRCNN represents Faster R-CNN.

We can conclude that results are quite similar to previous ones and the algorithms are not able to detect ischemia/infection or ulcer grades on human skin.

### 2.9.3. DFUNet: Convolutional Neural Networks for Diabetic Foot Ulcer Classification by M. Goyal, N. D. Reeves, A. K. Davison, S. Rajbhandari, J. Spragg and M. H. Yap.[3]

This work has a different point of view from the previous ones, its goal is to develop a custom convolutional neural network for image classification which is called DFUNet. This net is able to classify human skin between two groups, normal and anormal. This paper doesn't face the problem as an object detection problem, actually it faces it as an image classification one. Train images show parts of the human body containin DFU and healthy skin with no background artifacts, so the algorithm doesn't have to detect de location of the object, it only has to classificate between two groups.

DFUNet reduces its network depth by increasing the size of the convolution kernel by applying parallel convolution. This type of convolution extracts concatenate information from the inputs by applying to it different filters in parallel. Previously to this layer, traditional convolution is applied to the input data. Finally, a dense layer is applied.



**Fig. 31.** DFUNet architecture.

Results obtained by this method improve the ones shown in previous works, they are provided in the following figure.

	<i>Sensitivity</i>	<i>Specificity</i>	<i>Precision</i>	<i>Accuracy</i>	<i>F-Measure</i>	<i>MCC</i>
GoogLenet	0.783	0.882	0.784	0.846	0.784	0.665
Proposed DFUNet	<b>0.867</b>	<b>0.930</b>	<b>0.867</b>	<b>0.907</b>	<b>0.867</b>	<b>0.796</b>

**Fig. 30.** DFUNet and GoogLenet results.

#### 2.9.4. Recognition of ischaemia and infection in diabetic foot ulcers: Dataset and techniques by Manu Goyal, Neil D.Reeves, Satyan Rajbhandari, Naseer Ahmad, Chuan Wang and Moi Hoon Yap [4]

This work is based on solving an image classification problem by the application of several machine learning algorithms twice in order to make two binary classifications: ischemia/non-ischemia and ulcer/non-ulcer. These algorithms are trained by around 1500 images, the data imbalance appreciated at this images is solved using data augmentation before training as an upsampling technique. Networks are the following ones: BayesNet, Random forest, Multilayer perceptron, InceptionV3 (CNN), ResNet50 (CNN) and InceptionResNetV2 (CNN). In addition, ensemble method is applied as a post-processing technique to increase performance. The results are the following ones:



	<i>Accuracy</i>	<i>Sensitivity</i>	<i>Precision</i>	<i>Specificity</i>	<i>F-measure</i>
BayesNet	0.639 ± 0.036	0.619 ± 0.018	0.653 ± 0.039	0.660 ± 0.015	0.622 ± 0.079
Random forest	0.605 ± 0.025	0.608 ± 0.025	0.607 ± 0.037	0.601 ± 0.069	0.606 ± 0.012
Multilayer perceptron	0.621 ± 0.026	0.680 ± 0.023	0.622 ± 0.057	0.570 ± 0.023	0.627 ± 0.074
InceptionV3 (CNN)	0.662 ± 0.014	0.693 ± 0.038	0.653 ± 0.015	0.631 ± 0.034	0.672 ± 0.019
ResNet50 (CNN)	0.673 ± 0.013	0.692 ± 0.051	0.668 ± 0.023	0.654 ± 0.051	0.679 ± 0.019
InceptionResNetV2 (CNN)	0.676 ± 0.015	0.688 ± 0.052	0.672 ± 0.015	0.664 ± 0.039	0.680 ± 0.024
Ensemble (CNN)	0.727 ± 0.025	0.709 ± 0.044	0.735 ± 0.036	0.744 ± 0.050	0.722 ± 0.028

**Fig. 32** Performance of binary classification of infection.

	<i>Accuracy</i>	<i>Sensitivity</i>	<i>Precision</i>	<i>Specificity</i>	<i>F-measure</i>
BayesNet	0.785 ± 0.022	0.774 ± 0.034	0.809 ± 0.034	0.800 ± 0.027	0.790 ± 0.020
Random forest	0.780 ± 0.041	0.739 ± 0.049	0.872 ± 0.029	0.842 ± 0.034	0.799 ± 0.033
Multilayer perceptron	0.804 ± 0.022	0.817 ± 0.040	0.787 ± 0.046	0.795 ± 0.031	0.800 ± 0.023
InceptionV3 (CNN)	0.841 ± 0.017	0.784 ± 0.045	0.886 ± 0.018	0.898 ± 0.022	0.831 ± 0.021
ResNet50 (CNN)	0.862 ± 0.018	0.797 ± 0.043	0.917 ± 0.015	0.927 ± 0.017	0.852 ± 0.022
InceptionResNetV2 (CNN)	0.853 ± 0.021	0.789 ± 0.054	0.906 ± 0.017	0.917 ± 0.019	0.842 ± 0.027
Ensemble (CNN)	0.903 ± 0.012	0.886 ± 0.035	0.918 ± 0.019	0.921 ± 0.021	0.902 ± 0.014

**Fig. 33** Performance of binary classification of ischemia.

### 2.9.5. Summary

In previous papers, we can observe that DFU database has been used for wound detection as an object detection problem, for normal and anormal skin classification and for ischemia an infection recognition as a classification problem. In this case, according to the wanted solution, the DFU database will be used to train a model which can detect both: ischemia and infection.

According to the purpose of this work, it is acceptable to think that the best way of facing the problem is a classification one, as the “object recognition task“ is carried out by the person who would take the picture. Then, this image is normalized so a classification algorithm would achieve the best

performance. In this context, reference [4] is the paper where this project will be landed. In this work, we will try to improve results obtained in article [4] without using upsampling technique for solving imbalance problem. In addition, we will look for a reduced-size model as it would be an important characteristic at the time of implementing the algorithm in underdeveloped regions.

### **3. Working environment**

#### **3.1. Python**

Python and R are the most common programming languages in the field of machine learning and data science. Specifically, in computer vision tasks, python is a must use. This work will be completely developed using python as programming language. As it can be read in python organization official web: “Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive”. There are plenty of machine learning libraries in python, the ones used in this work will be introduced below.

##### **3.1.1. Tensorflow**

Tensorflow is a python library which contains all necessary tools to carry out almost every single step when solving a machine learning problem is needed. It allows to: create neural networks (high-level or low-level), train models, test models, carry out data augmentation techniques, etc. In this paper a lot of functions from this library will be used and explained.

##### **3.1.2. NumPy**

It is a scientific computing library, which constitutes a highly powerful way to work with multidimensional data. It is based in objects with plenty of methods which are designed to make algebraic operations in a faster and more intuitive way. By the time of introducing data into the machine learning algorithms, all of it will be introduced as a NumPy object.

##### **3.1.3. OpenCV**

As its name indicates (Open Source Computer Vision Library), it is a library which includes more than 2500 algorithms which make possible working with images for computer vision problem solving.

##### **3.1.4. Scikit-learn**

Scikit-learn is a python library which contains many functions for machine learning problem solving. Tensorflow and Scikit-learn are the two most used machine learning python libraries.

#### **3.2. Google Colaboratory**

Google colaboratory is a tool to write python tool through the browser especially suitable for machine learning, education and data analysis. In addition, it provides online computational capacity, including GPUs and large RAM and disk capacity. Formally, Google Colaboratory is a hosted Jupyter notebook service which permit users to access this software without downloading it.

The code on this paper will be entirely run in Google Colaboratory Pro which is a subscription which includes better GPUs and storage capacities (RAM: 25.46GB and disk: 166.83GB).

## 4. Binary classification for infection and ischemia recognition

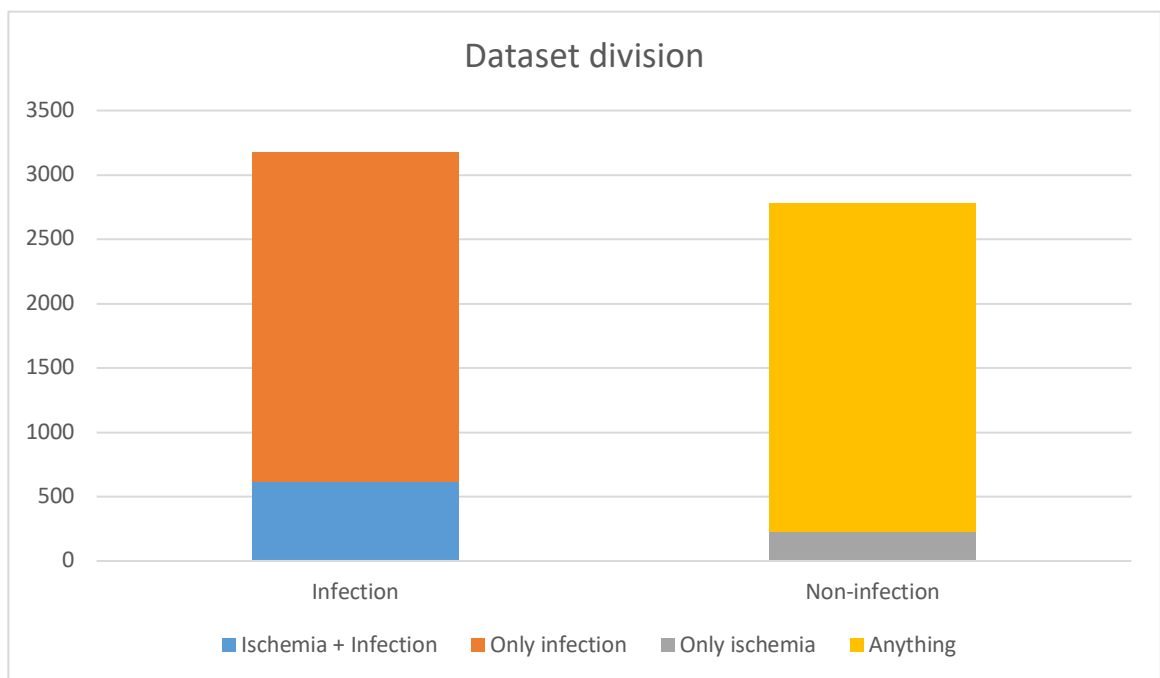
The problem will be faced both: as two binary classification problems and as a multilabel classification problem. On the one hand, in binary classification problem we will obtain one output bit which will determine if we have or not a specific object in the input picture (in this case: infection and ischemia). On the other hand, in multilabel classification problem we will have two output bits which will determine if we have or not ischemia and if we have or not infection in the output picture. Attending to what we have explained we will develop two binary classification algorithms and one multilabel classification algorithm.

Below, two independent python code will be exposed (ischemia and infection), they are executed in different scripts, so it is possible to find variables with the same name.

### 4.1. Dataset preparation

#### 4.1.1. Infection imbalance solving

In DFU dataset we have 5955 images as we mentioned in section 2.9, attending infection and non-infection cases we will have the following representation:

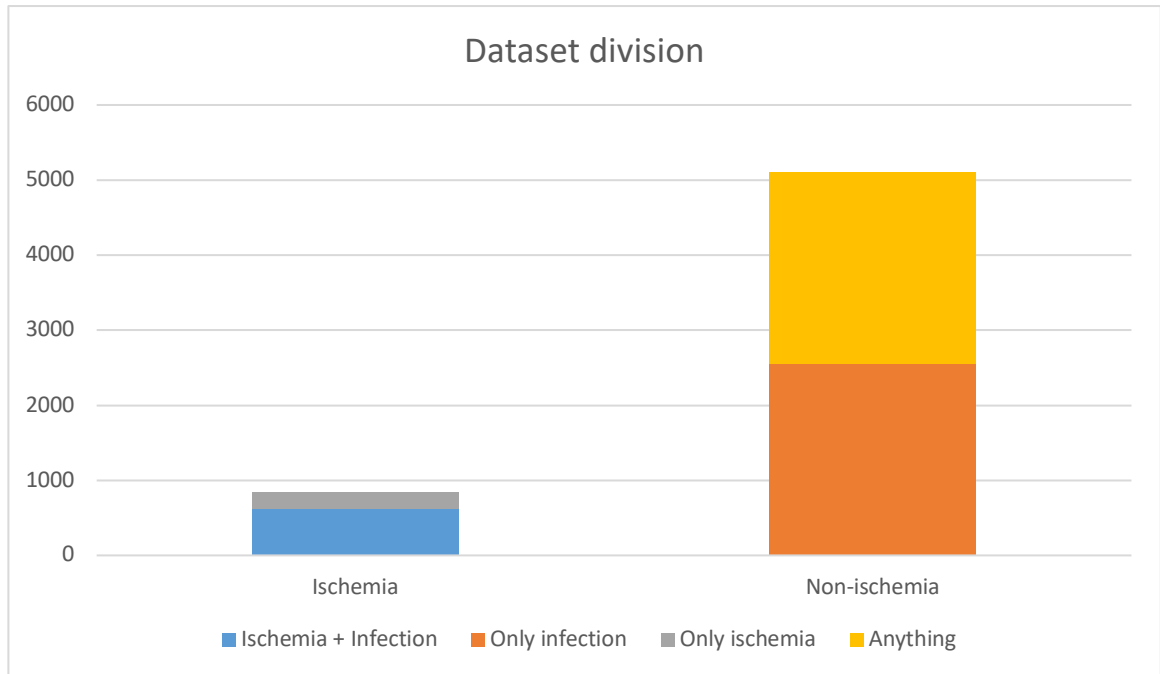


**Fig. 34** Dataset division for infection recognition.

For machine learning algorithms it is important to have a balanced dataset with same representation of the classes that we are going to predict. If we have an imbalanced dataset the algorithm will obtain a good accuracy by always predicting the larger class, but this algorithm won't face properly our problem. In this case, attending infection and non infection separation we have 3176 infection cases and 2779 non-infection cases, constituting the 53% and the 47% of the hole dataset, respectively. This is not a completely imbalanced system, but we are going to make both classes to have 50% of the hole amount of data by down sampling the infection class (as we would lose only 6% of the data).

### 4.1.2. Ischemia imbalance solving

As mentioned before, DFU dataset is made up of 5955 images, for binary infection classification the dataset was balanced, however, for ischemia binary classification this is completely different. As it is shown in Fig. 35, there is a significant imbalance as there are 879 ischemia cases and 5076 non-ischemia cases.



**Fig. 35** Dataset distribution for ischemia recognition.

This inconvenience will be solved by downsampling the majority class to the number of examples of the minority class. It is important to express why data augmentation is not applied to upsample the minority class. Data augmentation in computer vision problems is not applied before training in order to make bigger the amount of data, what it is applied is real-time data augmentation.

Real-time data augmentation is based on randomly generating batches of augmented images (rotations, zooms, shears, etc) from the training set, this way in each epoch the model is receiving different images. This technique is quite positive for avoiding overfitting. Thus, the network will not receive images from training data but only a randomly augmentation of them in each epoch.

Applying data augmentation to the minority class in order to balance the dataset would create adverse effects. The goal of machine learning algorithms is to create a model able to generalize so that it can predict a correct output from a real-world input, to obtain this purpose it is essential to have a wide database able to represent in a general way problem inputs. By applying previous data augmentation, an homogeneous database made up of images with very similar visual characteristics is being created. Consequently, we will obtain apparently quite good results in our data but very poor ones with inputs out of our database.

### 4.1.3. Python code

**Table 1.** Importing input images

```
file1=open("/content/gdrive/My Drive/TFG/DFUC2021_train/orden.txt","r")
file1_str=file1.read()
order = ast.literal_eval(file1_str)
train_images = []
for item in orden:
    img = cv.imread('/content/gdrive/My Drive/TFG/DFUC2021_train/images/'+item)
    train_images.append(img)
```

When images from Google Drive are downloaded, it may suffer variations in their order, so we read an order file to make it constant and compatible with Y labels. As we can see we are obtaining list of images which will be later processed.

Now it is time to import Y labels, which will be downloaded as an image format of 5955x4, as we can see in Table 2.

**Table 2** Downloading Y labels

```
y_input_bgr=cv.imread('/content/gdrive/My Drive/TFG/DFUC2021_train/y.png')
y_cv = cv.cvtColor(y_input_bgr, cv.COLOR_BGR2GRAY)
```

By default *cv.imread()* function reads the image format as BGR so we are obtaining a 5955x4x3 vector, by applying a colour space transformation into gray space we will obtain the desired 5955x4 vector.

As shown in section 2.9, the dataset is divided into 4 groups “no infection or ischemia”, “only infection”, “only ischemia”, “both”. So, Y labels describe those four groups in the following way: [1 0 0 0], [0 1 0 0], [0 0 1 0] and [0 0 0 1] respectively. Each one of the following algorithms create two independent lists of ischemia and non-ischemia images and infection and non-infection images.

**Table 3** Separating input images for infection recognition.

```
y_list=[]
infection=[]
no_infection=[]
i=0
for item in y_listtype:
    if item==[1,0,0,0]:
        no_infection.append(train_images[i])
    elif item==[0,1,0,0]:
        infection.append(train_images[i])
    elif item==[0,0,1,0]:
        no_infection.append(train_images[i])
    elif item==[0,0,0,1]:
        infection.append(train_images[i])
    i+=1
```

**Table 4** Separating input images for ischemia recognition.

```
y_list=[]
ischemia=[]
no_ischemia=[]
i=0
for item in y_listtype:
    if item==[1,0,0,0]:
        no_ischemia.append(train_images[i])
    elif item==[0,1,0,0]:
        no_ischemia.append(train_images[i])
    elif item==[0,0,1,0]:
        ischemia.append(train_images[i])
    elif item==[0,0,0,1]:
        ischemia.append(train_images[i])
    i+=1
```

Now, downsampling can be done by simply adding desired elements from both lists. Then, the list of images will be converted into a NumPy array as it is essential to introduce into the CNN that type of data.

**Table 5.** Converting into a NumPy array for infection recognition.

```
X = np.array(no_infection + infection[: -400], np.float32)
```

**Table 6** Converting into NumPy array for ischemia recognition.

```
X = np.array( ischemia+no_ischemia[: -4200], np.float32)
```

NumPy array of floating values is created.

To create the new vector of characteristics, it will be concatenated 1s to a list as many times as images there are in down sampled infection list. The same process will be carried out for 0s. Then the list is converted into a NumPy array of floating values.

**Table 7.** Creating characteristic vector for infection recognition.

```
y_list=[]
for i in range(len(infection[: -397])):
    y_list.append(1)
for i in range(len(no_infection)):
    y_list.append(0)

y = np.array(y_list, np.float32)
```

**Table 8** Creating characteristic vector for ischemia recognition

```

y_list=[]
for i in range(len(ischemia)):
    y_list.append(1)
for i in range(len(no_ischemia[: -4200])):
    y_list.append(0)

y = np.array(y_list, np.float32)

```

## 4.2. Splitting the data

In data science and machine learning is quite important to evaluate and test models in order to know how they are going to behave in real world problems. Typically, data is divided into two groups: 90/80% for training and 10/20% for validation and testing.

Focusing on validation data, it is used to evaluate a model at the same time of training, always at the end of an epoch. The model doesn't learn from this data as it only goes through the net in a forward way. However, many times this validation data is used to stop the training if we are not obtaining better results than previous epochs, so, somehow it is influencing the model.

Due to previous reason, the most correct way to split the data would be in three groups: training, validation and testing. In this paper, data will be splitted in 80% training, 10% validation and 10% testing as it is shown in Table 9.

**Table 9** Data splitting.

```

X, X_test, y, y_test = train_test_split(
    X,
    y,
    test_size = 0.10,
    random_state = 32)
X_training, X_validation, y_training, y_validation = train_test_split(
    X,
    y,
    test_size = 0.10,
    random_state = 32)

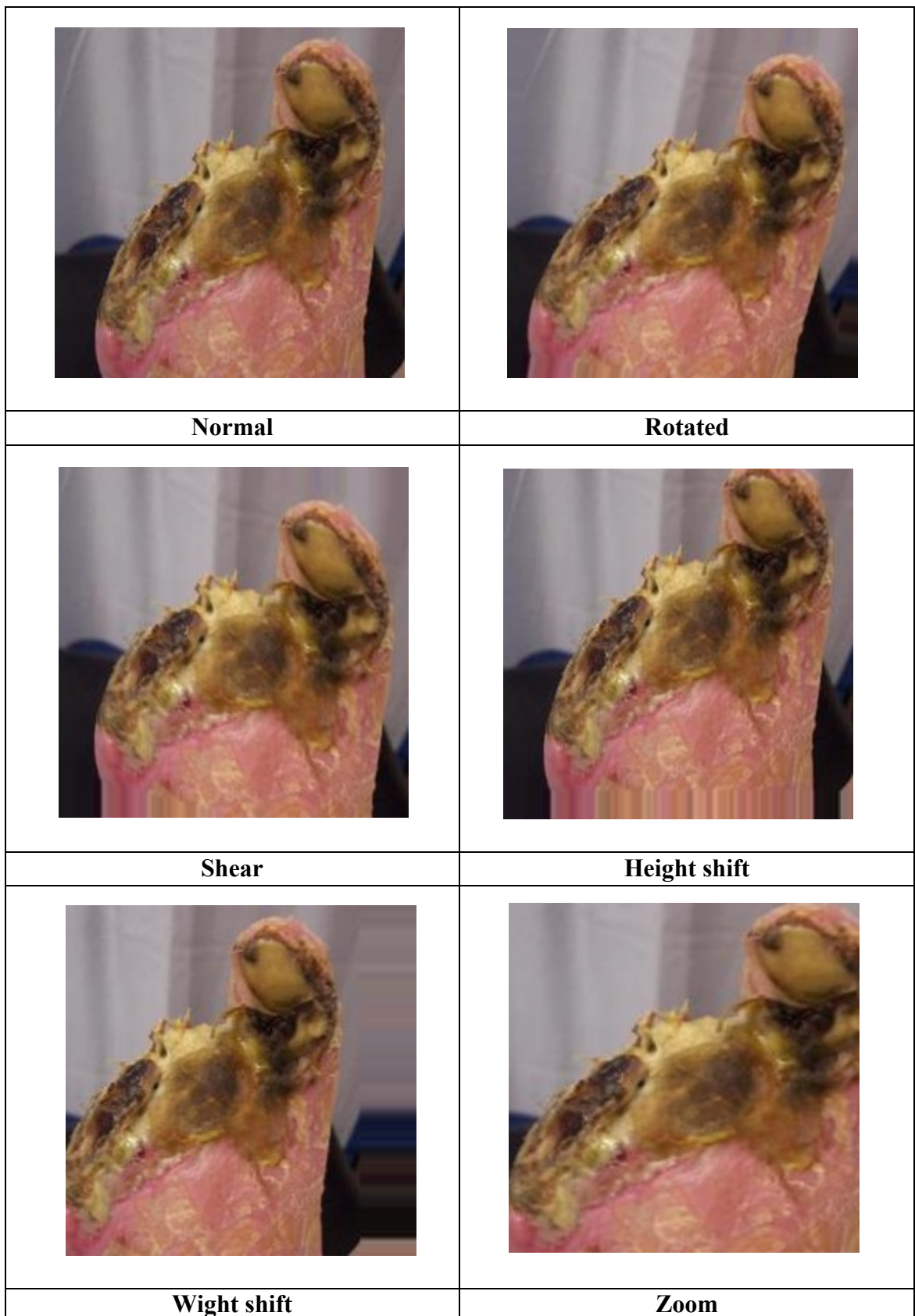
```

*Train\_test\_split()* is a function from sklearn library which is able to split data randomly. The argument *random\_state* refers to the seed to make the random separation.

## 4.3. Data augmentation

Data augmentation is a term which refers to the action of create artificial data making some mathematical operations to the dataset. In the field of computer vision, these mathematical operations can be translated into visual ones as: rotations, zooms, height and width shifts and shears.





**Fig. 36** Data augmentation transformations.

In this work, the function *ImageDataGenerator()* from tensorflow library will be used to carry out in-place data augmentation. This function can generate batches of images using real-time data augmentation. It returns an iterator which will be called at training time. This is a great advantage, as memory costs are small.

**Table 10** Creating data generator.

```
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.25,
    height_shift_range=0.25,
    shear_range=20,
    zoom_range=[0.6, 1.4],
)
```

Next step is to create the iterator, which will be called by the training function, using the method *datagen.flow()*.

**Table 11** Creating data augmentation iterator

```
data_gen_entrenamiento = datagen.flow(
    X_entrenamiento,
    y_entrenamiento,
    batch_size=32)
```

#### 4.4. Proposed network

For this purpose, fine tuning technique will be used as it is the best technique to obtain quite good results with a limited dataset. In addition, the problem will be faced with different CNNs in order to create a final ensemble method using all of them. Below, the general method will be explained taken as example ResNet101 but it will be the same for all CNNs.

First step to apply fine tuning to a specific CNN is to download the net with pretrained weights. That is what the following python code does:

**Table 12** Downloading the CNN.

```
from tensorflow.keras.applications.resnet import ResNet101

base_model = ResNet101(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

From tensorflow library we are able to download the architecture with predefined weights which are trained in ImageNet dataset, which is a public database with a number of images around 1.5 million belonging to 1000 classes. In this piece of code, it is also defined the input image dimension: (224,224,3).

Next step is to make this pretrained model to fit our problem by adding a classification layer.

In Table 13, it is shown how the final net is built. First, we have to unlock for training all parameters. Then by applying *GlobalAveragePooling2D()* function from tensorflow library we are converting the output of the net into a one dimension vector by making an average operation between all the output layers. Finally, the unique output unit is added; preceded by a dense layer of 32 neurons.

Last part of the code locks some parameters of the net, process which improves training duration and accuracy.

**Table 13** Creating the complete net.

```
base_model.trainable = True

input = tf.keras.Input(shape=(224, 224, 3))
x = base_model(input, training=True)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = Dense(32, activation='relu')(x)
output = Dense(1, activation='sigmoid')(x)
resnet101 = tf.keras.Model(input, output)

fine_tune_at = 150
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

As it is shown in Fig. 37, we have around 43 million parameters of which 33 million will be trainable. In all CNNs around 20% of the parameters will be locked for training because it increases performance as it has been observed empirically. However, NASNet will not have locked parameters.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
resnet101 (Functional)	(None, 7, 7, 2048)	42658176
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense (Dense)	(None, 32)	65568
dense_1 (Dense)	(None, 1)	33
-----		
Total params: 42,723,777		
Trainable params: 32,943,169		
Non-trainable params: 9,780,608		

**Fig. 37** Final CNN summary based on ResNet 101.

Following step is to train the model, which is made by these lines of code:

**Table 14** Model training.

```
Mobile_fine.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

#-----
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1)
callbacks=[early_stop]

history=Mobile_fine.fit(
    data_gen_entrenamiento,
    epochs=130, batch_size=32,
    validation_data=(X_validacion,y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    #class_weight={0:1.0,1:2.74},
    callbacks=callbacks
)
```

With *compile()* method from tensorflow library we declare optimizer, loss function and metrics as it was detailed in section 2.5.

Callbacks parameter are functions which can be added to the training process. In this case, *EarlyStopping()* function has been added, which let the model to stop before the predefined epochs are finished if the validation loss does not improve in a predefined number of epochs (which is defined using *patience* parameter).

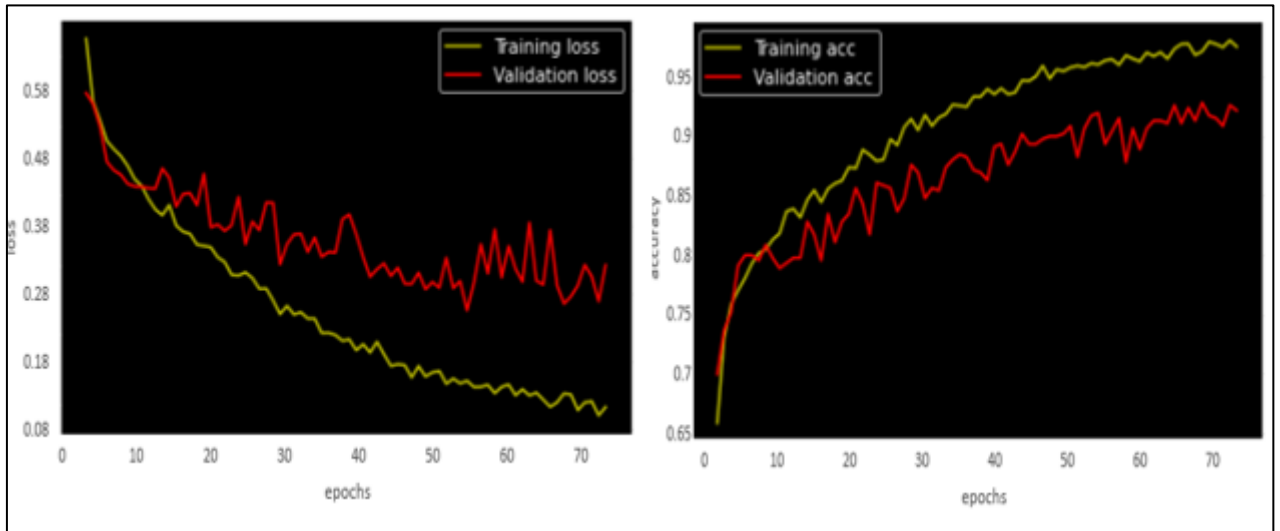
## 4.5. Networks' results

Below the results for each individual network will be exposed.

### 4.5.1. Results for infection recognition

#### ResNet101

Using ResNet 101, the following graphs for accuracy and loss have been obtained. The number of epochs is 76 until validation loss is stabilised, Curves have an appropriate form, no overfitting can be appreciated. Training time is 1.35 hours and the model size is 415.5 MB.



**Fig. 38** ResNet101 accuracy and loss

On testing data, the result obtained are the following ones. Precision and recall are balanced and accuracy obtained is high.

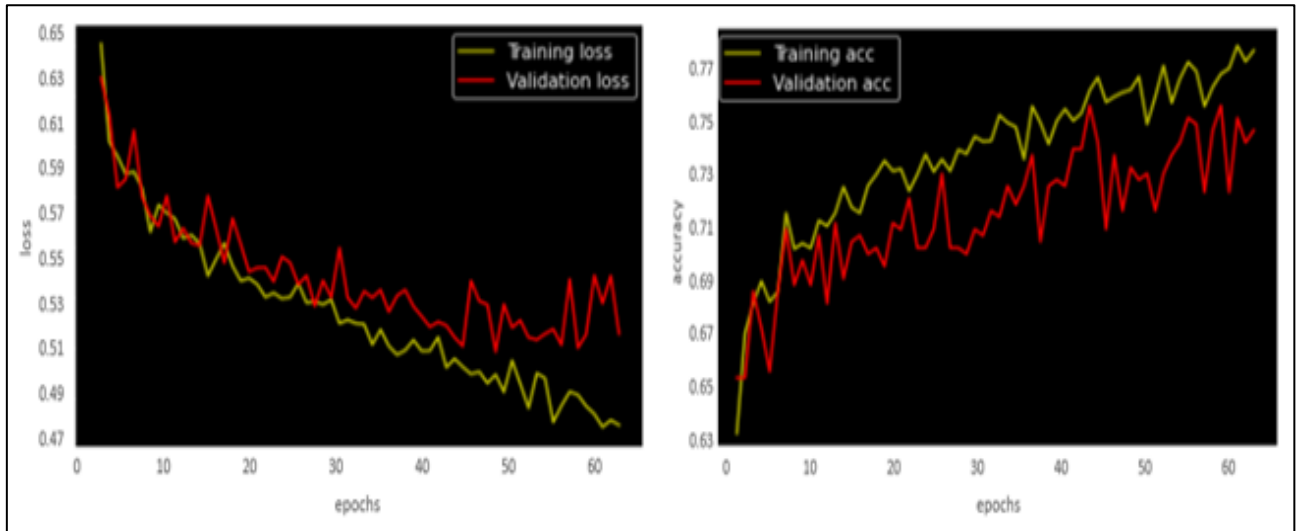
Truth	0	247	28
	1	20	261
		0	1
		Predicted	

	precision	recall	f1-score	support
0.0	0.93	0.90	0.91	275
1.0	0.90	0.93	0.92	281
accuracy			0.91	556

**Fig. 39.** Testing data metrics and confusion matrix for ResNet101

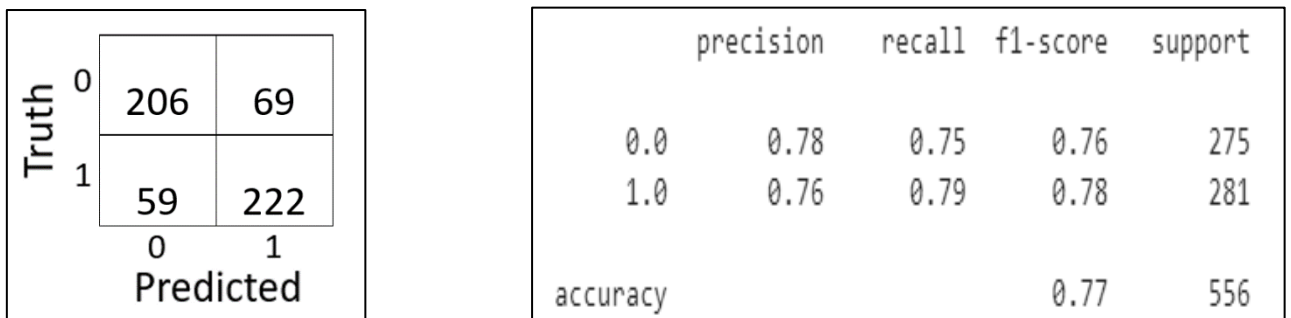
## MobileNet

Using MobileNet, the following graphs for accuracy and loss have been obtained. The number of epochs is 64, from epoch 45 we can observe little bit of over fitting as the training loss continues decreasing but the validation loss stays constant. Training time is 0.87 hours which is the 60% of the one obtained in ResNet101 due to the model size which is 23.7 MB.



**Fig. 40** MobileNet accuracy and loss.

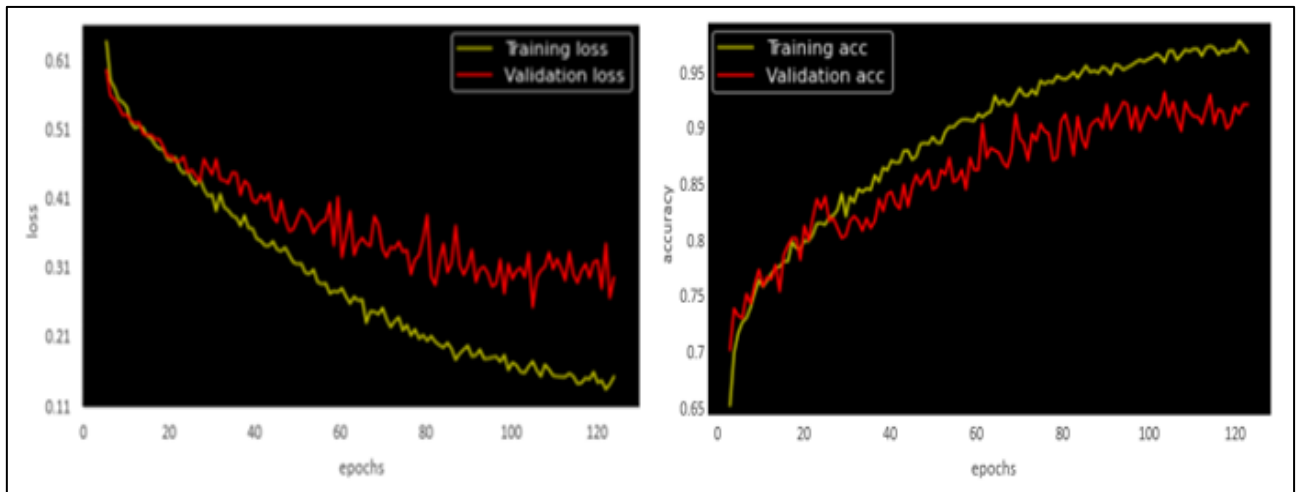
On testing data, the result obtained are the following ones. Performance is worst that in ResNet101 but the model size is less. However, results are acceptable, so this network can be later used for the small-size solution.



**Fig. 41** MobileNet Metrics obtained on testing set and confusion matrix

## Inception-ResNet V2

Using Inception\_ResNet V2, the following graphs for accuracy and loss have been obtained. The number of epochs is 126, from epoch 90 a stabilisation can be observed in accuracy a loss so training could have been stopeed then. Training time is 2.41 hours and the model size is 618.4 MB.



**Fig. 42** Inception-ResNet V2 accuracy and loss.

On testing data, the result obtained are the following ones. Metrics obtained are balanced and very similar to ResNet ones.

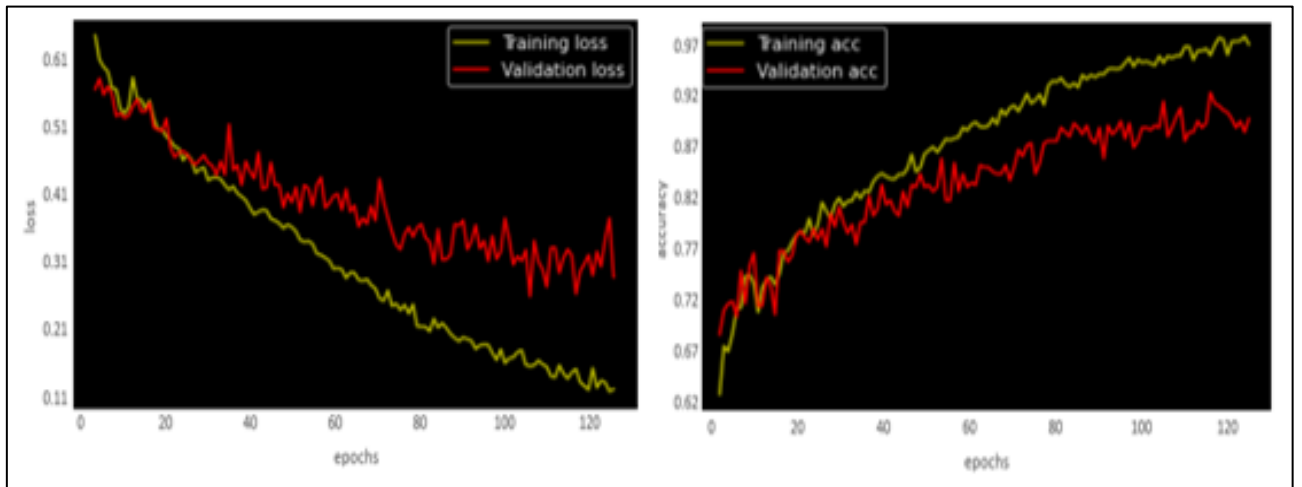
Truth	0	247	28
	1	22	259
		0	1
		Predicted	

	precision	recall	f1-score	support
0.0	0.92	0.90	0.91	275
1.0	0.90	0.92	0.91	281
accuracy			0.91	556

**Fig. 43** Inception-ResNet V2 metrics obtained on testing set and confusion matrix.

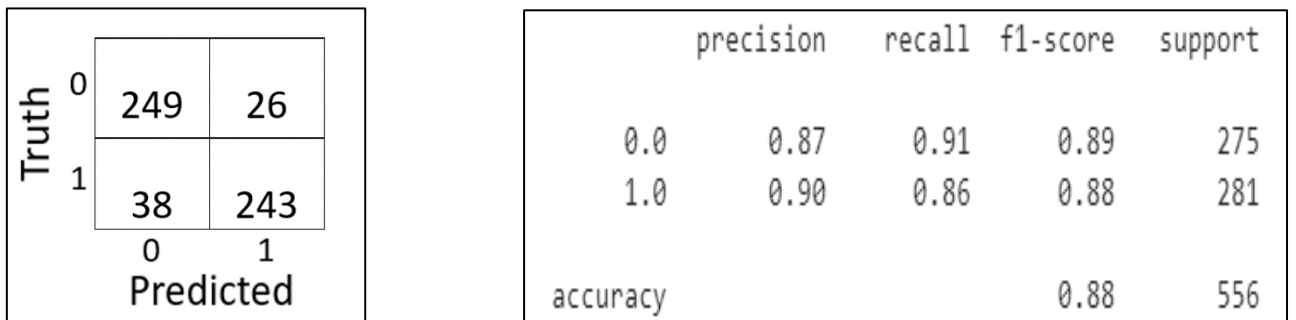
## NasNet

Using NasNet, the following graphs for accuracy and loss have been obtained. The number of epochs is 125, training time is 5.66 hours and the model size is 975.7 MB.



**Fig. 44** NasNet accuracy and loss.

On testing data, the result obtained are the following ones. Although this net is the biggest one, the results are good but not the best.

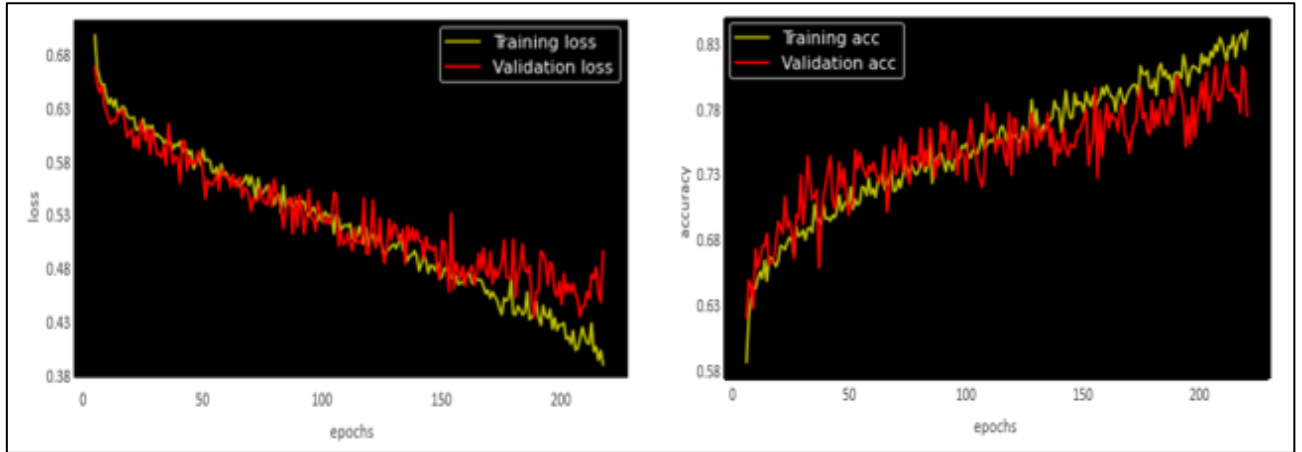


**Fig. 45** NasNet metrics obtained on testing set and confusion matrix.



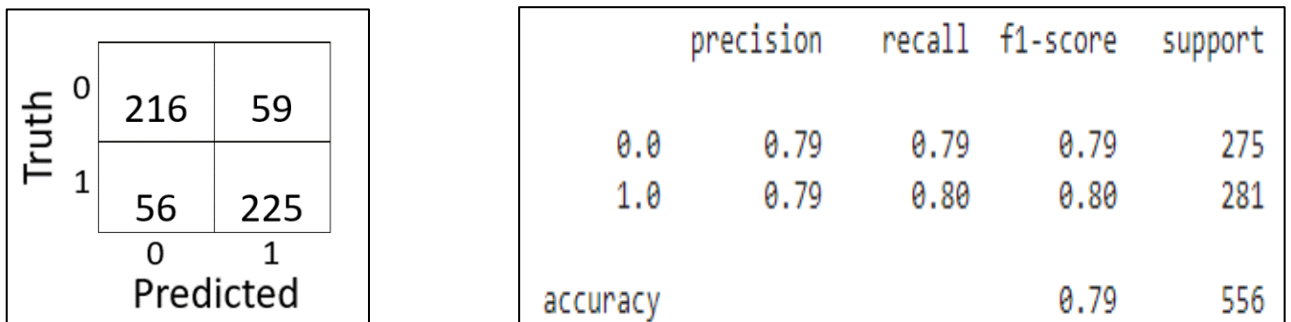
## Inception net

Using NasNet, the following graphs for accuracy and loss have been obtained. The number of epochs is 225, training time is 3.5 hours and the model size is 221.3 MB. Validation curves seem to start to be slightly constant, but maybe we could have obtained better results with a longer training time. However due to the computational cost compared with the metrics obtained, the training was stopped.



**Fig. 46** Inception accuracy and loss

On testing data, the result obtained are the following ones. They are very poor if the model size is taken into account.

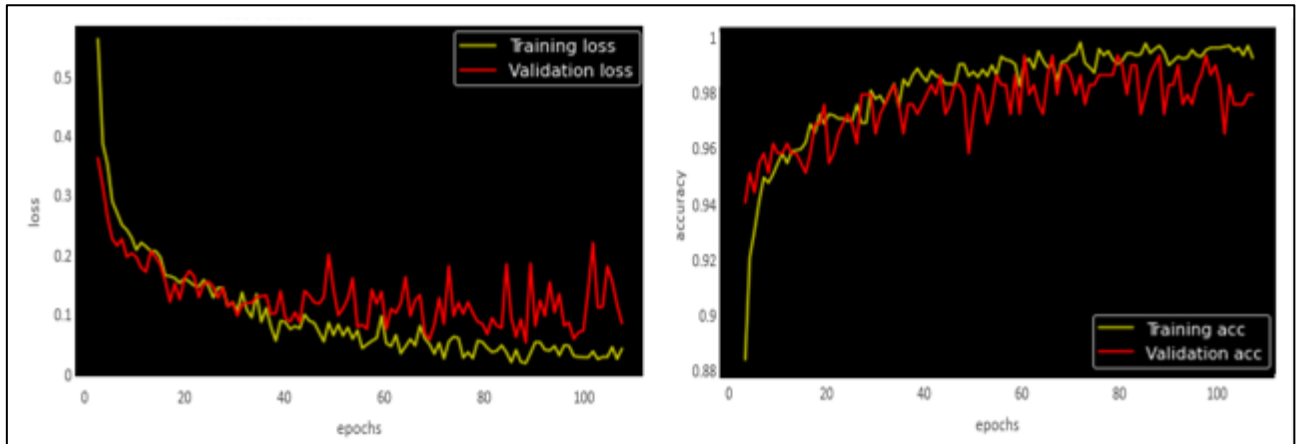


**Fig. 47** Inception net metrics obtained on testing set and confusion matrix.

## 4.5.2. Results for ischemia recognition

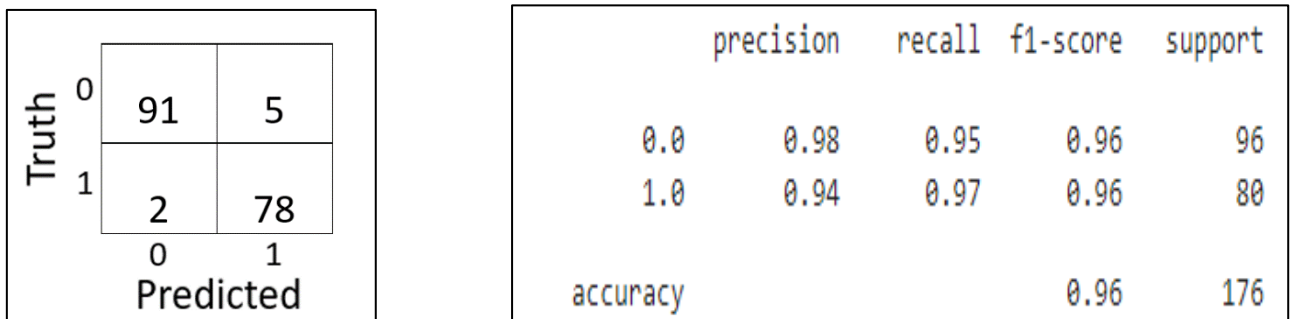
### ResNet101

Using ResNet 101, the following graphs for accuracy and loss have been obtained. The number of epochs is 110, training time is 44 minutes and the model size is 415.5 MB. Curves are constant from epoch 40, however, due to the big number used for patience parameter in *EarlyStopping()* function the training wasn't stopped.



**Fig. 48** ResNet101 accuracy and loss.

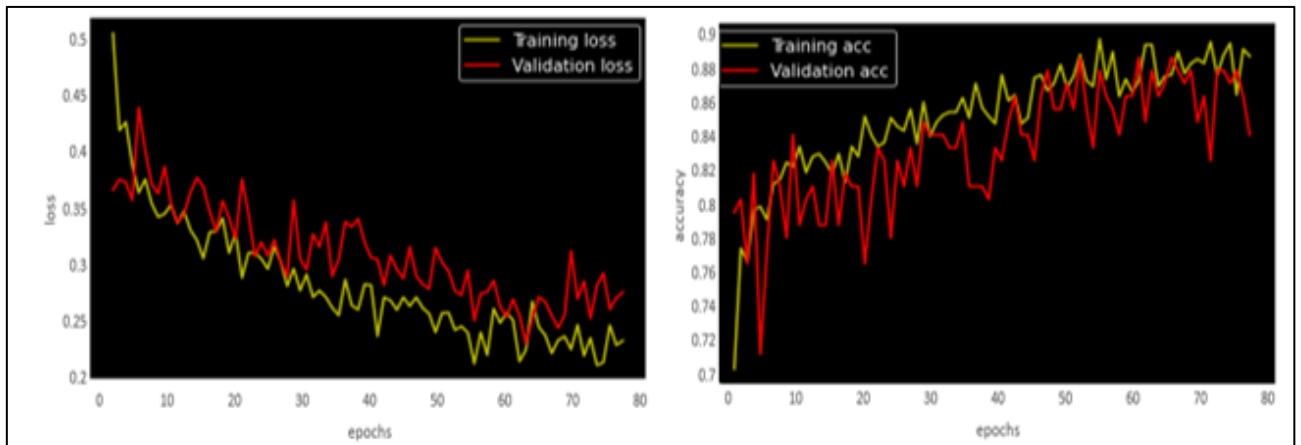
On testing data, the result obtained are the following ones. Performance is outstanding in ischemia recognition as it has easier visual characteristics to be recognised.



**Fig. 49** ResNet101 metrics obtained on testing set and confusion matrix

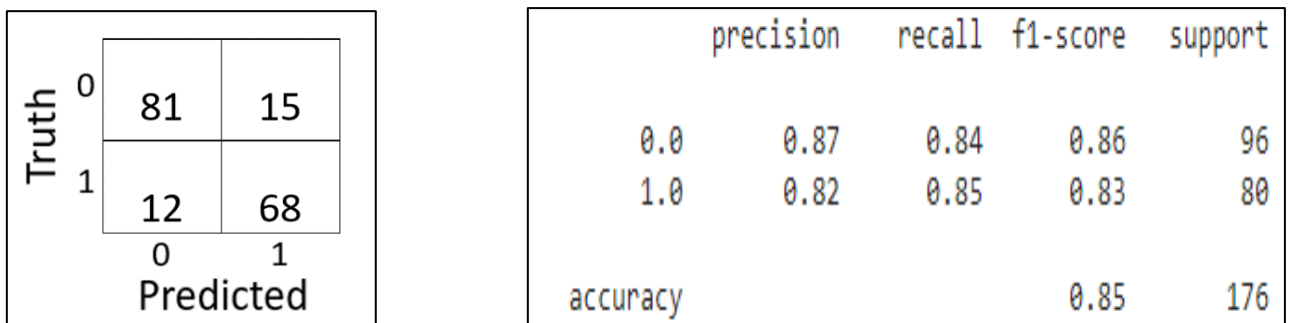
## MobileNet

Using MobileNet, the following graphs for accuracy and loss have been obtained. The number of epochs is 80, training time is 25 minutes and the model size is 23.7 MB. We can observe instability in training and loss curves due to the small number of training examples.



**Fig. 50** MobileNet accuracy and loss.

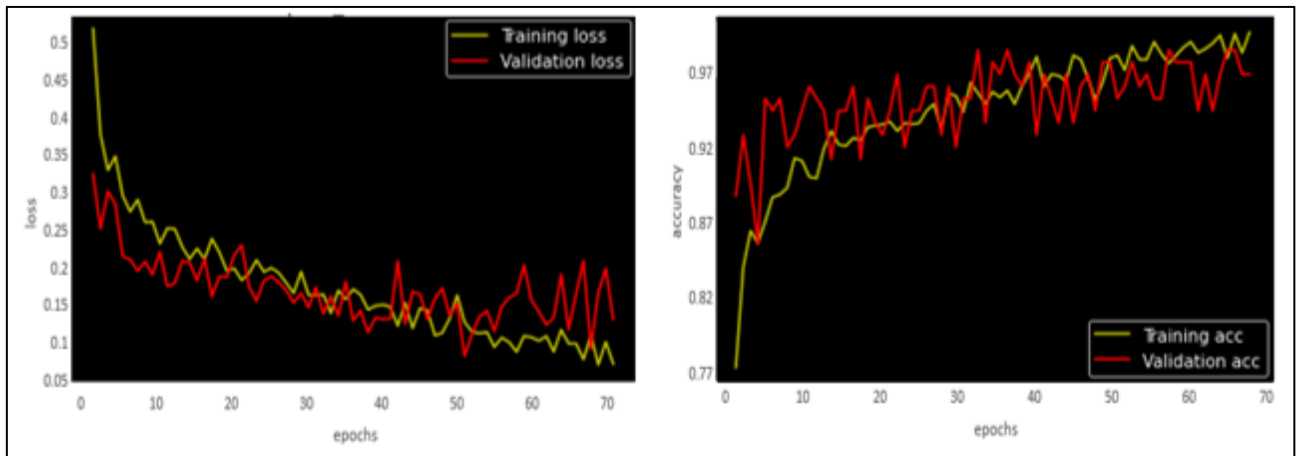
On testing data, the result obtained are the following ones. As happened in infection recognition, MobileNet results are poorer compare with bigger size models, although they are still useful.



**Fig. 51** MobileNet metrics on testing set and confusion matrix.

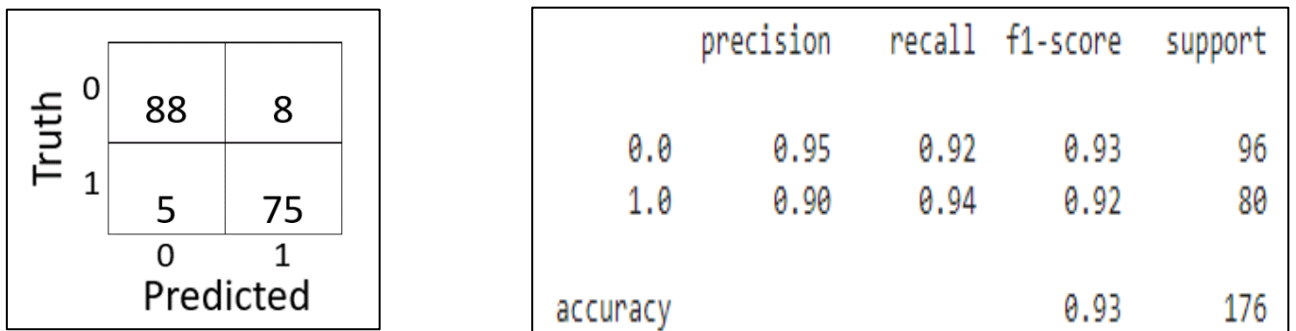
## Inception-ResNet V2

Using Inception\_ResNet V2, the following graphs for accuracy and loss have been obtained. The number of epochs is 71, training time is 28 minutes and the model size is 618.4 MB. We can observe instability in both curves and how fast they become almost constant.



**Fig. 52** Inception-ResNet V2 accuracy and loss

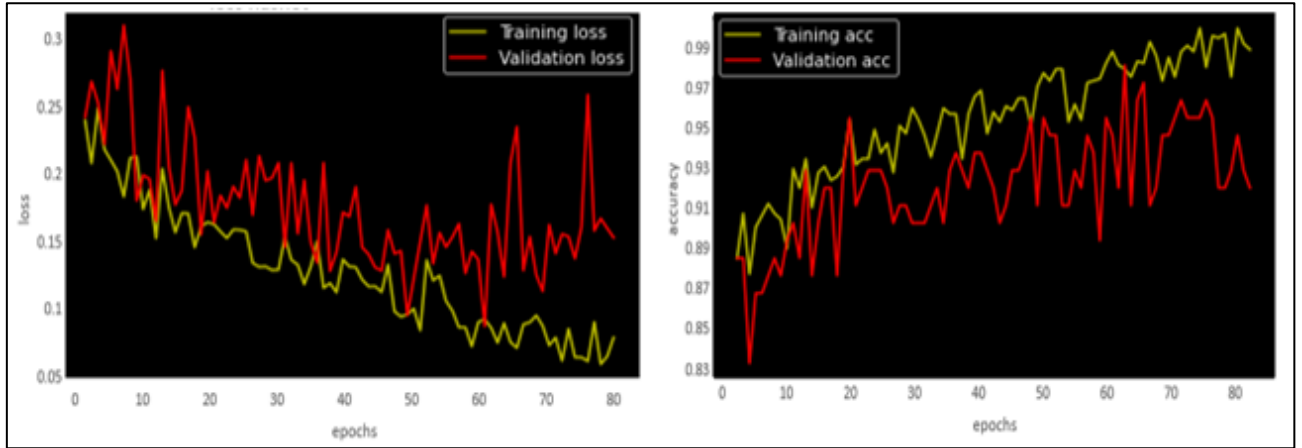
On testing data, the result obtained are the following ones. Results are outstanding, from 176 images only 13 are wrong predicted.



**Fig. 53** Inception-ResNet V2 metrics obtained on testing set and confusion matrix

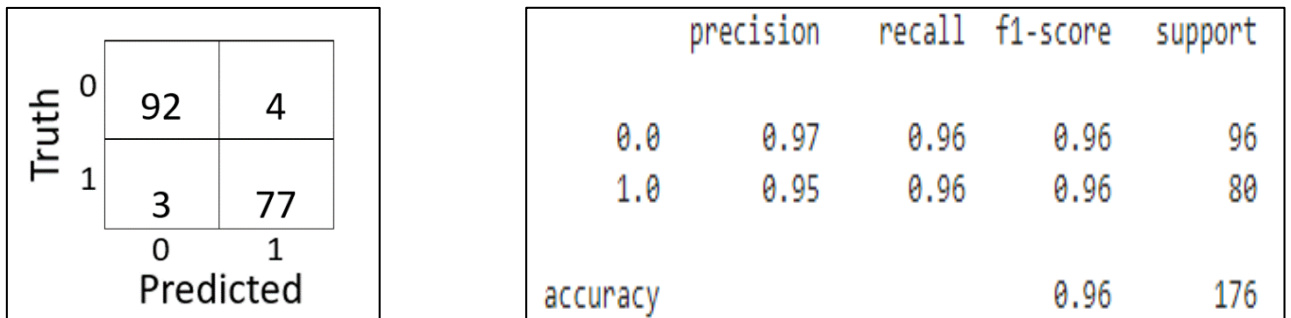
## NasNet

Using NasNet, the following graphs for accuracy and loss have been obtained. The number of epochs is 83, training time is 1.25 hours and the model size is 975.7 MB. The curves seem to be the most unstable, however they aren't due to the y axis scale as a quite good performance is obtained since very early epochs.



**Fig. 54** NasNet accuracy and loss.

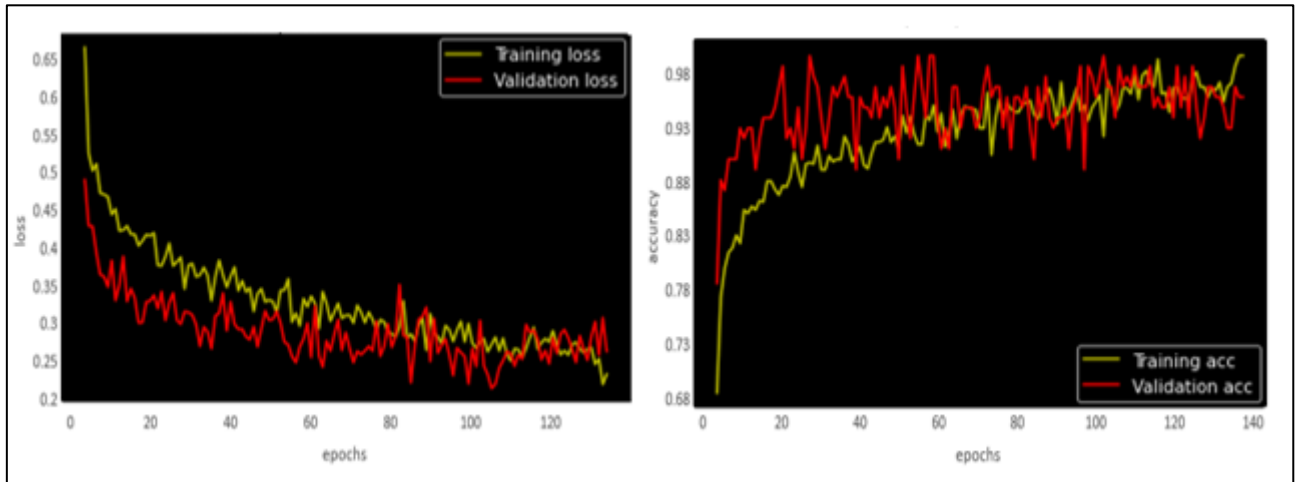
On testing data, the results are balanced and performance is high.



**Fig. 55** NasNet metrics obtained on testing set and confusion matrix.

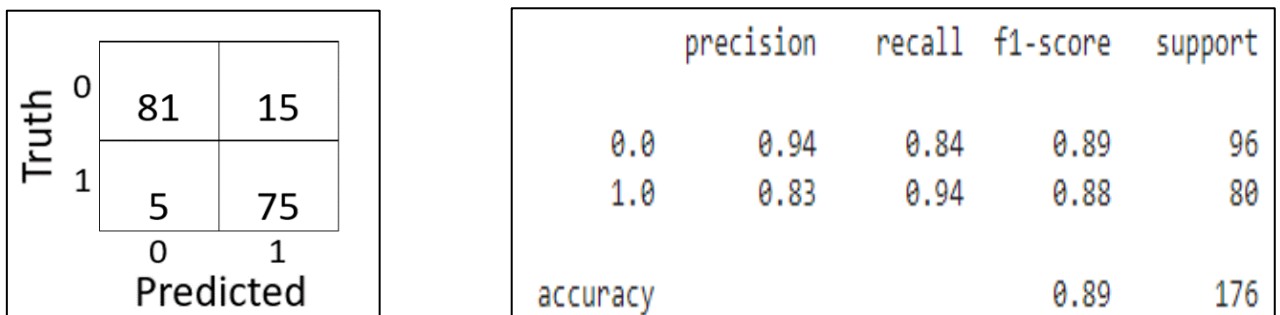
## Inception net

Using Inception Net, the following graphs for accuracy and loss have been obtained. The number of epochs is 137, training time is 25 minutes and the model size is 221.3 MB. As happened before, curves reach stability at very early epochs as the visual recognition of ischemia is simpler task compare with infection.



**Fig. 56** Inception accuracy and loss.

On testing data, the result obtained are the following ones. We can observe imbalance between recall and precision values, however it is acceptable.



**Fig. 57** Inception net metrics obtained on testing set

#### 4.6. Ensemble method for binary classification

Considering all networks' predictions and making a simple algorithm, it is possible to improve performance by creating an ensemble method. Predictions will be based on the majority vote of the best three models in isxhemia and infection recognition (NasNet, ResNet and Inception-ResNet). Method implementation is carried out by the following code. The algorithm is very simple, prediction values are stored in a list which is converted into a NumPy array. Finally, all the predictions values are added, if the addition is greater than one the final output will be 1 if not it will be 0.

**Table 15** Ensemble method for binary classification algorithm

```

y_np=[]
for model in models:
    y_np.append(model.predict(np.array(X_test, np.float32)))

y_pred=[]
y_elemen=[]
for prediction in y_np:
    for element in prediction:
        if element>0.5:
            y_elemen.append(1)
        else:
            y_elemen.append(0)
    y_pred.append(y_elemen)
    y_elemen=[]
y_pred=np.array(y_pred)

y_pred=y_pred.sum(axis=0)
Y=[]
for pred in y_pred:
    if pred>1:
        Y.append(1)
    else:
        Y.append(0)
Y = np.array(Y, np.float32)

```

##### 4.6.1. Final metrics for infection recognition

Using this method, it is possible to obtain the following results. As we can see, the results have improved from a maximum of 91% of accuracy to a 94%.

Truth	0	258	17
	1	18	263
		0	1
		Predicted	

	precision	recall	f1-score	support
0.0	0.93	0.94	0.94	275
1.0	0.94	0.94	0.94	281
accuracy			0.94	556

**Fig. 58** Ensemble method metrics obtained in testing set and confusion matrix.

#### 4.6.2. Final metrics for ischemia recognition

Using this method, performance is improved as we have obtained 2% better accuracy from 96% to 98%.

Truth	0	93	3
	1	1	79
		0	1
		Predicted	

	precision	recall	f1-score	support
0.0	0.99	0.97	0.98	96
1.0	0.96	0.99	0.98	80
accuracy			0.98	176

**Fig. 59.** Ensemble method metrics obtained in testing set and confusion matrix.

#### 4.7. Results overview

As we can see that results are outstanding, as we have obtained 98% of accuracy for ischemia recognition and 94% for infection recognition. However, the size of each ensemble method is 2GB, so for a hole wound detection we would need to use a 4GB-size model. Handling with this amount of data could be a limitation for many of mobile devices and computers so a hosted solution should be implemented. The model would be installed in a remote server, powerful enough to run the model, then, pictures will be taken and sent to the server via internet, where the prediction would be made.

However, this model is called to be used in places where diabetic foot specialists are inaccessible, so it is appropriate to think that many of these places will be underdeveloped regions where internet access could be limited. In order to cover all possible situations, we will develop a small-size model that could be run in any mobile device without having to use a hosted solution.



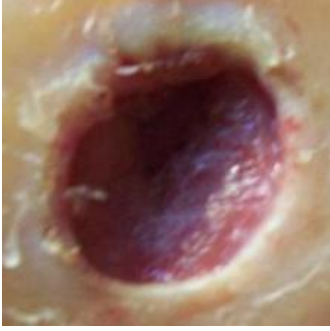



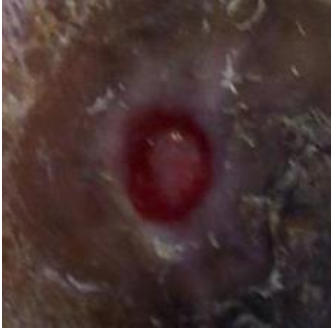

Final results for hosted solution are:

	Accuracy	Precision	Recall	F1-score
<b>Infection</b>	0.94	0.93	0.94	0.94
<b>Ischemia</b>	0.98	0.99	0.97	0.98

**Fig. 60** Final metrics for hosted solution.




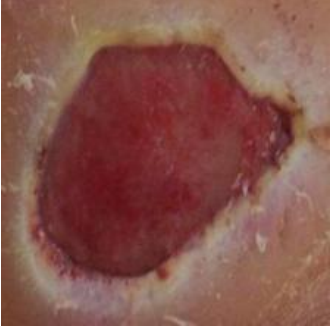






Some accurate and no accurate predictions for ischemia recognition:

	
<b>True positive</b>	<b>True positive</b>
	
<b>False positive</b>	<b>False positive</b>
	
<b>True negative</b>	<b>True negative</b>
	
<b>False negative</b>	<b>False negative</b>

**Fig. 61** Prediction examples for ischemia recognition.

Some accurate and no accurate examples for infection recognition:

	
<b>True positive</b>	<b>True positive</b>
	
<b>False positive</b>	<b>False positive</b>
	
<b>True negative</b>	<b>True negative</b>
	
<b>False negative</b>	<b>False negative</b>

**Fig. 62** Prediction examples for infection recognition.

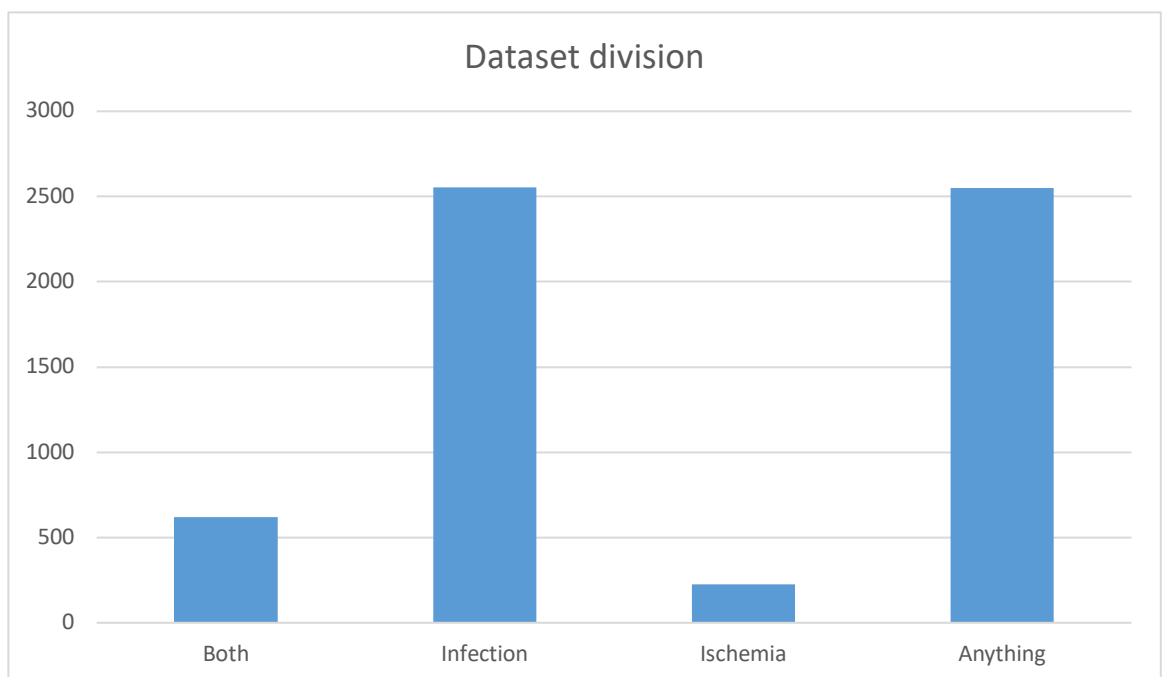
## 5. Multilabel classification for infection and ischemia recognition

Facing the problem as a multilabel recognition task reduces the size of the model by half, as only one algorithm will be needed in order to carry out the whole task.

### 5.1. Dataset preparation

#### 5.1.1. Imbalance solving

Multilabel classification algorithms are able to detect the presence or absence of several objects in a certain image, so using a unique CNN we are able to solve infection and ischemia recognition problem, reducing the final model size. As we are trying to detect two objects so our system output will be: [0 0], [1 0], [0 1], [1 1]. Each one of these outputs can be considered as a class so all of them must be balanced in order to obtain good results. However, a significant imbalance can be appreciated.



**Fig. 63.** Dataset distribution for multilabel classification.

As it was explained in section 4.1.2, solving imbalance with upsampling techniques could lead to misleading results. Due to this fact, using downsampling techniques would be the best options in order to appreciate more realistic results. So, only 226 images of each group will be introduced into the algorithm. It could seem to be few images, but they will be enough for a transfer learning method.

### 5.1.2. Python code

As done in previous method, all data will be imported using the same process.

**Table 16** Importing data.

```
file1=open("/content/gdrive/My Drive/TFG/DFUC2021_train/orden.txt","r")
file1_str=file1.read()
order = ast.literal_eval(file1_str)
train_images = []
for item in orden:
    img = cv.imread('/content/gdrive/My Drive/TFG/DFUC2021_train/images/'+item)
    train_images.append(img)

y_input_bgr=cv.imread('/content/gdrive/My Drive/TFG/DFUC2021_train/y.png')
y_cv = cv.cvtColor(y_input_bgr, cv.COLOR_BGR2GRAY)
```

Now, the dataset will be divided into 4 groups: “no infection or ischemia”, “only infection”, “only ischemia”, “both”. As we know, Y labels describe those four groups in the following way: [1 0 0 0], [0 1 0 0], [0 0 1 0] and [0 0 0 1] respectively. The following algorithms create four independent lists which will be later used to downsample the suitable groups.

**Table 17** Separating input into four groups

```
case1=[0,0]
case2=[1,0]
case3=[0,1]
case4=[1,1]
y_list=[]
ischemia_img=[]
infection_img=[]
both_img=[]
any_img=[]
i=0
for item in y_listtype:
    if item==[1,0,0,0]:
        any_img.append(train_images[i])
    elif item==[0,1,0,0]:
        infection_img.append(train_images[i])
    elif item==[0,0,1,0]:
        ischemia_img.append(train_images[i])
    elif item==[0,0,0,1]:
        both_img.append(train_images[i])
    i+=1
```

Now, downsampling can be done by simply adding the lists first 226 items. Then, list of images will be converted into a NumPy array as it is essential to introduce into the CNN that type of data.

**Table 18** Creating input array for multilabel classification.

```
X = np.array(ischemia_img+infection_img[:227]+both_img[:227]+any_img[:227],
            np.float32)
```

NumPy array of floating values is created.

To create the new vector of characteristics, it will be concatenated [0 0], [0 1], [1 0], [1 1] to a list as many times as images of each group there are in down sampled list.

**Table 19** Creating characteristic vector for multilabel classification.

```
y_list=[]
for i in range(len(ischemia_img)):
    y_list.append([0,1])
for i in range(len(infection_img[:227])):
    y_list.append([1,0])
for i in range(len(both_img[:227])):
    y_list.append([1,1])
for i in range(len(any_img[:227])):
    y_list.append([0,0])

y = np.array(y_list, np.float32)
```

## 5.2. Splitting the data

Data will be splitted in 80% training, 10% validation and 10% testing, using the same function as in previous method.

**Table 20** Data splitting for multilabel classification.

```
X, X_test, y, y_test = train_test_split(
    X,
    y,
    test_size = 0.10,
    random_state = 32)
X_training, X_validation, y_training, y_validation = train_test_split(
    X,
    y,
    test_size = 0.10,
    random_state = 32)
```

The same function as in previous method is used.

## 5.3. Data augmentation

As in previous sections, the function *ImageDataGenerator()* from tensorflow library will be used to carry out in-place data augmentation.

**Table 21** Creating data generator for multilabel classification.

```
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.25,
    height_shift_range=0.25,
    shear_range=20,
    zoom_range=[0.6, 1.4],)
```

Next step is to create the iterator, which will be called by the training function, using the method `datagen.flow()`.

**Table 22** Creating data augmentation iterator for multilabel classification.

```
data_gen_entrenamiento = datagen.flow(
    X_entrenamiento,
    y_entrenamiento,
    batch_size=32)
```

#### 5.4. Proposed networks

For this purpose, fine tuning technique will be used as it is the best technique to obtain quite good results with a limited dataset. The problem will be faced with different small-size CNNs in order to create a final ensemble method using all of them. Below, the general method will be explained taken as example MobileNet but it will be the same for all CNNs.

First step to apply fine tuning to a specific CNN is to download the net with a pretrained weights. That is what the following python code does:

**Table 23** Downloading the CNN

```
from tensorflow.keras.applications import MobileNetV2

base_model = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

We are using pretrained weights from tensorflow library as in the previous method. In this piece of code, it is also defined the input image dimension: (224,224,3).

Next step is to make this pretrained model to fit our problem by adding a classification layer.

**Table 24** Creating the complete net

```
base_model.trainable = True

input = tf.keras.Input(shape=(224, 224, 3))
x = base_model(input, training=True)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = Dense(32, activation='relu')(x)
output = Dense(2, activation='sigmoid')(x)
```

```

resnet101 = tf.keras.Model(input, output)

fine_tune_at = 150
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

```

The code is the same used for binary classification with the exception of the two-neuron output in order to classify into the four different groups.

Last part of the code locks some parameters of the net, process which improves training duration and accuracy. As it is shown in Fig. 64, we have around 2.3 million parameters of which 1.9 million will be trainable.

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 224, 224, 3)]	0
mobilenetv2_1.00_224 (Functional)	(None, 7, 7, 1280)	2257984
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 1280)	0
dense_4 (Dense)	(None, 32)	40992
dense_5 (Dense)	(None, 2)	66
=====		
Total params: 2,299,042		
Trainable params: 1,902,498		
Non-trainable params: 396,544		

**Fig. 64** Final CNN summary based on MobileNetV2

Following step is to train the model, which is made by these lines of code:

**Table 25** Model training.

```

Mobile_fine.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

#-----
early_stop = EarlyStopping(monitor='val_loss', patience=20, verbose=1)
callbacks=[early_stop]

history=Mobile_fine.fit(
    data_gen_training,
    epochs=130, batch_size=32,
    validation_data=(X_validation,y_validation),
    steps_per_epoch=int(np.ceil(len(X_training) / float(32))),
    validation_steps=int(np.ceil(len(X_validation) / float(32))),
    #class_weight={0:1.0,1:2.74},
    callbacks=callbacks
)

```

The code is exactly the same than the one used for binary recognition problem. The only parameter is worth talking about is the “loss” one. Each neuron of the output should behave as independent binary classification nodes so *binary\_crossentropy* parameter is chosen. This loss function gives an independent real value from 0 to 1 for each neuron.

### 5.5. Network’s results

Three different CNNs have been trained in order to built the final ensemble method. In this section, individual CNN results will be presented. The algorithms used are: MobileNet, NASNet mobile and EfficientNet.

As the amount of data is very small, the accuracy and loss graphics are very unstable so in this section will be omitted.

#### MobileNet

The model size is 23.7MB and the training time is 10 minutes. The results obtained are the following ones.

	precision	recall	f1-score	support
0	0.82	0.80	0.81	45
1	0.91	0.78	0.84	51

**Fig. 65** MobileNet metrics for multilabel classification.

The confusion matrices are the following ones:

Truth	0	38	8
	1	9	36
		0	1
		Predicted	

Truth	0	36	4
	1	11	40
		0	1
		Predicted	

**Fig. 66** Confusion matrix for infection recognition on the left and for ischemia recognition on the right using MobileNetV2.

Applying accuracy formula, the following result is obtained for infection recognition:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{38 + 36}{91} \cdot 100 = 81.3\%$$

Accuracy for ischemia recognition is:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{36 + 40}{91} \cdot 100 = 83.5\%$$



## NASNet Mobile

NASNet mobile is a model based in the same principles than the standard one but reducing the number of parameters, and therefore the size of the algorithm. The model size is 46.6MB and the training time is 14 minutes. The results obtained are the following ones.

	precision	recall	f1-score	support
0	0.75	0.80	0.77	45
1	0.94	0.86	0.90	51

**Fig. 67** NASNet Mobile metrics for multilabel classification.

The confusion matrices are the following ones:

Truth	0	34	12
	1	9	36
		0	1
		Predicted	

Truth	0	37	3
	1	7	44
		0	1
		Predicted	

**Fig. 68** Confusion matrix for infection recognition on the left and for ischemia recognition on the right using NASNet Mobile.

Applying accuracy formula, the following result is obtained for infection recognition:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{36 + 34}{30 + 38 + 8 + 15} \cdot 100 = 76.9\%$$

Accuracy for ischemia recognition is:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{37 + 44}{91} \cdot 100 = 89\%$$

## EfficientNet

The model size is 70.6 MB and the training time is 6 minutes. The results obtained are the following ones.

	precision	recall	f1-score	support
0	0.84	0.84	0.84	45
1	0.94	0.92	0.93	51

**Fig. 69.** NASNet Mobile metrics for multilabel classification.

The confusion matrices are the following ones:

Truth	0	39	7
	1	7	38
		0	1
		Predicted	

Truth	0	37	3
	1	4	47
		0	1
		Predicted	

**Fig. 70** Confusion matrix for infection recognition on the left and for ischemia recognition on the right using EfficientNet.

Applying accuracy formula, the following result is obtained for infection recognition:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{39 + 38}{91} \cdot 100 = 84.6\%$$

Accuracy for ischemia recognition is:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{47 + 37}{91} \cdot 100 = 92.3\%$$

## 5.6. Ensemble method

As used in section 4.6, an ensemble method will be used in order to improve performance. The algorithm used will be the same, all predictions will be based in the majority vote. The python code is presented below.

**Table 26** Ensemble method for multilabel classification.

```
#----- making predictions-----
y_np=[]
for model in models:
    y_np.append(model.predict(np.array(X_test, np.float32)))

#-----obtaining binary predictions-----
y_pred=[]
y_elemen=[]
for prediction in y_np:
    for element in prediction:
        if element[0]>0.5 and element[1]>0.5:
            y_elemen.append([1,1])
        if element[0]>0.5 and element[1]<=0.5:
            y_elemen.append([1,0])
        if element[0]<=0.5 and element[1]<=0.5:
            y_elemen.append([0,0])
        if element[0]<=0.5 and element[1]>0.5:
            y_elemen.append([0,1])
    y_pred.append(y_elemen)
    y_elemen=[]
y_pred=np.array(y_pred)

#-----applying the majority voting algorithm-----
y_pred=y_pred.sum(axis=0)
Y=[]
for element in y_pred:
    if element[0]>1 and element[1]>1:
        Y.append([1,1])
    if element[0]>1 and element[1]<=1:
        Y.append([1,0])
    if element[0]<=1 and element[1]<=1:
        Y.append([0,0])
    if element[0]<=1 and element[1]>1:
        Y.append([0,1])

Y = np.array(Y, np.float32)
```

The model size is 140.9 MB and the metrics obtained are the following ones.

	precision	recall	f1-score	support
0	0.85	0.89	0.87	45
1	0.94	0.86	0.90	51

**Fig. 71** Ensemble method metrics for multilabel classification.

The confusion matrices are the following ones:

Truth	0	39	7
	1	5	40
		0	1
		Predicted	

Truth	0	37	3
	1	7	44
		0	1
		Predicted	

**Fig. 72** Confusion matrix for infection recognition on the left and for ischemia recognition on the right using ensemble method.

Applying accuracy formula, the following result is obtained for infection recognition:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{39 + 40}{91} \cdot 100 = 86.8\%$$

Accuracy for ischemia recognition is:

$$Accuracy (\%) = \frac{TP + TN}{Total\ Cases} \cdot 100 = \frac{37 + 44}{91} \cdot 100 = 89.0\%$$

As we can see, using an ensemble method, an accuracy of 86.8% for infection and 89.0% for ischemia is obtained. However, using only the EfficientNet model results for ischemia recognition are better, this happens because other two models have a significant worse performance in this aspect, being prejudicial for the final model. On the other hand, infection results are improved using the ensemble method.



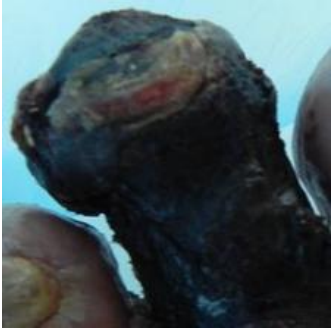

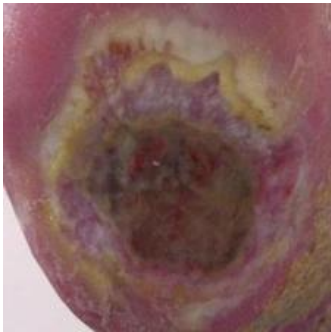

## 5.7. Results overview

In conclusion, to obtain the best possible results, the ensemble method will be used for infection recognition and the EfficientNet model will be used for ischemia recognition. Final model size is 140.9 MB. Final metrics are:

	Accuracy	Precision	Recall	F1-score
Infection	0.868	0.85	0.89	0.87
Ischemia	0.923	0.94	0.92	0.93

**Fig. 73** Results for multilabel classification solution.

Some examples of multilabel ischemia and infection recognition:

	
<p><b>Ischemia: 0 (FN)</b>  <b>Infection: 1 (TP)</b></p>	<p><b>Ischemia: 0 (TN)</b>  <b>Infection: 1 (FP)</b></p>
	
<p><b>Ischemia: 1 (TP)</b>  <b>Infection: 0 (TN)</b></p>	<p><b>Ischemia: 0 (TN)</b>  <b>Infection: 0 (FN)</b></p>
	
<p><b>Ischemia: 1 (FP)</b>  <b>Infection: 0 (FN)</b></p>	<p><b>Ischemia: 1(TP)</b>  <b>Infection: 1(TP)</b></p>

**Fig. 74** Examples for multilabel recognition task.

## Conclusions and results

1. In this paper two methods to recognise ischemia and infection in diabetic foot wound have been created. Best results have been obtained with the model obtained in section 4, which has a total size of 4GB and its metrics are shown in Fig. 75. This algorithm would be useful to implement a hosted solution due to the computational capability required.
2. Solution developed in section 6 obtains accurate results and it considerably reduces computational costs. As we can see in Fig 75, accuracy is reduced a 6% for ischemia recognition and a 7% for infection recognition by reducing the model size by a 96.5%. This model can be used in mobile and embebed devices.

		Accuracy	Precision	Recall	F1-score	Model size
<b>Multilabel model</b>	<b>Infection</b>	0.87	0.85	0.89	0.87	140.9 MB
	<b>Ischemia</b>	0.92	0.94	0.92	0.93	
<b>Classification model</b>	<b>Infection</b>	0.94	0.93	0.94	0.94	4GB
	<b>Ischemia</b>	0.98	0.99	0.97	0.98	

**Fig. 75** Results compare with the model size.

3. By implementing real-time data augmentation and avoiding using pretraining data augmentation to balance the dataset, models obtained are more likely to be used in real environments.
4. After understanding the task and people needs, it is much more valuable to develop classification models in diabetic foot field rather than object detection models. Person who is taken the picture can follow some normalization steps.
5. Google Colab is an essential tool to carry out this type of taks as it provides computational capabilities which are impossible to reach with standard computers. In this work Google Colaboratory Pro has been used, but in order to improve training times Google Colaboratory Pro + could be considered.

## Improvements

1. Due to dataset distribution, multilabel solution can be trained with a small dataset. On the other hand, using two binary classification algorithms allows to use a bigger database. For future works, trying to create a small-size model based on two classification algorithms could obtain good results.
2. The creation of an algorithm able to classify the wound grading, would be an important step to create a tool which carries out a complete diabetic foot disease diagnosis.
3. In future works, the models will improve their performance in real world applications by introducing more images into the database and repeating the training process. The more different images introduced in the training process the better general performance the models will achieve.

## List of references

- [1] Moi Hoon Yap, Ryo Hachiuma, Azadeh Alavi, Raphael Brüngel, Bill Cassidy, Manu Goyal, Hongtao Zhu, Johannes Rückert, Moshe Olshansky, Xiao Huang, Hideo Saito, Saeed Hassanpour, Christoph M. Friedrich, David B. Ascher, Anping Song, Hiroki Kajita, David Gillespie, Neil D. Reeves, Joseph M. Pappachan, Claire O'Shea, Eibe Frank, Deep learning in diabetic foot ulcers detection: A comprehensive evaluation, *Computers in Biology and Medicine*, Volume 135, 2021, 104596, ISSN 0010-4825, <https://doi.org/10.1016/j.compbiomed.2021.104596>.
- [2] Cassidy, B., Reeves, N. D., Pappachan, J. M., Gillespie, D., O'Shea, C., Rajbhandari, S., Maiya, A. G., Frank, E., Boulton, A. J., Armstrong, D. G., Najafi, B., Wu, J., Kochhar, R. S., & Yap, M. H. (2021). The DFUC 2020 Dataset: Analysis Towards Diabetic Foot Ulcer Detection. <https://doi.org/10.17925/EE.2021.17.1.5>
- [3] M. Goyal, N. D. Reeves, A. K. Davison, S. Rajbhandari, J. Spragg and M. H. Yap, "DFUNet: Convolutional Neural Networks for Diabetic Foot Ulcer Classification," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 5, pp. 728-739, Oct. 2020, doi: 10.1109/TETCI.2018.2866254. <https://ieeexplore.ieee.org/document/8464076>
- [4] Manu Goyal, Neil D. Reeves, Satyan Rajbhandari, Naseer Ahmad, Chuan Wang, Moi Hoon Yap, Recognition of ischaemia and infection in diabetic foot ulcers: Dataset and techniques, *Computers in Biology and Medicine*, Volume 117, 2020, 103616, ISSN 0010-4825, <https://doi.org/10.1016/j.compbiomed.2020.103616>.
- [5] Pendsey S. P. (2010). Understanding diabetic foot. *International journal of diabetes in developing countries*, 30(2), 75–79. <https://doi.org/10.4103/0973-3930.62596>.
- [6] The diabetic foot: a global view by Andrew Boulton on April 14, 1999 <https://www.woundsinternational.com/resources/details/the-diabetic-foot-a-global-view>
- [7] Zoph, Barret & Vasudevan, Vijay & Shlens, Jonathon & Le, Quoc. (2018). Learning Transferable Architectures for Scalable Image Recognition. 8697-8710. 10.1109/CVPR.2018.00907.
- [8] Papanas, N, and E Maltezos. "The diabetic foot: a global threat and a huge challenge for Greece." *Hippokratia* vol. 13,4 (2009): 199-204.
- [9] Diabetes-Related Foot Conditions, Cleveland Clinic. <https://my.clevelandclinic.org/health/diseases/21510-diabetic-feet>

- [10] Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
- [11] Fine-tuning ResNet with Keras, TensorFlow, and Deep Learning by Adrian Rosebrock on April 27, 2020. <https://pyimagesearch.com/2020/04/27/fine-tuning-resnet-with-keras-tensorflow-and-deep-learning/>
- [12] Tan, Mingxing and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." ArXiv abs/1905.11946 (2019)
- [13] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016 pp. 770-778 url: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90>
- [14] Szegedy, Christian & Ioffe, Sergey & Vanhoucke, Vincent & Alemi, Alexander. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. AAAI Conference on Artificial Intelligence.
- [15] J. Shijie, W. Ping, J. Peiyi and H. Siping, "Research on data augmentation for image classification based on convolution neural networks," *2017 Chinese Automation Congress (CAC)*, 2017, pp. 4165-4170, doi: 10.1109/CAC.2017.8243510.



## Information sources

- 1) Diabetic Foot Ulcer Challenge. <https://dfu2020.grand-challenge.org>
- 2) What is artificial intelligence? IBM Cloud Education. <https://www.ibm.com/cloud/learn/what-is-artificial-intelligence>.
- 3) Machine Learning. Coursera <https://www.coursera.org/learn/machine-learning/home>
- 4) Neural networks. IBM Cloud Education <https://www.ibm.com/cloud/learn/neural-networks>
- 5) Computer vision. IBM Cloud Education <https://www.ibm.com/topics/computer-vision>
- 6) Convolutional neural networks. IBM Cloud Education <https://www.ibm.com/cloud/learn/neural-networks>
- 7) Understanding and visualizing ResNets by Pablo Ruiz on October, 8 ,2018 <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>.  
[16]
- 8) What is batch size in neural network? <https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network>
- 9) Understanding Backpropagation Algorithm by Simeon Kostadinov on August, 8, 2019. <https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>
- 10) Binary Cross Entropy/Log Loss for Binary Classification by Shipra Saxena on March 3, 2021 <https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/#:~:text=What%20is%20Binary%20Cross%20Entropy,from%20the%20actual%20value>
- 11) Metrics to Evaluate your Machine Learning Algorithm by Aditya Mishra on February 24, 2018 <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>
- 12) NASNet - A brief overview <https://iq.opengenus.org/nasnet/#:~:text=NASNet%20stands%20for%20Neural%20Search,major%20breakthrough%20in%20AI%20soon>
- 13) Ranking de los países con mayor número de enfermos de diabetes en 2019. <https://es.statista.com/estadisticas/612458/paises-con-mayor-numero-de-personas-con-diabetes/#:~:text=Se%20estima%20que%20la%20diabetes.personas%20m%C3%A1s%20que%20en%202011>

14) EfficientNet <https://paperswithcode.com/method/efficientnet>