Interpretation of neural networks and advanced image augmentation for visual control of drones in human proximity

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics

presented by
Marco Ferri

under the supervision of
Prof. Alessandro Giusti
co-supervised by
Dr. Dario Mantegazza

February 2021

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Marco Ferri
Lugano, 22 February 2021

To someone

"Sometimes it is the people no one
can imagine anything of, who do
the things no one can imagine."
The Imitation Game

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam vulputate erat quis justo varius vehicula. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; In ut placerat velit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In elementum egestas turpis et auctor. Vestibulum gravida lorem nec egestas ornare. Duis varius arcu imperdiet, feugiat odio in, facilisis est. Quisque interdum vitae odio ut vehicula. Etiam molestie enim non risus maximus, vitae efficitur mauris sollicitudin. Phasellus consequat nulla at nulla tempus varius. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Nulla faucibus aliquam nisl, vel luctus arcu semper vel. Aliquam ipsum risus, feugiat quis nulla ac, aliquet imperdiet ante. Ut et massa sem. Donec eu ex augue. Nam urna nunc, commodo ac nunc et, auctor mattis nunc. Nam pellentesque laoreet purus, a tristique nisi auctor non. Integer sed congue lorem. Maecenas faucibus turpis nec ultrices tempor. Vivamus condimentum nibh sit amet molestie tempor. Pellentesque cursus diam maximus nisi gravida, sed consequat mauris malesuada. Fusce eget nisl vehicula, porta enim sit amet, ornare massa. Nunc et ex a eros tempor mattis in quis quam. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis volutpat ex nec ante tempus, quis bibendum nisi posuere. Cras augue nulla, ornare vel risus quis, vulputate bibendum sem.

Proin placerat euismod cursus. Nulla ornare lobortis ligula et pellentesque. Nullam cursus neque ut fermentum euismod. Sed sit amet luctus orci. Nunc convallis urna id nisl vestibulum ullamcorper. Aliquam ullamcorper porta dui et aliquam. Pellentesque urna nibh, finibus sed condimentum eget, interdum ut tellus. Morbi aliquet, erat et rhoncus cursus, lectus nibh vulputate nisl, eu interdum dui dolor quis ipsum. Donec id libero sit amet orci gravida pretium. Mauris in magna non nunc posuere consequat. Donec diam tortor, viverra posuere velit et, convallis commodo massa. Fusce consectetur posuere ex, nec tincidunt neque posuere tempus. Vivamus vitae accumsan ligula. Nulla facilisi. Donec pellentesque commodo lorem ac semper.

Acknowledgements

Thanks to...

Contents

Figures

Tables

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.1 Objective

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2 Outline

The thesis is composed of X chapters:

- Chapter **??** summarises ;
- Chapter **??** provides ;
- Chapter **??** presents ;
- Chapter **??** illustrates ;
- Chapter **??** explores ;
- Chapter **??** addresses .

Chapter 2

Theoretical Foundation

2.1 Robotics

2.2 Machine Learning

2.3 Human-drone Interaction

A good variety of research can be found on human-robot interaction and a lot is yet to come. In the field, drones represent a specific segment due to their ability to freely move in the 3D space, opening access to new use cases while representing a real challenge for professionals and researchers.

In this section, we firstly present a general overview on the topic, then we focus on related work by IDSIA.

2.3.1 The State of the Art of Human–Drone Interaction

A recent article, published in Nov 2019 for IEEE Access (Tezza and Andujar [2019]), explores literature and state of the art for human-drone interaction. Drones range from small toy-grade remote-controlled aircraft to fully-autonomous systems capable of decision-making through a large variety of sensors. Their usage grew a lot in the last years and it is expected to keep growing, thanks to decreasing costs and powerful features they can provide both for personal, commercial, and social usage.

United States Federal Aviation Administration (FAA) expects that total drone registrations will increase by more than 60% between 2018 and 2022 with a particular increment in the commercial sector rather than the hobbyist one, even though the latter still counts the largest number of units. Moreover, FAA reports that almost half of drones usage is for aerial photography (48%), followed by industrial inspection (28%) and agriculture (17%). Accordingly to Tezza and Andujar [2019], drones will become ubiquitous to society and, in the next decade, they will be extensively used

in advertising, shipping, sports, emergency, and many other fields for augmenting human capabilities.

The main concerns about drones today regard safety issues caused by propellers and limited flight times, usually no longer than 30 minutes due to limited battery capacity. Research in the sector of human-drone interaction mainly focuses on their control (through gestures, voice, or custom interfaces), communication between the user and the drone itself (in terms of acknowledgment and intents), perception of users' safety during flight, and innovative use cases.

2.3.2 Vision-based Control of a Quadrotor in User Proximity

Our work is built upon the original master thesis (Mantegazza [2018]) and paper (Mantegazza et al. [2019]) *Vision-based Control of a Quadrotor in User Proximity: Mediated vs End-to-End Learning Approaches* from Dario Mantegazza, developed at IDSIA in Lugano. In his thesis, the author proposes a machine learning model for teaching a drone to interact with a person by continuously flying to face the user frontally, towards the direction of the head. The problem is approached as a reactive control procedure and addressed with supervised learning, thus provides an interesting starting point for many other robotics applications.

The author collects data and tests his model on the Parrot Bebop 2, a 500grams drone commonly used for photography and leisure purposes, capable of effective video stabilization. However, the software runs off-board, on a dedicated computer remotely connected through WiFi.

A considerable amount of flights is recorded for building the training data by programmatically flying the drone in front of a person, controlling it through an omniscient controller that knows both the drone's and user's pose. Images produced by the front-facing camera of the drone are used as input for a custom-designed Residual Neural Network (ResNet) architecture to infer the relative user's position with respect to (wrt) the drone. Practically, the neural network performs a regression on the four variables that form the user's pose (X, Y, Z, YAW) and learns to predict their values by using spacial information contained in the input images.

In the paper, the author also makes a comparison between the mediated approach described above and another end-to-end approach that directly learns control signals[1] for the drone, instead of the user's pose. Another experiment also considers a learned controller. All the solutions provide similar results, but the former can be adapted to other tasks by simply designing a custom controller, providing a more transparent and analyzable solution[2].

Even though this kind of problems on human recognition and pose estimation could be faced with more advanced deep learning algorithms, making a simple re-

[1] desired pitch, roll, yaw, and vertical velocity

[2] take a look at image A.1 for details about models architectures designed for the three approaches

gression on four variables allows the network to be fairly small, so that the prediction task is light, fast to execute, and possibly portable on low-end devices.

Network and dataset defined in Mantegazza et al. [2019] have been used for our entire project, so the original code repository[3] is available for reference. The next chapters constantly make use of this particular model architecture, which will be further explained in section 3.2.

Having no official name, for enhancing readability, the custom ResNet architecture proposed by the author will be simply called *ProximityNet*. For a better understanding, also a good descriptive video is available at `https://drive.switch.ch/index.php/s/MlEDrsuHcSl5Aw5`.

2.3.3 Embedded Implementation of Controller for Nano-Drones

Autonomous navigation is an important and well-known area of research in robotics, which usually requires accomplishing complex and computationally-expensive tasks such as localization, mapping and path planning. Recent studies have started to approach autonomous driving through deep learning and imitation learningHussein et al. [2017a], where neural networks learn by imitating human behavior in specific tasks.

In 2018, researchers at the UZH University of Zürich have demonstrated that ResNets are able to provide satisfactory performance in the field (Loquercio et al. [2018]). They developed DroNet, a forked Convolutional Neural Network (CNN) that predicts, from a single gray-scale image, a steering angle and a collision probability. In other words, the model learns to steer and avoid obstacles from forward-looking videos recorded by cars and bikes while driving in real contexts. In this case, both the prediction and controller tasks were powered off-drone on a dedicated computer, remotely connected through WiFi.

A year later, ETH Zürich was able to develop PULP-DroNet, porting the CNN on the Crazyflie[4], a nano-drone with a size of only 3 3×3 centimeters for a weight of 27 grams. They propose a general methodology for deploying on-board deep learning algorithms for ultra-low-power devices Palossi et al. [2019b], without any need for an external laptop to run the software.

Inspired by PULP-DroNet, IDSIA adapted its ProximityNet to work on-board the Crazyflie with excellent results Zimmerman [2020]. The nano-drone is able to achieve good quantitative and qualitative performance, regardless of any problem deriving from working with such low-end devices. The main challenges are represented by low computational power, energy consumption management, and low-fidelity camera with no video stabilization[5].

[3] `https://github.com/idsia-robotics/proximity-quadrotor-learning`

[4] `https://www.bitcraze.io/products/crazyflie-2-1/`

[5] Himax HM01B0 camera, able to produce 320×320 megapixel (MP) at 60 FPS. However,

2.4 Network Interpretability

Deep Neural Network (NN)s learn abstract representations for finding a logical mapping between their input and output, determined by well-defined mathematical computations that involve the input itself and the progressively learned network parameters. Inspired by biological brains, this approach seems to be incredibly effective on a huge variety of tasks.

Unlike other Machine Learning (ML) techniques, NN are known to produce "black-box" models, particularly hard to understand even from domain experts. Their reasoning and comprehension are intrinsic in the network parameters, which are nothing but numbers.

However, when working with real-world problems, it is extremely important to be able to explain what a ML model is actually understanding. This is crucial for building trust in algorithms and to be sure there are no undesirable biases in the models, which could raise serious problems especially in critical fields such as medicine and law.

Explainable AI (XAI) is the field of study which tries to make ML results, and their underlying basis for decision-making, properly understandable to humans (Wikipedia [2021]).

Regarding XAI for CNNs, researchers have developed many techniques for understanding what a NNs actually care of when producing an output based on an input image. Main efforts regard feature visualization and attribution, but recent advanced studies have also shown how these methods can be used altogether (Olah et al. [2018]).

This section briefly explains these two major areas for CNN interpretability, with a particular focus on spatial attribution, the chosen methodology for our work.

2.4.1 Feature Visualization

Sources: Olah et al. [2017]

2.4.2 Spatial Attribution with GradCAM

Sources: Selvaraju et al. [2019], Chetoui [2019]

the frame rate is incredibly reduced during data collection due to platform limitation for image transfer.

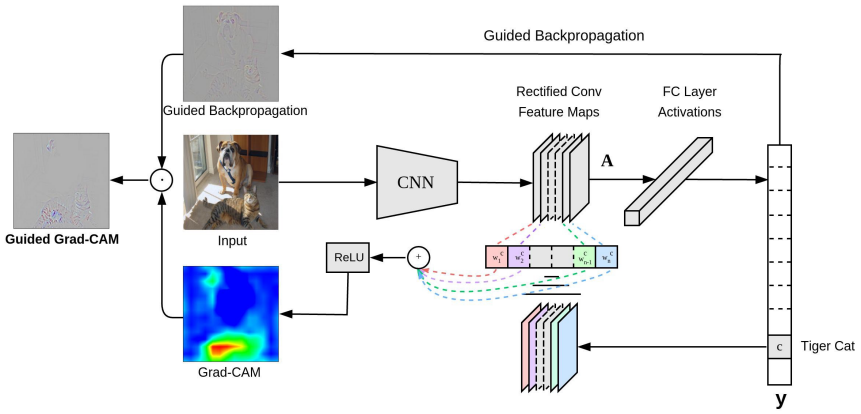


Figure 2.1: Grad-CAM schematic functioning (Selvaraju et al. [2019])

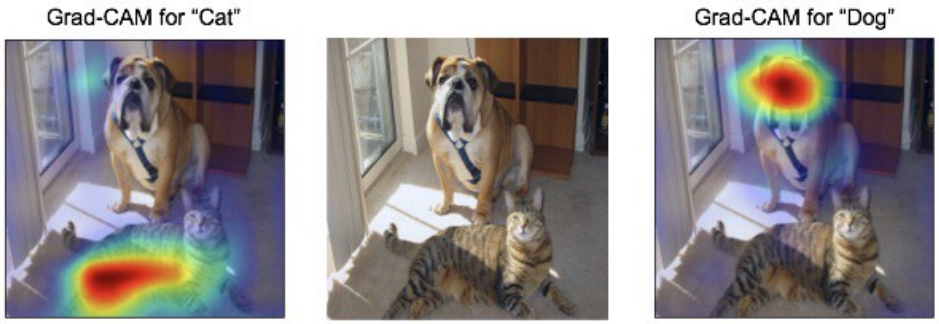


Figure 2.2: Grad-CAM example on dog-cat classification (Selvaraju et al. [2019])

2.5 Network Generalization

2.5.1 Data Augmentation

2.5.2 Domain Randomization

TODO

Summary from Mehta et al. [2019]

See Lilian [2019b]

"Domain randomization is a popular technique for improving domain transfer, often used in a zero-shot setting when the target domain is unknown or cannot easily be used for training. In this work, we empirically examine the effects of domain randomization on agent generalization. Our experiments show that domain randomization may lead to suboptimal, high-variance policies, which we attribute to the uniform sampling of environment parameters."

2.5.2.1 Virtual Simulation

Imitation learning through simulation is recently becoming an interesting and successful approach for both reinforcement learning (Hussein et al. [2017b] and Hussein et al. [2018]) object recognition (Tobin et al. [2017], Lilian [2019a]).

Robot and environment can be replicated through a dedicated simulator such as Gazebo[6], often used in robotics with Robot Operating System (ROS)[7] due to its straightforward integration, or even with general-purpose graphic engines. Unreal Engine[8] Unity[9] are well-known simulators designed for game-development, but recently used for Virtual Reality (VR) and Augmented Reality (AR) applications. They give developers unlimited possibilities, carefully supported by solid physics engines and active communities.

Given the difficulty of collecting data for our task, exploring the possibility of simulating the entire scenario in a 3D virtual-world is intriguing, especially to replace the need for a complex motion capture (MoCap) system. Integrating odometry support, drone and people can be thoroughly modeled to act as in the real world, with similar movements and sensing capabilities, in order to collect the data very efficiently. The virtual simulation gives both the opportunity of reproducing real indoor/outdoor scenes, but also randomizing the background with artificially generated textures.

Even though the approach appears to obtain sub-optimal results, a complete and adaptable implementation requires a lot of effort, yet unlocking a huge number of possibilities. Considering the consistent amount of fine details to consider and issues that can arise during the development of such simulators, we opt instead to work with an easier generalization pipeline, that mostly concerns machine learning only.

2.5.2.2 Background Replacement with Mask R-CNN

TODO

Previous works on images: Yue et al. [2019], Takahashi et al. [2020]
MaskRCNN: He et al. [2018], Arasanipalai [2018], ArcGIS [2021]

[6] `http://gazebosim.org/`
[7] `https://www.ros.org/`
[8] `https://www.unrealengine.com/en-US/`
[9] `https://unity.com/`

Chapter 3

System Description

This chapter aims to provide a generic view of our system.

First, we briefly describe the existing environment, its main components, and how they interact for flying and controlling the drone. Next, we explain network architecture and data used for the machine learning model, able to predict the user's pose given an image. Finally, we list tools, software, and libraries to achieve the goal of making the drone able to fly in any other environment.

3.1 Environment

Since we mainly focus on improving the ProximityNet model mentioned in section 2.3.2, we need to deeply understand the environment in which the original research has been conducted, physically located at the Swiss AI Lab Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA) in Lugano.

3.1.1 Parrot Bebop Drone 2

Figure 3.1: Parrot Bebop Drone 2

The entire work is built around the Parrot Bebop Drone 2 (figure 3.1), a lightweight drone (500 grams) with a size of $382 \times 328 \times 89$ millimeters. A 2700 mAh swappable battery gives power to four brushless engines and a dual-core processor with a

quad-core GPU, for a maximum flight time of 25 minutes. Connectivity is provided through 2.4 GHz 802.11a/b/n/ac Wi-Fi that enables remote control via mobile app or Parrot Skycontroller (up to a distance of 2km).

The drone is equipped with many simultaneous sensors to compute velocities, orientation, altitude, attitude, and GPS coordinates to ensure maximum stability during the whole flight. However, for this project, we mainly care about its camera, able to shoot 14 megapixel (MP) photos and record Full HD 1080p videos at 30 frames per second (FPS). Even though the original field of view (FOV) is 180°, raw camera images pass through a software stabilization that produces 16:9 images with a horizontal FOV of 90°. The 3-axis digital stabilization technique implemented by Parrot is able to compensate for the drone's pitch and roll, in order to provide correct-oriented horizontal images and stable videos regardless of the drone's movements. Full specifications provided by the official Parrot Documentation [2015].

3.1.2 OptiTrack

For tracking drone's movement, a motion capture (MoCap) system is required, able to record 3D coordinate of objects and people in space. The technique is widely used for motion tracking in a large variety of fields such as film making and animation, virtual reality, sport, medicine, and even military. A common way to implement a MoCap systems is by using special cameras placed around the area to be tracked, able to collect optical signals from passive[1] or active markers[2] inside the area.

IDSIA adopt OptiTrack, which is producing real-time MoCap systems since 1996 and are today world's choice for low-latency and high-precision 6 degrees of freedom (DoF) tracking for ground and aerial robotics, both indoor and outdoor. Full documentation is available on the OptiTrack Website.

3.1.3 Drone Arena

At IDSIA, a dedicated room has been equipped with an OptiTrack MoCap system composed by 12 OptiTrack Prime^{x}13 infrared (IR) cameras for medium-sized areas (figure 3.3a, 1.3 MP, 240 FPS, ± 0.20 mm 3D accuracy in a 9×9 meters area with 14mm marker), to track movements of passive markers placed on the person's head facing the drone and on the drone itself. Schematic and actual representation of the arena are shown in figures 3.2 and 3.3b. Such composition is able to track a theoretical number of 18 drones inside an available area of 6×6 meters (here surrounded by a safety net), with a virtual fence of 4.8×4.8 meters which virtually constraints the total area in which the drone is allowed to fly.

[1] a passive marker reflect light

[2] an active marker emits its own light

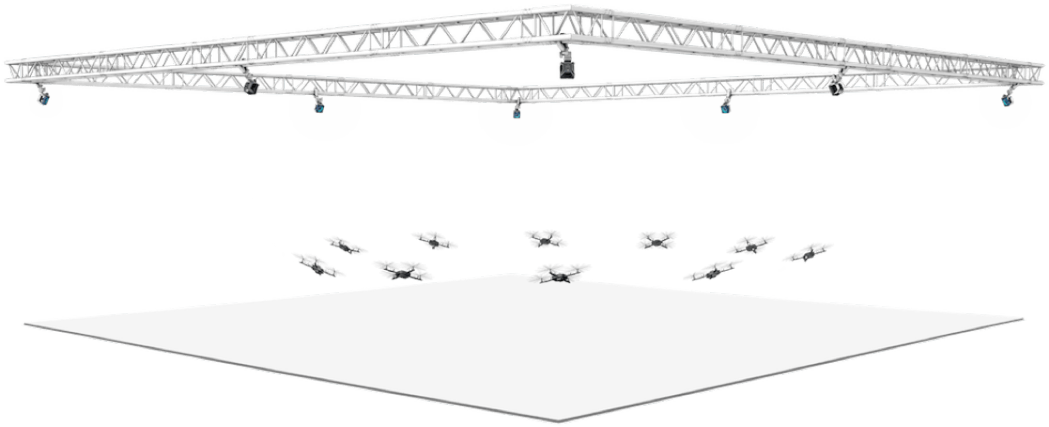
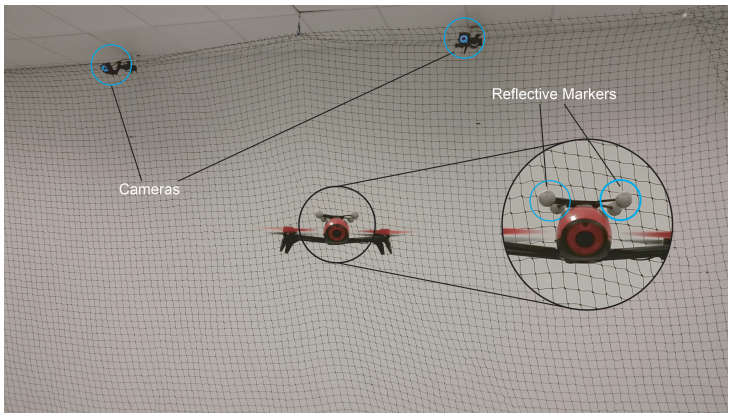


Figure 3.2: Schematic OptiTrack system with 12 OptiTrack Prime cameras

(a) Prime^{x}13 camera



(b) Drone with passive markers (Mantegazza [2018])

Figure 3.3: Drone arena at IDSIA

3.1.4 Robot Operating System (ROS)

ROS is an open-source robotics middleware suite of software libraries and tools for building distributed and modular robot applications. It provides hardware abstraction and orchestration, implementation of commonly-used functionality, message-passing between processes, and package management. ROS organizes its components in a graph architecture composed of nodes that communicates via a publish/-subscribe mechanism, supporting a wide variety of robots also used for education. The main client library is available in C++, Python, and Lisp.

Some of the most important ROS features include Standard Message Definitions, Robot Geometry and Description, Remote Procedure Calls, Diagnostics, Pose Estimation, Localization, Mapping, and Navigation. It also provides additional tools, such as Rviz (3D visualization of robots and various types of sensor data) and

Gazebo (3D indoor and outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine).

ROS has grown to include a large community of active users worldwide. Historically, the majority of the users were in research labs, but increasingly we are seeing adoption in the commercial sector, particularly in industrial and service robotics.

Further documentation is available on the official ROS Website.

3.1.5 Control & Data collection

Inside the arena, the drone is controlled by a ROS script which relies on the user's pose with respect to (wrt) the drone - from now on, the *target pose* (i.e., the pose of the user seen by the drone reference frame) - to compute acceleration commands for making the drone hover in front of the person, in the direction of the head at a predefined 1.5 meters distance.

During data collection, both user's and drone's poses are deduced by the OptiTrack system by using proper markers placed on the drone and on the person's head, as shown in picture 3.4. The target poses over time, mathematically computed by the script, are accurately synchronized with the video stream coming from the front-facing camera, and saved into `rosbag` files.

Figure 3.4: Markers placed on top of drone and user's head (Mantegazza [2018])

Data collected into the drone arena have been used to build the dataset for training a machine learning model, which should be able to infer the target pose by seeing a picture taken by the drone's camera. Figure 3.5 shows an illustration of the system from a bird-eye view.

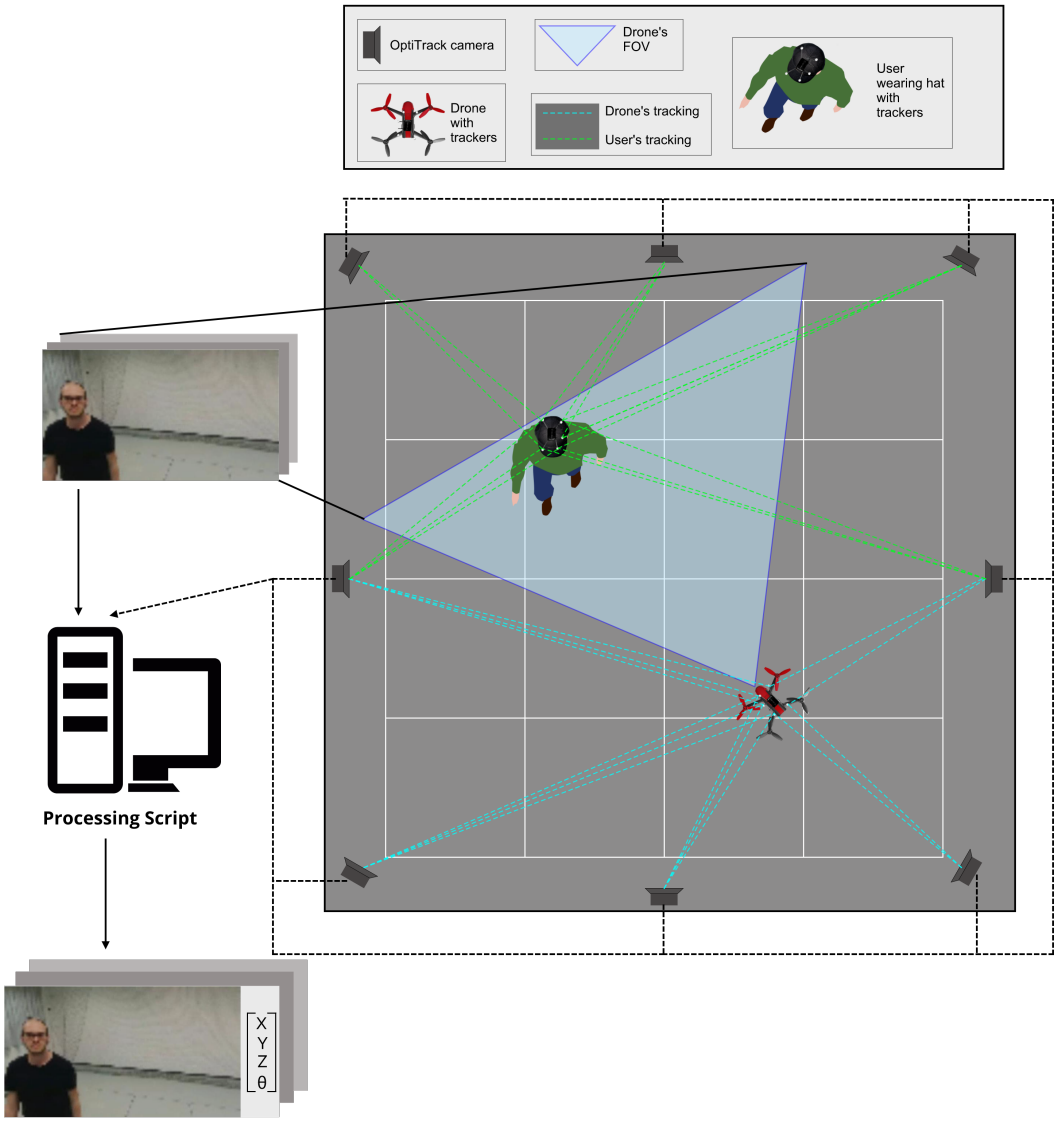


Figure 3.5: OptiTrack and data collection illustration (Mantegazza [2018])

3.2 Model

Our entire work is based on the work introduced in section 2.3.2, whose basic working and environment have been presented before. This section further inspects dataset composition, network architecture and model performance as declared in Mantegazza et al. [2019].

3.2.1 Dataset

Data have been entirely collected in the dedicated drone arena located at IDSIA. A good dataset should ideally provide images from various scenarios, but such kind of data is not easy to record since the ground truth must be given by a complex and expensive MoCap system, particularly difficult to be moved and reassembled outdoor.

For building both the training and the testing set, several flight sessions have been recorded using an omniscient controller, driving the drone towards the user's pose inferred by the OptiTrack. The dataset contains a total of 13 different people which differs both in physical characteristics and outfit, moving in different ways under various (artificial) light conditions. Many objects are present in the background of recorded images, and some experiments involve more than one person in front of the drone[3]. In total, 45 minutes of usable videos were used to compose the dataset, which counts about 63'000 and 11'000 frames respectively for training and testing sets.

A complete overview of images present in the training set is shown in figure 3.7. Please notice that a few frames in the dataset are affected by digital artifacts, mainly caused by connection issues during video recording (figure 3.6); also, there are frames in which no person is present at all, because of particular movements sequences during which the drone actually lose the user (figure 3.8).

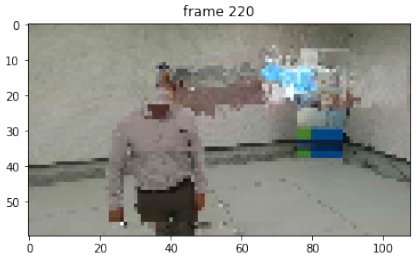


Figure 3.6: A frame with digital artifact caused by connection issues

[3] anyway, the drone always had to follow the nearest user (properly equipped with OptiTrack markers)

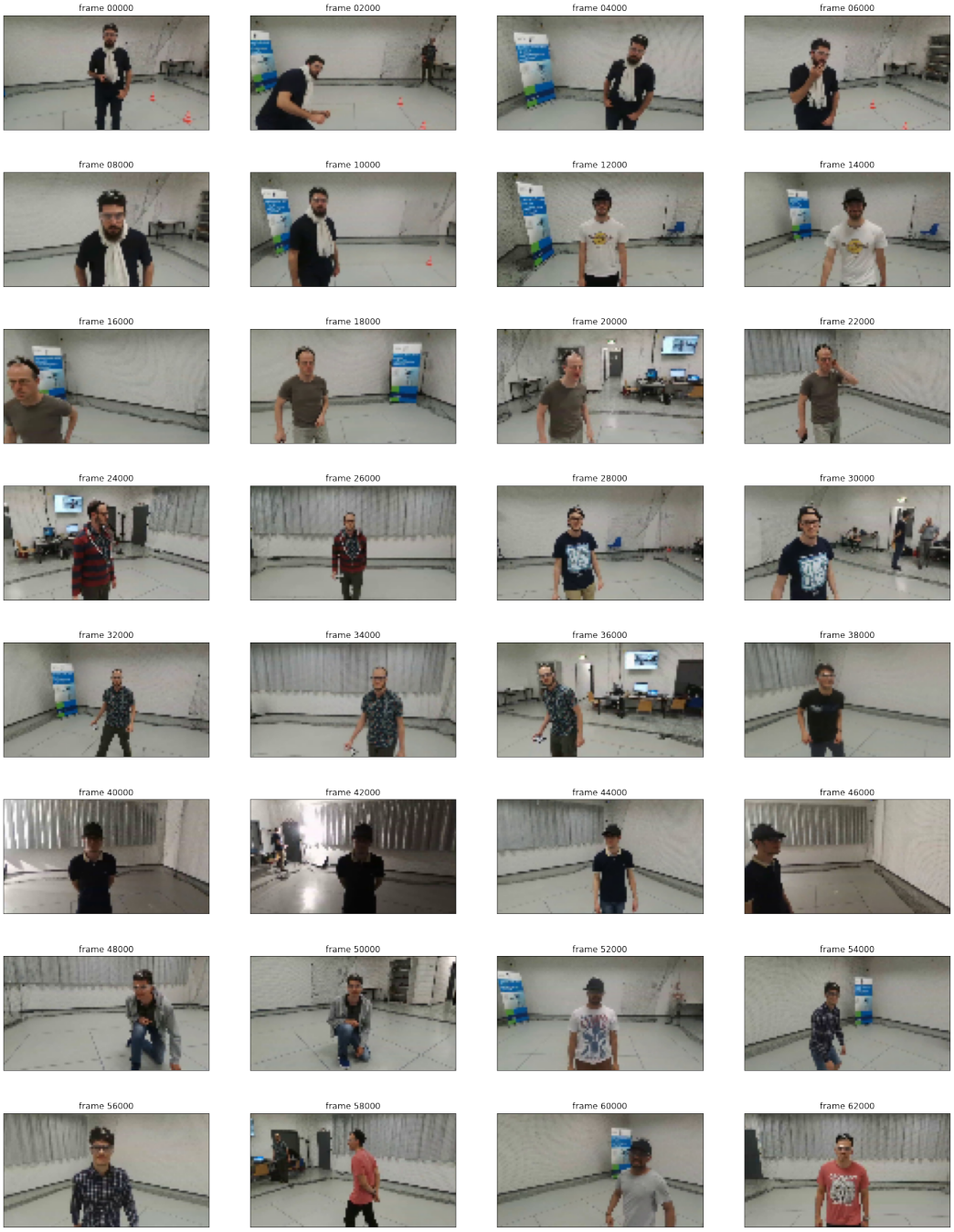


Figure 3.7: A complete overview of images in the training set

Figure 3.8: A movements sequence which led to images with no person presents

The ground truth is represented by the four variables, associated with each captured image, that explain the user's pose wrt the drone. Their interpretation is described here:

- `X`, stays for the distance of the user from the drone and affects the pitch (acceleration along the X-axis); if the user is at the correct distance in front of the drone, this variable will be equal to 1.5
- `Y`, represents the horizontal alignment of the user in front of the drone and affects the roll (acceleration along the Y-axis); if the user is horizontally centered in front of the drone, this variable will be equal to 0
- `Z`, represents the vertical alignment of the user in front of the drone and affects the velocity along the Z-axis; if the user is vertically centered in front of the drone, this variable will be equal to 0
- `W`, represents the angle created between head's pointing direction and drone position, is influenced by head orientation, and affects the yaw (angular velocity around the Z-axis); if the user is perfectly facing the drone, this variable will be equal to 0

From the distribution of the variables in the training set, shown in figure 3.9, we notice that most of the time the user is somehow centered in the image, which is an effect caused by the ROS controller based on known poses. The variation of the variables is affected by the user's movements in space, the more sudden they are, the greater the deviation.

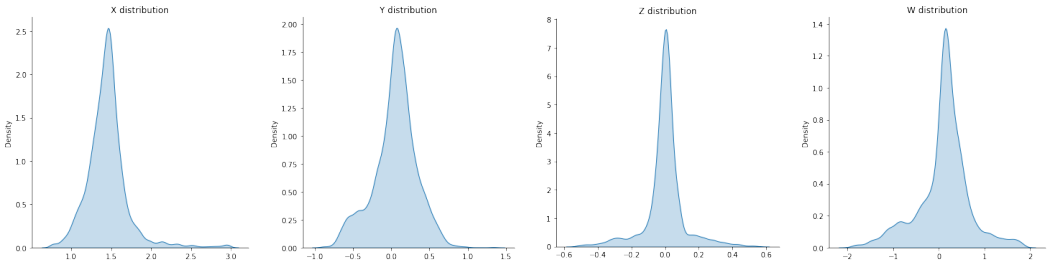


Figure 3.9: Target variables distribution for the regression task

3.2.2 Architecture

As perfectly suited for working with images, the proposed network is a Convolutional Neural Network (CNN) later improved by the author for being a Residual Neural Network (ResNet). The network is composed of a total of 1'332'484 trainable parameters, accepts a single 60×108 pixels image in input, and outputs 4 regression variables that correspond to the user's pose coordinates.

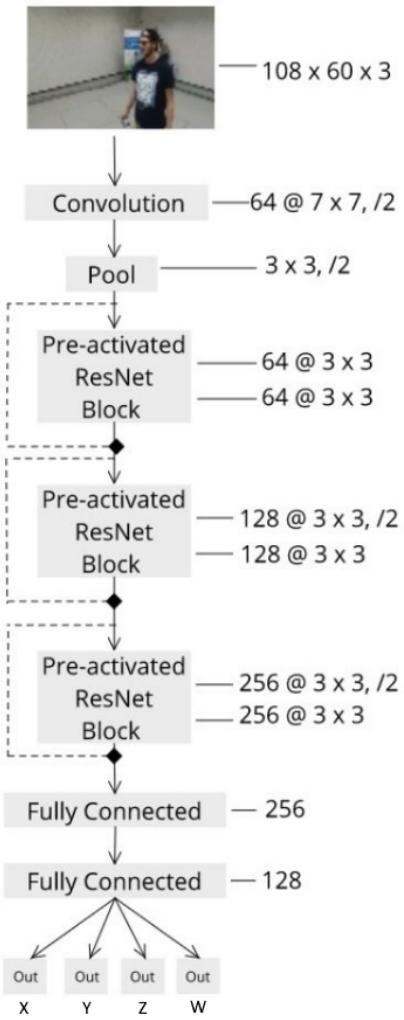


Figure 3.10: Schematic ProximityNet architecture (Mantegazza et al. [2019])

Figure 3.10 provides an illustration of the architecture, while a complete list of all layers is available in figures A.2 and A.2 of the appendix A.1. Each ResNet block is provided with batch normalization, ReLU activations (for info, Brownlee [2019]) are used for all layers except for the output neurons, which are associated with a linear activation function (for info, Z^2 Little [2020]).

3.2.3 Performance

In the original paper, ProximityNet is trained using the Mean Absolute Error (MAE) loss function with the Adaptive Moment Estimation (ADAM) optimizer (for info, Kingma and Ba [2014]) and a base learning rate of 0.001, progressively reduced on validation loss plateaus that last more than 5 epochs. A maximum of 200 epochs are run in total, but with an early stopping policy with patience of 10 epochs on the validation loss.

Performance is evaluated both quantitatively and qualitatively on the end-to-end model, rather than the mediated one considered for our work. However, as explained in section 2.3.2, both approaches obtain similar results so we can consider the following details to be valid also for the mediated approach.

For quantitative evaluation, the chosen metric is R^2[4], which has an interval of $[-\inf, 1]$ where 1 represents the optimality.

The author also conducts an experiment about the minimum cardinality of the dataset for obtaining acceptable performance. Results are available in figure 3.11, directly taken from the paper. As shown, decent performance requires at least 5'000 samples and keep improving as their number increases.

Specifically, predictions seem more accurate for variables Z and W with a R^2 score of 0.82 and 0.88, respectively. Different the findings for X and Y which only reach a R^2 of 0.59 and 0.57, respectively.

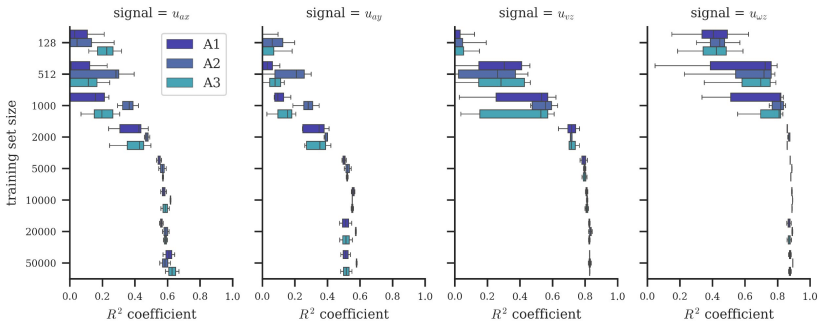


Figure 3.11: ProximityNet R^2 results from Mantegazza et al. [2019]

The previous considerations on the variables are confirmed by the qualitative evaluation, obtained by comparing ground truth and predictions during a short simulation. Figure 3.12[5] shows that X and Y predictions are considerably worse than results achieved by Z and W when compared with the ground truth.

[4] R^2 interpretation will be explained in the evaluation chapter 6

[5] A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results

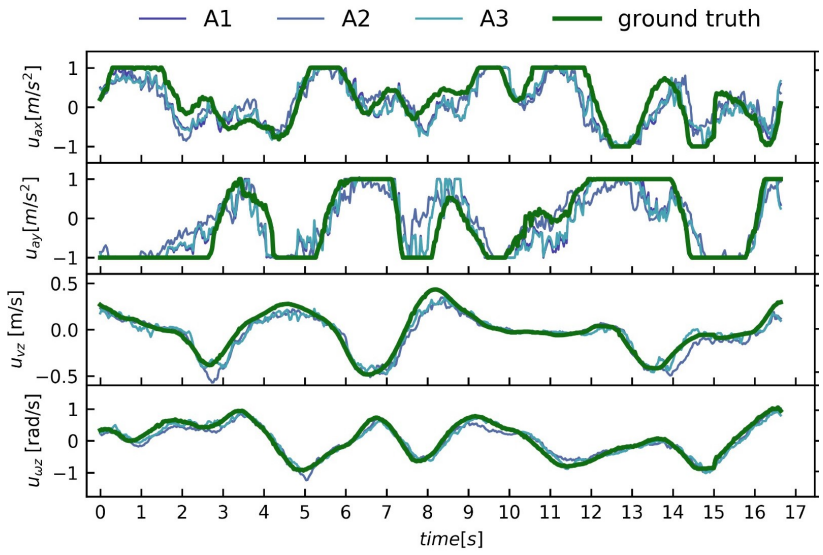


Figure 3.12: ProximityNen GT vs prediction results from Mantegazza et al. [2019]

3.2.4 Generalization

Even though the ProximityNet achieved quite good performance on the test set, its behavior must be proven on the real drone to certify the model usability.

Mantegazza et al. [2019] reports experiments conducted inside the arena by flying the drone without the MoCap system, only relying on the learned model for computing the user's pose. The outcome is incredibly good, with the drone actually performing its task without any issues. Figure 3.13 presents trajectories followed by the drone during five consecutive trials (with two different models) in which the quadrotor has to face a user initially rotated by 90 degrees. Although the paths are sometimes different from what designed by the omniscient controller (the ground truth), they are still reasonable and flying capabilities in the arena seem very promising.

Finally, we consider model performance in unknown environments, possibly outdoor. The official paper does not talk about the topic but during a direct discussion with the author, we discover that flying performance outside of the drone arena was not consistent with usual model behavior. Accordingly to this, it seems that the model is not able to generalize the task when outside of the environment it already knows.

The goal of our work is to explore ways of improvement, which aim to generalize the model to make it able to theoretically predict the user's pose in any other unknown scenario. The next chapters firstly try to understand the main issues

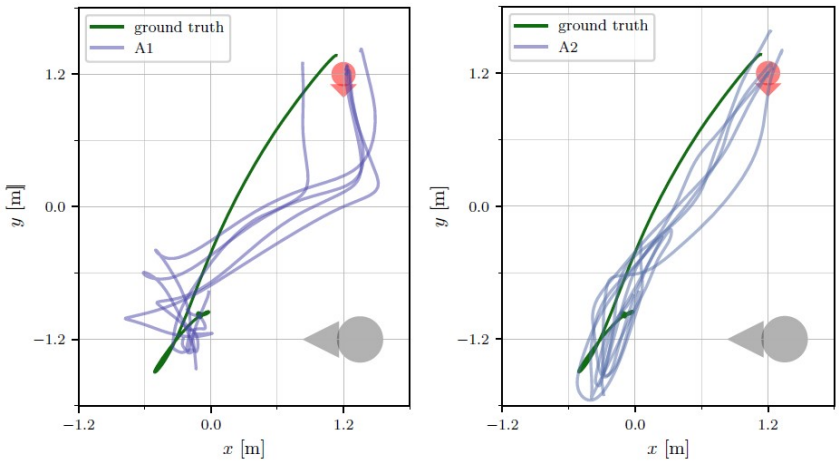


Figure 3.13: ProximityNet trajectories (Mantegazza et al. [2019]) for positioning in front of the user initially rotated by 90 degrees

and limitations of the model, then provide a possible solution for the generalization problem.

3.3 Development

Finally, this section presents tools and software used to conduct our research for improving the existing system from a machine learning point of view, according to the objective of the thesis.

3.3.1 Tools

The entire source code is written in Python 3. First experiments were carried out with Jupyter Notebooks via Google Colab on a GPU-accelerated runtime, while the final code is provided as classic Python scripts to be executable on a custom machine.

For debugging, we use a Windows 10 laptop equipped with an NVIDIA GeForce GTX 950M graphic card, while actual training is performed on a dedicated Ubuntu 18.04 workstation available at IDSIA mounting four NVIDIA GeForce RTX 2080 Ti [6].

3.3.2 Frameworks

The original work from Mantegazza et al. [2019] is written in Python and based on TensorFlow 1 and Keras. These libraries have been kept, but our project uses their

[6] multiple available GPUs are used for single-GPU computing, not simultaneously

updated versions for ease of use. The other main frameworks are listed below.

`Numpy` Largely used in the whole project for computation on arrays. Numpy is the fundamental package for scientific computing in Python, that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays.

`Pickle` Mainly used for saving and restoring Numpy arrays. The pickle module implements binary protocols for serializing and de-serializing a Python object structure.

`Matplotlib` The first choice for building charts, visualize images or any kind of figure. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Its `pyplot` module is inspired by MATLAB.

`OpenCV` Mainly used for efficient image/video manipulation and visualization, together with Matplotlib. OpenCV is an open-source library that includes several hundreds of computer vision algorithms.

`TensorFlow 2` All the project strongly relies on TensorFlow (TF) from start to end: network interpretation, person masking, training, and quantitative evaluation. Created by the Google Brain team, TensorFlow is an open-source library for numerical computation and large-scale machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. In version 2, it introduces a lot of comforts for easier development with a less steep learning curve.

`Keras` Used for defining the network architecture, training, and evaluating the model. Keras is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

`TensorBoard` Used with TensorFlow to precisely profile data generator performance for optimizing training time on the GPU. TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow. It enables tracking experiment metrics, visualizing the model graph, and much more.

`Sklearn` Only used for automatically computing some evaluation metrics, Sklearn is a simple and efficient tool for predictive data analysis reusable in various contexts built on NumPy, SciPy, and Matplotlib.

`tf-keras-vis` Used for applying GradCAM and other interpretability techniques. Open-source library for network interpretation, available on GitHub thanks to Kubota [2020]. Derived from the original keras-vis (Google [2020]) high-level toolkit for visualizing and debugging trained Keras neural network models.

`akTwelve Mask_RCNN` Used for human detection and segmentation in background replacement. An open-source implementation of Mask R-CNN on Python 3, Keras, and TensorFlow 2 is available on GitHub thanks to Kelly [2020]. The model generates bounding boxes, segmentation masks, and categorization labels for each instance of an object in the image.

Chapter 4

Experiments & Solution Design

This chapter describes the interpretation of the existing model for highlighting the main issues behind the lack of its generalization capabilities. Then, we propose a solution and present initial experiments on its feasibility.

4.1 Model Interpretation

Convolutional Neural Networks (CNN) are known to be suited for computer vision, because of their ability to gain spacial-related insights from images. Anyway, just like for any other Neural Network (NN), Convolutional Neural Network (CNN)s are "black-box" thus their internal behavior is particularly challenging for humans to understand.

In section 3.2.4, we discussed insufficient experimental results obtained by ProximityNet in predicting the user's pose in an unknown environment, thus outside of the drone arena. To find a solution to this problem, we first need to understand what the model is actually learning.

4.1.1 Applying GradCAM

Among network interpretability techniques introduced in section 2.4, we choose Gradient-weighted Class Activation Mapping (Grad-CAM) because it is indeed the most understandable way of visualizing what a model is actually seeing.

As explained in section 2.4.2, Grad-CAM is able to effectively visualize the most important parts of an input image which are responsible for predicting a certain output[1].

[1] for an easy understandable Grad-CAM example, please refer to the figure 2.2 which regards a simple dogs VS cats classifier

4.1.2 Regression to Classification

Grad-CAM is designed to be applied on classification tasks, rather than regression ones. Even though a porting of the algorithm for regression has been published and called Regression Activation Map (Wang and Yang [2019])[2], it appears to be an isolated case since the research in the field it is not developed yet. For this reason and only for network interpretation, we decide to transform our problem into a classification task.

Since the ground truth is composed of four variables with specific domains. Figure 3.9 in the previous chapter shows variables distribution, from which we can see that all variables in the dataset mainly lie on their "center" value because the user is mostly centered into the image. We decide to split continuous values into 3 different classes, which account for values smaller, around, and higher than the "center". We call these buckets respectively `low`, `medium`, and `high`.

- `X` values are splitted at 1.4 and 1.6
- `Y` values are splitted at -0.15 and $+0.15$
- `Z` values are splitted at -0.05 and $+0.05$
- `W` values are splitted at -0.20 and $+0.20$

So, for example, `X` values greater than 1.6 will be classified as `high`, while `Z` values between -0.05 and $+0.05$ will be classified as `medium`. These specific intervals have been manually defined for obtaining a good class distribution over the training set (figure 4.1). Please notice that this class partition is fundamental for understanding some Grad-CAM visualizations later.

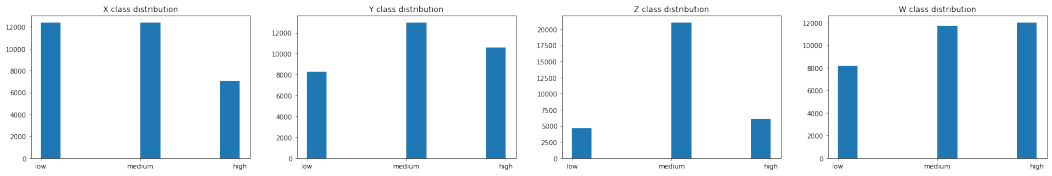


Figure 4.1: Target variables distribution for the classification task

4.1.3 Re-training

Considering the new ground truth, we define a new model architecture by replacing regression outputs with classification ones. We perform a re-training on the new model, by using the `categorical_crossentropy` loss and the `accuracy` metric, both

[2]open-source code available at `https://github.com/cauchyturing/kaggle_diabetic_RAM`

suited for multi-class models. We use the ADAM optimizer and a base learning rate of 0.001, progressively reduced on validation loss plateaus.

Results are shown in figure 4.2, which after 30 epochs shows a loss slightly smaller than 1, both for training and validation, and accuracy over the 80% for all variables. These values are not ideal for gaining some sort of conclusion but have instead been used for comparing different training experiments conducted on the new classification model.

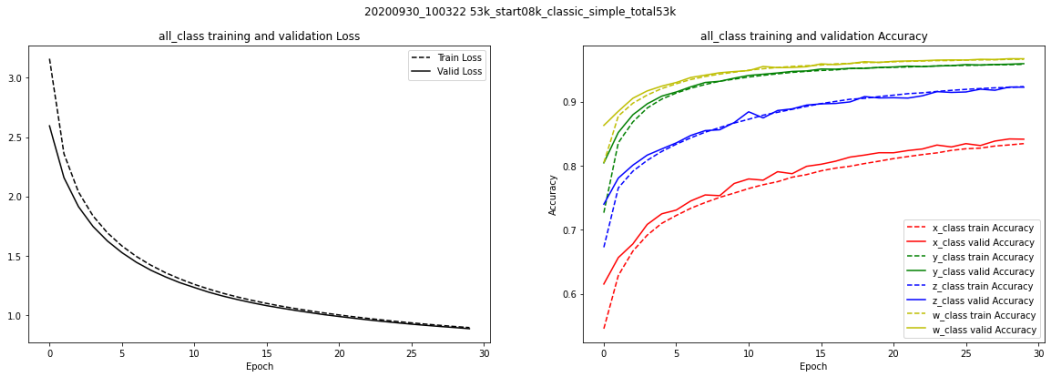


Figure 4.2: Grad-CAM: loss and accuracy of the new classification model

4.1.4 Framework

For practically applying Grad-CAM to the Keras trained model, we use the open-source `tf-keras-vis` available on GitHub[3] by Kubota [2020], porting for TensorFlow 2.0+ of the most famous `keras-vis` by Google [2020].

Finding the right library for this purpose has not been easy, since research is still on-going and most of the available resources are for TensorFlow 1 only. The most famous and powerful is `Lucid`[4], from the official TensorFlow team.

4.1.5 Reading charts

A proper understanding of this section requires a thorough ability on reading the following charts. Basic knowledge on how the related library works would also be helpful[5].

Grad-CAM application requires to specify the class for which we want to compute the activation mapping, which in our case would be one of the 3 introduced in section 4.1.2: `low`, `medium` or `high`.

[3] `https://github.com/keisen/tf-keras-vis`

[4] `https://github.com/tensorflow/lucid`

[5] Tutorial:`https://github.com/keisen/tf-keras-vis/blob/8f83773520069367902becc0a668dda90ab76349/examples/attentions.ipynb`

When working on standard classification problems, things are pretty easy: if we classify animals, we indicate Grad-CAM a specific animal class (e.g., `lion`), and the algorithm will provide a heatmap that overlay the portion of the image which is mainly associated with that animal (e.g., hopefully, the lion will be highlighted if it is present in the image).

However, our ML does not look like a standard classification problem, instead, it has been adapted from regression. Classes only serve as categorical values for variables that are actually numerical; also, does not exist a specific portion for each input image that can be associated with one of the three classes in particular. In other words, considering the class `low`, we do not expect its related heatmap to be different from the one generated for the class `high`: our discriminator is always the person, and its position in the image.

Moreover, ProximityNet predicts four different variables, then also the classification model will have multiple outputs. This further introduces another difficulty on reading Grad-CAM results, since we expect - once again - that heatmaps will only highlight the person in the image, regardless of the inspected variable.

Finally, in some cases, the network predicts the right class (coherent with the ground truth (GT)), while in other cases the outputs are incorrect. When examining wrong predictions, both the predicted and actual class could be taken into consideration for understanding what is going wrong with the model.

Of course, this assumption can be wrong, as the model could actually use other parts of the images (e.g., objects in the background) for actually determining classes to predict. However, this would be an undesired behavior since we expect that a correctly working NN will only care about the person and nothing else.

Figure 4.3 displays a typical full example on Grad-CAM application for each variable (`X, Y, Z, W`) and each class (`low, medium, high`). Heatmaps are not easy to interpret due to a large variety of parameters to consider. As a guideline, for each variable, you could only consider the column which corresponds to the actual (or predicted) class.

GT and predicted values are available in the right-most parenthesis, as "GT" and "PR" respectively. Rows define variables, while columns stand for the classes. It is clearly visible how no specific correlation is available between variables, classes, and computed predictions.

The last column is available as an average result of all classes, obtained by calling Grad-CAM without specifying a particular class to consider. The same reasoning can be also applied for variables, so that we are able to obtain also single meaningful images which represent Grad-CAM global average, for every variable and class at once (figure 4.4).

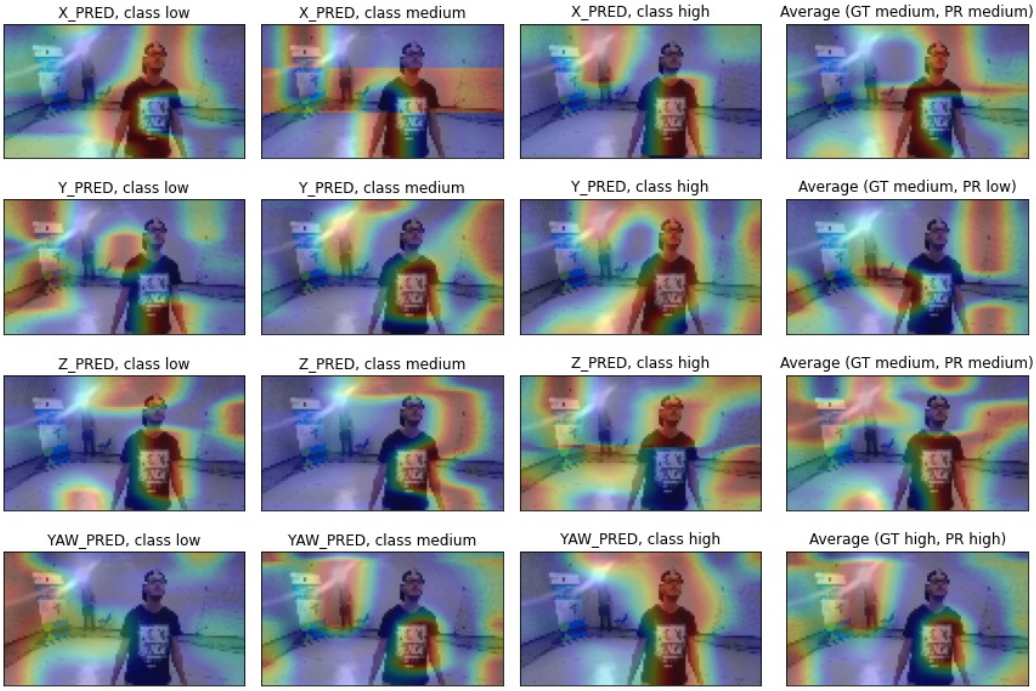


Figure 4.3: Grad-CAM: example of application for each variable and class

Figure 4.4: Grad-CAM: example of application on a global average

4.1.6 Results

As already shown in figure 4.3, it seems the network is not only considering the person in the frame for computing its output but instead relies on the whole image with particular attention on some spots.

From the previous section, we understand that reasoning with Grad-CAM heatmaps is not trivial, and separating visualization by variables and classes is not totally convenient when we can simply plot the global average Grad-CAM instead. For simplicity, this section will only focus on single-image results, while full Grad-CAM visualizations are available in the appendix A.2 for further inspection.

Reasonable detections

First of all, figure 4.5 displays examples of correctly working scenarios, in which the person is correctly detected from Grad-CAM. Three cases are observed during our studies: most of the times, the entire user is highlighted by Grad-CAM, while sometimes just the body or the head get major attention.

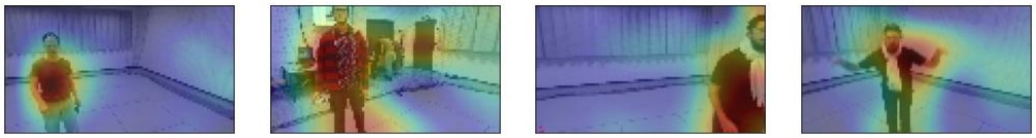


Figure 4.5: Grad-CAM: Correctly detected people

Such precise results are not the standard. In many cases, heatmaps are unstable, going in and out of the target person. The two frame sequences below fairly describe the usual behavior of the model seen by Grad-CAM (figures 4.6 and 4.7).

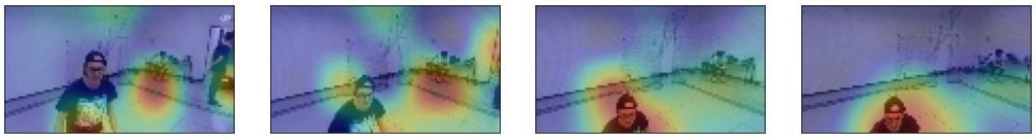


Figure 4.6: Grad-CAM: Sequence transitioning from wrong to correct detections

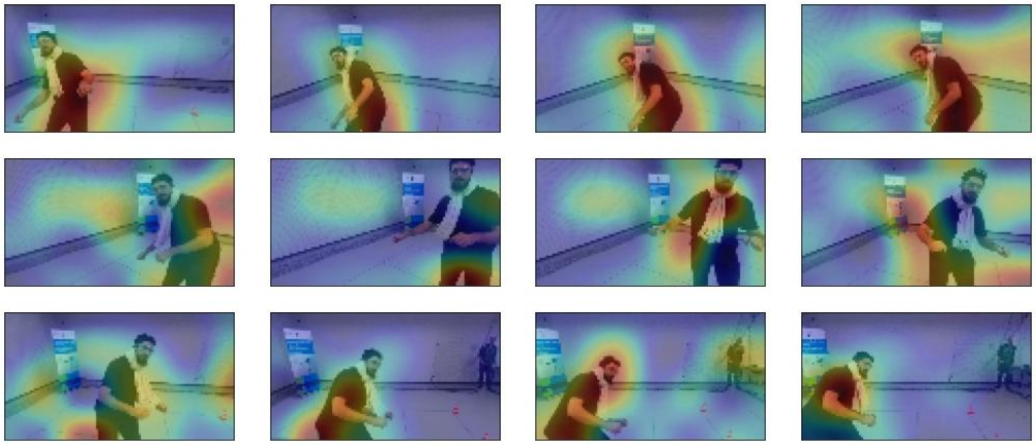


Figure 4.7: Grad-CAM: Sequence of unstable detections in and out of the person

Problematic detections

The examples presented above mostly reflect the model expected behavior. However, our network interpretation also reveals a lot of flaws in the prediction task. Grad-

CAM exhibits several situations in which the model output is affected by recurrent elements in the dataset.

- Objects in the background are prone to be considered important (figure 4.8)
- Curtains seem often particularly attractive (figure 4.9)
- Many parts of the room can easily distract the model, such as borders and baseboards or even blank spots on the walls (figure 4.10)
- When dealing with multiple people in front of the camera, sometimes not only the nearest person is considered (figure 4.11)
- Artificial glitches are sometimes ignored, sometimes distractive (figure 4.12)

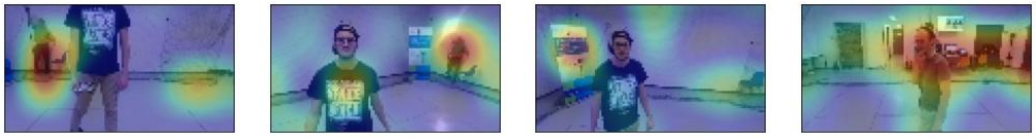


Figure 4.8: Grad-CAM: Objects in the background detected

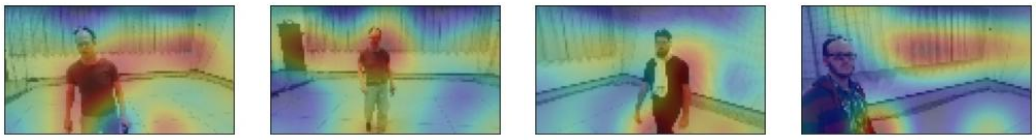


Figure 4.9: Grad-CAM: Curtains often distract the model

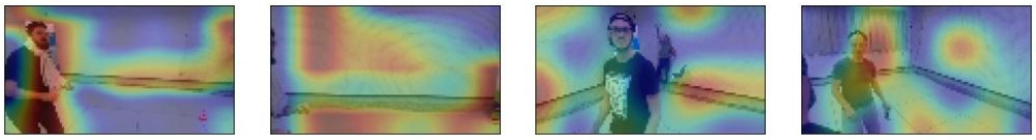


Figure 4.10: Grad-CAM: Model get easily distracted by various elements

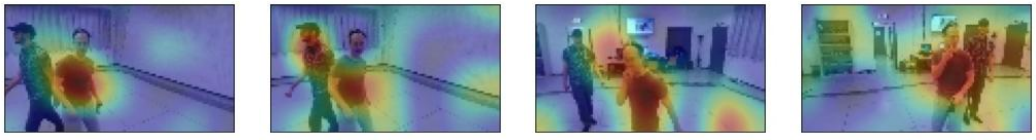


Figure 4.11: Grad-CAM: Detections when two people are present in the image

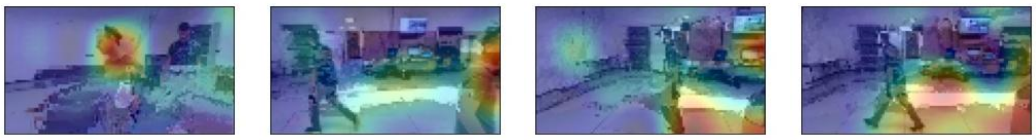


Figure 4.12: Grad-CAM: Model reactions to artificial glitches

4.1.7 Summary

Reported results demonstrate that the model is not robust enough to only focus on the user who is actually facing the drone's camera. Instead, various portions of the input images appear to be taken into consideration when the model makes predictions: many distractors are coming from the background.

In light of this, we can reasonably assume that the ResNet model has undesirably learned some details about the drone arena in which the dataset has been collected. This is most likely the reason why the model is unable to control the drone outside of that environment, as discussed in section 3.2.4.

4.2 Person Masking

From Grad-CAM results in the previous section, we conclude that the model is not capable of generalization. We have demonstrated that the main cause of the problem is inherent in the drone arena, thus we want to remove the room from the equation. We propose a solution that consists of performing advanced data augmentation by just keeping the person in the images, masking out the background to be randomly replaced with something else.

This section explores various algorithms and experiments for creating the mask of a person in an image, that ended with the final adoption of Mask R-CNN. This algorithm is used for image augmentation in the next chapters.

Before going deep into the tests with the available dataset, we discuss an alternative for achieving a similar result with a different methodology.

Chroma key

A known approach for implementing background replacement is the chroma key. Widely used in entertainment, it is a technique that makes use of colors in images and videos for splitting between actual content and background. Usually, a chroma key is achieved through a blue or green screen placed behind the subject, making sure that such color is not present in the foreground image. Then, a post-production software takes care of creating the appropriate mask which separates the two parts and enables background replacement.

Although this technique is particularly popular in many fields, placing several green screens into the drone arena is not the easiest task since it requires a lot of material and physical work to set up the proper environment, with possible issues related to the room composition or its illumination.

An experiment in this direction has been conducted years ago, with a portion of the arena walls covered by a green screen. Masking results were actually satisfying, but the limitations of such a small green screen are huge both in terms of user's movements and background coverage. In fact, figure 4.13 displays the setup, which reveals a lot of classic background still appearing in the images.

In addition, even with the capability of building a well-designed chroma key environment, the solution would be highly dependent on the geographical location of the setup. On the contrary, software-only approaches would be much more portable and reusable together with any other motion capture system in the world.

Figure 4.13: Experimental green screen setup in the drone arena

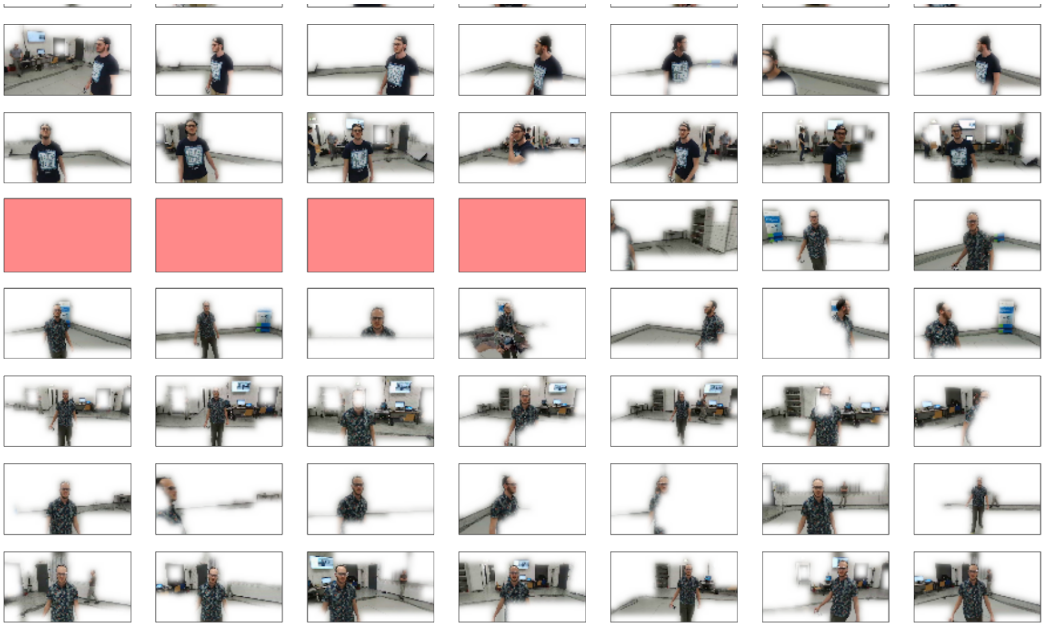


Figure 4.14: Canny edge detection overview on the training set

4.2.1 Canny

The first experiments are based on a classic computer vision technique called Canny Edge Detection. A custom algorithm[6] applies the related function from OpenCV[7] to find the edges inside the image, which are then used for also finding the contours. Only the biggest contour is taken into consideration for building a mask around the subject in the image.

A core aspect of the Canny function for finding appropriate contours is the choice of its parameters `minVal` and `maxVal`, used for distinguishing between *sure-edges*, *probable-edges* and *no-edges*. Several experiments have been made with different values, but no combination of the two parameters is optimal on our dataset.

Figure 4.14 shows what happens with `minVal` and `maxVal` respectively set to 100 and 400. Most of the time the person is well detected, while other times it completely disappears or even results in a fatal error (red frames). The room baseboard (the line between the floor and the wall) is often still present in the image, while many samples seem to preserve a huge portion of the background.

Figure 4.15: Canny edge enhanced algorithm demonstration

[6] adapted from `https://stackoverflow.com/a/29314286/10866825`

[7] `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html`

In some cases, it even happens that the body of the person is present in the image but its face disappears. For mitigating this problem, an enhanced version of the algorithm has been considered, designed to keep the person's face and body in the resulting image, assuming their positions are known. Figure 4.15 illustrates the problem and demonstrates that results obtained from the enhanced version are still not acceptable, since we are not able to appropriately mask the background away.

4.2.2 Grabcut

Given the Canny Edge Detection limits in removing the background from the images, the GrabCut algorithm has been tried. It operates using the subject position in the image and some statistical inference for labeling each pixel of the image as background or foreground. It works with the following algorithm (OpenCV Documentation):

- User inputs the rectangle. Everything outside this rectangle will be taken as sure background. Everything inside the rectangle is unknown. Similarly, any user input specifying foreground and background are considered as hard-labeling which won't change in the process.
- Computer does initial labeling depending on the data we gave. It labels the foreground and background pixels (or it hard-labels).
- From now on, a Gaussian Mixture Model (GMM) is used to model the foreground and background.
- Depending on the data we gave, GMM learns and creates a new pixel distribution. That is, the unknown pixels are labeled either *probable-foreground* or *probable-background* depending on their relationship with other hard-labeled pixels in terms of color statistics (like clustering).
- A graph is built from this pixel distribution. Nodes in the graphs are pixels. Additional two nodes are added, the Source node and the Sink node. Every foreground pixel is connected to the Source node and every background pixel is connected to the Sink node.
- The weights of edges connecting pixels to source or end nodes are defined by the probability of a pixel being foreground or background. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight.
- Then a min-cut algorithm is used to segment the graph. It cuts the graph into two, separating the source node and the sink node with a minimum cost

function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to the Source node become foreground and those connected to the Sink node become background.

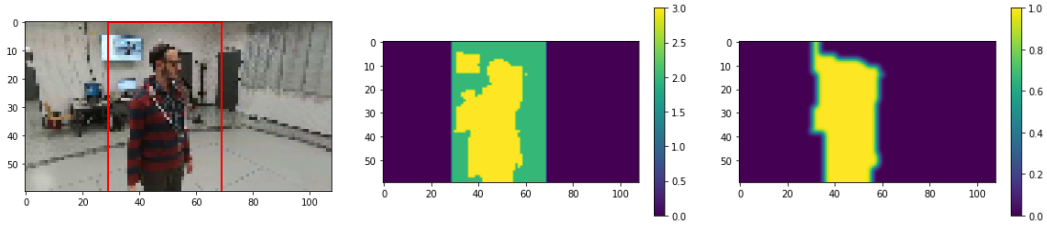
- The process is continued until the classification converges.

OpenCV GrabCut function[8] has two initialization modalities. You can only pass the rectangle, as described in the algorithm, or a mask of the image in which you specify whether a certain pixel is *sure-background*, *probable-background*, *probable-foreground* or *sure-foreground*. These classes[9] are also used by the library during the algorithm itself.

Both approaches require previous knowledge about the subject position in the image. For now, let's analyze both by manually providing this information for the example images. Later on, we will also consider an algorithm for automatic human detection.

4.2.2.1 Rectangle initialization

This approach requires that a rectangle, entirely containing the subject, is given in input to the function (figure 4.16a). GrabCut proceeds as follows. Area inside the rectangle is marked as *probable-background* (green), while the pixels outside are *sure-background* (blue). As the algorithm keeps going, it finds pixels inside the rectangle which can be foreground, marking them as *probable-foreground* (yellow) (figure 4.16b). Later, we binarize and smooth the mask (figure 4.16c) for finally removing the background from the original image.



(a) Initialization **(b)** Pixel classes **(c)** Final mask

Figure 4.16: Grabcut algorithm explained: rectangle initialization

Performance obtained by the algorithm is available in figure 4.17. Results seem better than the ones produced by Canny Edge Detection. However, it happens that the face or the entire person is filtered out of the image.

[8] https://docs.opencv.org/4.1.2/d7/d1b/group__imgproc__misc.html#ga909c1dda50efcbeaa3ce126be862b37f

[9] https://docs.opencv.org/master/d7/d1b/group__imgproc__misc.html#gad43d3e4208d3cf025d8304156b02ba38

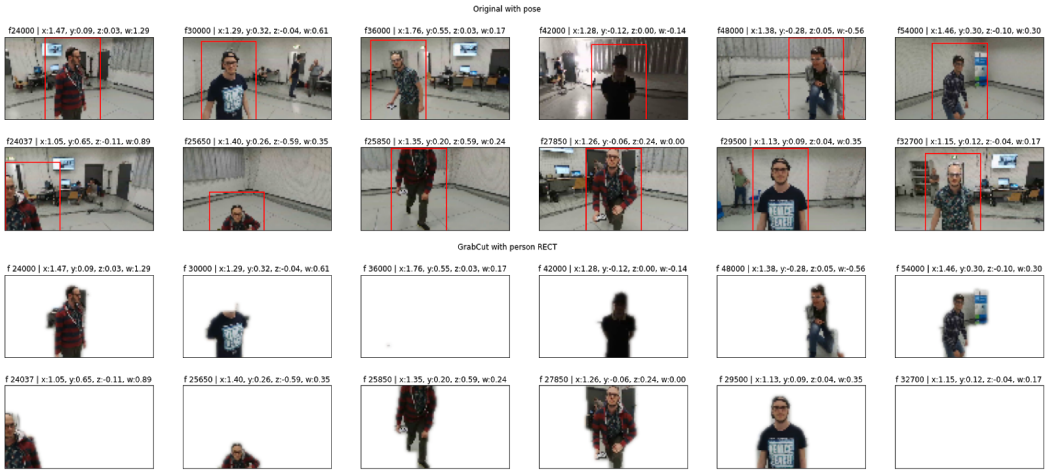
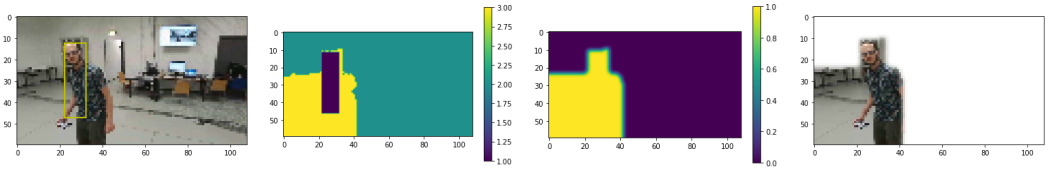


Figure 4.17: Grabcut demonstration: rectangle initialization

4.2.2.2 Mask initialization

In order to improve the previous technique, we try mask initialization, which allows us to initially classify the pixels in the image as we prefer. This time, we do consider the pose information for telling GrabCut we already know that some part of the image is *sure-foreground*: face and part of the body.

Let's now consider an example of an image for which the previous solution was completely missing the person in the result, despite the well-initialized rectangle. In figure **??**, we notice manually assigned *sure-foreground* pixels in blue, while GrabCut inferred *probable-foreground* in yellow and *probable-background* in green. Results are undoubtedly better, but we notice that the left-most background has been kept in the final image, while we could easily identify it as *sure-background* using the person pose we assume as known, like before.



(a) Initialization **(b)** Pixel classes **(c)** Final mask **(d)** Result

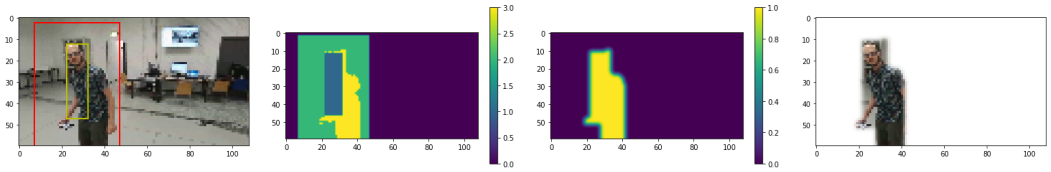
Figure 4.18: Grabcut algorithm explained: mask initialization

4.2.2.3 Hybrid initialization

We finally try a mixed approach between rectangle and mask initializations, by specifying both the person and face positions in the image. It allows us to initially set

sure-background, probable-background, and sure-foreground pixels. Only probable-foreground pixels have to be found by the GrabCut algorithm.

In figure 4.19 the usual explanation. Please notice that *sure-background* is dark blue, *probable-background* is green, *probable-foreground* is yellow and *sure-foreground* is light blue. Image 4.20 shows results of hybrid initialization applied to the same samples introduced in the figure 4.17. Performance is sub-optimal, as they present excellent segmentation capabilities.



(a) Initialization **(b)** Pixel classes **(c)** Final mask **(d)** Result

Figure 4.19: Grabcut algorithm explained: hybrid initialization

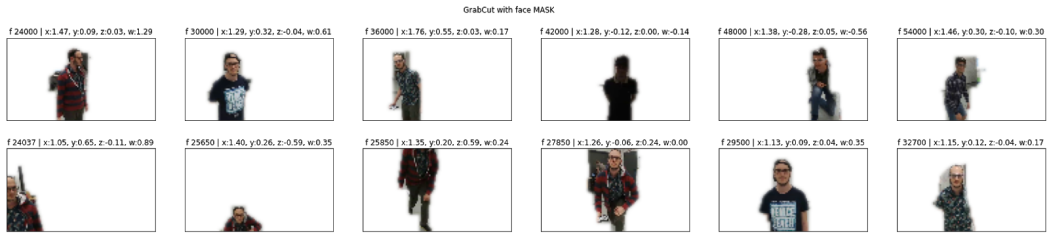


Figure 4.20: Grabcut demonstration: hybrid initialization

4.2.2.4 Automatic Human Detection

Now that we have a promising background removal algorithm, to let it work we need to infer person and face position from the image. GT data is not sufficiently precise for providing such information, so we try other state-of-the-art object detection techniques.

For properly using hybrid initialization, which demonstrated to be the most precise solution, we need two pieces of information: the bounding boxes associated with both the entire person and its head. For this reason, distinct detectors are necessary.

We adopt YOLO[10] for object detection, through the `cvlib` library that implements a YOLOv3 model trained on the COCO[11] dataset, capable of detecting 80 common objects in context. Underneath, it uses the OpenCV dnn module[12].

[10] `https://pjreddie.com/darknet/yolo/`
[11] `https://cocodataset.org/`
[12] `https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html`

A demo is shown in figure 4.21, where we notice that YOLO overall provides quite good results. However, in 19-20% of the cases, it does not detect any object in the image, most probably because of their low resolution.

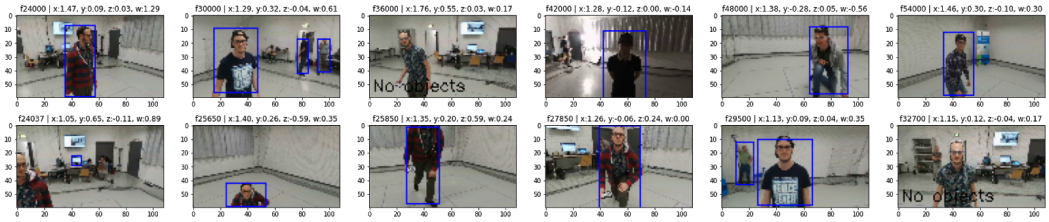


Figure 4.21: YOLO demonstration

For enabling mask initialization, also the face position is needed. We firstly try to heuristically infer its position based on the bounding box of the entire person provided by YOLO. Even though the majority of samples are compliant with this heuristic, a non-ignorable percentage of samples is not compatible. For this reason, we also try some head detectors[13], miserably failing in their task due to the small size of our images.

While searching for a solution compatible with such low-fidelity images, we find an all-in-one solution, presented in the next section, that immediately became our choice for its surprising results.

4.2.3 Mask R-CNN

Mask R-CNN (He et al. [2018]) is a state of the art deep learning framework for object detection and instance segmentation, whose technical details have been illustrated in section 2.5.2.2. Originally developed by Facebook researchers in PyTorch[14], now available in the `Detectron2`[15] package (Wu et al. [2019]), the algorithm has been ported to TensorFlow 1 (Abdulla [2017]) and later adapted for TensorFlow 2 by Kelly [2020][16]. The latter has been used for applying Mask R-CNN on our dataset images.

Results are incredibly precise and the method undoubtedly outperforms any other previously experimented, as it is able to provide both human detection and segmentation at once. Figure 4.22 below presents how Mask R-CNN easily detects people in our video frames, regardless of their low resolution and any light condition or person position. In many cases, but not always, multiple people or objects in the background are correctly detected, even if they are very small.

[13] `https://github.com/AVAuco/ssd_head_keras`
[14] `https://pytorch.org/`
[15] `https://github.com/facebookresearch/detectron2`
[16] `https://github.com/akTwelve/Mask_RCNN`

For our purposes, we are just interested in the actual user facing the drone, which is the person closest to the camera, thus usually the biggest one among all the people found by Mask R-CNN. Bounding boxes (not shown in the picture) and labels associated with each object instance are used to identify the correct mask (i.e., the one corresponding to the user), which is later used during training.

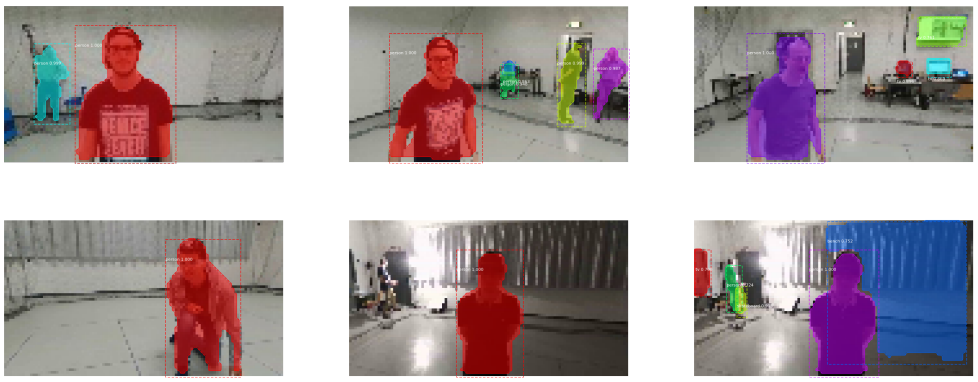


Figure 4.22: Mask R-CNN applied to our training set

This high-level of accuracy in detection and segmentation comes with an extremely high computing power requirement[17]. For reference, running Mask R-CNN on the test set - composed of about 11'000 images - requires a total computing time[18] of approximately 55 minutes on Google Colab with a GPU runtime[19].

Because of this, the inference on the images must be done offline. The training procedure will only receive, together with each input image, the previously computed user's mask.

[17] accordingly to the original paper, Mask R-CNN can only run at 5 FPS

[18] we observe, using the `time` command for IPython, the following CPU times: user 35min, sys 20min, total 55min; and Wall time: 1h 4min

[19] equipped with NVIDIA® T4 GPU

Chapter 5

Model Implementation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 6

Evaluation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 7

Conclusion

7.1 Final Thoughts

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

7.2 Future Works

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Appendix A

Extra Figures

Here a bunch of other images not inserted in the main chapters, in order to keep some section shorter and enhance general readability. Following figures are not crucial for the understanding of our work, but they add minor details.

A.1 ProximityNet

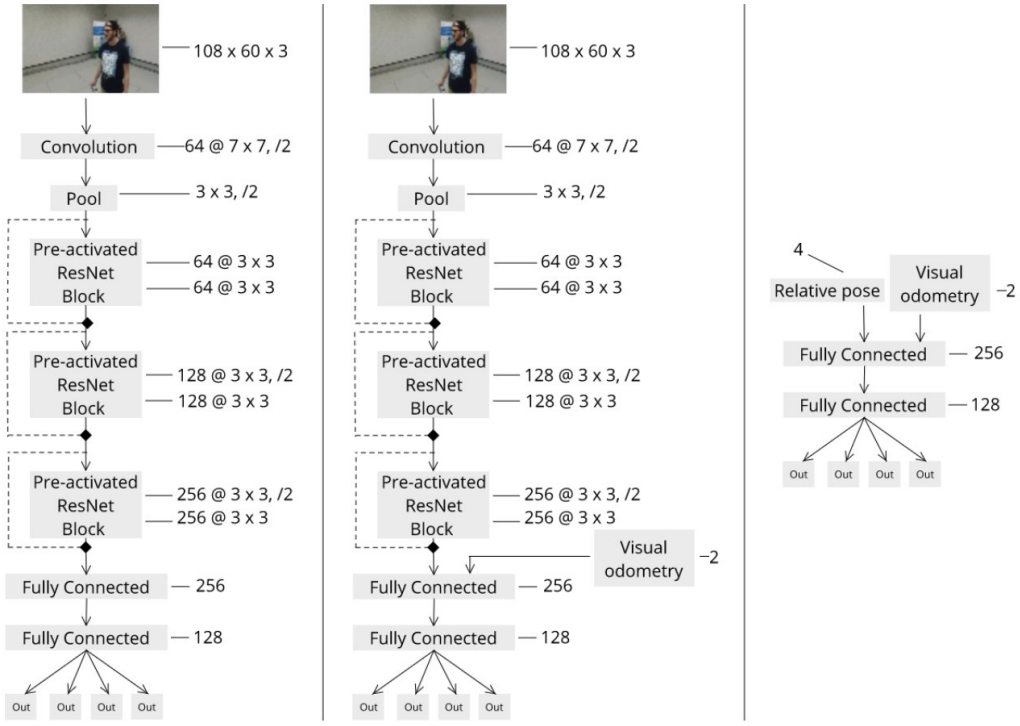


Figure A.1: Models by Mantegazza et al. [2019]: mediated, end-to-end, learned controlled

Please note that following architecture is for classification purposes presented in section 4.1.2. Standard model outputs have shapes `(?, 1)` instead.

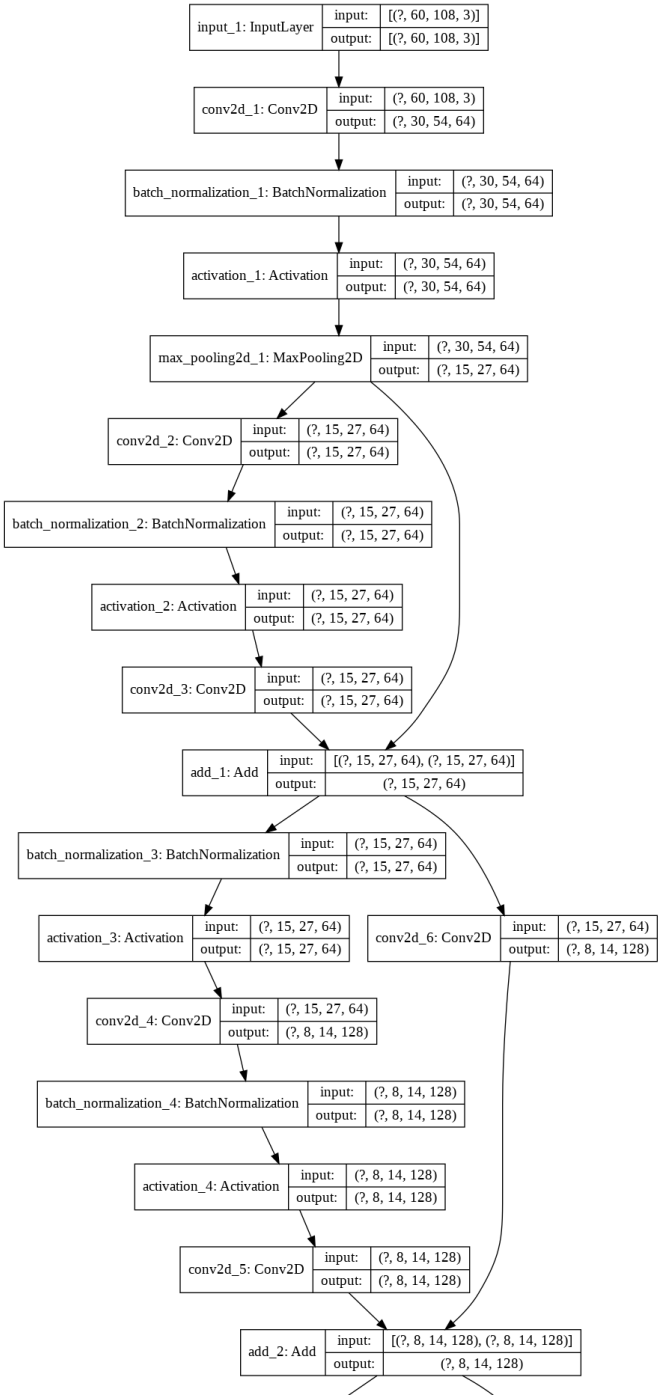


Figure A.2: ProximityNet complete architecture (part 1, from input to layer 18)

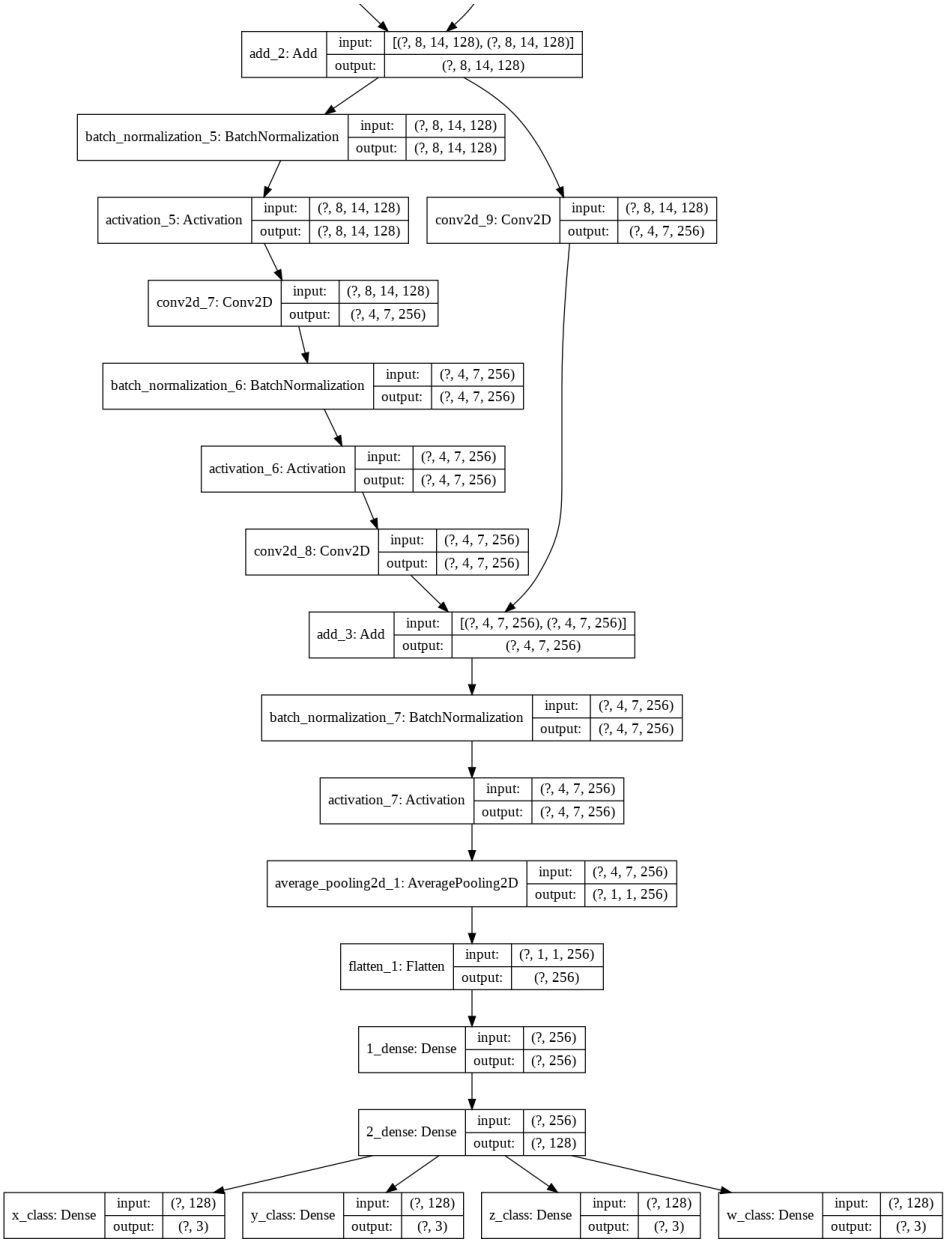


Figure A.3: ProximityNet complete architecture (part 2, from layer 18 to outputs)

A.2 Grad-CAM

This sections reports Grad-CAM applications appropriately divided into variables and classes, in contrast with the single-image approach followed in section 4.1.6.

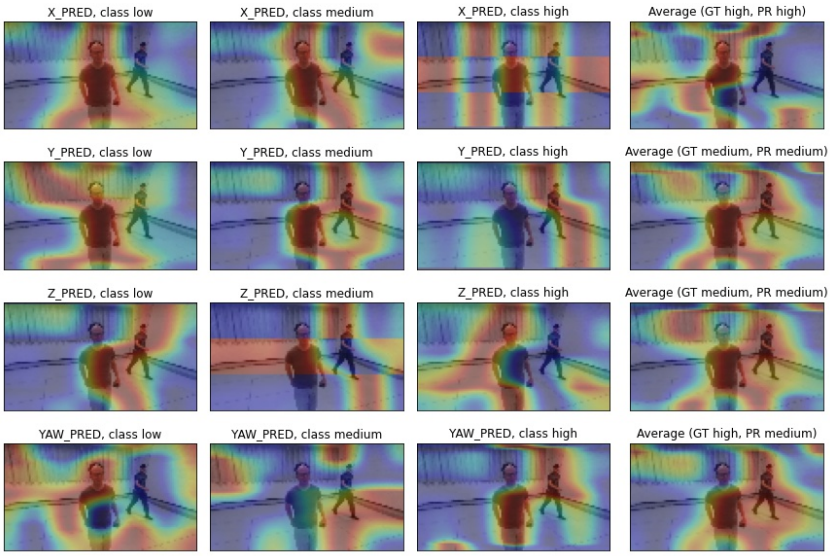


Figure A.4: Full Grad-CAM: two people in the frame

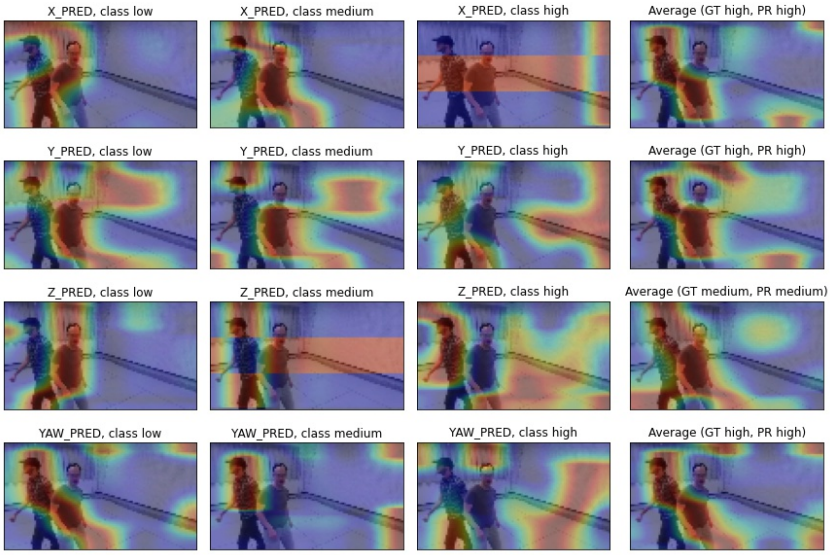


Figure A.5: Full Grad-CAM: two people in the frame

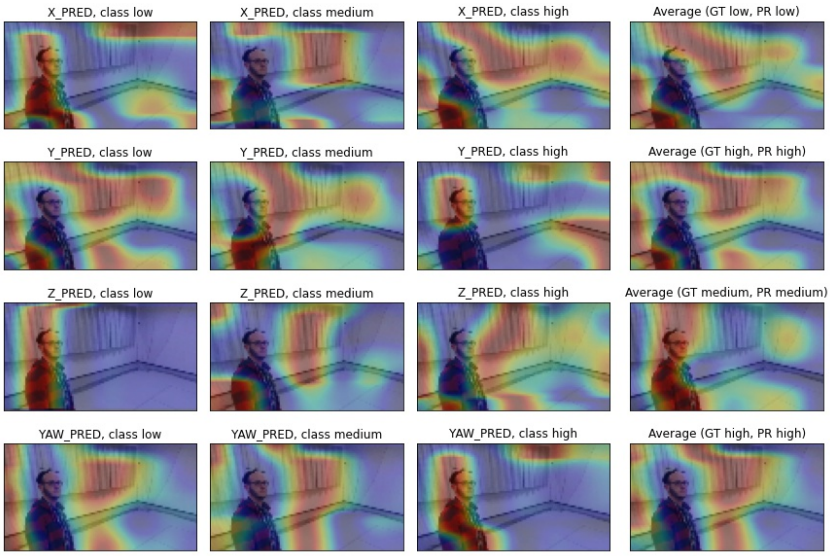


Figure A.6: Full Grad-CAM: model is attracted by curtains

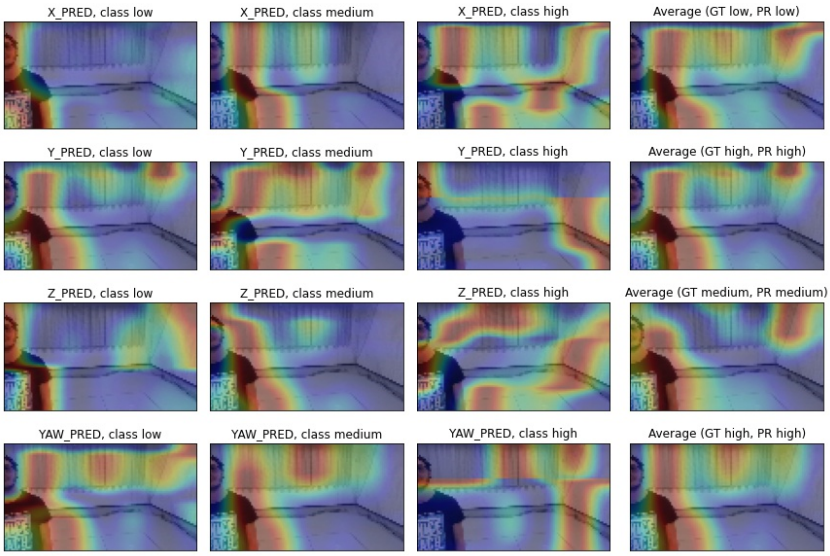


Figure A.7: Full Grad-CAM: model is attracted by curtains

Figure A.8: Full Grad-CAM: model is attracted by background objects

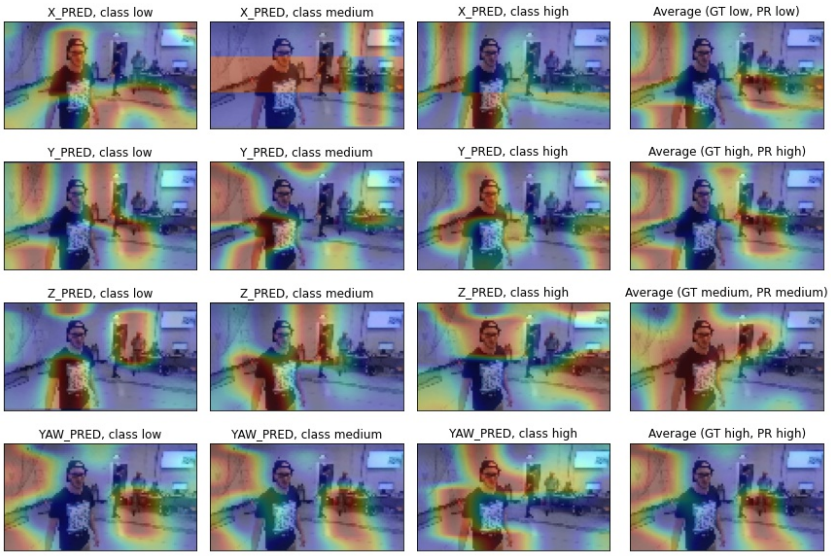


Figure A.9: Full Grad-CAM: model is attracted by background objects

Appendix B

Acronyms

ADAM Adaptive Moment Estimation

AR Augmented Reality

CNN Convolutional Neural Network

DoF degrees of freedom

FAA United States Federal Aviation Administration

FOV field of view

FPS frames per second

GMM Gaussian Mixture Model

Grad-CAM Gradient-weighted Class Activation Mapping

GT ground truth

IDSIA Istituto Dalle Molle di Studi sull'Intelligenza Artificiale

IR infrared

MAE Mean Absolute Error

ML Machine Learning

MoCap motion capture

MP megapixel

NN Neural Network

R^2 R Squared

ResNet Residual Neural Network

ROS Robot Operating System

VR Virtual Reality

wrt with respect to

XAI Explainable AI

Bibliography

Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017.

Ajay Arasanipalai. State of the art deep learning: an introduction to mask r-cnn, 2018. URL https://medium.com/free-code-camp/mask-r-cnn-explained-7f82bec890e3.

ArcGIS. How maskrcnn works?, 2021. URL https://developers.arcgis.com/python/guide/how-maskrcnn-works/.

Jason Brownlee. A gentle introduction to the rectified linear unit (relu), 2019. URL https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks.

Mohamed Chetoui. Explanation of gradient-weighted class activation mapping, 2019. URL https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a.

Gazebo. Robot simulation made easy. URL http://gazebosim.org/.

Google. keras-vis is a high-level toolkit for visualizing and debugging your trained keras neural net models. https://github.com/raghakot/keras-vis, 2020.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.

Ahmed Hussein, Mohamed Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys*, 50, 04 2017a. doi: 10.1145/3054912.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: a survey of learning methods. *ACM computing surveys*, 2017b. ISSN 0360-0300. URL http://hdl.handle.net/10059/2298.

Ahmed Hussein, Eyad Elyan, Mohamed Gaber, and Chrisina Jayne. Deep imitation learning for 3d navigation tasks. *Neural Computing and Applications*, 04 2018. doi: 10.1007/s00521-017-3241-z.

Adam Kelly. Mask r-cnn in tensorflow 2. https://github.com/akTwelve/Mask_RCNN, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Yasuhiro Kubota. tf-keras-vis is a visualization toolkit for debugging tf.keras models in tensorflow2.0+. https://github.com/keisen/tf-keras-vis, 2020.

Weng Lilian. Domain randomization for sim2real transfer. *lilianweng.github.io/lil-log*, 2019a. URL http://lilianweng.github.io/lil-log/2019/05/04/domain-randomization.html.

Weng Lilian. Paper explanation: Domain randomization for sim2real transfer, 2019b. URL https://lilianweng.github.io/lil-log/2019/05/05/domain-randomization.html.

Antonio Loquercio, Ana Isabel Maqueda, Carlos R. Del Blanco, and Davide Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 2018. doi: 10.1109/lra.2018.2795643.

Dario Mantegazza. Defning a new controller for a drone with user partial pose estimation, 2018.

Dario Mantegazza. Vision-based control of a quadrotor in user proximity: Mediated vs end-to-end learning approaches. https://github.com/idsia-robotics/proximity-quadrotor-learning, 2020.

Dario Mantegazza, Jérôme Guzzi, Luca M. Gambardella, and Alessandro Giusti. Vision-based control of a quadrotor in user proximity: Mediated vs end-to-end learning approaches, 2019.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher J. Pal, and Liam Paull. Active domain randomization, 2019.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

OpenCV Documentation. Interactive foreground extraction using grabcut algorithm. URL https://docs.opencv.org/4.1.2/d8/d83/tutorial_py_grabcut.html.

OptiTrack Website. Professional motion capture systems for robotics. URL https://optitrack.com/applications/robotics/.

D. Palossi, F. Conti, and L. Benini. An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 604–611, May 2019a. doi: 10.1109/DCOSS.2019.00111.

D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini. A 64mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 2019b. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2917066.

Parrot Documentation. Bebop drone 2 full specifications, 2015. URL https://s.eet.eu/icmedia/mmo_35935326_1491991400_5839_16054.pdf.

ROS Website. Professional motion capture systems for robotics. URL https://www.ros.org/.

Rviz. General purpose 3d visualization tool for ros. URL http://wiki.ros.org/rviz.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(9):2917–2931, Sep 2020. ISSN 1558-2205. doi: 10.1109/tcsvt.2019.2935128. URL http://dx.doi.org/10.1109/TCSVT.2019.2935128.

D. Tezza and M. Andujar. The state-of-the-art of human–drone interaction: A survey. *IEEE Access*, 7:167438–167454, 2019. doi: 10.1109/ACCESS.2019.2953900. URL https://ieeexplore.ieee.org/document/8903295.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

Zhiguang Wang and Jianbo Yang. Diabetic retinopathy detection via deep convolutional networks for discriminative localization and visual explanation, 2019.

Wikipedia. XAI, 2021. URL https://en.wikipedia.org/wiki/Explainable_artificial_intelligence.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

Xiangyu Yue, Yang Zhang, Sicheng Zhao, Alberto Sangiovanni-Vincentelli, Kurt Keutzer, and Boqing Gong. Domain randomization and pyramid consistency: Simulation-to-real generalization without accessing target domain data, 2019.

Z^2 Little. Activation functions (linear/non-linear) in deep learning, 2020. URL https://xzz201920.medium.com/activation-functions-linear-non-linear-in-deep-learning-relu-sigmoid-softmax-swish-leaky-relu-a6333be712ea.

Nicky Zimmerman. Embedded implementation of reactive end-to-end visual controller for nano-drones, 2020.