ANTONINE UNIVERSITY

Faculty of Engineering

Department of Computer and Communications

Bone fracture detection from X-RAY images using Artificial Intelligence and Computer Vision

Student(s)	Mansour Georges,201720539
Major	SW
Campus	BA
Supervisor(s)	Dr. Georges Badr

Spring 2023

ACKNOWLEDGMENT

My thanks go to Dr. Chady Abou Jaoude, Dean of the Faculty of Engineering – Department of Computer and Communications. Also, my thanks go to Dr. Georges Badr, my supervisor in this journey. His guidance and assistance through different stages of the project that greatly contributed to the development of this work.

Also, I would like to thank my friends and family for the mental support during the development of this work specially Charbel Mansour, my brother for helping me with ideas and thoughts to give the best result possible.

Specially, I appreciate the hard work of all the instructors of the Antonine University that prepared me to the market, and become the engineer that everyone can be proud of.

ABSTRACT

Technology in our days is evolving very quickly especially artificial intelligence. Artificial intelligence can be used in many fields especially the medical one. Bone fracture is a real issue that can occur to people at different ages, that is why it’s very important to have a practical solution that can assure for the patient if he has a fracture on more than one part of the body. As a result, the proposed solution is a desktop application where the user can upload an X-RAY image and the system will check if he has a fracture, location of the fracture (elbow, hand, finger, etc…) and finally, the fracture is marked with a square and a report will be sent to the user. During the development, many experiments and tests were made that led to a good accuracy of detection of the bone that can help the patient.

RESUME

De nos jours, la technologie évolue très rapidement, notamment l'intelligence artificielle. L'intelligence artificielle peut être utilisée dans de nombreux domaines, surtout dans le domaine médical. La fracture osseuse est un problème réel qui peut survenir à des personnes de différents âges, c'est pourquoi il est très important d'avoir une solution pratique qui peut assurer le patient s'il a une fracture sur plus d'une partie du corps. En résultat, la solution proposée est une application de «Desktop» où l'utilisateur peut télécharger une image radiographique(X-RAY) et le système vérifiera s'il a une fracture, l'emplacement de la fracture (coude, main, doigt, etc…) et enfin, la fracture est marquée avec un carré et un rapport sera envoyé à l'utilisateur. Pendant le développement, de nombreuses expériences et tests ont été effectués qui ont conduit à une bonne précision de détection de l'os qui peut aider le patient.

TABLE OF CONTENT

INTRODUCTION

This desktop application is related to the domain of application of the medical field especially the healthcare topic. Also, it’s related to the artificial intelligence field where it’s embedded in it.

To start with our current situation, going to a hospital is very difficult due to the high prices and the economic prices that is hitting the country. To add, the just the idea of going to the doctor for any simple checkout after dollarizing their cost is just way to expensive, next to the high gas prices, money trapped in the bank, NSSF not helping people due to the lack of cash flow and exactly and most important the high rate of the Lebanese Pound (LBP) in the face of the USD where 1$ is almost 100,000 LBP. All of the stated above issues can help the user and minimize the cost by using our application by helping him check for any possible fracture existing after just uploading a single X-RAY image. Moreover, the artificial intelligence (AI) now is at its finest, where it has a huge impact on the technology and everything happening around us. That is why, this desktop application will be simplifying the user’s need to pay less and visualize more to know if they need a medical interference or not.

People nowadays are afraid of the cost that they might pay if they visit a doctor or any specialist after a fall that may or may not hurt them, more precise aged people. Also, all the people can talk about is the crisis and the lack of medical equipment in the hospital next to its high cost and its low quality of service and treatment. As a result, to help minimize the cost and save money and time, this project falls into the artificial intelligence area. To solve the mentioned problem, this desktop application is formed only of four inputs field where one of them is the X-RAY image, and the others are the first name, last name, and email of the patient on which he will receive a report indicating if it has any fracture or not, health advice for which it indicated if he needs to check a doctor, go to hospital, etc .., a copy of the image uploaded with a square marker around the fracture if it has any, and finally the type of the bone which is selected, for example: hand, forearm, finger, and many more. Therefore, our goal is to minimize the cost on the patient, simplify his life and his needs, and eventually help him make more responsible and understandable decision because after all it’s his life and health we are talking about.

The first chapter includes the general idea of the project with some background study, related works and existing solutions next to a detailed understanding of the application area. In Chapter 2, the proposed solution will be explained the system methods and the algorithms used. Chapter 3 will include the software implementation in order to achieve the proper result and solution. In Chapter 4, a better understanding of the system, with all the tests, results, experiments done. And finally, a conclusion that contains an overview and a brief of the project with future works that is to be applied.

CHAPTER I: Related Work

1. Introduction

For the past few years, the artificial intelligence field has been the talk of the world by spreading its capabilities. Due to that, the choice was taken to develop a desktop application that aims to help people and facilitate their lives by uploading a simple X-RAY image that will be received as report about the medical case. This application's main focus is to help people and save them money and time in our hard times.

In this section, we will discuss the context and the domain of application concerning our desktop app, review and check the already existing solutions that helps solve this problem and compare it to the solutions found and developed. Next, this comparison will be used to state the main problem and the main objectives that this applications solves, and finally end our chapter by a brief conclusion.

2. Context and Domain of application

Topic of interest:

Artificial Intelligence:

Artificial Intelligence (AI) makes it possible for machines to learn from experience, adjust to new inputs and perform human-like tasks [1]. So this term refers to creating system that think, identify, and reason just like human beings do and also learn from past experiences to get better results in the future.

Domain of application:

Healthcare:

Healthcare was always the number one priority for any person ever existed on this earth, because it's the only thing that can't be bought. So taking care of yourself is always a priority, which is becoming better for diagnosing issue in the human body due to the advanced technology that entered the medical field.

Project Idea:

Therefore, this project will hit the objective directly by providing the user the ability to upload an X-RAY image and checking if it contains a fracture or not, where the fracture is, that means which bone (if it exists), and finally mark it so the user can see it better after receiving the report that shows the result and make the user take better decision concerning their health after just reading the health advice that exists for every case that can occur during the process.

So the technology is not always a bad choice that makes the person lazy but can also help him improve, making him take better choice and taking care of his self and his loved ones.

3. Existing Solutions/Methods

3.1. Deep Learning Approach

This study was made to develop a model that is able to classify and detect the fracture and the healthy bone. The target is to present a computer based system that reduces the time and improve effectiveness to reduce the error probability of the diagnosis.

This system can help solve many issues, most important the time reduction. Bone fractures are mostly caused by automobile accident or bad fall, also the risk of the bone fracturing is high with old or aged people due to its weakness [2]. This system is made to detect the fracture with a faster and more precise way because the small fractures are difficult to be analyzed by the doctors.

In a general way, this model and solving architecture is based on a Convolutional Neural Network (CNN) model where a big dataset is implemented so that the model can train in a better way to receive more accurate result in the testing phase. The dataset implemented has many sources.

As for the implementation, the four steps of the deep convolution neural network was implemented which were: convolution, pooling, flatten and dense layer. After implementing these steps, three experiments were made, but we will talk about the first two:

In the first experiment, the model was trained with 90% of the sample data and the rest 10% were the testing. Instead, in the second experiment, the model was trained with 80% of the sample data, and the rest 20% were used for testing purpose.

As for the results, **Table 1** shows the results of the first experiment where the performance of the function *adamax* is much better than the *softmax* in terms of accuracy and the loss metrics.

Table 1: Results of experiment 1

Experiment Id	Experiment description	Accuracy Type		Loss Type	
Experiment 1		Tr-acc	VL-acc	Tr-loss	VL-loss
	Healthy Bone Using	96.12%	92.25%	0.075	0.1023

	softmax				
	Fracture Bone Using softmax	97.52%	93.04%	0.067	0.9704
	Healthy Bone Using Adam	98.03%	95.23%	0.046	0.9807
	Fracture Bone Using Adam	98.45%	95.67%	0.035	0.5608

As for the results of the second experiments, **Table 2** shows it where the function *adamax* was also much better than the *softmax* in the terms of accuracy and the loss metics

Table 2: Results of experiment 2

Experiment Id	Experiment description	Accuracy Type		Loss Type	
Experiment 2		Tr-acc	VL-acc	Tr-loss	VL-loss
	Healthy Bone Using softmax	95.03%	91.42%	0.085	0.1723
	Fracture Bone Using softmax	96.42%	92.02%	0.077	0.9812
	Healthy Bone Using Adam	96.18%	92.25%	0.052	0.1107
	Fracture Bone Using Adam	97.45%	94.67%	0.045	0.6102

3.2. Artificial Neural Network Based on Line Features

The purpose of this paper is to detect fracture lines from X-RAY images using extracted features. As for the implementations, two methods were made.

The first method is the Standard line based fracture detection, that takes the processed image and then apply a line detection technique (Probabilistic Hough Transform). And after the line detection, the line based feature extraction with 13 feature is used to train and test the Artificial Neural network (ANN) as we can see in **Figure 1**. Meanwhile, a reduction was performed through the process by using the principal component analysis (PCA).

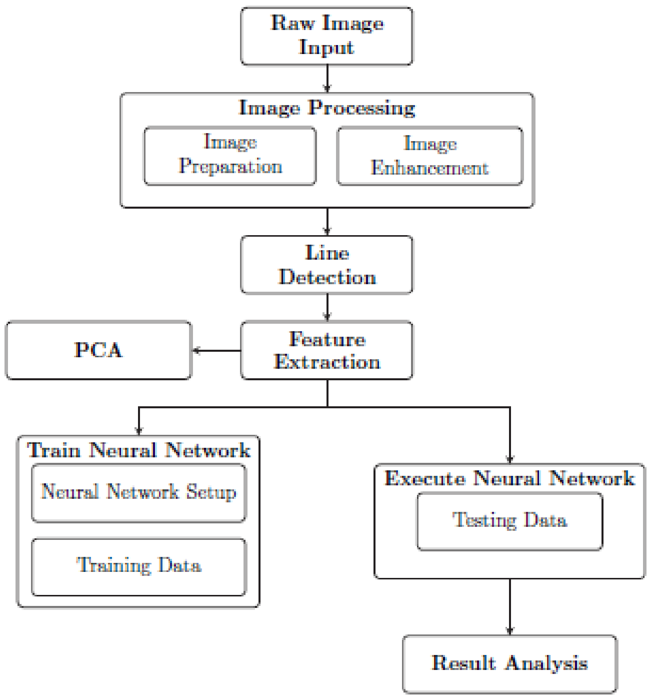


Figure 1: Flow chart of Method 1

As for the second method, it's the Adaptive Differential Parameter Optimized (ADPO). In this method the same methodology was used but without the line extraction technique and different parameters that is passed were changed (threshold, the minimum line length, maximum line gap) which has different values from method.

After implementing the two methods, we have two results:

- For the Standard line based fracture, the accuracy is around 71.57%
- For the ADPO, the accuracy is around 72.89

Also, the sensitivity of the ADPO is better than the Standard line based fracture.

3.3. Fusion Technique

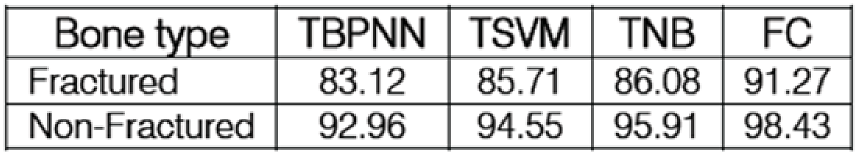
The goal of this paper of this paper is to detect Tibia fracture detection. *Tibia,* also called *shin,* inner and larger of the two bones of the lower leg in vertebrate [3].

To implement this model, the solution is divided into 3 main steps:

- Pre-processing to enhance the features of the X-RAY image using SACEN algorithm (Simultaneous Automatic, Contrast adjustment, Edge Enhancement and Noise removal).

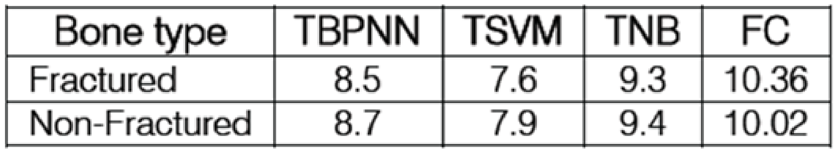
- Segmentation which is divided into two algorithms. The first one segments the bone image from the X-RAY and the second one describes the technique that identified the center of a long bone (Tibia) region of a bone image. These two are used to narrow down the search during the fracture detection step.
- Fracture detection uses multiple classifiers where all the classifiers works as binary, that means it checks only if it has a fracture or not. And the basic fusion rule is used that says if more than two classifiers reports fracture, then the image contains a fracture.

As for the results, **Figure 2** and **Figure 3** shows the detection rate and the detection speed of the model for the four different classifiers used:



Bone type	TBPNN	TSVM	TNB	FC
Fractured	83.12	85.71	86.08	91.27
Non-Fractured	92.96	94.55	95.91	98.43

Figure 2: Detection rate



Bone type	TBPNN	TSVM	TNB	FC
Fractured	8.5	7.6	9.3	10.36
Non-Fractured	8.7	7.9	9.4	10.02

Figure 3: Detection speed

The disadvantage of this model that it’s not easy to use and this works only and specifically on Tibia and not every bone in the human body.

3.4. Feature Ambiguity Mitigate Operator (FAMO)

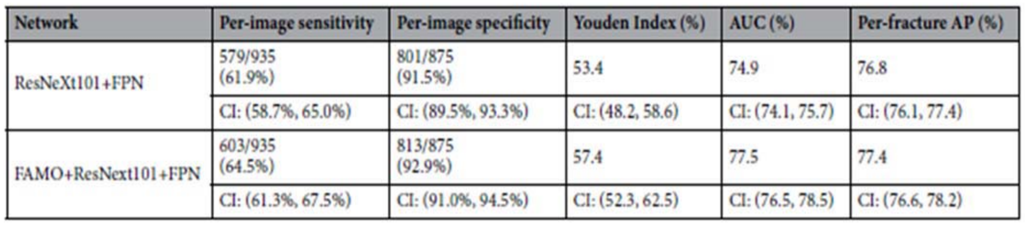
The main target of this paper was to propose a mitigate feature ambiguity in bone fracture detection by using FAMO + ResNext101 + FPN and compare it with the result in ResNext101 + FPN.

To implement this model, three main steps where made:

- Pre-processed radiographs were firstly sent to the initial encoder which is ResNext101.

- The feature pyramid network (FPN) collected the feature maps in different scales produced by ResNext101, and fused them to generate features.
- Forwarded by region proposal decoder (RPN) decoder to make global box predictions. To add, for the image labeling check, the DCNN (Deep Convolutional Neural Network) was used. [4]

The following figure, **Figure 4,** shows the result and the comparison of the model stated above



Network	Per-image sensitivity	Per-image specificity	Youden Index (%)	AUC (%)	Per-fracture AP (%)
ResNeXt101+FPN	579/935 (61.9%)	801/875 (91.5%)	53.4	74.9	76.8
	CI: (58.7%, 65.0%)	CI: (89.5%, 93.3%)	CI: (48.2, 58.6)	CI: (74.1, 75.7)	CI: (76.1, 77.4)
FAMO+ResNext101+FPN	603/935 (64.5%)	813/875 (92.9%)	57.4	77.5	77.4
	CI: (61.3%, 67.5%)	CI: (91.0%, 94.5%)	CI: (52.3, 62.5)	CI: (76.5, 78.5)	CI: (76.6, 78.2)

Figure 4: Result and comparative table of the models

This solutions has a disadvantages that the knee fracture were hard to be recognized. On the other side, the advantages that it increased the detection, and also increased the sensitivity and the specificity.

3.5. Comparative Study

Our proposed solution is an upgraded version that can easily be a competitor to the other solutions. It is user friendly, it's also application based that users can check and see the results and understand it with a simple sentence and way. The following **Table 3** shows the difference between the similar solutions explained above and our proposed solution.

Table 3: Comparative table between other solutions

	Deep Learning Method	Line Features Method	Fusion Method	FAMO	Proposed Solution
Free	N/A	N/A	N/A	N/A	Yes
Easy to use	No	No	No	No	Yes
Fracture location specification	No	Yes	Yes	No	Yes
More than one bone	Yes	N/A	No	Yes	Yes
Report	No	No	No	No	Yes

4. Problem Statement and Objectives

Lebanon is going through multiple crisis one of the huge economic crisis that started in the late 2019 and was ranked one among the worst economic crisis globally since the mid-nineteenth century by the World Bank [5]. This crisis was reflected on many levels where the people's money is sitting in the bank not allowed to be used even in case of emergencies.

To add, everything has changed due to the exchange rate in the black market where 1 USD is somehow 100 000, which also led to the lack of equipment in the hospital with the massive price to do a simple test [6], and led to a lower quality of caring towards the patient.

So, how can we help and save money and time to the patient? People are searching for the anything that can allow them to save some money in these hard times, also they are preferring not to go to the doctors and the hospitals because of the high prices and not only of the medical field, but also of the gas prices, the hard to travel using public transportation and much more.

Our topics of interest are Artificial intelligence that is included in the medical field and the healthcare option. The problem stated above falls into this topic because we are already in the AI Era where technology and especially AI is being used for the benefit and help of the human being to solve all our daily problems and facilitate the way of living.

In our project, after getting an X-RAY image, the user must enter its first name, last name and email on which he will receive a report. Next, the user must import an X-RAY image on which he want to know if it has any fracture or not, which bone it is and finally if it contains the fracture mark it and know it's correct location on the image. Last but not least, on the email provided by the user, he will be receiving a pdf file containing all the results that needs to be informed:

- If it has any fracture or not
- A health advice that is given to the user to know what to do(go to the doctor, it's safe, …)
- The bone on which it has the fracture, if it has any (hand, forearm, finger, etc…)
- The image that was uploaded by the user, marked if it has a fracture, and the same image if it doesn't has any fracture.

Our goal is to help the patient by saving him money and time but providing the user with the necessary informations about his case and help him make better decisions in the current situation that the entire country is living.

5. Conclusion

In conclusion, a lot of research and studies has been made to help the patients and the hospitals with a safe result. Some of these studies were explained and described in this chapter along with a comparison table between these already existing solutions and the one we were working on.

CHAPTER II: Proposed Solution/ Design/ Method

1. Introduction

In this chapter, we will dive into the details of the proposed solution and its design architecture, where basically it's divided into two main parts which is the artificial intelligence part and the desktop application part. The artificial intelligence it's also divided into three parts.

We will start about talking the project's architecture which is in the core made with machine learning and computer vision, then we will talk about the system architecture and how the user will interact with the desktop app. And finally, the flow of the project will be explained.

2. Design of the proposed solution

2.1. Solution architecture (project architecture)

In general, our solution is based on WPF .Net Framework. WPF is a subset of the .NET Framework that allows the user to use an efficient desktop app. While developing a desktop app, it's highly recommended to use WPF, it's easy to use, controls with highly efficient and powerful data-binding capabilities and create the user interface (UI) with limitless possibilities [7]. The .Net Framework facilitate the development phase by providing a big amount of packages that helps get the result needed. Also, the AI and computer vision models were developed using Python, the most common language for machine learning in our days because of the many libraries that exists.

2.2. Refined solution architecture (system architecture)

In this section, we will talk about the solution architecture, explaining how our application works, and how the result is received.

In the following figure, **Figure 5**, shows the flow chart of the system:

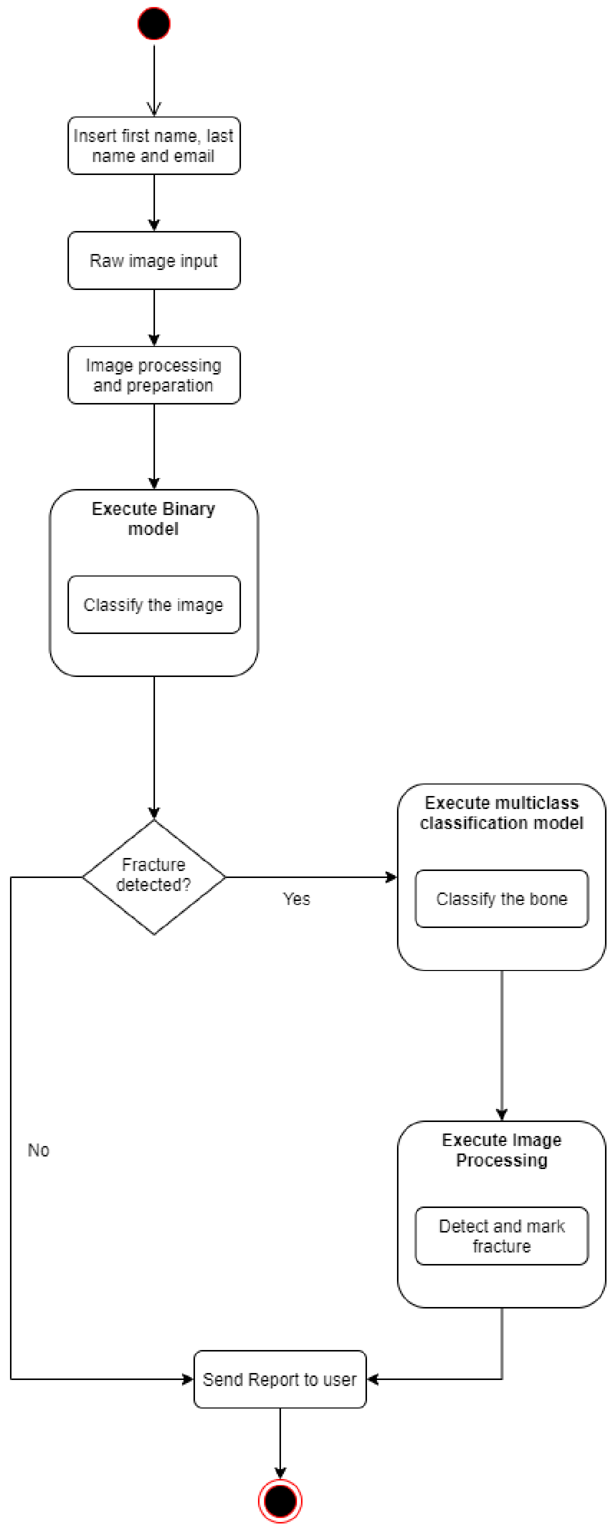


Figure 5: Flow chart of the application

The flow chart shown in **Figure 5** summarize the whole user usage of the application. First, the user should add his first name, last name and email, then he should browse and select the X-RAY image that the user want to check for any fractures. Before processing the image, the email is validated for being in the correct format, and that all the fields are not empty especially the image that is selected. Secondly, the image uploaded is taken, processed and prepared so it matched the input of the models. The first model executed is the binary model, which gives a result only if the image contains a fracture (positive), or the bone is healthy and doesn't contain any fracture (negative).

After this binary classification, if the image doesn't contain a fracture, a report will be sent to the user, otherwise, if it has a fracture, the processed image will be given to another model to classify which bone is being fracture. For example, this model will tell the user that the bone in this image goes to the forearm, or the fingers (depends on each image).

Furthermore, after this multi class classification process, the image is given to a computer vision model to mark the fracture in the image. The image is mark with a red square and the image is saved to be sent later on to the user in the report received by email.

3. Practical solution

This proposed solution is the most practical and optimal solution the user can use. To start with, this solution contains a simple interface that the user or the patient can interact with, different that the existing solutions.

Secondly, it provides the user with the necessary informations to take better choice and can help know more about the case he is checking on this desktop application. As for the application, this solution doesn't contains any identification needs like log in or sign up, just because the user might be in a hurry if it's a critical case. So why bother him with sign up or sign in forms just to get after all a report that he will be waiting so bad?

Thirdly, the accuracy hit by the AI is better than most of the already proposed solutions. This accuracy is better on the models that is provided by the system above with are one model for object detection and the other two for the classification where in all the solutions, the classification doesn't work on more than one bone, always works on only one bone and it's not very user friendly and explainable.

4. Conclusion

The goal of this chapter is to understand more how to solution works. First, we gave brief description of our language used which is WPF in .Net Framework.

Secondly, an explanation was made to understand more the flow chart provided and how the applications works. And finally, a small explanation was provided to understand why this solution is the most practical and optimal solutions.

CHAPTER III: Development and Implementation

1. Introduction

In this chapter, we will explain how we created our solution and implemented it step by step. First of all, we will explain a bit about our development environment. Second, we will cover all the software functionality by going through all the steps, from programming languages to libraries and tools used, and the combination to create the desktop application.

2. Software development and implementation

2.1. Development environment

To start with, the main core of our project for any tests and development were made on “kaggle”, **Figure 6**. “kaggle” is an online community platform for data scientists and machine learning enthusiasts [8], that allows users to develop, test, and create AI models, and also allows users to interact with each other and share datasets and notebooks.

Figure 6: kaggle logo

We first started with “Google Colab”. “Google Colab” limits it’s user to a maximum of 12.7 GB RAM and can’t use GPU in an easy way to speed up the process on the free version. That is why we switched to “kaggle”. “kaggle” has a higher limitation to its users on the free version, it allows you to use up to 30 GB RAM with a runtime session continuous for 12 hours and a storage size of 73.1 GB when no accelerator is being used. On the other hand, when an accelerator is being used (GPU or TPU), the maximum use is 13 GB RAM and 14.8 GB GPU memory but with a runtime of 9 consecutive hours instead of 12 hours.

So basically in “kaggle”, we developed our models and saved the models and then we downloaded it to get the final result that we will be explaining.

2.2. Software functionalities

To more understand how we achieved the following results, we will be dividing this section into 5: Data pre-processing, binary classification, multi class classification, object detection, and finally the desktop application that combines all three models together.

a) Data pre-processing:

Data pre-processing is an important stage when training AI models. It usually transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks [9].

There is many ways to pre-process data, for example change brightness, rotate an image, zoom in, and many more. While developing the project, we tried many ways of the stated above and additionally we tried the edge detection, binary images (convert images to black and white), and a combination of together.

The most optimal solution was the data augmentation. Data augmentation is when you have a dataset but the models are not being able to train correctly to get accurate result, so we use data augmentation to extends the number of dataset by changing the images where the models considers it as new inputs and create a new and a larger dataset. The following two figures (**Figure 7 and Figure 8**) shows the data before and after augmenting it for the same image.

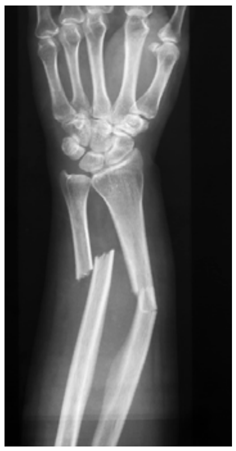


Figure 7: Image of a bone before augmentation

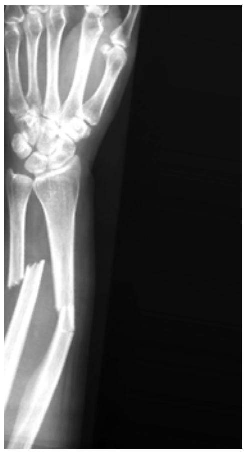


Figure 8: Image of the same bone after augmentation

The image shown in **Figure 8**, is being zoomed, shifted and rescaled, brightness changed within a specific range, and rotated. Other images may be flipped horizontally and vertically.

As part of the augmentation, each image is being augmented 5 times. First rotated with 60 degrees, change brightness, zoomed in, flipped horizontally and vertically, and finally the last one combines the above next to copying the original image. So from 530 images we were able to create 2571 images in total as a new dataset and splitting it into training dataset and testing dataset, where the testing dataset is 15% of the full dataset.

b) Binary classification:

Binary classification in simple words, is when a model is trained to classify an image between two classes only. In our case, these two classes are: positive and negative. The positive class, as it says, refers when an image contains a fracture, on the hand, the negative is the opposite.

The most accurate model we got is the VGG16 model. VGG16 is a CNN that is 16 layers deep. **Figure 9**, shows the architecture of the used model.

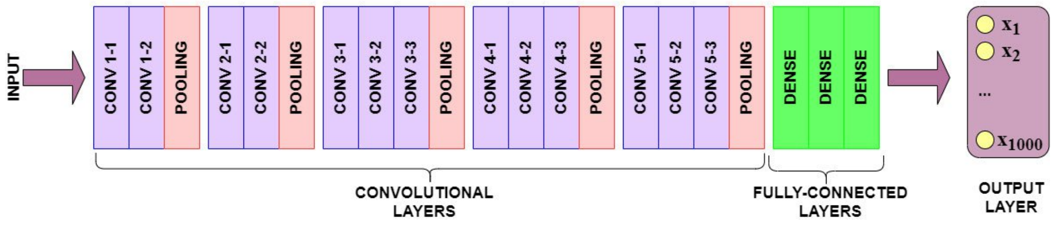
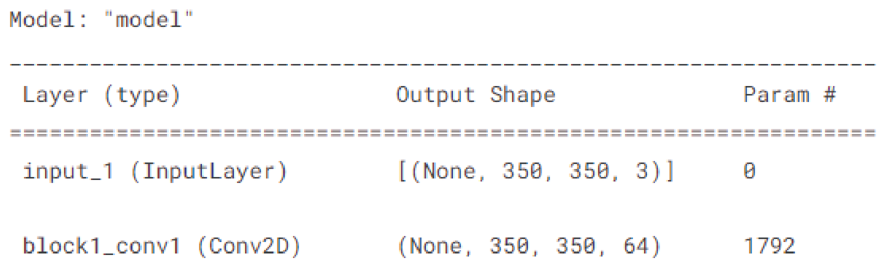


Figure 9: VGG16 architecture

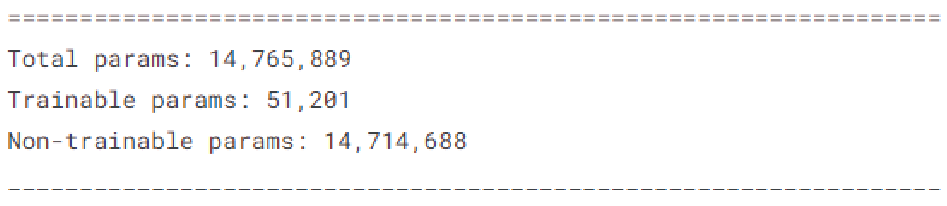
It’s better for the model to set the input images on a specific size, so the size we selected was 350x350. First, we imported the needed libraries like “numpy”, “matplot” and “keras” with preprocessing, models, applications and many more. Then we created the VGG16 model and gave the input shape as a 4 dimensions input, and we loaded the pre-trained weights “imagenet”. Next, we set the layers to not be trainable, and a created a flatten layer. After these initial steps, we assigned the inputs and outputs to a model and checked it’s summary where it shows that the output of the first layer is ”[(None, 350,350,3)], which indicates the size of image and the dimensions of the colors (RGB) as shows **Figure 10**, and contains total parameters as 14,765,889 as shows **Figure 11.**



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 350, 350, 3)]     0

 block1_conv1 (Conv2D)       (None, 350, 350, 64)      1792
```

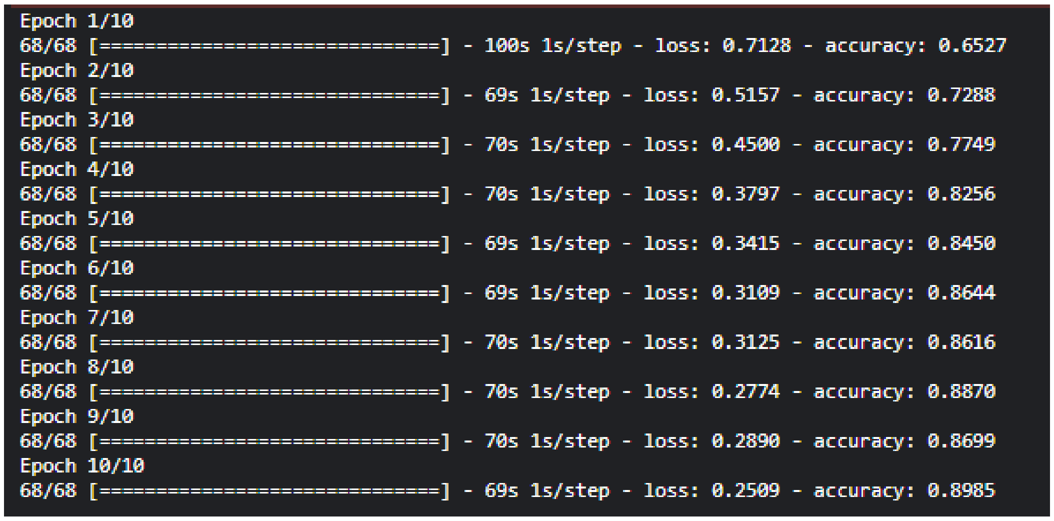
Figure 10: VGG16 binary model first layer output



```
=================================================================
Total params: 14,765,889
Trainable params: 51,201
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 11: VGG16 binary number of parameters

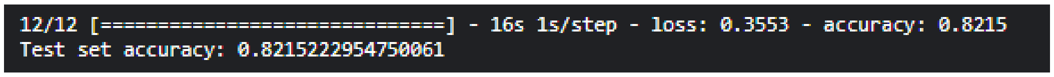
After checking these informations, we compiled our model with the *adam* optimizer and set the metrics as accuracy. Next, we loaded the images with a batch size of 32, class mode to binary, and assigned it to the model as training set, set the number of epochs to 10 and the steps per epoch is the length of the training set. The obtained results hit a training accuracy around 89.86% as shows **Figure 12**.



```
Epoch 1/10
68/68 [==============================] - 100s 1s/step - loss: 0.7128 - accuracy: 0.6527
Epoch 2/10
68/68 [==============================] - 69s 1s/step - loss: 0.5157 - accuracy: 0.7288
Epoch 3/10
68/68 [==============================] - 70s 1s/step - loss: 0.4500 - accuracy: 0.7749
Epoch 4/10
68/68 [==============================] - 70s 1s/step - loss: 0.3797 - accuracy: 0.8256
Epoch 5/10
68/68 [==============================] - 69s 1s/step - loss: 0.3415 - accuracy: 0.8450
Epoch 6/10
68/68 [==============================] - 69s 1s/step - loss: 0.3109 - accuracy: 0.8644
Epoch 7/10
68/68 [==============================] - 70s 1s/step - loss: 0.3125 - accuracy: 0.8616
Epoch 8/10
68/68 [==============================] - 70s 1s/step - loss: 0.2774 - accuracy: 0.8870
Epoch 9/10
68/68 [==============================] - 70s 1s/step - loss: 0.2890 - accuracy: 0.8699
Epoch 10/10
68/68 [==============================] - 69s 1s/step - loss: 0.2509 - accuracy: 0.8985
```

Figure 12: VGG16 binary model training accuracy

After training the model, it’s saved, loaded, and ready to be tested where it hit an accuracy around 82.15% on the testing dataset as shows **Figure 13**.



```
12/12 [==============================] - 16s 1s/step - loss: 0.3553 - accuracy: 0.8215
Test set accuracy: 0.8215222954750061
```

Figure 13: VGG16 binary model test accuracy

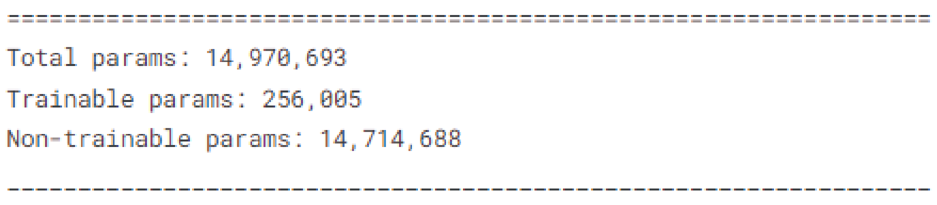
c) Multi class classification:

Multi class classification in simple terms is when classifying a specific image into more than two classes. In our case, the classification of a bone image can refer to hand, finger, forearm, humerus and elbow. That means we have defined and trained our model based on 5 classes.

The model used here is the same as the binary which is VGG16, but with different behavior whereas, in this case, it’s informing us which bone is having the fracture.

Without repeating the above, the dataset used is the augmented dataset for each class contains in total 2571 images, the same as binary, but when importing the model, different activation is used.

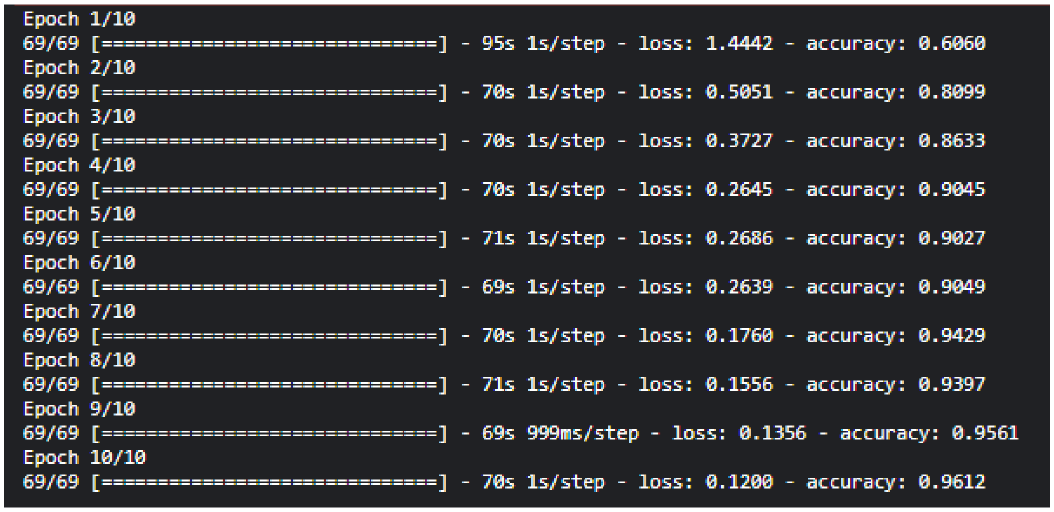
After importing the corresponding libraries, we first started by setting the size of images to 350x350, and then created the VGG16 model, assigned the input shape as 4 dimensions like the binary, set the weights to pre-defined weights which is “imagenet”, set the trainable layers to false. Next, we created a flatten layer and assigned the output to it, created a dense layer as length is the number of classes (5 classes in our case), but the activation as "softmax" which is used for multi class classification. To add, we assigned the inputs and outputs to a model and checked its summary where we got a total of 14,970,693 parameters that the model will train in. The following figure, **Figure 14**, will show more details about the parameters.



```
=================================================================
Total params: 14,970,693
Trainable params: 256,005
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 14: VGG16 multi class model total parameters

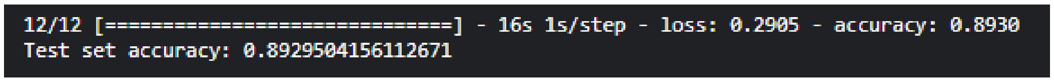
To add, to compile the model, we set the loss as "categorical_crossentropy" because it's classification with the same optimizer and metrics as binary which are adam for optimizer and accuracy for metrics. Then we imported the data to train the model where we got 2188 image belonging to 5 classes after setting the batch size to 32 and the class mode to categorical. We added the images loaded earlier to the model, we set the number of epochs to 10 and the steps per epochs is equal to the length of the training set. The obtained result hit a training accuracy with around 96.12%% as indicated **Figure 15**.



```
Epoch 1/10
69/69 [==============================] - 95s 1s/step - loss: 1.4442 - accuracy: 0.6060
Epoch 2/10
69/69 [==============================] - 70s 1s/step - loss: 0.5051 - accuracy: 0.8099
Epoch 3/10
69/69 [==============================] - 70s 1s/step - loss: 0.3727 - accuracy: 0.8633
Epoch 4/10
69/69 [==============================] - 70s 1s/step - loss: 0.2645 - accuracy: 0.9045
Epoch 5/10
69/69 [==============================] - 71s 1s/step - loss: 0.2686 - accuracy: 0.9027
Epoch 6/10
69/69 [==============================] - 69s 1s/step - loss: 0.2639 - accuracy: 0.9049
Epoch 7/10
69/69 [==============================] - 70s 1s/step - loss: 0.1760 - accuracy: 0.9429
Epoch 8/10
69/69 [==============================] - 71s 1s/step - loss: 0.1556 - accuracy: 0.9397
Epoch 9/10
69/69 [==============================] - 69s 999ms/step - loss: 0.1356 - accuracy: 0.9561
Epoch 10/10
69/69 [==============================] - 70s 1s/step - loss: 0.1200 - accuracy: 0.9612
```

Figure 15: VGG16 multi class model training accuracy

The model is trained, saved, loaded and ready to be tested where it hit a testing accuracy around 89.3% as it's showed in **Figure 16**.



```
12/12 [==============================] - 16s 1s/step - loss: 0.2905 - accuracy: 0.8930
Test set accuracy: 0.8929504156112671
```

Figure 16: VGG16 multi class testing accuracy

d) Object detection:

Object detection is a computer vision technique for locating instances of objects in images or videos [10]. Object detection can happen on live camera images, live or normal videos on any types of images. Object detection can simplify the needs when detecting an object and localizing it in a specific image given to the model.

The most used object detection models in the computer vision world is the YOLO (You Only Look Once). YOLO is an important model for detecting, predicting, and localizing an object in an image. It has many versions starting from version 3 (YOLOv3) to the newest version that is version 8 (YOLOv8). The difference between the versions of YOLO is the precision of the detection and the prediction, and each version is faster while compiling and doing it's necessary job that the version before. To add, YOLO train its model by treating the "thing" to detect as an object. The input is by selecting the object in the training images so it can have a clear vision while the training is happening.

In the proposed solution, the latest YOLO version is used which is YOLOv8. We used "Roboflow", **Figure 17**, to mark the images. "Roboflow" is a software that allows the developers to create and maintain their own computer vision models in a simple and effective way.

Figure 17: Roboflow logo

The dataset used to train our model is the augmented dataset but filtered and marked. First, we uploaded our dataset to "roboflow" and we doubled check if all the images were clear, the fractures were visible to mark it before training the model.

Then we created one class named “Fracture”. This class is the one that the custom model will be detecting out of the bone fracture images given to it. Next, all the uploaded images were marked, as it can be seen in **Figure 18**.

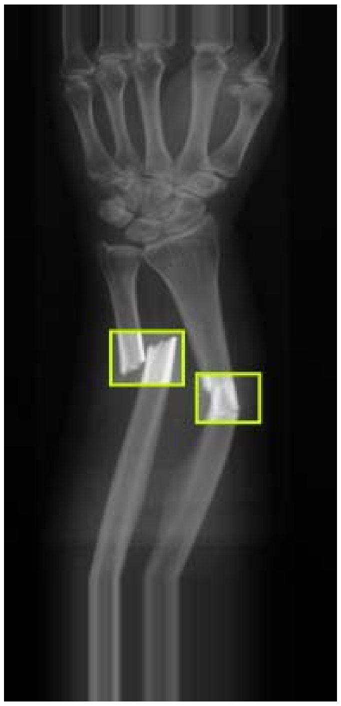
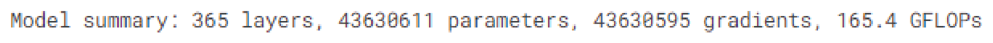


Figure 18: Example of images on roboflow after marking

After the marking is finished, the dataset is split into training set, validation set and testing set. The total images uploaded were 1036 image, where 15% randomly chosen to be the test dataset and 10% is the validation dataset. After creating the images that will be used to train the YOLO, a script was generated for the specific version of YOLO (YOLOv8), to download the images and start the training without physically owning the dataset.

To start, YOLOv8 on “kaggle” was treated as its being used on the command line. For example, the command “!pip install ultralytics” was used to download the specific library from which the YOLO is used. As for the library of “Roboflow”, it was installed to be able to import the images marked and saved in the application through the generated script given. After importing the specific dataset, the model to be saved is a custom trained model, which means that the result that we will obtain will only be able to detect the fractures due to its class singularity and specificity when being pre-processed on the “Roboflow” application.

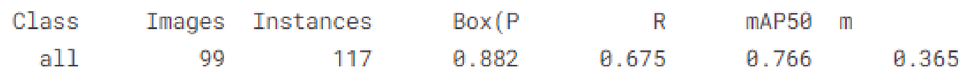
To train the model, we chose the task to train on is the detect task, the mode is train and we chose the “yolov8l.pt” model. YOLOv8 contains many model with different architecture, the selected one is the large scale that is used for large projects and datasets. The batch size chosen is 32, the number of epochs is 300 and the image size is 350. The model summary is shown in **Figure 19**, where it’s being shown the total number of parameters which is 43,630,611 parameters with 365 layers to train the model.



```
Model summary: 365 layers, 43630611 parameters, 43630595 gradients, 165.4 GFLOPs
```

Figure 19: YOLOv8 model total parameters and layers

After checking the summary, the model is trained and saved the best model with a training accuracy of 76.6% as shown in **Figure 20.**



```
Class     Images  Instances      Box(P          R      mAP50   m
  all         99        117      0.882      0.675      0.766      0.365
```

Figure 20: YOLOv8 model training accuracy

To understand if the model, the following graphs shown in **Figure 21** shows all the criteria on which the best model was chosen, starting the training set to the metrics accuracy.

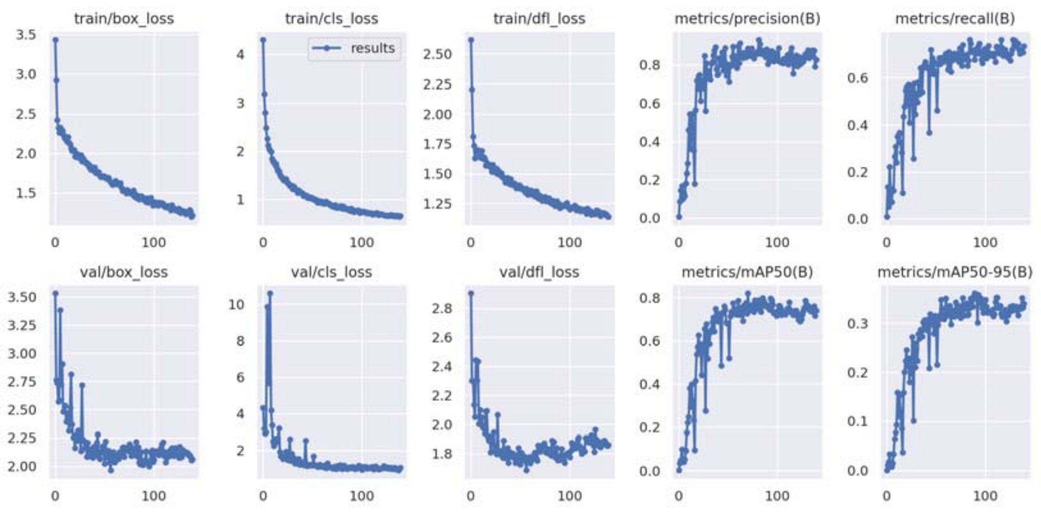


Figure 21: Results of YOLOv8 training

Figure 21, shows the stability in the training class loss, where it decreased in a correct way similar to the validation set. Also, the accuracy increased to hit the best option which is the 76.6% with the lowest class loss on both training and validation dataset. To add, after finishing the training of the model, it’s saved and loaded to be validated and tested. The validation precision it around 76.7% which is very close and similar to the training precision. **Figure 22**, shows some the images that the model was validated on.

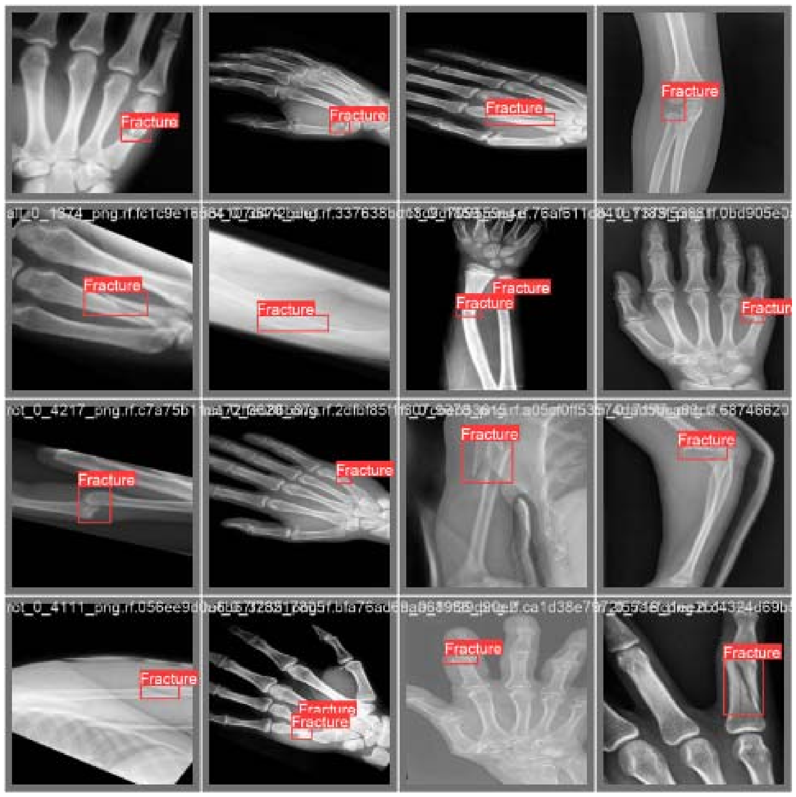


Figure 22: Some of the validation dataset images

After validating the model, we tested it on the testing dataset, and the results were saved, similar to the ones validated but with a precision measure. **Figure 23**, shows ones of the results of the testing dataset where the model detected the fracture.

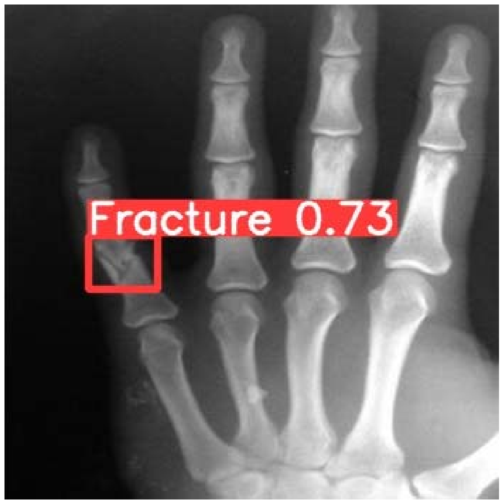


Figure 23: YOLOv8 testing result

e) Desktop application:

The idea of the desktop application is very simple. It's created to combine all three models together so that the user can simply work with it easily. The application was developed using visual studio 2022 with the language C# and .Net 6 framework (WPF).

The desktop application contains 4 inputs and 2 buttons. The first 3 inputs are first name, last name and email of the user. The 4th input is the image chosen. The browse button is used to browse for the image and the other button "check" is used to get the result and send the email directly to the inserted email after validating it first.

Before starting the development of the desktop application, all the models developed in the previous sections were saved in a folder named "models". Then a python file is written that contains for start, the specific libraries imported, all the models imported and all the conditions for classifications and detecting exists. For example, the images that will be loaded by the user is saved in a specific directory with a stable physical path, this image is passed to the models. If the image has a fracture (binary classification), the same image is forwarded to the categorical classification (multi class classification) to know the type of the bone, and also is being given to the YOLO model to detect the fracture. The results of each model is being written in a text file formatted as shown in **Figure 24**.

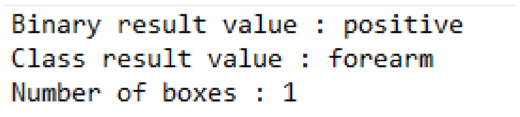
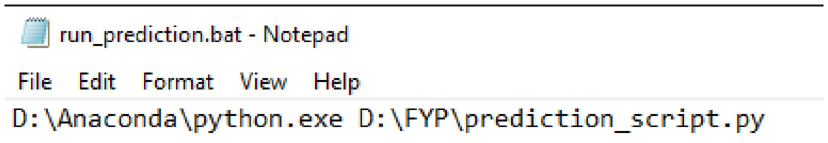

```
Binary result value : positive
Class result value : forearm
Number of boxes : 1
```

Figure 24: Results of the model saved in a txt file

To be able to run the python script from the desktop application, a batch file contains the path of the python script is saved and will be triggered from the code in visual studio. The batch file that contains the path is shown in **Figure 25**.



```
D:\Anaconda\python.exe D:\FYP\prediction_script.py
```

Figure 25: Batch file content

Now, let's start with the desktop application. The first name, last name and email are required fields next the image needs to be chosen. After pressing the "check" button, the batch file is ran triggered through a process being started in the code. Then, after the result of the python running is saved in the txt file, this last is read, and the result is filled in a dictionary containing keys and values for each case. After, this dictionary is passed to a function that adds the health advice to it, based on the results saved. In total there is three cases for the health advice:

1. The binary classification is negative, that means the user is safe
2. The binary classification is positive and the boxes drawn by the object detection model is 0, that means the application wasn't capable of making sure if it has a fracture
3. The binary classification is positive and the boxes drawn by the object detection model is greater than 0, which means there is a fracture detected for sure.

After securing the advice, the report is set to be ready. All the informations captured above is being delivered to an html file after converting to pdf. To convert the html file to pdf, the library "DinkToPdf" is being used next to an unmanaged library "libwkhtmltox.dll". The margins are set to default with a portrait format and a paper kind of A4. After saving the pdf with all the proper informations, it's being sent automatically to the email entered by the user using the "MailKit" library and the "BodyBuilder" to attach the pdf document.

All the explanation of the desktop app can be minimized with the flow chart in **Figure 26.**

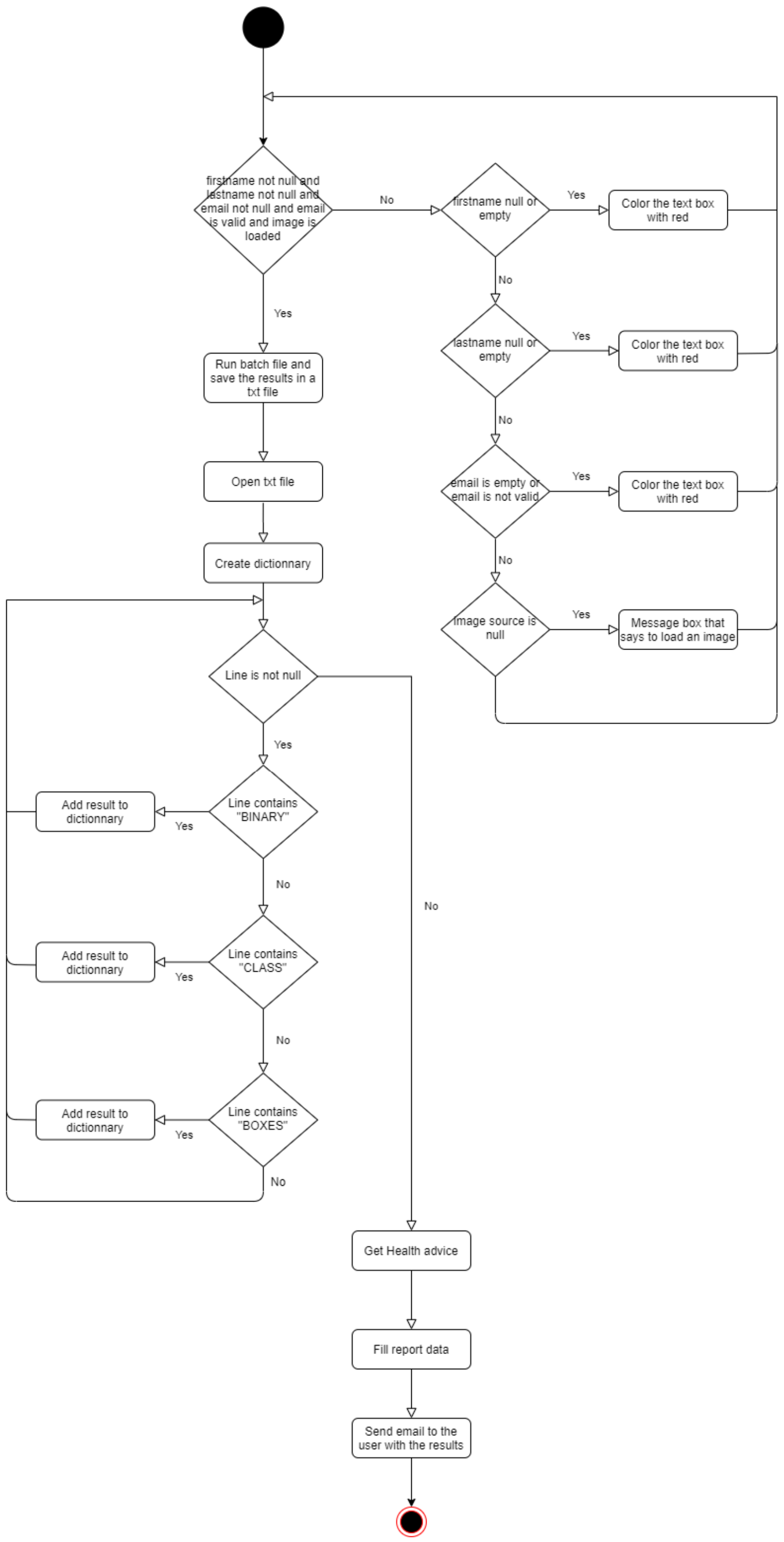

firstname not null and lastname not null and email not null and email is valid and image is loaded
No
Yes
firstname null or empty
Yes
Color the text box with red
No
lastname null or empty
Yes
Color the text box with red
No
email is empty or email is not valid
Yes
Color the text box with red
No
image source is null
Yes
Message box that says to load an image
Run batch file and save the results in a txt file
Open txt file
Create dictionnary
Line is not null
Yes
No
Line contains "BINARY"
Yes
Add result to dictionnary
No
Line contains "CLASS"
Yes
Add result to dictionnary
No
Line contains "BOXES"
Yes
Add result to dictionnary
No
Get Health advice
Fill report data
Send email to the user with the results

Figure 26: Desktop app results flow chart

3. Conclusion

To conclude, in this chapter we presented the features of this application and we divided into 5 steps in the order on which we worked on. We explained the flow and the methodology and the choice of the selected technologies. Furthermore, we explained how we implemented the functions and combine it together and showed the results obtained.

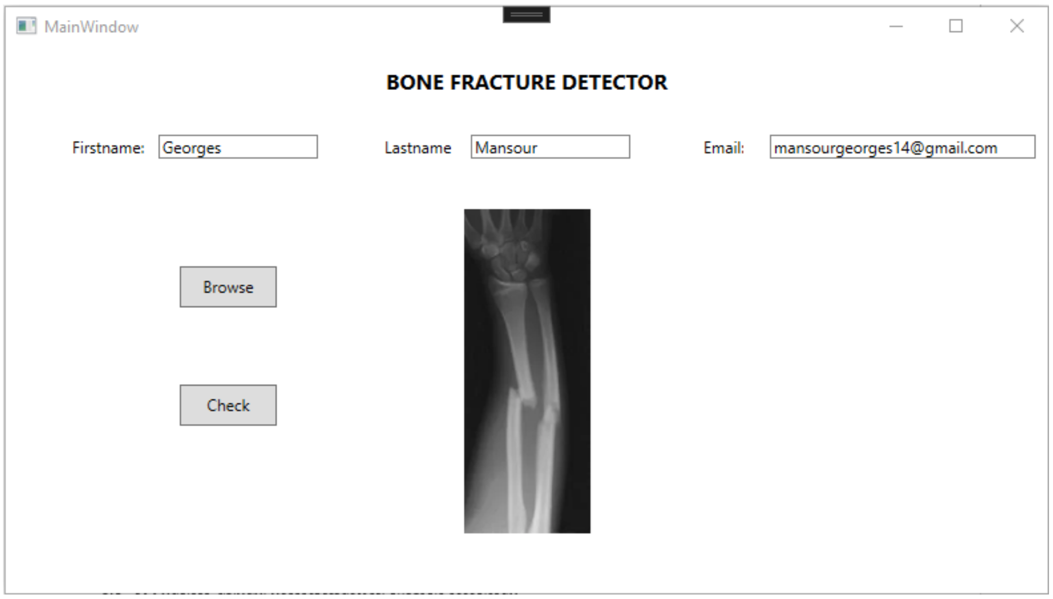
CHAPTER IV: Experiments and Results

1. Introduction

In this chapter, we will be presenting the prototype of our application, then we will show multiple tests done for every model created. Next, we will discuss system setup. Finally, we will take a look at the results obtained, interpret and evaluate the solution.

2. Prototype/Application

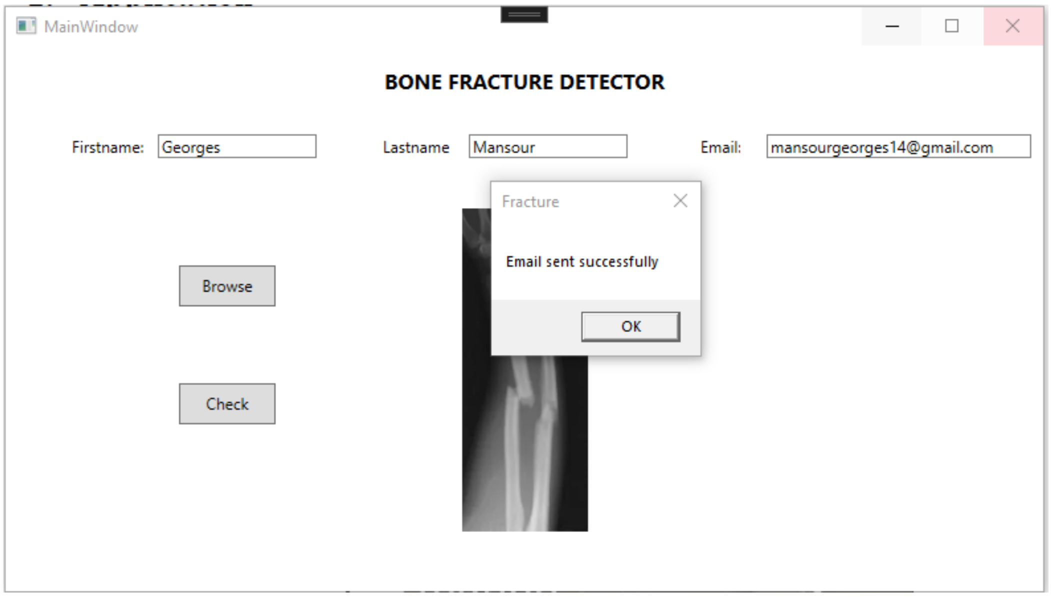
Our application is a desktop application with a single and simple use as shown in **Figure 27**.



Firstname: Georges
Lastname Mansour
Email: mansourgeorges14@gmail.com

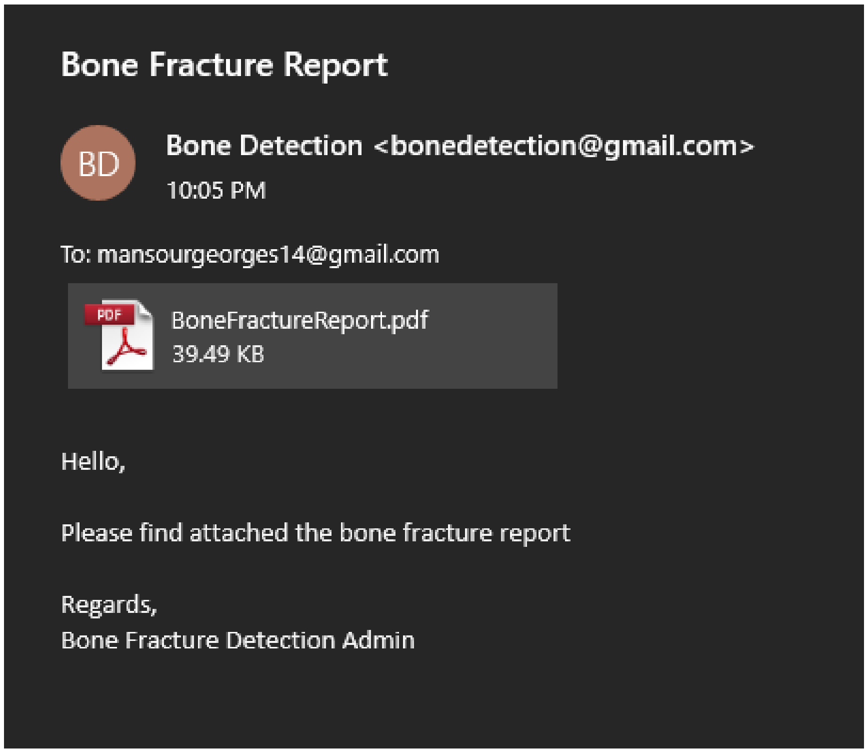
Figure 27: Main Window of the application

After pressing the button check, it will be predicting, then send the report in the following email format after receiving a popup message shown in **Figure 28, 29**.



Email sent successfully

Figure 28: Popup message that the email was sent


Bone Fracture Report

Bone Detection <bonedetection@gmail.com>
10:05 PM

To: mansourgeorges14@gmail.com

BoneFractureReport.pdf
39.49 KB

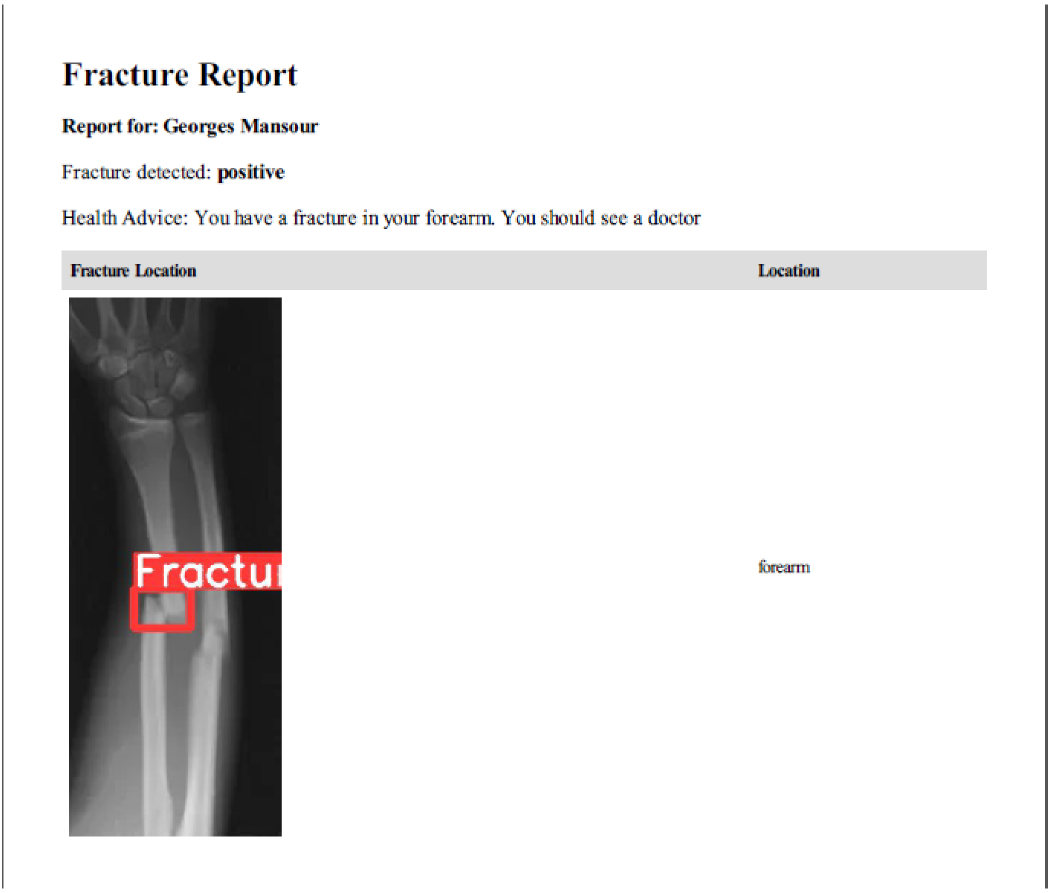
Hello,

Please find attached the bone fracture report

Regards,
Bone Fracture Detection Admin

Figure 29: Email format received

The attached pdf file is the report of the user and has all the results. In **Figure 30**, we will check the format of the report sent to the user. The location is the bone fractured, the fracture location is the fracture marked if it has any and everything else are the informations.



Fracture Report

Report for: Georges Mansour

Fracture detected: **positive**

Health Advice: You have a fracture in your forearm. You should see a doctor

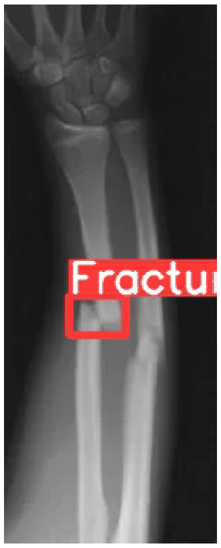
Fracture Location	Location
	forearm

Figure 30: Report received by the user

As for the validation, all the fields are required, if one is missing it will be colored red, **Figure 31**. If the image is missing a popup will show, **Figure 32**.

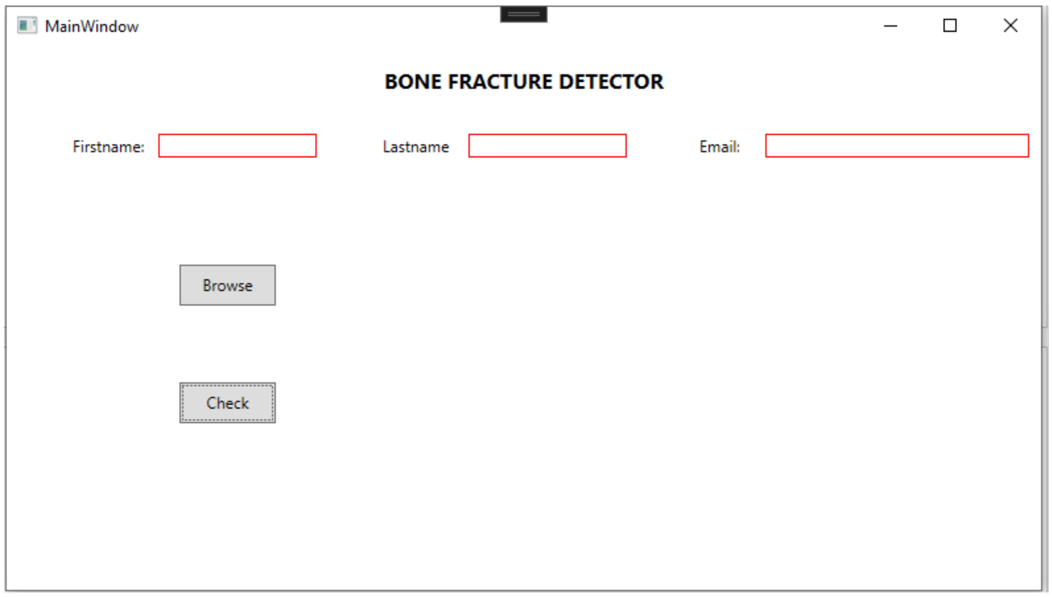
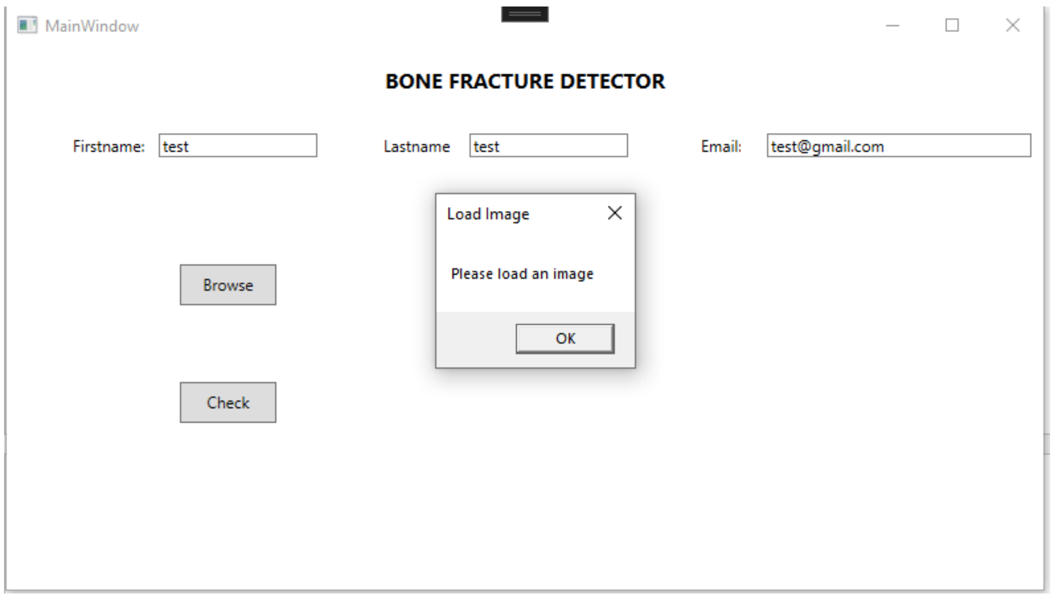


Figure 31: Required fields visibility



Firstname: test
Lastname test
Email: test@gmail.com

Please load an image

Figure 32: Image missing error popup

3. System tests/simulation/experiments

3.1. System setup

As for the setup, we were able to combine the python code with the .Net framework using an object called "Process". This object allows to run other applications from the code. In our case, the python code provided were all the classification and detection is made is being treated as an application that can be accessed through the CMD (command line). That is why we created a batch file with the path of python.exe and the python script and used the process object to run externally from our .Net code (As explained in chapter 3).

3.2. Parameters/metrics/variables

Although each model had his own difficulties and issue, the most important factor of them all was the dataset. The dataset had a huge impact on anything. So the dataset was cleaned, chosen carefully and finally was augmented so it can meet all the factors that can occur, from rotation, brightness difference, zooming and many more. So the augmented dataset was changed from 250 image in total to more than 2000 image in total where each bone had a minimum of 450+ images.

4. Results, discussion and interpretation

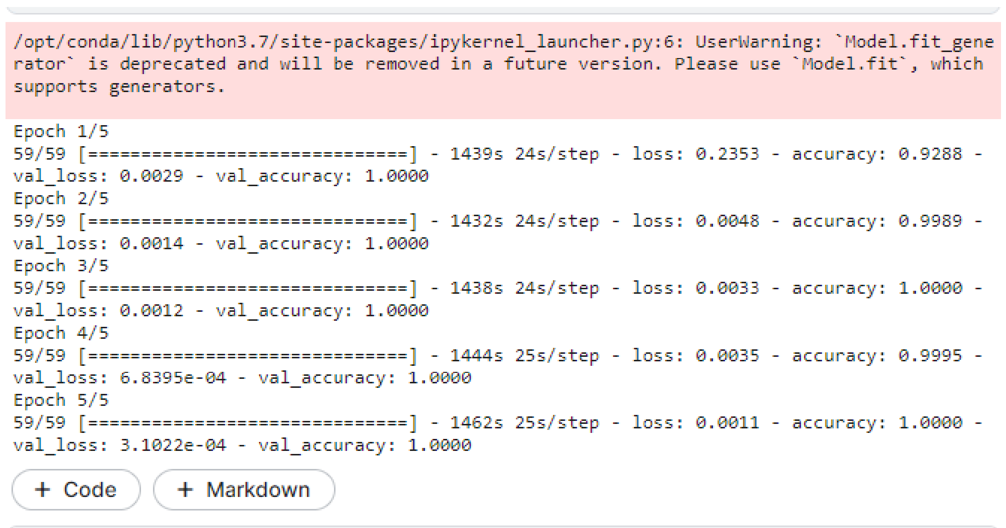
During the process at first, before data augmentation, each of the models hit an accuracy below 50% which means that something was wrong with the data. After the augmentation, for a start, for the binary classification four models were tested: ResNet50, InceptionV3, DenseNet169 and VGG16. Each of these models hit an accuracy of validation and testing below 50% which is totally unacceptable except the DenseNet169 model which hit accuracy between 64 and 71% when testing and the VGG16 which is our preferred solution. The InceptionV3 and ResNet50 was overfitting and not learning correctly where it hit an accuracy of 100% from the first five epochs of the training. **Table 4** shows the comparison between the models.

Table 4: Comparison between the models used for binary classification

	Accuracy
ResNet50	100% (over fit)
InceptionV3	100% (over fit)
DenseNet169	64-71%
VGG16	82%

As for the multiclass, three models were tested, Resnet50, DenseNet169 and VGG16. First, when the data was augmented, all the models were overfitting because the data was being duplicated. The problem faced was the same image was being processed and copied the same for 5 times, so the models were training on the same image over and over again. So the model hit an accuracy of 100% on training and

12% on testing and we got the following figures. **Figure 33**, shows the overfitting while training, and **Figure 34** shows its graph.



```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:6: UserWarning: `Model.fit_gene
rator` is deprecated and will be removed in a future version. Please use `Model.fit`, which
supports generators.

Epoch 1/5
59/59 [==============================] - 1439s 24s/step - loss: 0.2353 - accuracy: 0.9288 -
val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 2/5
59/59 [==============================] - 1432s 24s/step - loss: 0.0048 - accuracy: 0.9989 -
val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 3/5
59/59 [==============================] - 1438s 24s/step - loss: 0.0033 - accuracy: 1.0000 -
val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 4/5
59/59 [==============================] - 1444s 25s/step - loss: 0.0035 - accuracy: 0.9995 -
val_loss: 6.8395e-04 - val_accuracy: 1.0000
Epoch 5/5
59/59 [==============================] - 1462s 25s/step - loss: 0.0011 - accuracy: 1.0000 -
val_loss: 3.1022e-04 - val_accuracy: 1.0000
```

Figure 33: Overfitting while training of the VGG16 model while data duplication

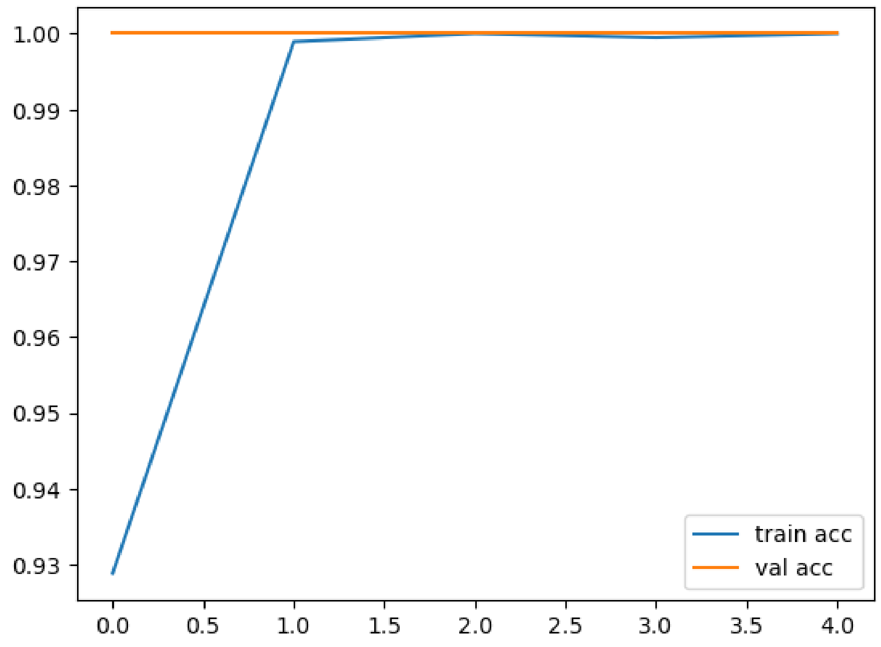


Figure 34: Overfitting when data is duplicated of the VGG16 model

After correction, VGG16 had the best accuracy as shown in the **Table 5**:

Table 5: Comparison between the models used for multi class classification

	Accuracy
ResNet50	66%
DenseNet169	76-78%
VGG16	82-89%

As for the object detection, first we tried to get the features of the fracture from the binary classification model were it was a dead end. As for the second methods, the VGG16 model can be treated also as an object detection, with some research and testing, the VGG16 wasn't able to mark the fracture were we were obliged to draw it by "OpenCV" library. That occurred to a miss marking and drawing over the fracture. That is why the YOLOv8 was chosen after hitting the highest accuracy between all the YOLOs family (from YOLOv5 to YOLOv8)

The results obtained can be increased if more data existed on these bones. Also, many things might change if our study was based on more than 5 bones. As for the processing time, the VGG model takes around 10 seconds to train and send the email which is good while it's passing through 3 models.

5. Conclusion

To conclude, in this chapter, we showed the application that is user friendly and its error handling through figures. Next, we presented many solutions and tested done during the development phase and we discussed the results obtained and the problems faced.

CONCLUSION

The objective of this project was to build a desktop application that might be able to simplify the life of the user by classifying and detecting bone fracture and set it in a report after uploading the chosen image. The developed solution was translated in a desktop application that the user adds its first name, last name, email and the X-RAY image, which needs to be checked, with all the fields are required and mandatory. In the process, the creation of two models of AI (Artificial Intelligence) was created and trained for classification. The first model was a binary classification to check if the bone has a fracture, and the second model was for multi class classification that was used to check what type of bone the user uploaded, and one computer vision model that was developed to mark and detect the fracture in the image given. The provided solution has a better effect than the existing solutions because it's user friendly, works on multiple bones (five bones to be exact), and finally gives the user a readable result by report with a health advice so he can know how to continue on the better track.

The domain of application of the solution is a huge field that can be extended to more features. For example, in a simple way, instead of working of five bones only, the models can be trained on the whole body. This can simplify the life of the users, doctors and hospitals especially, because it can detect on spot the fracture. For a better use, the application can be have a version as a mobile application that receives instead of image pdfs, mostly because in our days all the results of the X-RAY are received of a pdf form. That can lead that this solution becomes a worldwide, and can work anywhere not only in Lebanon. As for the future, we are studying the possibility to increase the accuracy of model for the beginning, and then start training models for other bones combining with the five existing.

REFERENCES

[1] SAS, "Artificial Intelligence. What it is and why it matters," [Online]. Available: https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html.

[2] S. Myint, A. S. Khaing and H. M. Tun, "Detecting Leg Bone Fracture In X-Ray Images," vol. 5, no. 06, p. 5, June 2016.

[3] The Editors of Encyclopaedia Britannica, "Tibia," 4 February 2020. [Online]. Available: https://www.britannica.com/science/tibia.

[4] R. Lindsey, A. Daluiski, S. Chopraa, A. Lachapelle, M. Mozer, S. Sicular, D. Hanel, M. Gardner, A. Gupta, R. Hotchkiss and H. Potter, "Deep neural network improves fracture detection," p. 6, 22 October 2018.

[5] World Bank Organization, "The World Bank," The World Bank, 02 November 2022. [Online]. Available: https://www.worldbank.org/en/country/lebanon/overview.

[6] Human Rights Watch, "Lebanon: Hospital Crisis Endangering Health," Human Rights Watch, 10 December 2019. [Online]. Available: https://www.hrw.org/news/2019/12/10/lebanon-hospital-crisis-endangering-health.

[7] SnapStack Digital Solutions, "What are the Benefits of Using WPF?," [Online]. Available: https://snapstack.cz/what-are-the-benefits-of-using-wpf/.

[8] Ç. Uslu, "What is Kaggle?," datacamp, March 2022. [Online]. Available: https://www.datacamp.com/blog/what-is-kaggle.

[9] G. Lawton, "Data Preprocessing," TechTarget, January 2022. [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/data-preprocessing.

[10] MathWorks, "What Is Object Detection? 3 things you need to know," MathWorks, [Online]. Available: https://www.mathworks.com/discovery/object-detection.html.

LIST OF ACRONYMS

AI: Artificial Intelligence

CNN: Convolution Neural Network

ANN: Artificial Neural Network

FPN: Feature Pyramid Network

DCNN: Deep Convolution Neural Network

UI: User Interface

RAM: Random Access Memory

GPU: Graphics Processing Units

TPU: Tensor Processing Units

YOLO: You Only Look Once

CMD: Command Line

LIST OF FIGURES

LIST OF TABLES