REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOHAMED BOUDIAF - M'SILA

Faculté des Mathématiques et de l'Informatique

Département d'Informatique

N° :……..

DOMAINE : Mathématiques et Informatique

FILIERE : Informatique

OPTION : INTELLIGENCE ARTIFICIELLE

Mémoire présenté pour l'obtention
Du diplôme de Master Professionnalisant

Par:

Bourahla Mohamed Zakaria

Djeriou Khaled

Intitulé

Development of an intelligent system for the prediction of dangerous patient situations: A diploma – A startup

Soutenu devant le jury composé de :

Dr. Kadri Said	Université de M'sila	Président
Pr. Bourahla Mustapha	Université de M'sila	Rapporteur
Dr. Meliouh Amel	Université de M'sila	Examinateur

Année universitaire : 2021 / 2022

خلاصة

في مذكرة الماستر هذه، قدمنا مشروعًا في إطار مراقبة المرضى المقيمين في المستشفى. يمكن أن تحتوي كل غرفة على العديد من المرضى حيث يرتبط المريض بسرير به جميع المرافق الطبية. الغرف مشغولة بكاميرات المراقبة وسيتم نقل تسلسل الفيديو إلى نظامنا الذكي المسمى .Aidy يحاول نظام Aidy أولاً اكتشاف أزواج الكائنات من المرضى والأسرة من أجل التقاط صورة تربط المريض بالسرير باستخدام نموذج تصنيف لاكتشاف الأشياء. سيتم إرسال هذه الصور التي تم اقتصاصها من الصورة الأصلية إلى نموذج تصنيف ثانٍ للتنبؤ بحالة المرضى. إذا كان التنبؤ يشير إلى حالة خطيرة لمريض ما، فسيتم إرسال إشعار إلى تطبيق الهاتف المحمول المثبت على الجهاز المحمول لطبيب أو ممرضة، وهو متصل بنظامنا الذكي.Aidy

الكلمات الرئيسية: الذكاء الاصطناعي ، التعلم العميق ، اكتشاف الأشياء ، رؤية الكمبيوتر ، الشبكة العصبية التلافيفية ، الرعاية الصحية ، مراقبة المريض ، تطبيقات الهاتف المحمول ، واجهة Tk ، خطط SHapley الإضافية.

Abstract

In this Master's thesis, we presented a project within the framework of monitoring patients staying in a hospital. Each room can contain several patients where a patient is associated with a bed with all the medical facilities. The rooms are occupied by surveillance cameras and the video sequences will be transmitted to our intelligent system called Aidy. The Aidy system first tries to detect object pairs of patients and beds in order to take an image that relates patient to bed using a classification model for object detection. These cropped images from the original image will be sent to a second classification model to predict the situation of patients. If the prediction indicates a dangerous case of a patient, a notification will be transmitted to a mobile application installed on the mobile device of a doctor or a nurse, which is connected to our intelligent system Aidy.

Keywords: Artificial Intelligence, Deep Learning, Object Detection, Computer Vision, Convolutional Neural Network, Healthcare, Patient Monitoring, Mobile Application, Tk Interface, SHapley Additive Explanations.

Résumé

Dans ce mémoire de Master, nous avons présenté un projet dans le cadre du suivi des patients séjournant dans un hôpital. Chaque chambre peut contenir plusieurs patients où un patient est associé à un lit avec toutes les installations médicales. Les salles sont occupées par des caméras de surveillance et les séquences vidéo seront transmises à notre système intelligent appelé Aidy. Le système Aidy essaie d'abord de détecter des paires d'objets de patients et de lits afin de prendre une image qui relie le patient au lit en utilisant un modèle de classification pour la détection d'objets. Ces images recadrées à partir de l'image originale seront envoyées à un deuxième modèle de classification pour prédire la situation des patients. Si la prédiction indique un cas dangereux d'un patient, une notification sera transmise à une application mobile installée sur l'appareil mobile d'un médecin ou d'une infirmière, qui est connecté à notre système intelligent Aidy.

Mots clés : Intelligence Artificielle, Apprentissage Profond, Détection d'Objets, Vision par Ordinateur, Réseau de Neurones Convolutif, Soins de Santé, Surveillance du Patient, Application Mobile, Interface Tk, Explications Additives SHapley.

شكـــــــر

الحمد لله الذي بنعمته استطعنا اكمال هذا المشروع

نشكر المشرف، البروفيسور بورحلة مصطفى، على ثقته بنا ومتابعتنا بشكل جيد في تحقيق هذا المشروع.

كما نشكر أعضاء لجنة التحكيم، الدكتور قادري سعيد و الدكتورة مليوح أمال، على قبولهم تقييم هذا العمل.

نشكر جميع أساتذتنا الكرام من كلية الرياضيات و الأعلام الآلي الذين ساهموا في تعليمنا و أوصلونا بمشيئة الله الى ما كنا نتطلع اليه من التحصيل العلمي.

بورحلة محمد زكرياء
جريـــــو خـــالـــد

Content

List of Figures 3

List of tables 3

GENERAL INTRODUCTION 4

Chapter 1 : Overview on the Aidy system 7

1.1 Introduction 7

1.2 Project challenges 7

1.2.1 Methodology and architecture framework 7

1.2.2 Real-time Object detection model algorithm 8

1.2.3 CNN model for Image Classification 9

1.3 Dataset description 10

1.4 Adding more Machine Learning techniques in the future 10

1.5 Description of used hardware 11

1.6 Notification to Mobile Application 11

1.7 Conclusion 12

Chapter 2 : Implementation of the project Aidy 13

2.1 Introduction 13

2.2 Dataset preparation 13

2.3 Dataset preprocessing 14

2.4 Creation of Convolutional Neural Network (CNN) model 16

2.4.1 Import of necessary libraries 16

2.4.2 Loading of dataset 16

2.4.3 Scaling the pixels 17

2.4.4 Definition of the CNN model 17

2.5 Saving of the CNN model 20

2.6 Evaluation of the final model 21

2.7 Classification test with image files and video sequences 21

2.8 Conclusion 25

Chapter 3 : Classification extension to many patients in one room 26

3.1 Introduction 26

3.2 Model for detection of objects 26

3.3 Creating the CNN model for object detection 28

3.4 Package of useful functions 30

3.5 Classification of cropped images 32
3.6 Mobile Application for notifications 35
3.7 Conclusion 36
GENERAL CONCLUSION 37
Bibliography 38
Appendix 40
Business plan 40

List of Figures

Figure 1.1 : The frame that camera captured....8
Figure 1.2 : The images that get treated by the Classification CNN Model....8
Figure 1.3 : Basic idea of YOLO algorithm (Redmon & Farhadi, YOLOv3: An Incremental Improvement, 2018)....8
Figure 1.4 : Bounding boxes....9
Figure 1.5 : Samples from the dataset....10
Figure 1.6 : Hardware used for model training....11
Figure 1.7 : Screenshots from mobile device....12
Figure 2.1 : samples from the original dataset....13
Figure 2.2 : transformed image....14
Figure 2.3 : Architecture of the CNN model (Jogin & Mohana, 2020)....17
Figure 2.4 : Flattening (Jogin & Mohana, 2020)....18
Figure 2.5 : Test results of the CNN model fitting....20
Figure 2.6 : The graphical user interface to upload and classify an image....23
Figure 2.7 : explanation of model prediction....24
Figure 2.8 : Classification of images from video sequence....25
Figure 3.1 ; Classification of cropped images from video sequence....35
Figure 3.2 : notification to mobile application....36

List of tables

Table 1.1 : Convolutional neural network structure....9

GENERAL INTRODUCTION

In general, artificial intelligence consists of computer programs that help in making intelligent and timely decisions. It has become widely used in many fields due to its economic benefits. Many countries have encouraged the use of artificial intelligence, which will push to benefit from this scientific development in almost all areas.

Artificial intelligence is currently used in several areas to help companies run their businesses. We can create smart systems to help in the areas of health, diagnostics, wastewater and rainwater management, transportation management, help people in their daily lives, prevent accidents, floods, fires, etc. The ultimate goal is to obtain economic benefits for these companies by making the right decisions at the right time while managing with artificial intelligence.

The use of artificial intelligence is now an economic necessity required by globalization (Dauverge, 2021). Many countries are encouraging the use of artificial intelligence because they realize that it will improve human lives with significant economic benefits. Since the development of intelligent systems relies on machine learning from real data, it would not make sense to use data collected from environments different from those of the system in which we want to deploy it.

Artificial intelligence is a programming technique used in developing intelligent computer systems (software) to simulate human behavior (Russell & Norvig, 2020). This technology has been around for several decades but has now developed due to the advanced electronic technology used in manufacturing computers that has become very powerful both in execution speed and in storage capacity.

We can classify intelligent systems into two broad categories: Knowledge-based intelligent systems developed from expert knowledge (Bourahla & Bourahla, 2022). Knowledge of a domain is represented using mathematical logic in the form of rules that will be used by the (automatic) inference system (Bourahla M. , Description and reasoning for vague ontologies using logic programming, 2018). These intelligent systems are called expert systems and they are accurate (giving accurate results) but require human expertise to be developed.

The second category of intelligent systems is developed based on mathematical models. They are mathematical structures (a set of equations) whose parameters (which we call weights) will be determined by machine learning (auto-training) algorithms using data collected from real cases (Bourahla M. , LTL transformation modulo positive transitions, 2017). These intelligent systems are not accurate and the degree of their accuracy depends on the design of the model used, the training algorithm and the data collected.

We can design efficient machine learning models and algorithms for a particular domain but will be less efficient for others, and the same for the collected data. We can get good quality data and poor quality data, even the amount of data plays a role in the automatic training process. To arrive at effective mathematical models, we can apply simulation methods in order to perform experiments that guide us to choose the correct mathematical model, i.e. the effective structure.

Intelligent systems based on models developed by training can be classified into two main categories: intelligent systems whose outputs are encoded in binary systems and intelligent systems whose outputs are real (continuous) numbers. The type of data used in training guides us to choose the appropriate model. For the best known models using binary coding, we find models inspired by neural networks, on the other hand, the best known models for intelligent systems whose outputs are continuous numbers, we find mathematical models based on linear regression where a machine learning algorithm tries to find the correlation between variables of these models.

If the training data were images rather than numbers, then these machine learning algorithms would not be effective. The AI technology in this case uses deep learning algorithms (Jogin & Mohana, 2020). Here the machine learning model is organized into several layers and each layer performs a certain (well defined) processing using a method that combines two mathematical functions of convolution and correlation by defining filters. This method is quite complex and requires faster processors and more memory space to store information.

Algorithms used in machine learning are either supervised learning algorithms, in which case the training data contains the desired output (category) value for each reference, unsupervised (in this case the desired output is unknown) or mixed (hybrid: only some data contains required output values). Each class of machine learning algorithms runs on specific models, that is, the machine learning algorithm is designed according to data (supervised, unsupervised or mixed) and runs on a specific mathematical model.

Generally, supervised machine learning algorithms use datasets consisting of two parts, the input and its desired output, and try to find outputs that are very close to the desired output. If the difference between the outputs calculated by a function called loss is too large, the algorithm repeats the calculation after updating the model weights until it reaches the minimum loss. Unsupervised machine learning algorithms use subgroup (clustering) algorithms according to the values of certain attributes. Hybrid learning algorithms try to find the missing output that is close to the current output that is available.

Once the intelligent system is developed, i.e. the mathematical model is quite specific; we can use it by sending data to it in real time to get smart predictions in order to make decisions. Google has developed functional libraries (and tools) to help developers of smart systems. These functions were written for use with development environments that integrate many programming languages. Thus, we can find functions for processing data and digital images, learning algorithms, functions for evaluating learning models, etc.

In this context we have developed an intelligent system to monitor in real time patients staying in hospital rooms, we called it Aidy (**Aid**s is on its wa**y**). This smart system will help doctors and nurses monitor patients whether they are in danger or not. For this we have developed a CNN (Convolutional Neural Network) model which is trained by a deep learning algorithm on a set of real data collected from images taken within the medical emergency department of the University of M'Sila.

This intelligent system is supported by a graphical user interface to predict the class of the loaded image and give an explanation of this class prediction (Jin, Sergeeva, Weng, Chauhan, & Szolovits, 2021). A mobile application has also been developed to notify nurses and doctors of patient cases via mobile devices.

This Master's thesis is organized as follows. We present an overview of this project in the first chapter. The second chapter presents the implementation of this intelligent system. An extension that uses object detection is developed to handle the case where multiple patients occupy the same room. At the end, we give conclusions and perspectives.

As this work is part of a diploma-a startup project, it is registered at the incubator of M'Sila, and we have obtained a label from the ministry of startups, which will help us to create a startup. The appendix contains its business plan.

Chapter 1 : Overview on the Aidy system

1.1 Introduction

Health systems are faced with growing demand for their services, costs and a workforce struggling to meet the needs of their patients. Especially in light of current events and conditions, pressure, tension, lack of manpower and time spent inside hospitals (Hossain, Muhammad, & Guizan, 2020). In this work, we present a project that uses artificial intelligence in health systems, and how this project can help doctors and nurses in their work and time in the hospital.

Monitoring of patients and their needs in hospitals mainly rely on doctors and nurses to give the most efficient and helpful to patients (Asan, 2020). However, with the deterioration of the weather, the increase in the number of patients and the concern of the nurses for other work, it became difficult to follow all the patients in terms of follow-up, which led to not discovering the patient's condition as soon as possible.

Here we introduce Aidy, the project which relies on the application of artificial intelligence to find out patient cases and send notifications and alerts to the mobile application deployed on doctor/nurse's mobile devices connected to it.

1.2 Project challenges

For visual systems, to realize this project, we face several difficult challenges, such as using the camera to detect an indeterminate number of patients in a room, training models developed using the technique of artificial intelligence, on a data set that was collected in a specific room, then applying it in another room, or by applying the project in peripheral devices.

1.2.1 Methodology and architecture framework

This artificial intelligence application is based on two models, an object detection model to detect the patients and their positions in the real-time image and a CNN classification model to classify the patient's cases. We used the two models working together for several purposes:

1- In order to classify an indeterminate number of patients in a room (Figure 1.1), we used the object detection model to detect each patient in the room by taking the patient's location and calculating its intersection to union ratio [IOU] with each bed in the room to find out which bed he is associated with, then split the picture frame into several small pictures (Figure 1.2), each picture contains only one patient and one bed, then send the image to be processed by the CNN model to classify the final case of a specific patient in the room.

2- Secondly, in order to give the program the highest possible classification accuracy, we used the object detection model to help the CNN model in its work in terms of splitting the image and making it suitable for processing by the CNN model and this by making the image similar to the data on which the CNN model was trained.

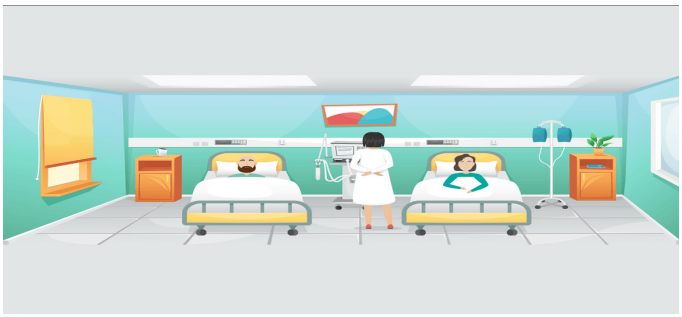

Figure 1.1 : The frame that camera captured

Figure 1.2 : The images that get treated by the Classification CNN Model

1.2.2 Real-time Object detection model algorithm

While there are many different object detection algorithms, in our project we used the Yolo (**Y**ou **o**nly **l**ook **o**nce) algorithm created by Joseph Redmon and Ali Farhadi (Redmon & Farhadi, YOLOv3: An Incremental Improvement, 2018). YOLO is a real-time object recognition system that can recognize multiple objects in a single frame. It applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region.

The basic idea of YOLO is exhibited in the figure below (Figure 1.3). YOLO divides the input image into an S × S grid (S stands for specific size) and each grid cell is responsible for predicting the object centered in that grid cell.

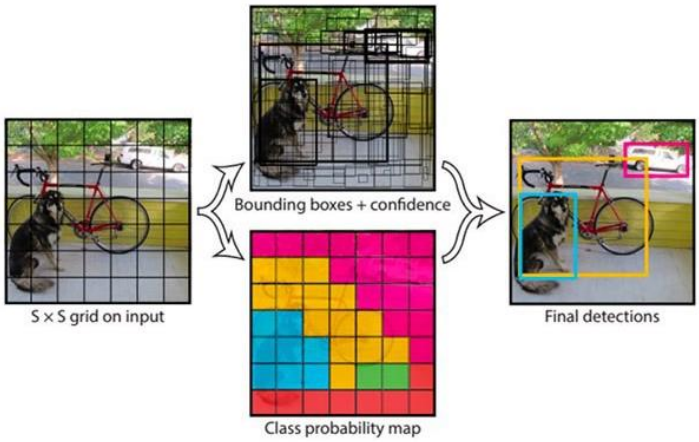

Figure 1.3 : Basic idea of YOLO algorithm **(Redmon & Farhadi, YOLOv3: An Incremental Improvement, 2018)**

Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts (Figure 1.4).

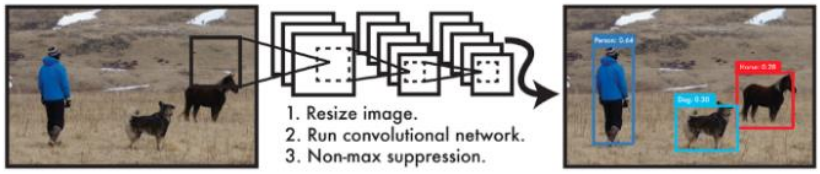

Figure 1.4 : Bounding boxes

The reason we chose this particular algorithm is that it is designed for fast and accurate real-time object detection, making it the perfect choice for embedded applications. See the following paper for more details on the full YOLO system (Redmon, Divvala, Girshick, & Farhadi, 2016).

For the purpose of our project, we implemented and coded this algorithm in Tensorflow Framework (see Chapter 3) and then we used the pre-trained Yolov3 416x416 weights that were trained on 80 classes of the coco object detection dataset. And because we only care about the person (patient) and bed classes, we took their indexes from the 80's classes and cleaned up the other objects from the frame.

1.2.3 CNN model for Image Classification

In order to classify patient's positions, we used a sequential Convolution Neural Network Model (Gua, Wangb, & Kuenb, 2017), which consists of an input layer of shape 32x32x3, convolution layers (Convolution2D), activation layers (Relu activation function), pool layers (MaxPooling2D), fully connected layers, and final output layer (four output nodes for 4 classes). The structure of the model is shown in the following table (Table 1.1)

Table 1.1 : Convolutional neural network structure

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchN ormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
batch_normalization_1 (Batc hNormalization)	(None, 32, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
dropout (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_2 (Batc hNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
batch_normalization_3 (Batc hNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
batch_normalization_4 (Batc hNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_5 (Batc hNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0

dense (Dense)	(None, 128)	262272
batch_normalization_6 (Batc hNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Totals		Total params: 552,100 Trainable params: 550,948 Non-trainable params: 1,152

1.3 Dataset description

Large computer vision applications require quality visual data and lots of data (Dash, Shakyawar, Sharma, & Kaushik, 2019), (Wu, et al., 2020). But in this project and this particular model, there is no data for it. We therefore choose to collect our own dataset.

We collected the image dataset using a camera to capture multiple patients in hospital rooms, each room contains many beds, and we used four different positions (bed is empty, patient is sleeping, patient is sitting on bed, and patient fell to the ground), which allows the model to detect the four classes.

Next, we used data augmentation techniques (Perez & Wang, 2017) to increase data size and diversity to avoid overfitting and to make the data more efficient. The following figure (Figure 1.5) shows samples of the data set.

Figure 1.5 : Samples from the dataset

By using combination of both models: the object detection model and the CNN model we were able to classify the classes: The patient is not on his bed, the patient is out of the room, the patient is sleeping on his bed, the patient is sitting on his bed or the patient fell to the floor.

1.4 Adding more Machine Learning techniques in the future

To implement this project in the real world to get more scalability, we can add more features, which consist of:

1- Adding object tracking technique to identify the same patients from the previous frame, this can help to know the people in the room as well as knowing the new people who enter the room.
2- Recreating the dataset to make it more effective in a real-world project.

3- Train the object detection model on a new dataset that is dedicated to this project only.

1.5 Description of used hardware

The concept of Edge AI consists of performing computations on an embedded real-time system. Since the training process requires a lot more computational power as compared to the inference process, it is not performed on the embedded system, but a dedicated workstation. Then, the model with the obtained weights is deployed on the target hardware in order to be executed.

For the embedded implementation of the model, different hardware platforms have been considered:

1- NVIDIA Jetson Nano Developer Kit
2- Raspberry Pi camera 2
3- Intel Neural Compute Stick accelerator

Figure 1.6 : Hardware used for model training

1.6 Notification to Mobile Application

The mobile application will display the patient's status in real time on UI dashboard: In bed, try to get up, fell, away from the bed, out of the room. The notification with the mobile application will be multiple alerts, which are easy to read, and the alert color changes according to the priority of attention.

The mobile application can attach a picture to an alert of a status, which may be useful in some cases. But we think of blurring the faces of local residents when we send them to the phones of the duty nurses. Of course there are: No live streaming, No audio and video recording, and blurred image on the patient's face (not done yet).

Figure 1.7 shows screenshots were the dashboard shows the real-time status (event alerts) of a patient with its image. The event alert displays multiple alerts, which are easy to read, and the alert color changes according to attention priority: blue, yellow, red.

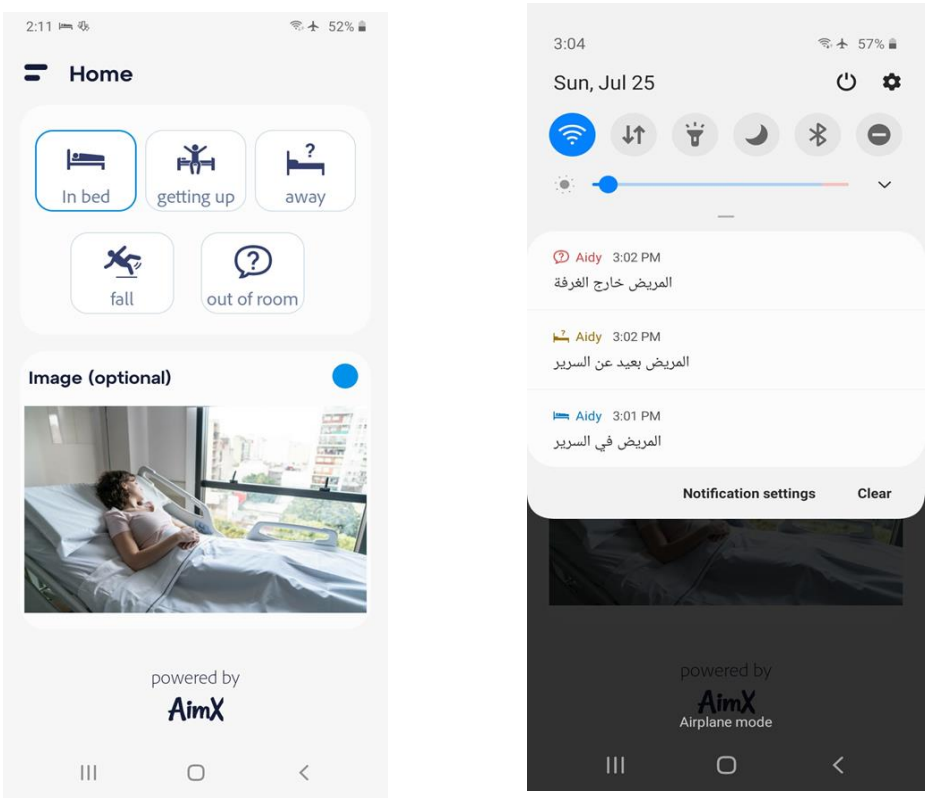

Figure 1.7 : Screenshots from mobile device

1.7 Conclusion

In this chapter, we presented an overview about this project which is dedicated to real-time control of different patient's situations staying in hospital rooms. The implementation of this project will be discussed in detail in the next two chapters.

Chapter 2 : Implementation of the project Aidy

2.1 Introduction

In this chapter, we explain in detail the implementation of this project. We have used Python (Guido & Drake Jr, 1995), an interpreter, high-level, general-purpose programming language, and Python libraries for implementing this project. The PyCharm IDE (Interface Development Environment) (Pesce, 2022) is used to help us debugging our code and facilitate the integration of the Python libraries.

2.2 Dataset preparation

The University of M'Sila gave us a room containing a hospital like bed to take pictures on patients in different situations like they fell on the ground, sleeping on the bed, sitting on the bed, out of the room, etc. Here are some pictures taken.

Figure 2.1 : samples from the original dataset

We augmented this original dataset with a set of images automatically generated by a transformation method described by the following Python program, where a set of transformations is applied on every image as rotations, horizontal flipping, color shifting and scaling.

```
# This module is for augmenting the data set by transformation of images using
# rotation, horizontal flip, color shift and shift of scales
import os
import cv2
from os import makedirs, listdir
from os.path import isfile, join
from shutil import copyfile
import albumentations as A
import numpy as np

# To define the transformation
transform = A.Compose(
    [
        A.Rotate(limit=45, p=0.9),
        A.HorizontalFlip(p=0.5),
        A.RGBShift(r_shift_limit=30, g_shift_limit=30, b_shift_limit=30, p=0.9),
        A.ShiftScaleRotate(scale_limit=0.2, shift_limit=0, rotate_limit=0, p=1)
    ]
)

# The augmentation of the original data set and the results are saved in the
# augmented dataset directory
def data_augmentation(original_dataset, augmentated_dataset):
    makedirs(augmentated_dataset + '/', exist_ok=True)
```

```
l = [f for f in listdir(original_dataset) if isfile(join(original_dataset, f))]
for img in l:
    count = 1
    copyfile(join(original_dataset, img), join(augmentated_dataset, img))
    img_array = cv2.imread(os.path.join(original_dataset, img),
                           cv2.IMREAD_COLOR)
    img_array = cv2.resize(img_array, (180, 180))
    img_array = np.array(img_array)
    for i in range(20):
        augmentations = transform(image=img_array)
        augmented_img = augmentations["image"]
        classe = img.split('.')[0]
        number = str(img.split('.')[1])
        cv2.imwrite(os.path.join(augmentated_dataset, '%s.%s_%d.jpg' % (classe,
                    number, count)), augmented_img)
        count += 1
```

By this code, each image is augmented by 20 other images, which means that we have a set of 20 times the number of original images.

Figure 2.2 : transformed image

2.3 Dataset preprocessing

The following Python code is for organizing the augmented dataset according to an appropriate structure. All images files are stored in the 'augmented_dataset' directory (generated from ‘original_dataset’), where an image is saved in a jpg file; its name is composed of the class name followed by a period, then a file number and the extension jpg <className.fileNumber.jpg>. The following Python code is for importing the useful packages.

```
# Import useful packages
from os import makedirs
from os.path import isfile, join
from shutil import copyfile
from random import seed
from random import random
from os import listdir
import numpy as np
from PIL import Image
```

The structure to create is as follows: under the dataset directory we create two subdirectories 'train' and 'test', which will themselves contain sub-directories where their names will bear the names of the classes, i.e. the patient’s situations names. A class subdirectory contains all the image files belonging to this class.

```
def create_structure(dataset_home):
	# starting random number generator
	seed(1)
	# define the frame rate to use for validation (test)
	val_ratio = 0.25 # 25 %
	# copy images from training dataset to subdirectories
	l = [f for f in listdir(dataset_home) if isfile(join(dataset_home, f))]
	for file in l:
		# source file
		src = dataset_home + '/' + file
		# The training subdirectory
		dst_dir = '/train/'
		if random() < val_ratio:
			# the test (validation) subdirectory
			dst_dir = '/test/'
		# the training or test class subdirectory
		classe = file.split('.')[0]
		makedirs(dataset_home + dst_dir + classe + '/', exist_ok=True)
		dst = dataset_home + dst_dir + classe + '/' + file
		copyfile(src, dst)
```

This function reformats the training and test data in the format that will be used by machine learning to build (train) the model. The image size is 32 pixels width and 32 pixels high and three color channels.

```
def load_data(dataset_home):
	Image_Width = 32
	Image_Height = 32
	Image_Size=(Image_Width,Image_Height)
	Image_Channels = 3
	train_class_directories = []
	for classe in listdir(dataset_home + '/train/'):
		train_class_directories = train_class_directories + [classe]
	num_train_samples = 0
	for classe in train_class_directories:
		num_train_samples = num_train_samples + len(listdir(dataset_home +
				'/train/' + classe))
	x_train = np.empty((num_train_samples, Image_Width, Image_Height,
			Image_Channels), dtype='uint8')
	y_train = np.empty((num_train_samples,), dtype='uint8')
	i = 0
	j = -1
	for classe in train_class_directories:
		j = j + 1
		for file in listdir(dataset_home + '/train/' + classe):
			image = Image.open(dataset_home + '/train/' + classe + '/' +
					file)
			image = image.resize(Image_Size)
			x_train[i] = np.asarray(image)
			y_train[i] = j
			i = i + 1
	test_class_directories = []
	for classe in listdir(dataset_home + '/test/'):
		test_class_directories = test_class_directories + [classe]
	num_test_samples = 0
	for classe in test_class_directories:
		num_test_samples = num_test_samples + len(listdir(dataset_home +
				'/test/' + classe))
	x_test = np.empty((num_test_samples, Image_Width, Image_Height,
			Image_Channels), dtype='uint8')
	y_test = np.empty((num_test_samples,), dtype='uint8')
	i = 0
	j = -1
	for classe in test_class_directories:
		j = j + 1
		for file in listdir(dataset_home + '/test/' + classe):
			image = Image.open(dataset_home + '/test/' + classe + '/' +
```

```
                                file)
                image = image.resize(Image_Size)
                x_test[i] = np.asarray(image)
                y_test[i] = j
                i = i + 1
    return (x_train, y_train), (x_test, y_test), len(listdir(dataset_home +
            '/train'))
```

2.4 Creation of Convolutional Neural Network (CNN) model

Its mode of operation is at first sight simple: the user provides as input an image in the form of a matrix of pixels. It has 3 sizes: two dimensions for a gray scale image. A third dimension, of depth 3 to represent the fundamental colors (Red, Green, and Blue). Unlike a classic MLP (Multi Layers Perceptron) model which only contains a classification part, the architecture of the Convolutional Neural Network has a convolutional part upstream and therefore has two very distinct parts:

1) A convolutional part: Its final objective is to extract characteristics specific to each image by compressing them in order to reduce their initial size. In summary, the input image passes through a succession of filters, creating at the same time new images called convolution maps. Finally, the obtained convolution maps are concatenated in a feature vector called CNN code.
2) A classification part: The CNN code obtained at the output of the convolutional part is provided as input in a second part, consisting of fully connected layers called multilayer perceptron (MLP for Multi Layers Perceptron). The role of this part is to combine the characteristics of the CNN code in order to classify the image.

To create a CNN model we used this Python code.

2.4.1 Import of necessary libraries

```
# import of necessary libraries
import sys
from matplotlib import pyplot
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
import dataProcessing
```

2.4.2 Loading of dataset

This python function will be called to load the dataset to train and test the CNN model.

```
# load the data set
def load_dataset():
    (trainX, trainY), (testX, testY), num_classes =
                          dataProcessing.load_data('augmentated_dataset')
    # data categorization
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY, num_classes
```

2.4.3 Scaling the pixels

This function will be called to normalize the image data (pixels) to be in the range [0, 1]

```
# scale pixels
def prep_pixels(train, test):
    # convert integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

2.4.4 Definition of the CNN model

The CNN model is composed of a stack of layers, where an input image (starting point) is handled by convolutional layers to realize convolution operations and pooling layers for pooling operation and input layer for the artificial neural network (flattening).

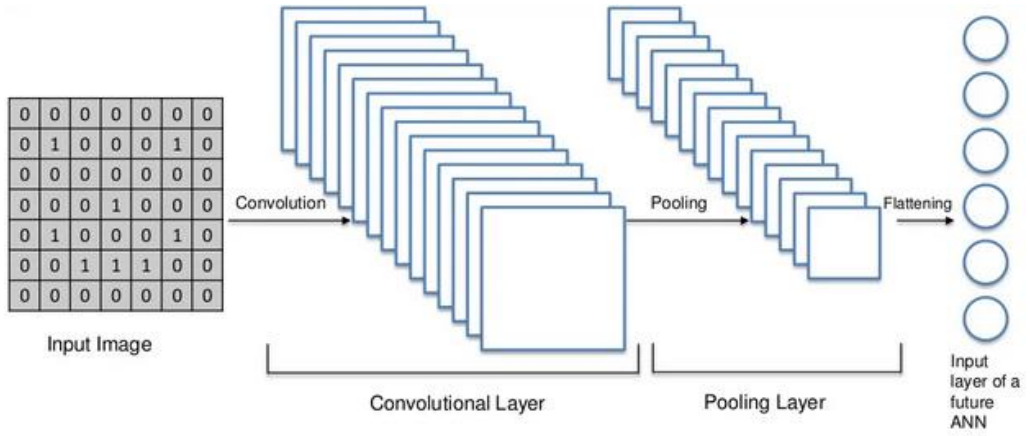

Figure 2.3 : Architecture of the CNN model **(Jogin & Mohana, 2020)**

We create a sequential model by calling the constructor Sequential(), which is appropriate for simple stacking of layers where each layer has exactly one input tensor and one output tensor. The function Conv2D() creates a 2D convolution layer to be added to the sequential model, this layer creates a convolution kernel which is wind with input layers which helps to produce output tensor.

The function BatchNormalization() creates layer to be added to the sequential model, this layer normalizes its inputs. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

The function MaxPooling2D() adds a layer to compute the max pooling operation for 2D spatial data. The layer added by the function Dropout(rate) applies drop out to the input. It randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent over fitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.

After creating the convolution and pooling layers, we're supposed to have a pooled feature map. The Flatten() function adds a flatten layer to the model to flat the input without affecting the batch size. If inputs are shaped (batch,) without a feature axis, then flattening adds an extra channel dimension and output shape is (batch, 1). Thus, we are going to flatten our pooled feature map into a column like in the image below.

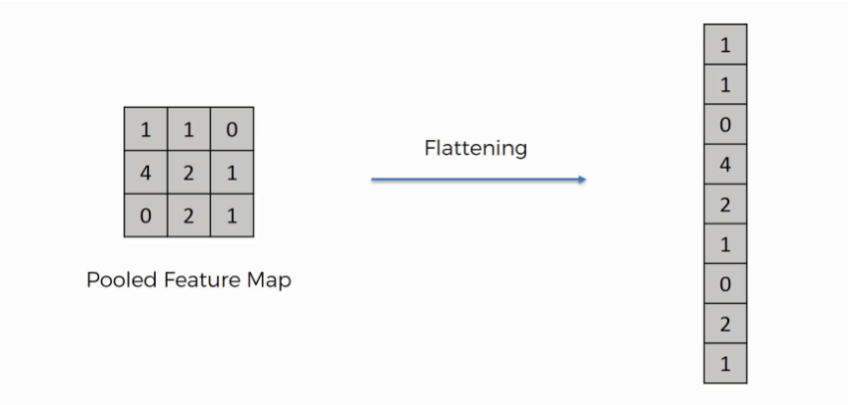

Figure 2.4 : Flattening **(Jogin & Mohana, 2020)**

The reason we do this is that we're going to need to insert this data into an artificial neural network later on.

The Dense() function implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer. At the end, this model is compiled using the SGD Gradient descent (with momentum) optimizer. Below is the Python code.

```
# define the CNN model
def define_model(num_classes):
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu',kernel_initializer='he_uniform',
              padding='same', input_shape=(32, 32, 3)))
    model.add(BatchNormalization())
    model.add(Conv2D(32, (3, 3), activation='relu',
              kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), activation='relu',
              kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(64, (3, 3), activation='relu',
              kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(128, (3, 3), activation='relu',
              kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(128, (3, 3), activation='relu',
              kernel_initializer='he_uniform', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
    model.add(Dropout(0.4))
```

```
    model.add(Flatten())
    model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    # num_classes is the number of classes
    model.add(Dense(num_classes, activation='softmax'))
    # compile the model with the SGD optimizer
    opt = SGD(lr=0.001, momentum=0.9)
    model.compile(optimizer=opt, loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

The following function will be called to show statistics about the CNN model. It will plot the learning diagnostic curves. The first curve will display the loss values and the second display the accuracy. At the end these curves are saved in a png file.

```
# Draw diagnostic learning curves
def summarize_diagnostics(history):
    # trace the loss
    pyplot.subplot(211)
    pyplot.title('Cross Entropy Loss and Classification Accuracy')
    pyplot.ylabel('Loss of cross-entropy')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot precision
    pyplot.subplot(212)
    pyplot.xlabel('Number of iterations')
    pyplot.ylabel('Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='blue', label='train')
    pyplot.plot(history.history['val_accuracy'], color='orange', label='test')
    # save the curve in a file
    filename = sys.argv[0].split('/')[-1]
    pyplot.savefig(filename + '_plot.png')
    pyplot.close()
```

The function shown below is to run the test harness to evaluate the CNN model. First it calls the function load_dataset(), to load the augmented dataset, which returns the training and testing data in addition to the number of classes. Then, it normalizes the images pixels to be in the range [0, 1]. A CNN model object is then created by calling the function define_model(). A data generator is then created. With this data generator, we create a training data flow to fit the model.

```
# run the test harness to evaluate a model
def run_test_harness():
    # load the dataset (the dataset)
    trainX, trainY, testX, testY, num_classes = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # set model
    model = define_model(num_classes)
    # create a data generator
    datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,
                                 horizontal_flip=True)
    # prepare the iterator
    #it_train = datagen.flow(trainX, trainY, batch_size=64)
    it_train = datagen.flow(trainX, trainY)
    # suitable model
    #steps = int(trainX.shape[0] / 64)
    steps = int(trainX.shape[0]/10)
    #history = model.fit_generator(it_train, steps_per_epoch=steps, epochs=400,
                                  validation_data=(testX, testY),verbose=1)
    history = model.fit(it_train, validation_data=(testX, testY), epochs=50,
                        batch_size=10, verbose=1)
```

```
    # evaluate the model
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('The precision is > %.3f' % (acc * 100.0))
    # learning curves
    summarize_diagnostics(history)
```

The result of the function summarize_diagnostics(history) is shown in Figure 2.5.

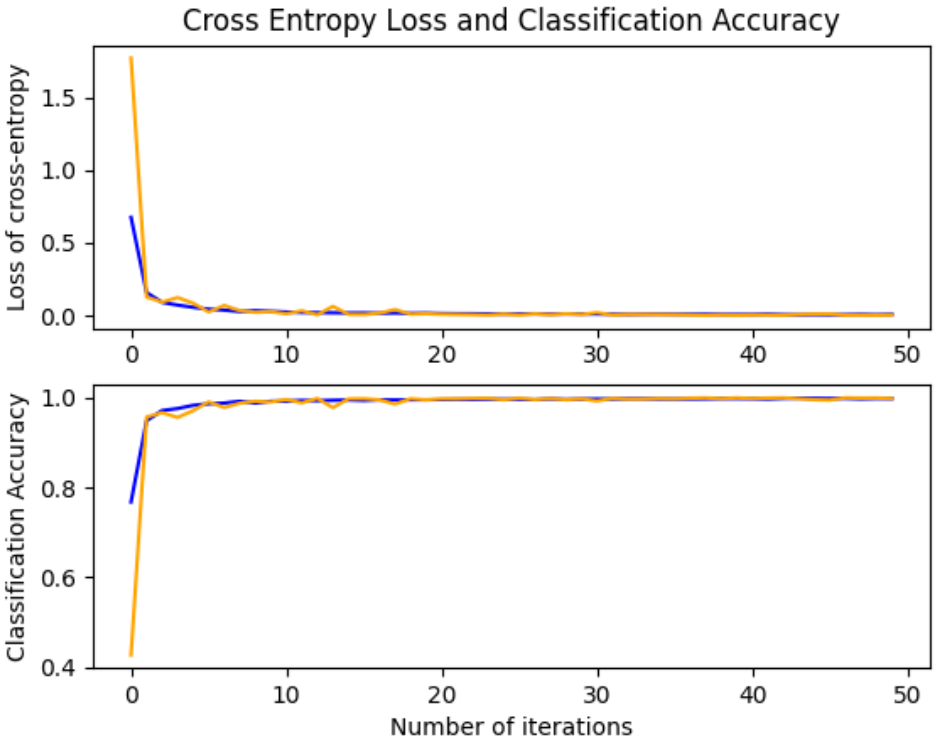

Figure 2.5 : Test results of the CNN model fitting

2.5 Saving of the CNN model

When the model training gave us good accuracy and loss, this model will be saved in a h5 file, which saves its weights. The final model to be saved should be trained using a appended dataset of training and testing data.

```
# run the test harness to evaluate a model
def run_test_harness():
    # load the data set
    trainX, trainY, testX, testY, num_classes = load_dataset()
    # prepare pixel data
    trainX, testX = prep_pixels(trainX, testX)
    # set model
    model = define_model(num_classes)
    # fit the model
    Xtrain = np.append(trainX, testX, axis=0)
    Ytrain = np.append(trainY, testY, axis=0)
    model.fit(Xtrain, Ytrain, epochs=10, verbose=1)
    # save model
    model.save('final_model.h5')
```

2.6 Evaluation of the final model

The following function will be used to evaluate the final model, which is saved in the file final_model.h5. The function evaluate() will give us its accuracy, which is 99.7%

```
# Run the test harness to evaluate a model
def run_test_harness():
    # Load data set
    testX, testY = load_dataset()
    # Prepare pixel data
    testX = prep_pixels(testX)
    # Load model
    model = load_model('final_model.h5')
    # evaluate the model on the test dataset
    _, acc = model.evaluate(testX, testY, verbose=0)
    print('The precision is > %.3f' % (acc * 100.0))
```

The following python script is used to run all modules for building the CNN model using the appropriate dataset.

```
# This script is for running all other modules
import createModel
import dataProcessing
import evaluateFinalModel
import saveFinalModel
from augmentation import data_augmentation

# Creation of the data structure
dataProcessing.create_structure('original_dataset')
# Data augmentation
data_augmentation('original_dataset', 'augmentated_dataset')
# Creation of the data structure
dataProcessing.create_structure('augmentated_dataset')
# Model creation and testing
createModel.run_test_harness()
# Saving the final model
saveFinalModel.run_test_harness()
# Evaluation of the final model
evaluateFinalModel.run_test_harness()
```

2.7 Classification test with image files and video sequences

Before deployment, we developed a graphical user interface to check the CNN model if it classifies exactly images. The graphical user interface (GUI) is developed with the package tkinter "Tk interface", which is the standard Python interface to the Tcl/Tk GUI toolkit (Moore, 2018).

This GUI contains a button to generate explanation for the model predication, which is developed with the package SHAP (SHapley Additive exPlanations), which is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions (Lundberg & Lee, 2017)

```
# This module contains the code for the graphical user interface.
# It contains a button to load the image to classify.
# Another button for classification and a third for explanation
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy
from tensorflow.keras.models import load_model
```

```
import shap
import dataProcessing
import tensorflow as tf

tf.compat.v1.disable_v2_behavior()

# Select a set of background examples to answer a prediction,
# we use the original dataset
(x_train, _), (x_test, _), _ = dataProcessing.load_data('original_dataset')
x = numpy.append(x_train, x_test, axis=0)
background = x[numpy.random.choice(x.shape[0], len(x), replace=False)]
# Loading the model
model = load_model('final_model.h5')
# Set Deep Explanation
e = shap.DeepExplainer(model, background)
# Dictionary to label all classes
classes = {
    0: 'empty',
    1: 'fell',
    2: 'sitting',
    3: 'sleep'
}
# initialize GUI
top = tk.Tk()
top.geometry('800x600')
top.title('Classification of patient\'s situations')
top.configure(background='#CDCDCD')
label = Label(top, background='#CDCDCD', font=('arial', 15, 'bold'))
sign_image = Label(top)
# The classification function
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((32, 32))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    image = image / 255
    pred = sum(model.predict([image]))
    pclasses = numpy.argmax(pred)
    sign = classes[pclasses]
    label.configure(foreground='#011638', text=sign)
    show_explain_button(file_path)
# Prediction explanation function
def explain(file_path):
    image = Image.open(file_path)
    image = image.resize((32, 32))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    image = image / 255
    # explain the model's predictions on the image
    shap_values = e.shap_values(image)
    # trace feature assignments
    shap.image_plot(shap_values, image,
                    labels=['empty', 'fell', 'sitting', 'sleep'])
# Show classification button
def show_classify_button(file_path):
    classify_b = Button(top, text="Classify the image",
                        command=lambda: classify(file_path), padx=10, pady=5)
    classify_b.configure(background='#364156', foreground='white',
                         font=('arial', 10, 'bold'))
    classify_b.place(relx=0.76, rely=0.36)
# Show explanation button
def show_explain_button(file_path):
    explain_b = Button(top, text="Explain the prediction",
                       command=lambda: explain(file_path), padx=10, pady=5)
    explain_b.configure(background='#364156', foreground='white',
                        font=('arial', 10, 'bold'))
    explain_b.place(relx=0.76, rely=0.46)
```

```
# Loading the image to classify
def upload_image():
    try:
        file_path = filedialog.askopenfilename()
        uploaded = Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width() / 2.25),
                            (top.winfo_height() / 2.25)))
        im = ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image = im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload = Button(top, text="Upload an image", command=upload_image, padx=10, pady=5)
upload.configure(background='#364156', foreground='white',
                 font=('arial', 10, 'bold'))
upload.pack(side=BOTTOM, pady=50)
sign_image.pack(side=BOTTOM, expand=True)
label.pack(side=BOTTOM, expand=True)
heading = Label(top, text='Classification of this situation', pady=20,
                font=('arial', 20, 'bold'))
heading.configure(background='#CDCDCD', foreground='#364156')
heading.pack()
top.mainloop()
```

After running this Python module, we will get this interface to upload an image by clicking on the button ‘Upload an image’ a dialog box will be shown to choose an image file, which will be displayed. After that, we can click on the button ‘Classify the image’ to display its predicted class. The button ‘Explain the prediction’ will be clicked to display a prediction explanation.

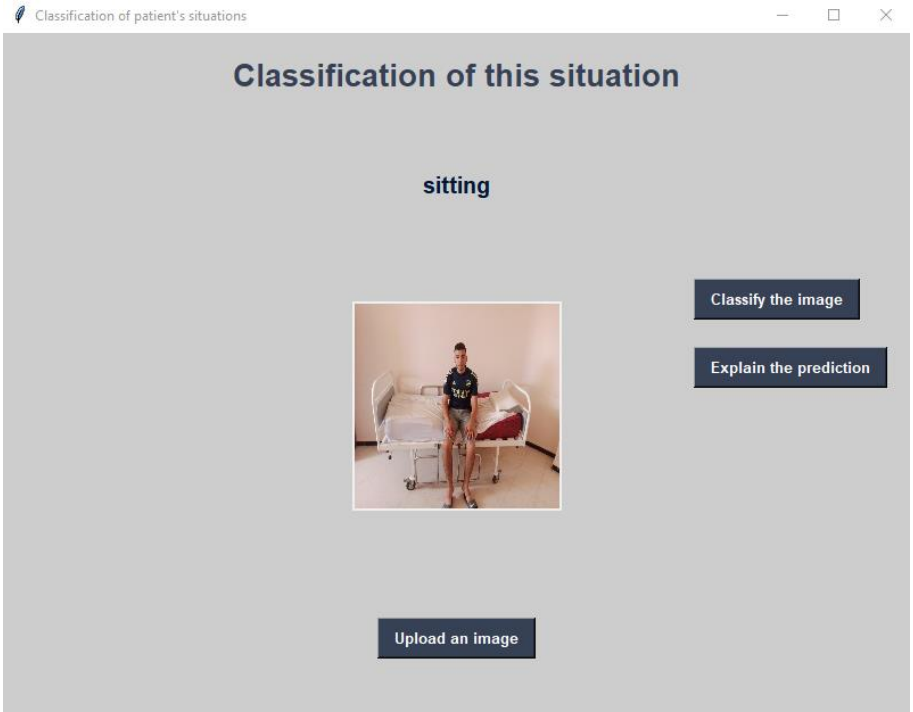

Figure 2.6 : The graphical user interface to upload and classify an image

Figure 2.7 shows explanation about the prediction. The sitting class was the closest to the given image, where the pixels of color red are dominating in the explanation, which give high SHAP value.

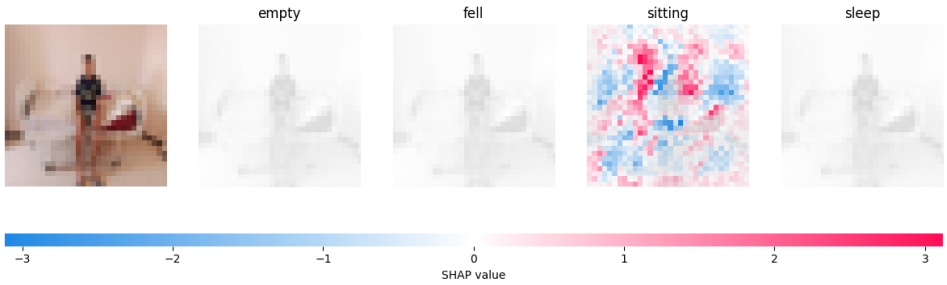

Figure 2.7 : explanation of model prediction

In addition to test the classification of png images, we can test the classification model using video sequences. The Python code for this testing approach is based on the package cv2 for computer vision (Bradski & Kaehler, 2000) shown below.

```
# This module is for the classification of patient's states
# if we have only one patient associated with a bed in one room
import numpy as np
from tensorflow.keras.models import load_model
import cv2
from absl import logging
import time

# Dictionary to label all classes
classes = {
    0: 'the bed is empty',
    1: 'the person is on the floor',
    2: 'the person is sitting on the bed ',
    3: 'the person is sleeping'
}
# load the classification model
model = load_model('./final_model.h5')
# The classification function
def classification(img):
    image = np.array(img)
    image = cv2.resize(image, (32, 32))
    image = np.expand_dims(image, axis=0)
    image = image / 255
    pred = sum(model.predict([image]))
    pclasses = np.argmax(pred)
    sign = classes[pclasses]
    print(sign)
# The main function, if type is 0 then the video capture is from camera
# else it is from video file
def main(type=0):
    vid = cv2.VideoCapture(type)
    while True:
        _, img = vid.read()
        cv2.imshow('Video Sequence', img)
        if img is None:
            logging.warning("Empty Frame")
            time.sleep(0.1)
            continue
        classification(img)
```

```
        if cv2.waitKey(1) == ord('q'):
            break
        time.sleep(0.1)
    cv2.destroyAllWindows()
# We test with video file named 'video7.mp4'
if __name__ == '__main__':
    main("./video7.mp4")
```

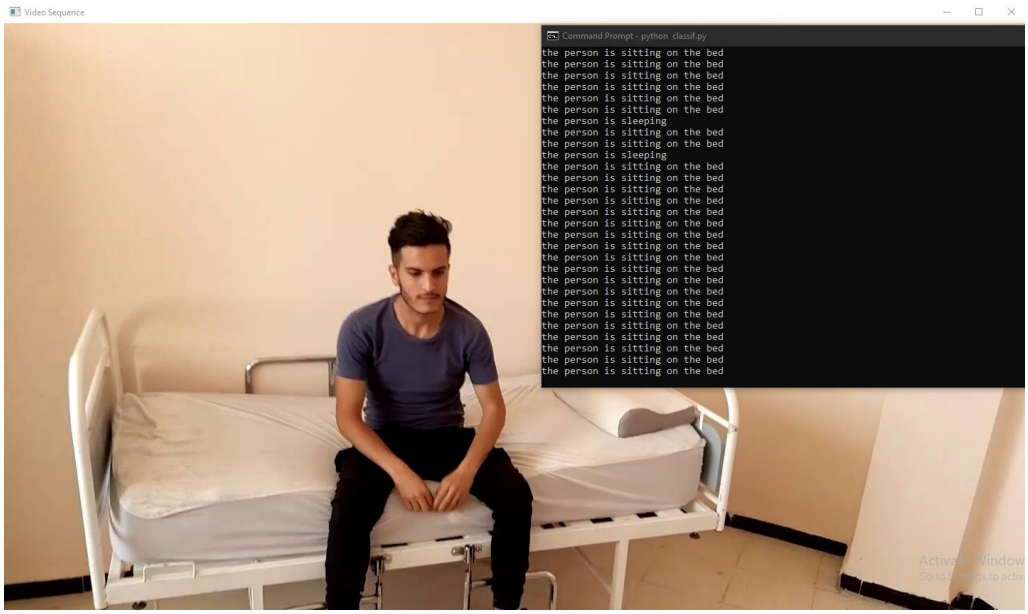

Figure 2.8 : Classification of images from video sequence

2.8 Conclusion

In this chapter, we have presented the implementation of our project called Aidy using the language Python and the PyCharm IDE. In the next chapter, we present implementation of an extension, where the classification will be applied in the case of many patients that are collocated in one room with multiple hospital beds using the technique YOLO for objects detection. In the same chapter, we will present how to use a mobile application for notifying doctors and nurses about dangerous situations facing their patients.

Chapter 3 : Classification extension to many patients in one room

3.1 Introduction

The objective is to place a surveillance camera in one hospital room to capture periodically (in well-determined time slots) images of the different patient's situations with respect to their associated beds.

The challenge is to determine to which bed a patient is associated with. Thus, the implementation presented in Chapter 2 is extended to detect first the objects of two classes (bed and patient) and then we calculate the intersection over the union of different pairs composed of an element from the beds class and another element from the patient's class.

If this value is strictly positive, we consider the pair of two objects (bed and patient) are related. An image of this pair of objects will be captured and send to the classification model (the CNN model presented in Chapter 2).

3.2 Model for detection of objects

We have developed another classification CNN model to detect different objects in hospital rooms. The inputs to this model are images, which can contain beds and patients. For the definition of this model, first we have changed the code of the layers "BatchNormalization" and "convolutional" to make them appropriate to our CNN model for detecting the objects.

```
import tensorflow as tf
import numpy as np

# The BatchNormalization layer
class BatchNormalization(tf.keras.layers.BatchNormalization):
    def call(self, x, training=False):
        if not training:
            training = tf.constant(False)
        training = tf.logical_and(training, self.trainable)
        return super().call(x, training)

# The convolutional layer
def convolutional(input_layer, filters_shape, down_sample=False,
                  activate=True, batch_norm=True, regularization=0.0005,
                  reg_stddev=0.01, activate_alpha=0.1):
    if down_sample:
        input_layer = tf.keras.layers.ZeroPadding2D(((1, 0), (1, 0)))(input_layer)
        padding = "valid"
        strides = 2
    else:
        padding = "same"
        strides = 1
    conv = tf.keras.layers.Conv2D(filters=filters_shape[-1],
             kernel_size=filters_shape[0], strides=strides, padding=padding,
             use_bias=not batch_norm,
             kernel_regularizer=tf.keras.regularizers.l2(regularization),
             kernel_initializer=tf.random_normal_initializer(stddev=reg_stddev),
             bias_initializer=tf.constant_initializer(0.))(input_layer)
    if batch_norm:
        conv = BatchNormalization()(conv)
    if activate:
        conv = tf.nn.leaky_relu(conv, alpha=activate_alpha)
    return conv
```

The following is to apply convolution operation on cuts of the image and at the end it returns a merge of the results.

```
def res_block(input_layer, input_channel, filter_num1, filter_num2):
    short_cut = input_layer
    conv = convolutional(input_layer, filters_shape=(1, 1, input_layer,
                         filter_num1))
    conv = convolutional(conv, filters_shape=(3, 3, filter_num1, filter_num2))
    res_output = short_cut + conv
    return res_output
```

YOLO (Redmon, Divvala, Girshick, & Farhadi, 2016), which stands for "You Only Look Once.", is a single-stage object detector that leverages the power of convolutional neural networks to detect multiple objects in a given image. It divides the image into a grid and predicts class probabilities and box coordinates for each grid cell of the image.

YOLO applies a single neural network (Redmon & Farhadi, YOLOv3: An Incremental Improvement, 2018) on the whole image and detects the objects in a single network pass. This architecture is similar in spirit to an image classification network, where the object is classified in a single forward pass. Thus, YOLO is faster than other state-of-the-art detectors, making it ideal for many industrial applications. YOLOv3 was one of the best models in terms of real-time object detection. The significant difference between YOLOv3 and its predecessors is in the network architecture called Darknet-53.

The Darknet-53 architecture consisting of 53 convolutional layers that act as a base for the object detection network or a feature extractor. The 53 layers are pre-trained on the image classification task using the ImageNet dataset (Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009). For the object detection task, 53 more layers are stacked on top of the base/backbone network, making it a total of 106 layers, and we get the final model known as YOLOv3.

```
def darknet53(input_data):
    input_data = convolutional(input_data, (3, 3, 3, 32))
    input_data = convolutional(input_data, (3, 3, 32, 64), down_sample=True)
    for i in range(1):
        input_data = res_block(input_data, 64, 32, 64)
    input_data = convolutional(input_data, (3, 3, 64, 128), down_sample=True)
    for i in range(2):
        input_data = res_block(input_data, 128, 64, 128)
    input_data = convolutional(input_data, (3, 3, 128, 256), down_sample=True)
    for i in range(8):
        input_data = res_block(input_data, 256, 128, 256)
    route_1 = input_data
    input_data = convolutional(input_data, (3, 3, 256, 512), down_sample=True)
    for i in range(8):
        input_data = res_block(input_data, 512, 256, 512)
    route_2 = input_data
    input_data = convolutional(input_data, (3, 3, 512, 1024), down_sample=True)
    for i in range(4):
        input_data = res_block(input_data, 1024, 512, 1024)
    return route_1, route_2, input_data

# This will be called to resize the input image
def upsample(input_layer):
    return tf.image.resize(input_layer, (input_layer.shape[1] * 2,
           input_layer.shape[2] * 2), method='nearest')
```

When the object detection model is defined, its weights will be updated by predefined weights from YOLO3 project, where the Darknet-53 architecture was trained on the image classification task with the ImageNet dataset in the pre-training step.

```
def Load_weights(model, weight_file):
    wf = open(weight_file, 'rb')
    major, minor, revision, seen, _ = np.fromfile(wf, dtype=np.int32, count=5)
    j = 0
    for i in range(75):
        conv_layer_name = 'conv2d_%d' % i if i > 0 else 'conv2d'
        bn_layer_name = 'batch_normalization_%d' % j if j > 0 else
                        'batch_normalization'
        conv_layer = model.get_layer(conv_layer_name)
        filters = conv_layer.filters
        k_size = conv_layer.kernel_size[0]
        in_dim = conv_layer.input_shape[-1]
        if i not in [58, 66, 74]:
            # darknet weights: [beta, gamma, mean, variance]
            bn_weights = np.fromfile(wf, dtype=np.float32, count=4 * filters)
            bn_weights = bn_weights.reshape((4, filters))[[1, 0, 2, 3]]
            bn_layer = model.get_layer(bn_layer_name)
            j += 1
        else:
            conv_bias = np.fromfile(wf, dtype=np.float32, count=filters)
        # darknet shape is (out_dim, in_dim, height,width)
        conv_shape = (filters, in_dim, k_size, k_size)
        conv_weights = np.fromfile(wf, dtype=np.float32,
                                   count=np.product(conv_shape))
        # tf shpae (height, width, in_dim, out_dim)
        conv_weights = conv_weights.reshape(conv_shape).transpose([2, 3, 1, 0])
        if i not in [58, 66, 74]:
            conv_layer.set_weights([conv_weights])
            bn_layer.set_weights(bn_weights)
        else:
            conv_layer.set_weights([conv_weights, conv_bias])
    assert len(wf.read(0)) == 0, 'failed to read all data'
    wf.close()
    return model
```

3.3 Creating the CNN model for object detection

This Python code will create a convolutional model to detect objects from an image. The function yoloV3() will define object detection model based on the darkNet-53 architecture, where its input layer is given as an argument of this function.

```
import tensorflow as tf
from commonBlocks import darknet53, upsample, convolutional
import numpy as np
from commonBlocks import Load_weights

# hyper parameters
NUM_CLASSES = 80
STRIDES = np.array([8, 16, 32])
ANCHORS = (
    1.25, 1.625, 2.0, 3.75, 4.125, 2.875, 1.875, 3.8125, 3.875, 2.8125, 3.6875,
    7.4375, 3.625, 2.8125, 4.875, 6.1875, 11.65625, 10.1875)
ANCHORS = np.array(ANCHORS).reshape(3, 3, 2)
weight_file = "yolov3.weights"

# This is the YOLO detection model that we used as the base of our CNN model
# to detect objets
def yoloV3(input_layer):
    route_1, route_2, conv = darknet53(input_layer)
    conv = convolutional(conv, (1, 1, 1024, 512))
    conv = convolutional(conv, (3, 3, 512, 1024))
```

```
    conv = convolutional(conv, (1, 1, 1024, 512))
    conv = convolutional(conv, (3, 3, 512, 1024))
    conv = convolutional(conv, (1, 1, 1024, 512))
    conv_lobj_branch = convolutional(conv, (3, 3, 512, 1024))
    conv_lbbox = convolutional(conv_lobj_branch, (1, 1, 1024, 3 * (NUM_CLASSES +
                               5)), activate=False, batch_norm=False)
    conv = convolutional(conv, (1, 1, 512, 256))
    conv = upsample(conv)
    conv = tf.concat([conv, route_2], axis=-1)
    conv = convolutional(conv, (1, 1, 768, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv = convolutional(conv, (3, 3, 256, 512))
    conv = convolutional(conv, (1, 1, 512, 256))
    conv_mobj_branch = convolutional(conv, (3, 3, 256, 512))
    conv_mbbox = convolutional(conv_mobj_branch, (1, 1, 512, 3 * (NUM_CLASSES +
                               5)), activate=False, batch_norm=False)
    conv = convolutional(conv, (1, 1, 256, 128))
    conv = upsample(conv)
    conv = tf.concat([conv, route_1], axis=-1)
    conv = convolutional(conv, (1, 1, 384, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv = convolutional(conv, (3, 3, 128, 256))
    conv = convolutional(conv, (1, 1, 256, 128))
    conv_sobj_branch = convolutional(conv, (3, 3, 128, 256))
    conv_sbbox = convolutional(conv_sobj_branch,
             (1, 1, 256, 3 * (NUM_CLASSES + 5)), activate=False, batch_norm=False)
    return [conv_sbbox, conv_mbbox, conv_lbbox]
```

The following function will be used to decode the detected objects from the convolutional layers. In this case, the sigmoid function is used for decoding the detected objects.

```
def decode(conv_out, i=0):
    conv_shape = tf.shape(conv_out)
    batch_size = conv_shape[0]
    output_size = conv_shape[1]
    conv_output = tf.reshape(conv_out, (batch_size, output_size, output_size, 3,
                             5 + NUM_CLASSES))
    conv_raw_dxdy = conv_output[:, :, :, :, 0:2]
    conv_raw_dwdh = conv_output[:, :, :, :, 2:4]
    conv_raw_conf = conv_output[:, :, :, :, 4:5]
    conv_raw_prob = conv_output[:, :, :, :, 5:]
    y = tf.tile(tf.range(output_size, dtype=tf.int32)[:, tf.newaxis],
                [1, output_size])
    x = tf.tile(tf.range(output_size, dtype=tf.int32)[tf.newaxis, :],
                [output_size, 1])
    xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :, tf.newaxis]], axis=-1)
    xy_grid = tf.tile(xy_grid[tf.newaxis, :, :, tf.newaxis, :],
                      [batch_size, 1, 1, 3, 1])
    xy_grid = tf.cast(xy_grid, tf.float32)
    pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) * STRIDES[i]
    pred_wh = (tf.exp(conv_raw_dwdh) * ANCHORS[i]) * STRIDES[i]
    pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
    pred_conf = tf.sigmoid(conv_raw_conf)
    pred_prob = tf.sigmoid(conv_raw_prob)
    return tf.concat([pred_xywh, pred_conf, pred_prob], axis=-1)
```

The function Model() creates a CNN model for object detection, which has an input layer defined with the keras function Input() and box tensors defined as feature maps of the YOLO architecture. The weights of this model architecture are loaded from weights of the pre-trained YOLO model.

```
def Model():
    input_layer = tf.keras.layers.Input([416, 416, 3])
    feature_maps = yoloV3(input_layer)
    bbox_tensors = []
    for i, fm in enumerate(feature_maps):
        bbox_tensor = decode(fm, i)
        bbox_tensors.append(bbox_tensor)
    model = tf.keras.Model(input_layer, bbox_tensors)
    model = Load_weights(model, weight_file)
    return model
```

3.4 Package of useful functions

A package called utils.py is developed to define functions that are useful for our application. These are used for objects (beds and persons) detection. We begin by importing useful packages.

```
import tensorflow as tf
import numpy as np
import cv2
```

The names of objects as bed and person are read from the class file name and returned as a set of names by this function.

```
def read_class_names(class_file_name):
    names = {}
    with open(class_file_name, 'r') as data:
        for ID, name in enumerate(data):
            names[ID] = name.strip('\n')
    return names
```

This code is to transform the image to the appropriate size and range of pixels [0, 1].

```
def transform_images(x_train, size):
    x_train = tf.image.resize(x_train, (size, size))
    x_train = x_train / 255
    return x_train
```

Since YOLOv3 is supposed to detect objects at multiple scales at each of the last residual layers, a detection layer is attached to make object detection predictions. This function will be used to detect the boxes from the predicted objects by the CNN model for the objects detection.

```
def box_detector(pred):
    center_x, center_y, width, height, confidence, classes =
                tf.split(pred, [1, 1, 1, 1, 1, -1], axis=-1)
    top_left_x = (center_x - width / 2.0) / 416.0
    top_left_y = (center_y - height / 2.0) / 416.0
    bottom_right_x = (center_x + width / 2.0) / 416.0
    bottom_right_y = (center_y + height / 2.0) / 416.0
    boxes = tf.concat([top_left_y, top_left_x, bottom_right_y, bottom_right_x],
                      axis=-1)
    scores = confidence * classes
    scores = np.array(scores)
    scores = scores.max(axis=-1)
    class_index = np.argmax(classes, axis=-1)
    final_indexes = tf.image.non_max_suppression(boxes, scores, max_output_size=10)
    final_indexes = np.array(final_indexes)
    class_names = class_index[final_indexes]
    boxes = np.array(boxes)
    scores = np.array(scores)
    class_names = np.array(class_names)
```

```
    boxes = boxes[final_indexes, :]
    scores = scores[final_indexes]
    boxes = boxes * 416
    return boxes, class_names, scores
```

This function is for drawing rectangles around the boxes of detected objects.

```
def drawbox(boxes, class_names, scores, names, img):
    data = np.concatenate([boxes, scores[:, np.newaxis],
                           class_names[:, np.newaxis]], axis=-1)
    data = data[np.logical_and(data[:, 0] >= 0, data[:, 0] <= 416)]
    data = data[np.logical_and(data[:, 1] >= 0, data[:, 1] <= 416)]
    data = data[np.logical_and(data[:, 2] >= 0, data[:, 2] <= 416)]
    data = data[np.logical_and(data[:, 3] >= 0, data[:, 3] <= 416)]
    data = data[data[:, 4] > 0.3]
    img = cv2.resize(img, (416, 416))
    for i, row in enumerate(data):
        img = cv2.rectangle(img, (int(row[1]), int(row[0])), (int(row[3]),
                             int(row[2])), (170, 1, 130), 1)
        img = cv2.putText(img, (names[row[5]] + ": " + "{:.4f}".format(row[4])),
                          (int(row[1]), int(row[0])),
                          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
    return img
```

The detected objects that have scores of detection less than or equal 0.3 of detection probability to be one of the targeted objects, will be cleaned from the list of objects. Our concerns are about beds and persons only (i.e. objects other than beds and persons can be cleaned from the list by this function).

```
def clean_objects(boxes, class_names, scores):
    new_boxes = []
    new_class_names = []
    nes_scores = []
    for i in range(len(class_names)):
        if scores[i] > 0.3:
            new_boxes.append(boxes[i])
            new_class_names.append(class_names[i])
            nes_scores.append(scores[i])
    return [new_boxes, new_class_names, nes_scores]
```

Calculate the intersection over union of two boxes to check if they are related. In our application, we want to check to which bed a patient is associated.

```
def calculate_iou(box1, box2):
    # calculating the inter area
    xa = max(box1[0], box2[0])
    ya = max(box1[1], box2[1])
    xb = min(box1[2], box2[2])
    yb = min(box1[3], box2[3])
    inter_area = abs(max((xb - xa), 0) * max((yb - ya), 0))
    if inter_area == 0:
        return 0
    # calculating both boxes area
    box1_area = abs(max((box1[2] - box1[0]), 0) * max((box1[3] - box1[1]), 0))
    box2_area = abs(max((box2[2] - box2[0]), 0) * max((box2[3] - box2[1]), 0))
    # calculating iou by dividing inter area by the sum of
    # the two boxes plus the inter area
    iou = inter_area / float(box1_area + box2_area - inter_area)
    return iou
```

This function is used to get the list of beds from the detected objects (boxes).

```
def get_beds(boxes, class_names, scores):
    beds = []
    for i in range(len(class_names)):
        if class_names[i] == 59:
            beds.append([boxes[i], class_names[i], scores[i]])
    return beds
```

This function is used to get the list of patients (i.e. persons) from the detected objects (boxes).

```
def get_persons(boxes, class_names, scores):
    persons = []
    for j in range(len(class_names)):
        if class_names[j] == 0:
            persons.append([boxes[j], class_names[j], scores[j]])
    return persons
```

This function is very useful for cropping an image to many small images. Each sub image contains only one bed or pair of a bed and a patient (person) if their intersection over union is strictly positive. The input to this function is a list of pairs (bed, person) or just a bed.

```
def image_crop(iou_count):
    cropped_annotation = []
    for count in iou_count:
        bed = count[0]
        if len(count) == 1: # the bed is empty
            y_min = bed[0][0] - 100
            if y_min < 0:
                y_min = 0
            y_max = bed[0][2] + 100
            if y_max > 416:
                y_max = 416
            x_min = bed[0][1] - 100
            if x_min < 0:
                x_min = 0
            x_max = bed[0][3] + 100
            if x_max > 416:
                x_max = 416
        else: # a pair of bed and patient
            person = count[1]
            y_min = min(bed[0][0], person[0][0]) - 100
            if y_min < 0:
                y_min = 0
            y_max = max(bed[0][2], person[0][2]) + 100
            if y_max > 416:
                y_max = 416
            x_min = min(bed[0][1], person[0][1]) - 100
            if x_min < 0:
                x_min = 0
            x_max = max(bed[0][3], person[0][3]) + 100
            if x_max > 416:
                x_max = 416
        cropped_annotation.append([y_min, y_max, x_min, x_max])
    return cropped_annotation
```

3.5 Classification of cropped images

Our objective is to classify cropped images, which are captured from video sequences taken from surveillance camera. Each cropped image represents a pair of associated bed with a patient. First, we begin by importing the packages.

```
from objects_model import Model
import app_script
import tensorflow as tf
import utils
from tensorflow.keras.models import load_model
import numpy as np
import cv2
from absl import logging
import time
import logging, os
```

We specify the dictionary to label all classes that the CNN model will predict from the cropped image ‘img’ using the function ‘classification(img)’

```
classes = {
    0: 'the bed is empty',
    1: 'the person is on the floor',
    2: 'the person is sitting on the bed ',
    3: 'the person is sleeping'
}

def classification(img):
    # Load the classification model
    model = load_model('./final_model.h5')
    image = np.array(img)
    image = cv2.resize(image, (32, 32))
    image = np.expand_dims(image, axis=0)
    image = image / 255
    pred = sum(model.predict([image]))
    pclasses = np.argmax(pred)
    sign = classes[pclasses]
    return sign
```

We use the function Model() from the package objects_model that we developed, to predict the objects in hospital rooms, the function utils.box_detector() will be used to detect the object boxes and the function utils.clean_objects() to clean the objects list from the object that their prediction probability is less than or equal 0.3.

Then we create a list of beds and a list of patients (persons) by calling utils.get_bed() and utils.get_person(), respectively. Then, a list ‘iou_count’ of pairs of beds and patients is created, where a pair of two objects (bed and patient) should have an intersection over union ‘iou’ strictly positive, we compute the probability of their coexistence (intersection over union) using the function utils.calculate_iou().

The list ‘iou_count’ will be used to crop the original image to small images each one contains only a pair of related bed and person. This image cropping is realized by the function utils.image_crop(iou_count), which returns an annotations list called ‘crop_annotations’ to be proceeded by the classification model presented in Chapter 2.

Each cropped image will be send to the classification() function to predict the patient situation vis-à-vis its bed using the trained CNN model. This status of patient will be used to change the state in the mobile application database using the function app_script.changeState(True, 'img.jpg', status) from the package app_changeState of the mobile application package. The parameter True is to show the cropped image on the mobile device with the patient status. The original image with its cropped images, are shown in different windows to check the predictions, which are printed. The boxes corresponding to a cropped image are drawn.

```
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
def main(type=0):
    model = Model()
    names = utils.read_class_names("classes.names")
    video = cv2.VideoCapture(type)
    while True:
        _, img = video.read()
        if img is None:
            logging.warning("Empty Frame")
            time.sleep(0.1)
            continue
        img_in = tf.expand_dims(img, 0)
        img_in = utils.transform_images(img_in, 416)
        pred_bbox = model.predict(img_in)
        pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for x in pred_bbox]
        pred_bbox = tf.concat(pred_bbox, axis=0)
        boxes, class_names, scores = utils.box_detector(pred_bbox)
        boxes, class_names, scores = utils.clean_objects(boxes, class_names,
                                                         scores)
        beds = utils.get_beds(boxes, class_names, scores)
        persons = utils.get_persons(boxes, class_names, scores)
        iou_count = []
        for bed in range(len(beds)):
            l = [beds[bed]]
            for person in range(len(persons)):
                iou = utils.calculate_iou(beds[bed][0], persons[person][0])
                if iou > 0:
                    l.append(persons[person])
                    break
            iou_count.append(l)
        crop_annotations = utils.image_crop(iou_count)
        for annotation in crop_annotations:
            img_array = cv2.resize(img, (416, 416))
            cropped_img = img_array[int(annotation[0]): int(annotation[1]),
                                    int(annotation[2]): int(annotation[3])]
            cropped_img = cv2.resize(cropped_img, (400, 300))
            cv2.imshow('image treated with classifier', cropped_img)
            cv2.imwrite('img.jpg', cropped_img)
            cropped_img = tf.image.resize(cropped_img, (32, 32))
            cropped_img = np.array(cropped_img)
            sign = classification(cropped_img)
            print(sign)
            if sign == 'the bed is empty':
                app_script.changeState(True, 'img.jpg', 3)
            if sign == 'the person is on the floor':
                app_script.changeState(True, 'img.jpg', 4)
            if sign == 'the person is sitting on the bed ':
                app_script.changeState(True, 'img.jpg', 2)
            if sign == 'the person is sleeping':
                app_script.changeState(True, 'img.jpg', 1)
        bbox = utils.box_detector(pred_bbox)
        img = utils.drawbox(bbox[0], bbox[1], bbox[2], names, img)
        img = cv2.resize(img, (900, 500))
        cv2.imshow('Real-time detection', img)
        if cv2.waitKey(1) == ord('q'):
            break
        time.sleep(0.1)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main("./video_test2.mp4")
```

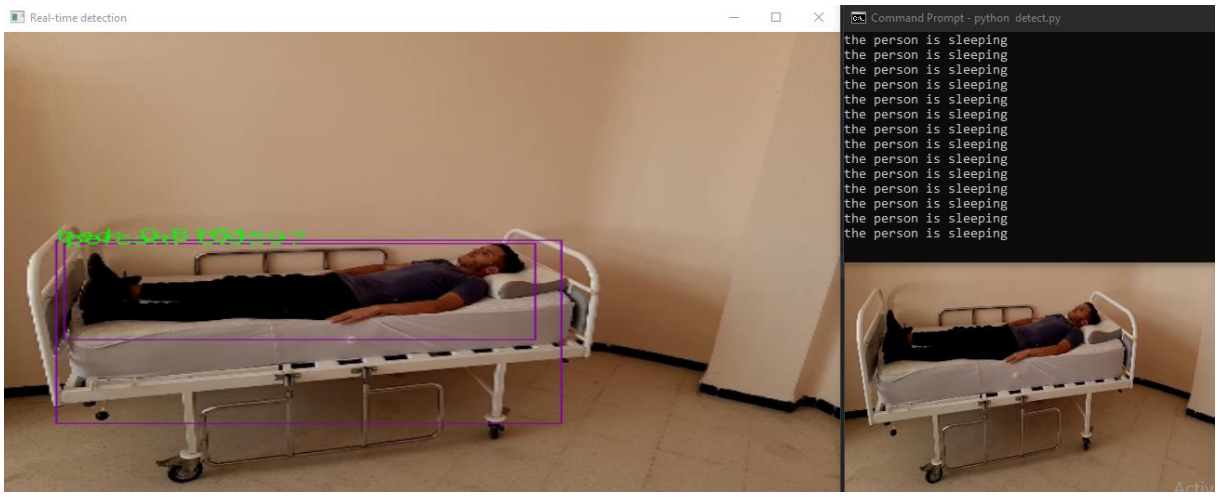

Figure 3.1 ; Classification of cropped images from video sequence

3.6 Mobile Application for notifications

A mobile application is developed to notify doctors/nurses in case where the patients are in dangerous situations. This python code is developed to create connections with the mobile application running on a mobile device to send these notifications about the patient's states.

```
import pyrebase

# state of patient in ORDER
# { 1 -> mean:  In bed }
# { 2 -> mean:  getting out from bed }
# { 3 -> mean:  patient is away from bed }
# { 4 -> mean:  fell }
# { 5 -> mean:  out of the room }
```

This function will be called to update the data base by the new state of the patient. If the show of image is required, which is given by the parameter 'local_image', the mobile application will display it on the mobile device with the patient state notification.

```
def changeState(is_img_required, local_img, n_state):
    db.child("home/patient_activity").set({'patient_activity': n_state})
    # To put the image if it is required
    if (is_img_required):
        path_on_cloud = "patient/event_image.jpg"
        path_local = local_img
        storage_0.child(path_on_cloud).put(path_local)
```

These parameters are used to setup the connection configuration.

```
config = {
    "apiKey": "AIzaSyC9SCwRVisyPeFdcq9Bzm_roeIwzaABzR8",
    "authDomain": "aidy-983b2.firebaseapp.com",
    "databaseURL": "https://aidy-983b2-default-rtdb.firebaseio.com",
    "projectId": "aidy-983b2",
    "storageBucket": "aidy-983b2.appspot.com",
    "messagingSenderId": "372739667315",
    "appId": "1:372739667315:web:af31fd5374f2cef5804106",
    "measurementId": "G-DRCL8WKLQT"
}
```

The mobile application objects are created and initialized with these lines of code to setup the objects references

```
firebase = pyrebase.initialize_app(config)
storage_0 = firebase.storage()
db = firebase.database()
```

Figure 3.2 shows an example of notification for mobile application running on mobile device and connected to the system Aidy.

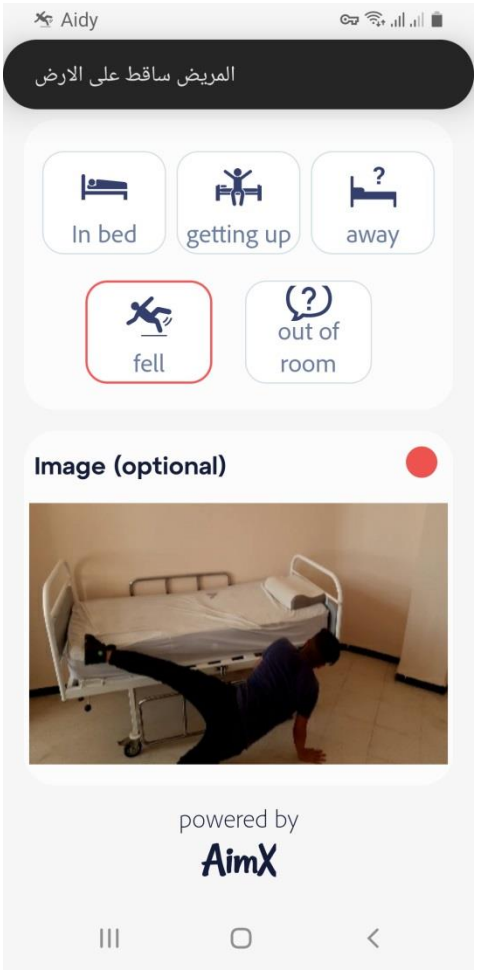

Figure 3.2 : notification to mobile application

3.7 Conclusion

In this chapter, we presented the implementation of the extension of our project which concerns the prediction of the situations of several patients in the same hospital room. This extension is essentially based on the detection of objects and the cutting of images. In addition we presented a mobile application to notify doctors and nurses about dangerous patient situations.

GENERAL CONCLUSION

In this Master's thesis, we presented a project within the framework of monitoring patients staying in a hospital. Each room can contain several patients where a patient is associated with a bed with all the medical facilities. The rooms are occupied by surveillance cameras and the video sequences will be transmitted to our intelligent system called Aidy.

The Aidy system first tries to detect object pairs of patients and beds in order to take an image that relates patient to bed using a classification model for object detection. These cropped images from the original image will be sent to a second classification model to predict the situation of patients. If the prediction indicates a dangerous case of a patient, a notification will be transmitted to a mobile application installed on the mobile device of a doctor or a nurse, which is connected to our intelligent system Aidy.

For the good realization of this intelligent system, we have developed a graphic interface in order to facilitate the tests by giving images of validation and to have explanations on the predictions to decide well if the system works well or not. All the tests carried out have given satisfactory and encouraging results to continue with this project.

As a perspective, we plan to deploy this intelligent system for patient monitoring in real cases at the hospital level. In addition, we also want to extend this work for other cases that allow diagnosing patients in order to give predictions to doctors. The extension project that we want to realize is on the intelligent control of patients with cardiac problems.

Bibliography

Asan, O. (2020). *Role of Artificial Intelligence in Patient Safety.* Outcomes:Systematic Literature Review.

Bourahla, A. E., & Bourahla, M. (2022). Approach for the development of mobile applications based on migrant objects. *Comput. Sci. J. Moldova*, 3-27.

Bourahla, M. (2017). LTL transformation modulo positive transitions. *IET Comput. Digit. Tech.*, 205-213.

Bourahla, M. (2018). Description and reasoning for vague ontologies using logic programming. *IET Software*, 397-409.

Bradski, G., & Kaehler, A. (2000). *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Media, Inc.

Dash, S., Shakyawar, S. K., Sharma, M., & Kaushik, S. (2019). *Big data in healthcare: management, analysis and future prospects.*

Dauverge, P. (2021). The globalization of artificial intelligence: Consequences for the politics of environmentalism. *Globalizations*, 285-299.

Deng, J., Dong, W., Socher, R., Li, L. -J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248-255.

Gua, J., Wangb, Z., & Kuenb, J. (2017). *Recent Advances in Convolutional Neural Networks.*

Guido, V. R., & Drake Jr, F. L. (1995). *Python reference manual.* Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica.

Hossain, M. S., Muhammad, G., & Guizan, N. (2020). Explainable AI and Mass Surveillance System-Based Healthcare Framework to Combat COVID-I9 Like Pandemics. *IEEE Network*, 126-132.

Jin, D., Sergeeva, E., Weng, W.-H., Chauhan, G., & Szolovits, P. (2021). *Explainable Deep Learning in Healthcare: A Methodological Survey from an Attribution View.*

Jogin, M., & Mohana, D. G. (2020). *Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning.*

Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. In I. G. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett, *Advances in Neural Information Processing Systems 30* (pp. 4765--4774). Curran Associates, Inc.

Moore, A. D. (2018). *Python GUI programming with Tkinter.* Packt.

Perez, L., & Wang, J. (2017). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning.*

Pesce, R. (2022, April 13). *Introducing PyCharm 2022.1!* Retrieved from https://blog.jetbrains.com/pycharm/2022/04/2022-1/

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement.*

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection.

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach, 4th Edition.* Pearson.

Wu, W.-T., Li, Y.-J., Feng, A.-Z., Li, L., Huang, T., Xu, A.-D., et al. (2020). *Data mining in clinical big data: the frequently used databases, steps, and methodological models.*

Appendix

Business plan