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
Dr. Alessandro Giusti
co-supervised by
PhD Stud. 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 works by IDSIA.

2.3.1 The State of the Art of Human–Drone Interaction

An 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 and paper from Dario Mantegazza named *Vision-based Control of a Quadrotor in User Proximity: Mediated vs End-to-End Learning Approaches* from Dario Mantegazza, developed at IDSIA between 2018 and 2019 (Mantegazza [2018], Mantegazza et al. [2019]).

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. 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.

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.

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

Our entire project makes use of the original network architecture and dataset

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

defined in Mantegazza et al. [2019], respectively explained in section 2.3.2.1 and 3.3, For a better understanding, the original code repository[2] and a descriptive video are available[3]. For enhancing readability of this thesis, having no official name, the custom ResNet architecture proposed by the author will be simply called *FrontalNet*.

2.3.2.1 FrontalNet Architecture

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. 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]).

Figure 2.1 provides an illustration of the mediated architecture we consider for this thesis[4]. A complete list of all layers is also available in figures A.2 and A.2 of the appendix.

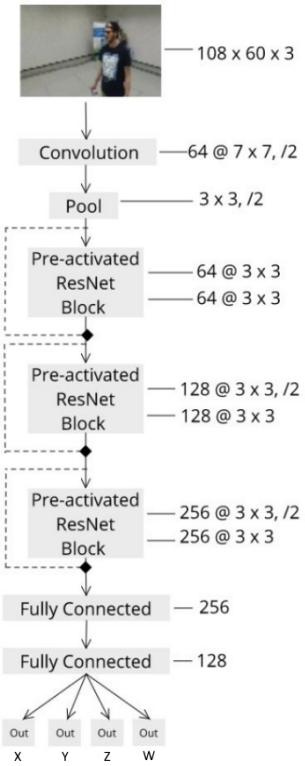


Figure 2.1: Schematic FrontalNet architecture (Mantegazza et al. [2019])

[2] `https://github.com/idsia-robotics/proximity-quadrotor-learning`
[3] `https://drive.switch.ch/index.php/s/MlEDrsuHcSl5Aw5`
[4] take a look at image A.1 for details about the architectures related to the other approaches

2.3.2.2 FrontalNet Performance

FrontalNet 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 before, both approaches obtain similar results. We can accordingly consider the following evaluation to be valid for the mediated approach too.

For quantitative evaluation, the chosen metric is R Squared (R^2)[5], 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 2.2, 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 an R^2 score of 0.82 and 0.88, respectively. Different the findings for X and Y which only reach an R^2 of 0.59 and 0.57, respectively.

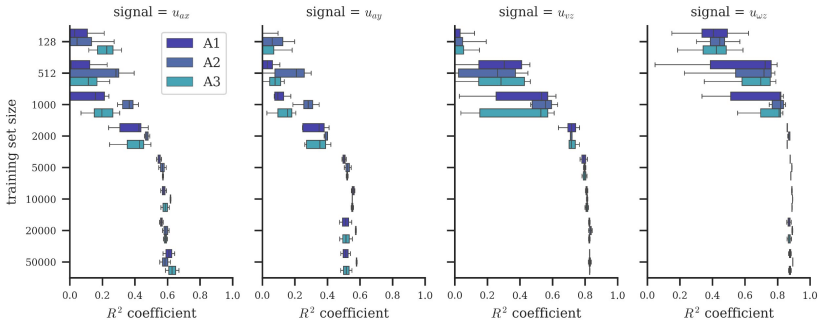


Figure 2.2: FrontalNet R^2 results (Mantegazza et al. [2019]). A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

The previous considerations on the variables are confirmed by the qualitative evaluation, obtained by comparing ground truth and predictions during a short simulation. Figure 2.3 shows that X and Y predictions are considerably worse than results achieved by Z and W when compared with the ground truth.

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

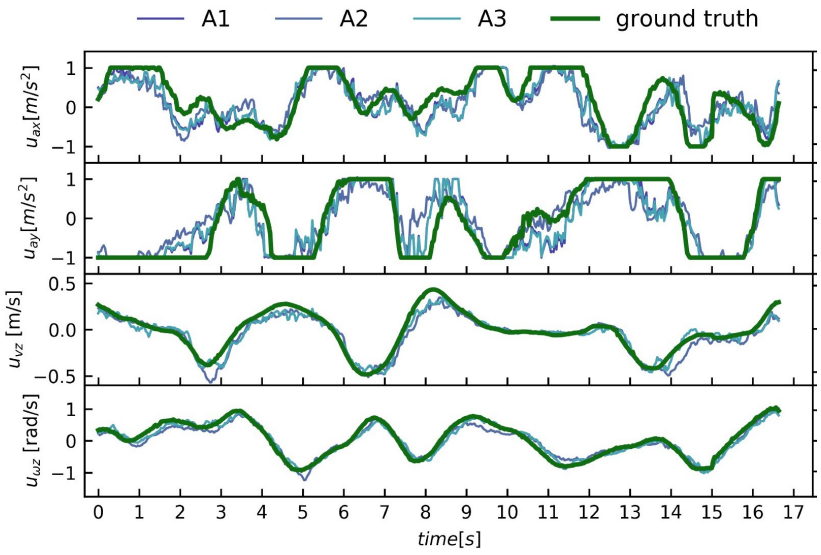


Figure 2.3: FrontalNet GT vs prediction results (Mantegazza et al. [2019]). A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

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 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[6], 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 FrontalNet to work on-board

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

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[7].

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.

However, 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.

When working with real-world problems, it is extremely important to be able to explain what a ML model is actually understanding. This builds trust in algorithms and makes 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]). For CNNs, researchers have developed many techniques for understanding what a NN 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]

[7] Himax HM01B0 camera, theoretically able to produce 320×320 megapixel (MP) images at 60 FPS. However, the frame rate is incredibly reduced during data collection due to some platform limitation in image transfering.

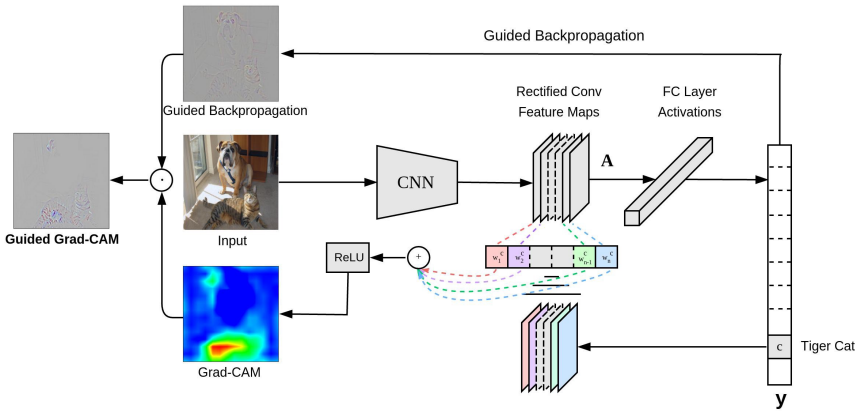


Figure 2.4: Grad-CAM schematic functioning (Selvaraju et al. [2019])

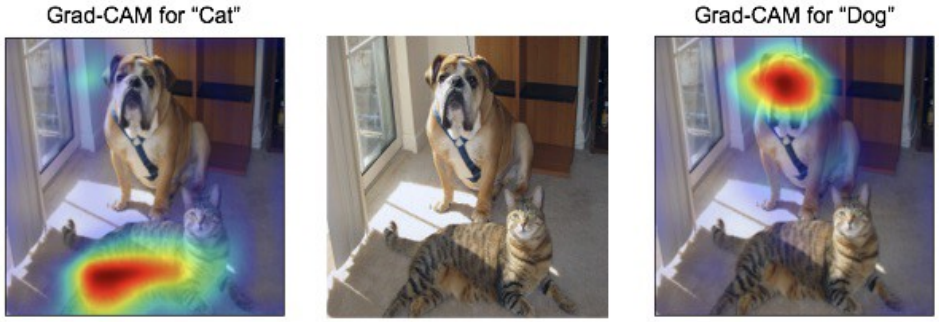


Figure 2.5: Grad-CAM example on dog-cat classification (Selvaraju et al. [2019])

2.5 Network Generalization

Asd

2.5.1 Data Augmentation

TODO

Random Erasing Data Augmentation: Zhong et al. [2017]

Previous works on images: Yue et al. [2019], Takahashi et al. [2020], Xie et al. [2020]

AutoAugment: Cubuk et al. [2019] Learning Data Augmentation Strategies for Object Detection: Zoph et al. [2019]

2.5.2 Domain Randomization

TODO

Summary from Mehta et al. [2019]

See Lilian [2019b], Tobin et al. [2017]

"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."

Imitation learning through simulation is recently becoming an interesting and successful approach for both reinforcement learning (Hussein et al. [2017b], 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[8], often used in robotics with Robot Operating System (ROS)[9] due to its straightforward integration, or even with general-purpose graphic engines. Unreal Engine[10] Unity[11] 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.

2.6 Human Detection and Segmentation

TODO

[8] http://gazebosim.org/
[9] https://www.ros.org/
[10] https://www.unrealengine.com/en-US/
[11] https://unity.com/

2.6.1 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 during development of Mantegazza et al. [2019], by placing a green screen on a portion of the arena walls. 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 2.6 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 2.6: Experimental green screen setup in the drone arena

2.6.2 Mask R-CNN

TODO

MaskRCNN: He et al. [2018], Arasanipalai [2018], ArcGIS [2021]

Chapter 3

System Description

We focus on improving the original work from Mantegazza et al. [2019], introduced in section 2.3.2. This chapter aims to provide a generic view of the system, starting from its main components and how they interact for flying and controlling the drone. Then, we present tools, libraries and dataset we are working with for achieving the goal of enhancing the model generalization capabilities.

3.1 Hardware

Working from a machine learning perspective, our task does not require to interact with physical components. However, it is helpful to know the environment in which the drone was originally taught to fly. This is physically located in Manno (Switzerland) at Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)[1].

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 motors and a dual-core processor with a

[1] now relocated to Lugano

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, and GPS coordinates, in order to ensure maximum stability during the whole flight. For this project, we mainly focus on 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 about 80°. 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

To monitor the drone and the user's movement we require a motion capture (MoCap) system, that is 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. These are able to collect optical signals from passive[2] or active markers[3] 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.

3.1.3 Drone Arena

At IDSIA, a dedicated room has been equipped with a MoCap system composed by 12 OptiTrack Prime^{x}13 infrared (IR) cameras[4] for medium-sized areas (figure 3.3a). They track the movements of passive markers, placed both 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). A virtual fence of 4.8×4.8 meters virtually constraints the total area in which the drone is allowed to fly.

[2] a passive marker reflect light

[3] an active marker emits its own light

[4] 1.3 MP, 240 FPS, ± 0.20 mm 3D accuracy in a 9×9 meters area with 14mm markers

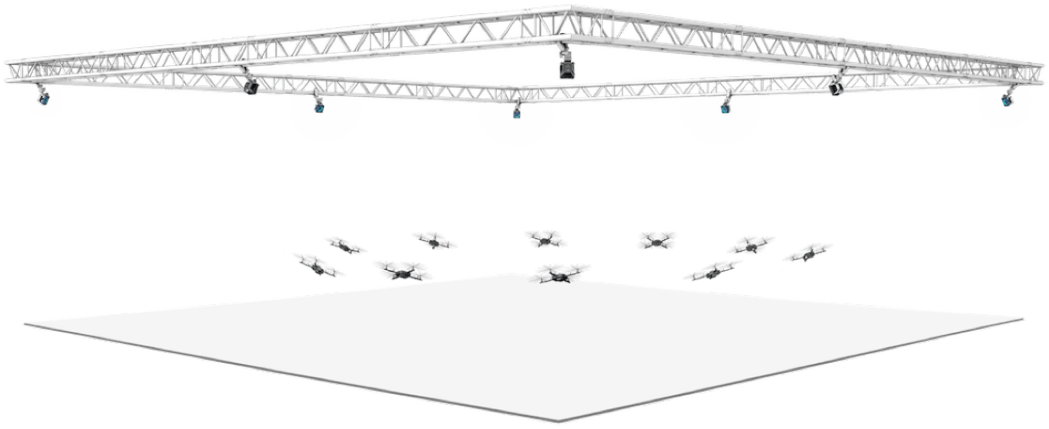
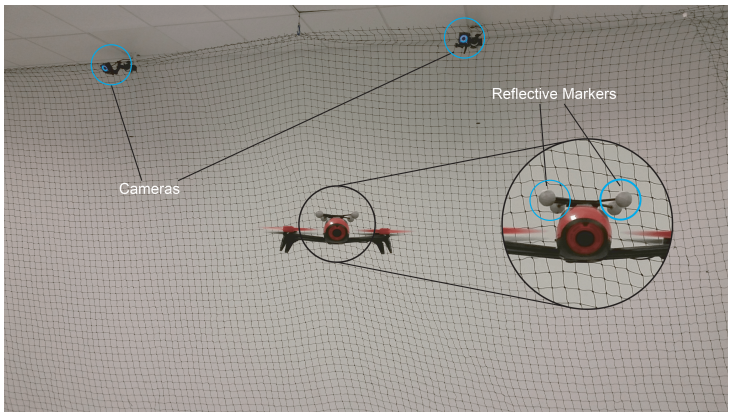


Figure 3.2: Schematic OptiTrack system with 12 OptiTrack Prime cameras

(a) Prime^{x}13 camera



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

Figure 3.3: Drone arena at IDSIA

3.2 Software

This section presents tools and software used to conduct our research for improving the existing system. The original work is written in Python 3 and based on ROS, TensorFlow 1, and Keras. We adapt the code for working with TensorFlow 2.

Our first experiments were carried out with Jupyter Notebooks via Google Colab on a GPU-accelerated runtime, while the final code is provided as Python scripts. For debugging, we use a Windows 10 laptop equipped with an NVIDIA GeForce GTX 950M graphic card, while training is performed on a dedicated Ubuntu 18.04 workstation available at IDSIA, mounting four NVIDIA GeForce RTX 2080 Ti [5].

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

3.2.1 Frameworks

Here is a list of the main libraries which compose our software.

`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 loading Numpy arrays. The pickle module implements binary protocols for (de-)serializing Python object structures.

`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` The entire 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.

`Albumentation` Used during training for implementing classic image augmentation. Albumentations efficiently implements a rich variety of image transform operations that are optimized for performance through a concise yet powerful interface. Widely used in industry, deep learning research, machine learning competitions, and open source projects.

`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. Open-source implementation of Mask R-CNN on Python 3, Keras, and TensorFlow 2 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.

`Robot Operating System (ROS)` Even tough not used in our work, it is crucial in Mantegazza et al. [2019] to actually control the drone during flight. ROS is an open-source robotics middleware suite[6] for building robot applications, providing hardware abstraction, implementation of commonly-used functionality, message-passing between processes, and package management. It also integrates with additional tools, such as Rviz and Gazebo, for real-time 3D visualizations and simulations.

3.3 Data

This section refers to the dataset acquired for Mantegazza et al. [2019], which is also used for our research.

3.3.1 Collection

Data have been entirely collected in the dedicated drone arena, presented in section 3.1.3. A good dataset should ideally provide images from various scenarios, but such a kind of data is not easy to record. The ground truth must be acquired by a complex and expensive MoCap system, particularly difficult to be moved and reassembled outdoor.

The drone is controlled by a ROS script, running on a dedicated computer remotely connected through WiFi. It relies on the known user's pose - from now on, the *target pose* (i.e., the pose of the user seen by the drone reference frame) - to

[6] available for C++, Python, and Lisp

compute acceleration commands for the drone. The script is responsible of making the drone hovering in front of the person, facing the head orientation at a predefined 1.5 meters distance.

During data collection, both user's and drone's poses are captured by the OptiTrack using proper markers, placed on the drone and on the person's head (picture 3.4). The target poses over time[7], are synchronized with the video stream coming from the camera, and saved into `rosbag` files. Figure 3.5 shows an illustration of the system from a bird-eye view.

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

3.3.2 Composition

Data collected inside the arena have been used to build the dataset for training a machine learning model, able to infer the target pose from a picture. For building both the training and the testing set, several flight sessions have been recorded using the omniscient controller described above.

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[8]. 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 composing 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).

[7] mathematically computed by a script from Mantegazza [2018]

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

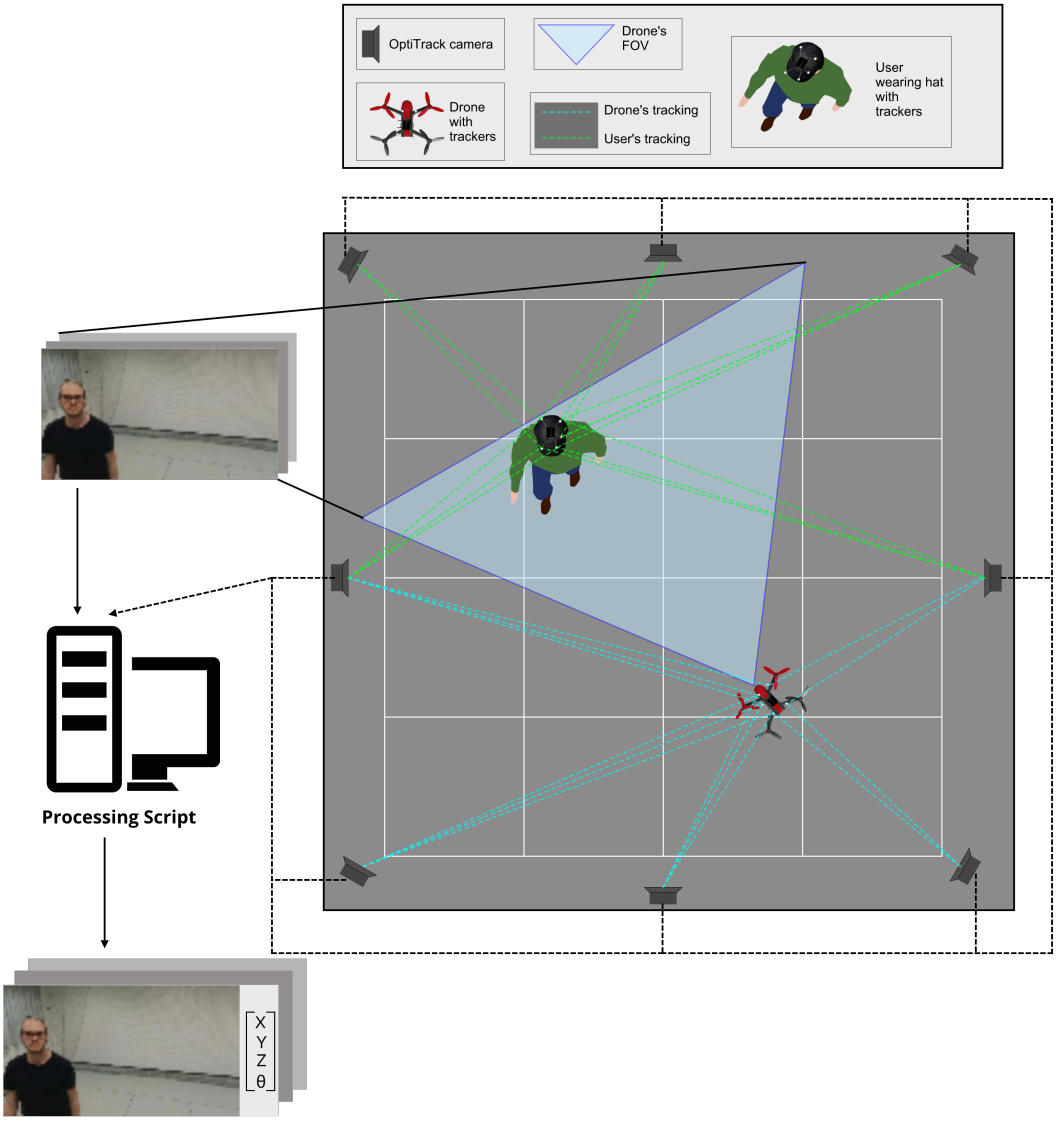


Figure 3.5: OptiTrack and data collection illustration (Mantegazza [2018])

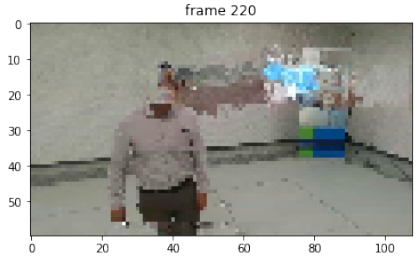


Figure 3.6: A frame with digital artifact caused by connection issues

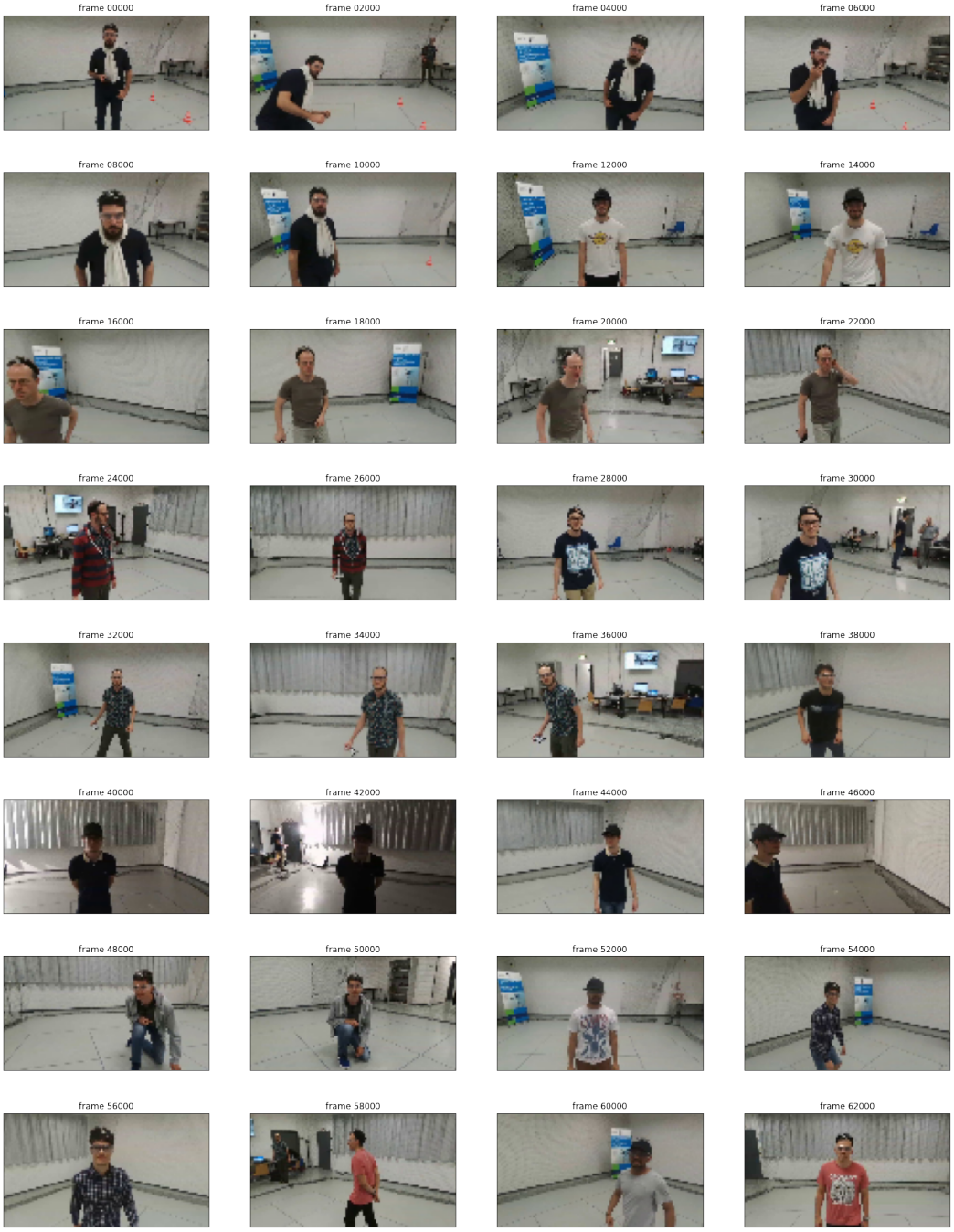


Figure 3.7: A complete overview of images in the training set

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. The variables explain the user's pose with respect to (wrt) the drone, and their interpretation is described here:

- `X` is the distance of the user from the drone and affects the pitch (acceleration along the X-axis). Usually, the drone flies at 1.5 meters from the user.
- `Y` represents the horizontal alignment of the user in front of the drone and affects the roll (acceleration along the Y-axis). When 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. When 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. This 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.

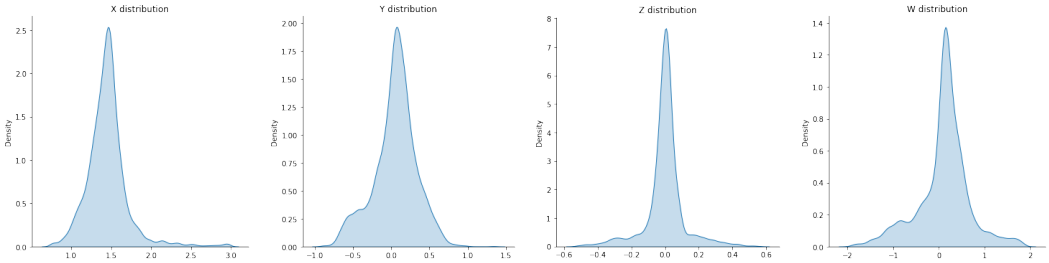


Figure 3.9: Target variables distribution for the regression task

Chapter 4

Solution Design

This chapter explores the issues of the existing model, proposes a solution, and presents initial experiments on its feasibility.

4.1 Problem Summary

In section 2.3.2, we introduce the original paper we are working on and present its architecture and basic performance. Chapter 3 illustrates the main components for controlling the drone and the dataset used to train the machine learning model, as declared in Mantegazza et al. [2019].

FrontalNet achieved quite good performance on the test set, as explained in section 2.3.2.2. However, 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[1].

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 discovered that flying performance outside of the drone arena was not consistent with usual model behavior. The drone was not able to follow the user and its movements were unexplainable. We conclude that the model is not able to generalize the task when outside of the environment it already knows.

Our goal is to explore ways of improvement, to generalize the model and make it able to theoretically predict the user's pose in any other unknown scenario. The next sections firstly try to understand the main issues and limitations of the model, then provide a possible solution for the generalization problem.

[1] see figure A.4 in the appendix for further details

4.2 Model Interpretation with Grad-CAM

In the previous section, we discussed insufficient experimental results obtained by FrontalNet in predicting the user's pose in an unknown environment (i.e., outside of the drone arena). This section highlights the main issues behind the lack of its generalization capabilities, by understanding what the model is actually learning.

Convolutional Neural Networks (CNN) are suited for computer vision thanks to 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". This means that their internal behavior is particularly challenging for humans to understand.

Among network interpretability tecnhiques introduced in section 2.4, we choose Gradient-weighted Class Activation Mapping (Grad-CAM). The algorithm is indeed the most understandable way of visualizing what a CNN 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 actually responsible for predicting a certain output[2].

Research in the field is still on-going and most of the available resources are for TensorFlow 1. The most powerful and famous library for network interpretability is `Lucid`[3], from the official TensorFlow team, but it is not easily applicable to Keras models.

Then, we use the open-source library `tf-keras-vis` by Kubota [2020], porting for TensorFlow 2.0+ of `keras-vis` by Google [2020].

4.2.1 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 (Regression Activation Map, Wang and Yang [2019]), it appears to be an isolated case, since the research in the field it is not developed yet. For this reason, and for network interpretation only, we decide to transform our problem into a classification task.

The ground truth is composed of four variables with specific domains, and figure 3.9 in the previous chapter shows their distribution. Every variable has a particular "center" value, which is obtained when the user is somehow centered in 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`.

[2] for an easy understandable Grad-CAM example, please refer to the figure 2.5 which regards a simple dogs VS cats classifier

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

- `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). This class partition is fundamental for understanding some Grad-CAM visualizations later.

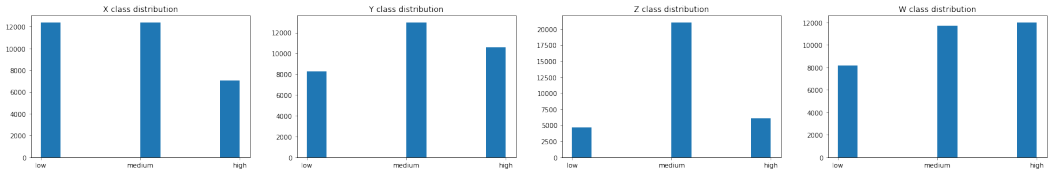


Figure 4.1: Target variables distribution for the classification task

4.2.2 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 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.

4.2.3 Reading charts

A proper understanding of this section requires a thorough ability on reading the following charts[4].

Grad-CAM application requires to specify the class for which we want to compute the activation mapping. In our case, it would be one of the 3 defined in section 4.2.1: `low`, `medium` or `high`.

[4]Basic knowledge on how the related library works would also be helpful. Tutorial:`https://github.com/keisen/tf-keras-vis/blob/8f83773520069367902becc0a668dda90ab76349/examples/attentions.ipynb`

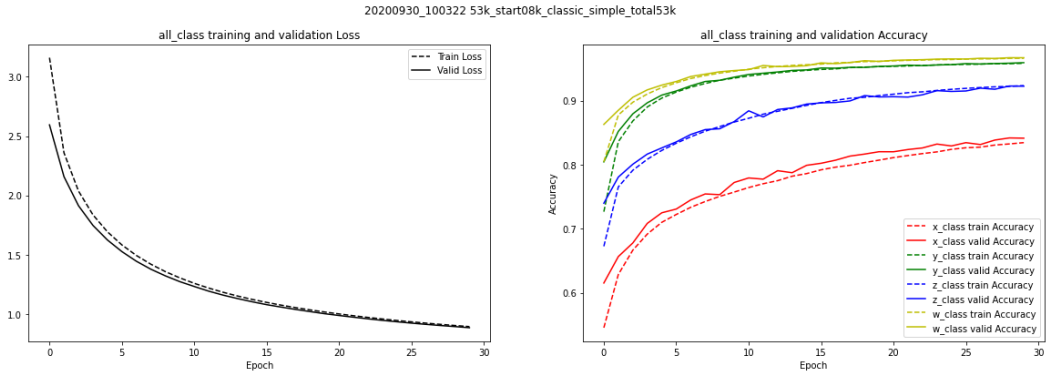


Figure 4.2: Grad-CAM: loss and accuracy of the new classification model

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, FrontalNet 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 assumptions 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 CNN 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.

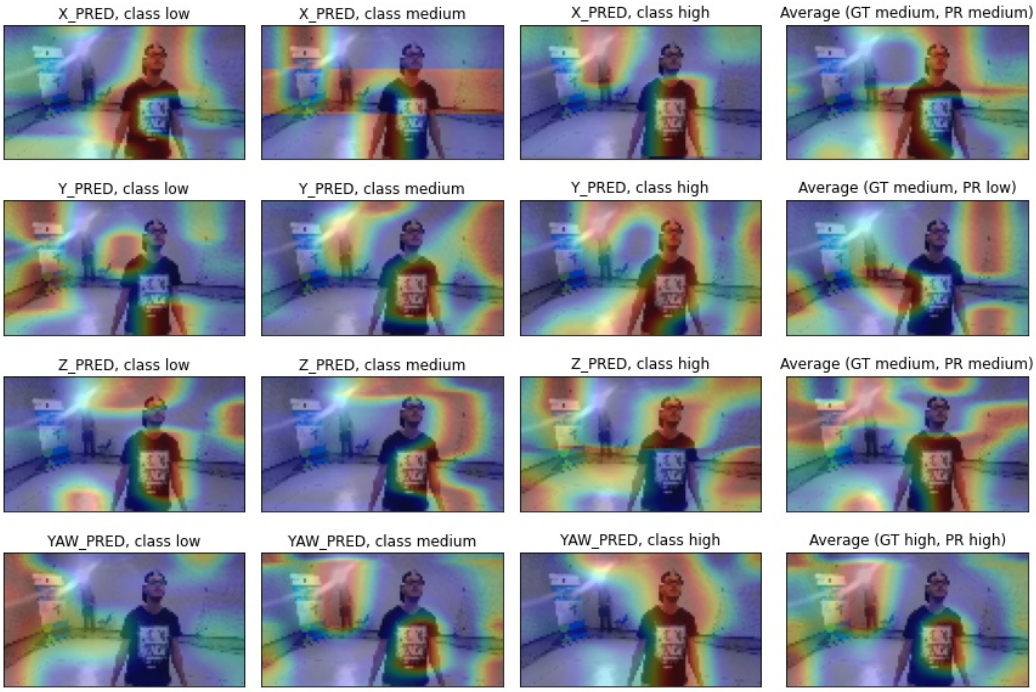


Figure 4.3: Grad-CAM: example of application for each variable and class

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).

4.2.4 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

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

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.

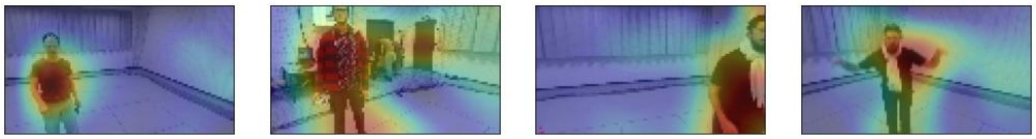


Figure 4.5: Grad-CAM: Correctly detected people

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).

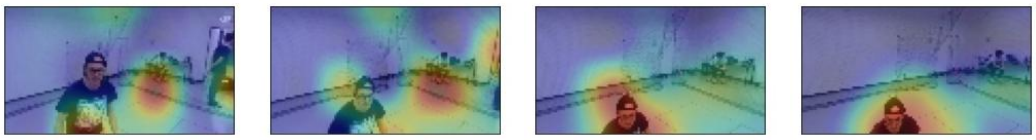


Figure 4.6: Grad-CAM: Sequence transitioning from wrong to correct detections

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-

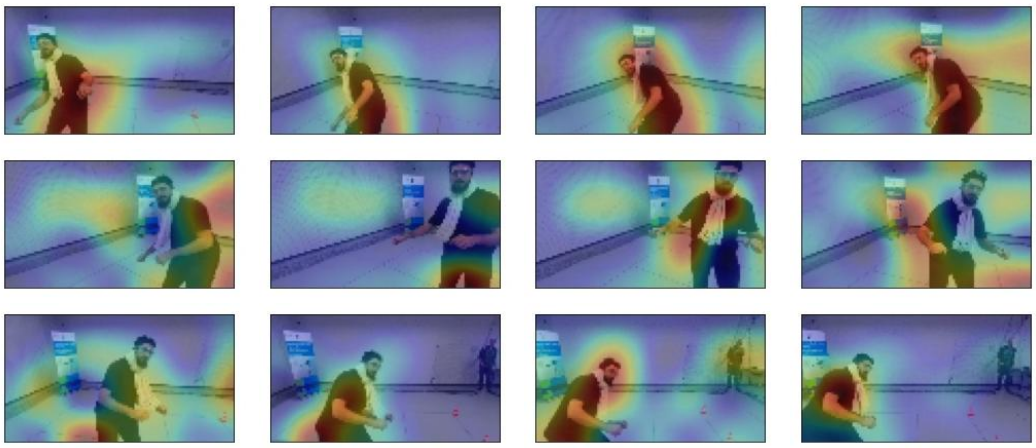


Figure 4.7: Grad-CAM: Sequence of unstable detections in and out of the person

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)

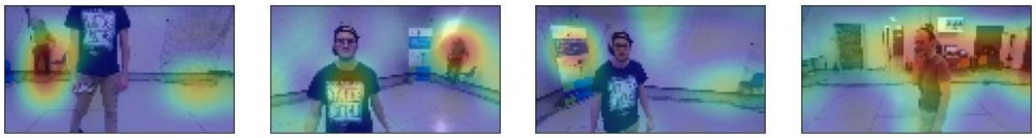


Figure 4.8: Grad-CAM: Objects in the background detected

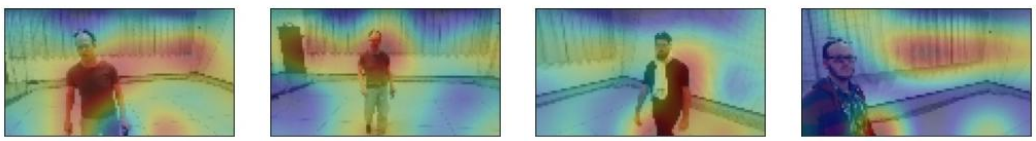


Figure 4.9: Grad-CAM: Curtains often distract the model

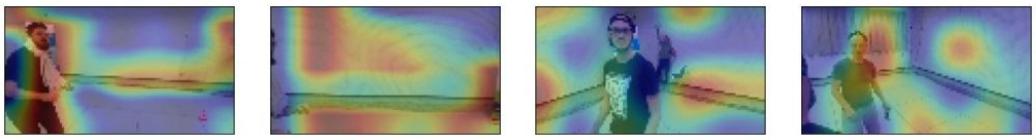


Figure 4.10: Grad-CAM: Model get easily distracted by various elements

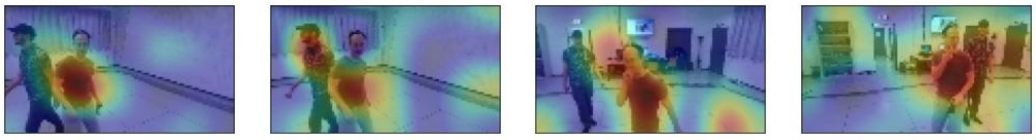


Figure 4.11: Grad-CAM: Detections when two people are present in the image

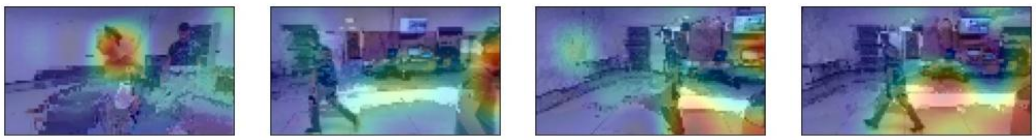


Figure 4.12: Grad-CAM: Model reactions to artificial glitches

4.2.5 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 its 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 4.1.

4.3 Person Masking

From Grad-CAM results presented 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.

4.3.1 Canny

The first experiments are based on a classic computer vision technique called Canny Edge Detection. A custom algorithm[5] applies the related function from OpenCVOpenCV [a] 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 4.13 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.

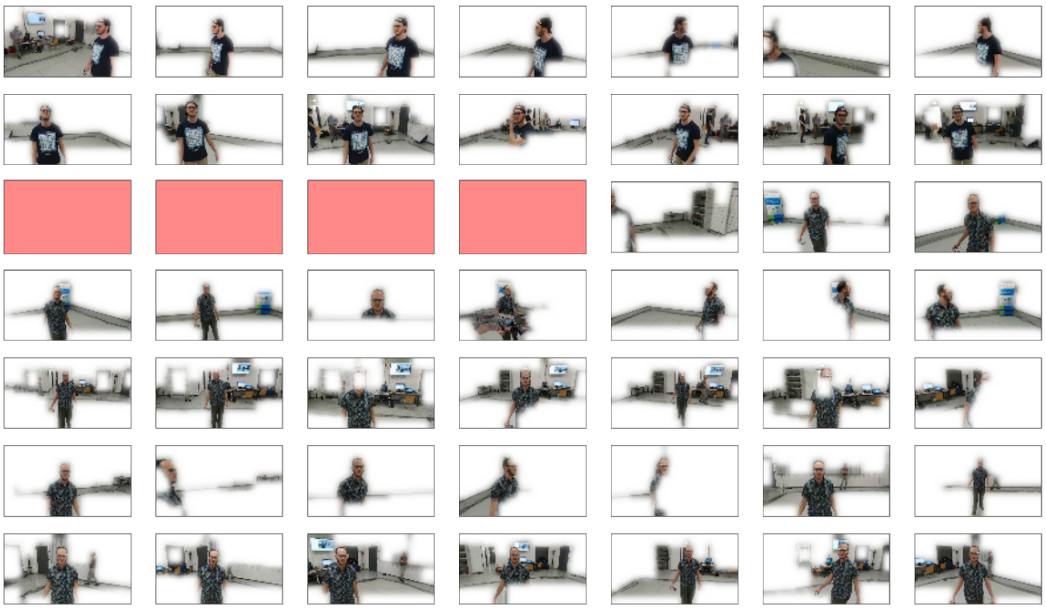


Figure 4.13: Canny edge detection overview on the training set

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 always keep the person's face and part of the body in the resulting image, assuming their positions are known. Figure 4.14

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

illustrates the problem and demonstrates that results obtained from the enhanced version are still not acceptable, since we are not able enough to appropriately remove the background from the scene.

Figure 4.14: Canny edge enhanced algorithm demonstration

4.3.2 Grabcut

Given the Canny Edge Detection limits, 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 [b]):

- 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[6] 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[7] 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 manually provide this information for the example images. Later on, we will consider an algorithm for automatic human detection.

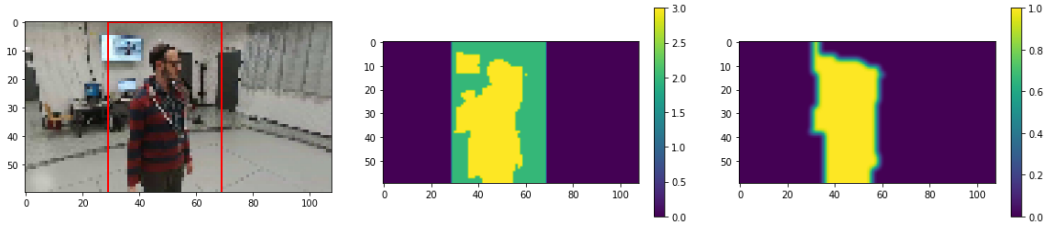
4.3.2.1 Rectangle initialization

This approach requires that a rectangle, entirely containing the subject, is given in input to the function (figure 4.15a). 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

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

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

rectangle which can be foreground, marking them as *probable-foreground* (yellow) (figure 4.15b). Later, we binarize and smooth the mask (figure 4.15c) for finally removing the background from the original image.



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

Figure 4.15: Grabcut algorithm explained: rectangle initialization

Performance obtained by the algorithm is available in figure 4.16. 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.

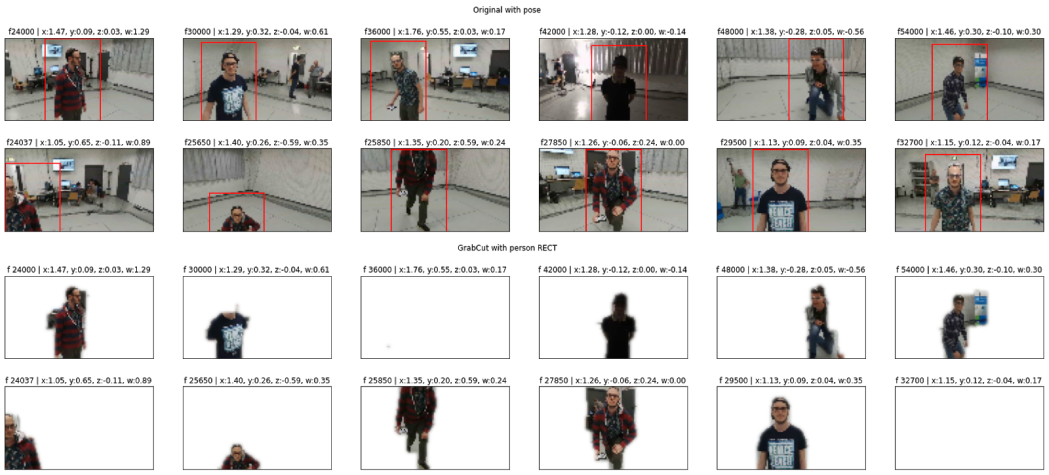


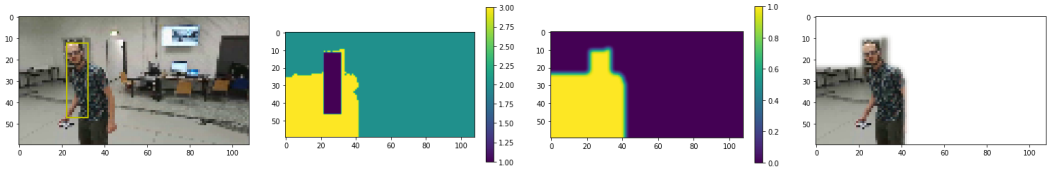
Figure 4.16: Grabcut demonstration: rectangle initialization

4.3.2.2 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 4.17, 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 in the previous rectangle initialization.



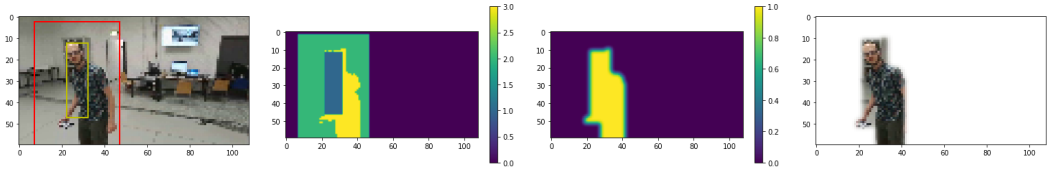
(a) Initialization (b) Pixel classes (c) Final mask (d) Result

Figure 4.17: Grabcut algorithm explained: mask initialization

4.3.2.3 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 4.18 the usual explanation. Here, *sure-background* is dark blue, *probable-background* is green, *probable-foreground* is yellow and *sure-foreground* is light blue. Image 4.19 shows results of hybrid initialization applied to the same samples introduced in the figure 4.16. Results are sub-optimal, with excellent segmentation.



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

Figure 4.18: Grabcut algorithm explained: hybrid initialization

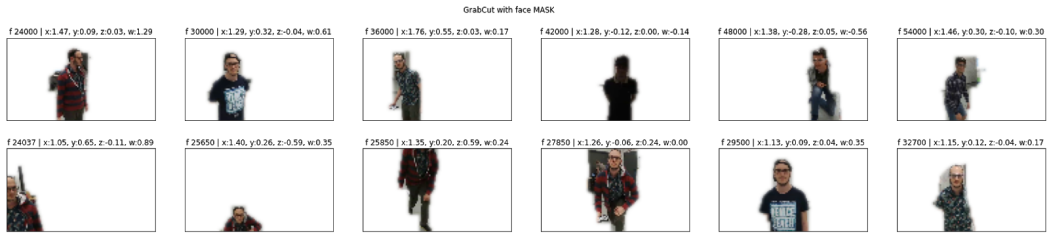


Figure 4.19: Grabcut demonstration: hybrid initialization

4.3.2.4 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 (Redmon et al. [2016]) for human detection, through the `cvlib`[8] library that implements a YOLOv3 model trained on the Microsoft COCO dataset (Lin et al. [2015]), capable of detecting 80 common objects in context. Underneath, it uses the OpenCV dnn module[9].

A demo on our dataset is shown in figure 4.20, where we notice that YOLO overall provides quite good results. However, in 10-20% of the cases, it does not detect any object in the image, most probably because of their low resolution.

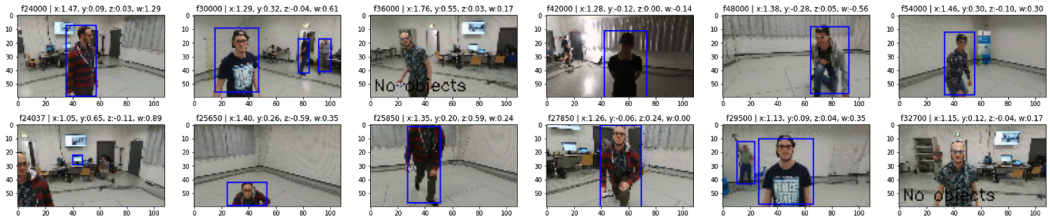


Figure 4.20: YOLO demonstration, which shows failures for 2 images

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. The majority of samples are compliant with this heuristic, but a non-ignorable percentage of samples is not compatible. For this reason, we also try an open-source head detector[10], miserably failing in its 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.

[8] `https://www.cvlib.net/`
[9] `https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html`
[10] `https://github.com/AVAuco/ssd_head_keras`

4.3.3 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 2.6.2. Originally developed by Facebook researchers in PyTorchPyTorch, now available in the `Detectron2` package (Wu et al. [2019]), the algorithm has been ported to TensorFlow 1 (Abdulla [2017]) and later adapted for TensorFlow 2 by Kelly [2020][11]. 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 4.21 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.

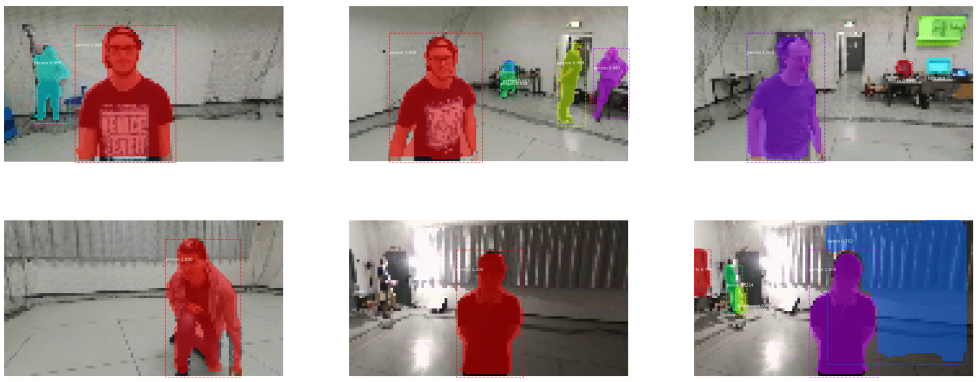


Figure 4.21: Mask R-CNN applied to our training set

This high-level of accuracy in detection and segmentation comes with an extremely high computing power requirement[12]. For reference, running Mask R-CNN on the test set - composed of about 11'000 images - requires a total computing time[13] of approximately 55 minutes on Google Colab with a GPU runtime[14].

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. This will be used to perform background replacement.

[11] `https://github.com/akTwelve/Mask_RCNN`

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

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

[14] equipped with NVIDIA T4 GPU

Chapter 5

Model Implementation

In this chapter, we explain how the proposed solution has been implemented. Our main contribution concerns the way the dataset is treated and processed to reduce overfitting.

The chapter is composed as follows:

- Sections 5.1 and 5.2 focus on the implementation of our generalization strategy.
- Section 5.3 provides an overview of our different model alternatives, that will be considered for comparison and evaluation in chapter 6.
- Sections 5.4 and 5.5 describe the training procedure, with a particular focus on time performance.

5.1 Background Replacement

As demonstrated in section 4.2.4, the approach defined by Mantegazza et al. [2019] is lacking generalization capabilities. The main reason behind this problem is attributable to its dataset composition. More specifically, the model is biased by many elements appearing in the drone arena, in which the data have been originally collected. As a solution to eliminate the problem, we modify the training set by performing background replacement on its images.

In section 4.3.3, we anticipated the use of Mask R-CNN to preprocess the dataset. The algorithm detects and creates a mask for all the objects appearing in the input images, labeling each mask with the category to which the object belong (e.g., person, TV, bike, car, ...). However, for our purposes, we are only interested in the mask corresponding to the user who is actually facing the drone. Since the user is always the nearest person to the drone's camera, its mask must be the one with the largest size among all people's masks found by Mask R-CNN.

To perform the background replacement, the training procedure receives together with each sample also the corresponding user's mask. This is used for distin-

guishing the subject from the rest of the image, to accordingly blend the camera's frame with another image, serving as the background. The ground truth remains unchanged, and this is the main advantage of our approach. We can simulate a dataset acquired in different environments without the need of actually collecting it, which would otherwise require a dedicated MoCap system. Our method is fairly similar to domain randomization (section 2.5.2), a technique widely applied in robotics to train ML models in simulated virtual environments.

By providing a considerable amount of images to use as backgrounds, the ML model trained on the modified dataset should be able to actually ignore the background. The CNN, instead, will hopefully learn the concept of a person, in order to predict its position in any condition.

For our work, we select the publicly available dataset[1] for Indoor Scene Recognition presented during the 9th Conference on Computer Vision and Pattern Recognition (CVPR). Created by researchers from MIT (Quattoni and Torralba [2009]), the dataset contains a total of 15'620 images divided into 67 indoor categories. For our task, categorization is not actually needed, but it ensures a good variety of scenarios to present to the model. For shortness, in this thesis, the dataset will be referred to as *CVPR*.

During training, each sample is assigned to a randomly chosen background from the CVPR dataset. Figure 5.1 shows a demonstration applied to the samples previously presented in figure 3.7.

[1] `http://web.mit.edu/torralba/www/indoor.html`

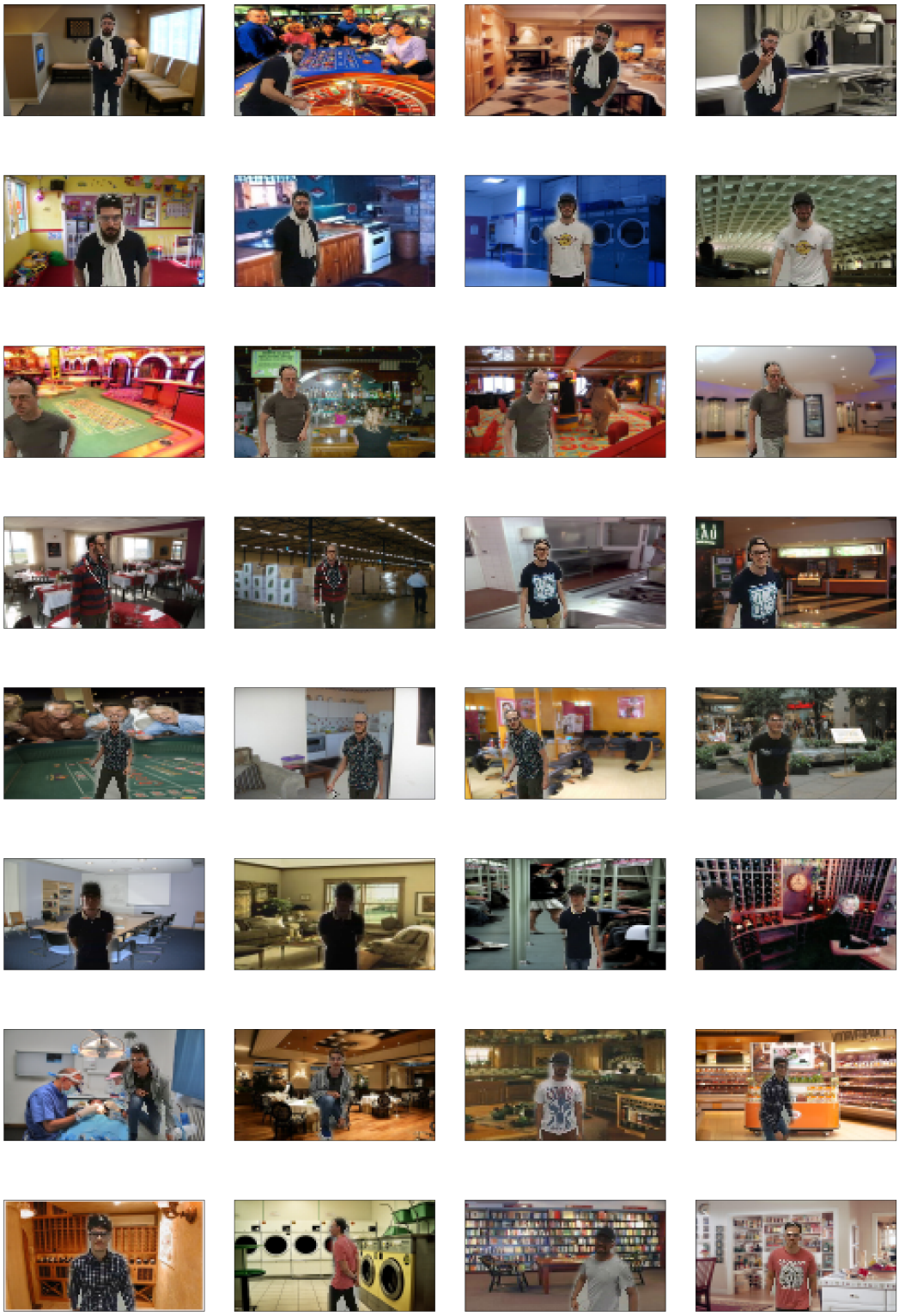


Figure 5.1: Example of background replacement on the training set

5.2 Classic Augmentation

Data augmentation is another common regularization technique for improving the ability of neural networks to generalize their task on previously unseen samples. Introduced in section 2.5.1, image augmentation is ubiquitously used with CNNs for reducing overfitting on the training images by applying random transformations.

Our implementation relies on Albumentations (Buslaev et al. [2020]), a state of the art Python library which provides a huge variety of image augmentations. Constantly updated and well-documented, Albumentations can boast the best benchmarking performance in the field[2].

Image transformations can be divided into two types:

- Spatial-level augmentations: also called affine transformations, they relocates pixels by cropping, scaling, rotating, translating, mirroring and shearing the images. These transformations also have to accordingly modify additional elements associated with the images such as ground truth, masks, bounding boxes or keypoints.

- Pixel-level augmentations: algorithms which only modify pixels (or group of pixels) without relocating them or changing the image shape. These transformations leave unchanged any other additional element associated with the images such as ground truth, masks, bounding boxes or keypoints.

Our CNN learns to predict the user's pose, but the relation between the person's position in the image and the ground is not known a priori. Being not able to modify the ground truth according to affine transformations, spatial-level augmentation is not an option in our case. The only exception is for horizontal mirroring, which only requires to invert the `Y` coordinate in the ground truth[3]. For this reason, we only apply pixel-level augmentations.

According to user-defined probabilities, Albumentations gives the opportunity to apply a set of different augmentations with variable intensities. Among pixel-level transformations, the possibilities are limitless and they can produce aggressively augmented images, which might be very different from the originals. Figure 5.2 provides a good example of augmentation, applied both on original and background-augmented samples. Results below are obtained combining transformations on brightness and contrast, multiplicative noise, channels manipulation, and rectangular dropouts.

The combination of different transformations composes a custom pipeline, which time performance depends on custom choices and probabilities. Among all augmen-

[2] `https://github.com/albumentations-team/albumentations#benchmarking-results`

[3] `Y` coordinate represent the horizontal alignment, as explained in section ??

Figure 5.2: Example of image augmentation with Albumentations

tations available[4] and tested[5], some of them only requires about 0.5 seconds[6] or even less[7] to be applied on a set of 10'000 60×108 images. The most expensive ones take up to 12 seconds[8] under the same conditions.

```
augmenter = A.Compose([
    A.RandomBrightnessContrast(brightness_by_max = True, p = 0.75),
    A.RandomGamma(p = 0.5),
    A.CLAHE(p = 0.05),
    A.Solarize(threshold = (200, 250), p = 0.2),
    A.OneOf([
        A.Equalize(by_channels = False, p = 0.5),
        A.Equalize(by_channels = True, p = 0.5),
    ], p = 0.1),
    A.RGBShift(p = 0.3),
    A.OneOf([
        A.ChannelDropout(fill_value = 128, p = 0.2),
        A.ChannelShuffle(p = 0.8),
    ], p = 0.1),
    A.MultiplicativeNoise(per_channel = True, elementwise = True, p =
                                    0.05),
    A.CoarseDropout(holes = (20, 70), size = (1, 4), p = 0.2),
    A.ToGray(p = 0.05),
    A.InvertImg(p = 0.05),
    A.OneOf([
        A.Blur(blur_limit = 4, p = 0.5),
        A.MotionBlur(blur_limit = 6, p = 0.5),
    ], p = 0.05),
], p = aug_prob)
```

Listing 5.1: Chosen Albumentations pipeline

[4] https://albumentations.ai/docs/api_reference/augmentations

[5] tests performed on a notebook equipped with an Intel Core i7-6700HQ CPU @ 2.60 GHz

[6] Blur, Equalize, RandomBrightnessContrast, RandomGamma

[7] up to 0.30 seconds for InvertImg, Solarize, ChannelDropout, ChannelShuffle, ToGray

[8] HueSaturation 1.60 sec, RGBShift 1.50 sec, CLAHE 3.50 sec, CoarseDropout 3.50 sec, MultiplicativeNoise 7 sec, GaussNoise 10 sec, ISONoise 12 sec

The Albumentations pipeline adopted for our work is shown in listing 5.1. It is not particularly aggressive, and according to its probabilities it leaves unchanged the 3.5% of images. To be executed on the above test, the chosen augmentations require about 4 seconds. The code also accepts an `aug_prob` parameter that is the a priori probability of applying the pipeline to an input image. Some examples of resulting augmentations are available in figure 5.3.

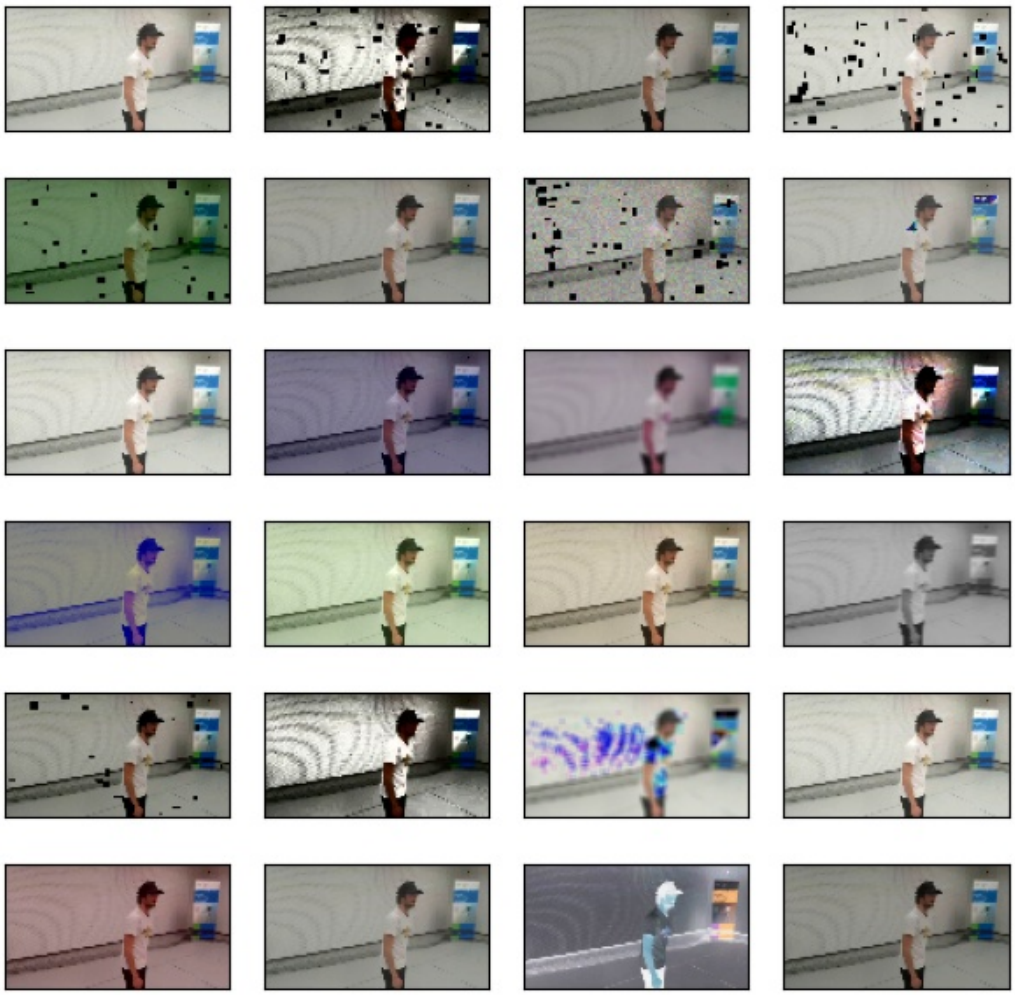


Figure 5.3: Examples of the chosen image augmentation pipeline

In addition, at the end of the pipeline, we also apply Perlin noise (**?**) with a probability of 20%. Injecting noise into images can greatly help CNNs avoid overfitting (Shorten and Khoshgoftaar [2019]), and Perlin is particulary useful to generate procedural textures. Since noise generation is highly expensive, a set of Perlin noise examples has been generated offline. During training, we randomly

choose, crop and flip one of them to be multiplied with an input image. Half of the time it is applied on a grayscale basis, and the other half over RGB channels, as shown in picture 5.4.

Figure 5.4: Perlin noise example. From left to right: (1) randomly chosen, cropped and flipped texture (2) applied uniformly (3) applied by channel

5.3 Model Alternatives

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.

5.4 Data Generator

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.

5.5 Training

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 FrontalNet

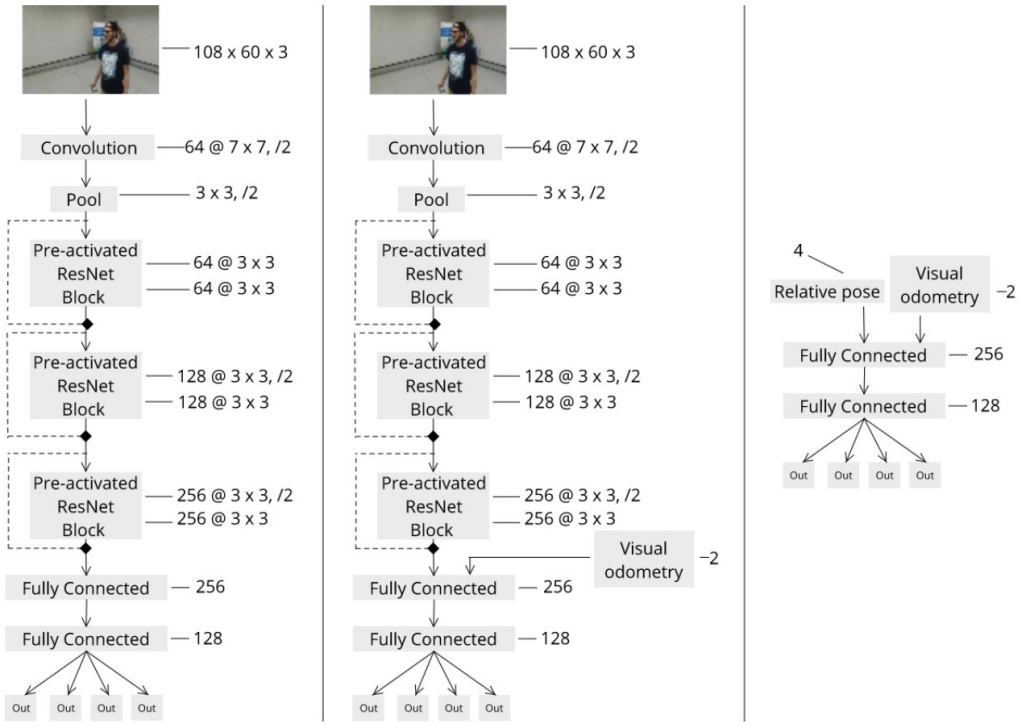


Figure A.1: Models by Mantegazza et al. [2019]: mediated, end-to-end, learned controlled

Please note that following architecture is for classification purposes presented in section 4.2.1. Standard model outputs have shapes `(?, 1)` instead.

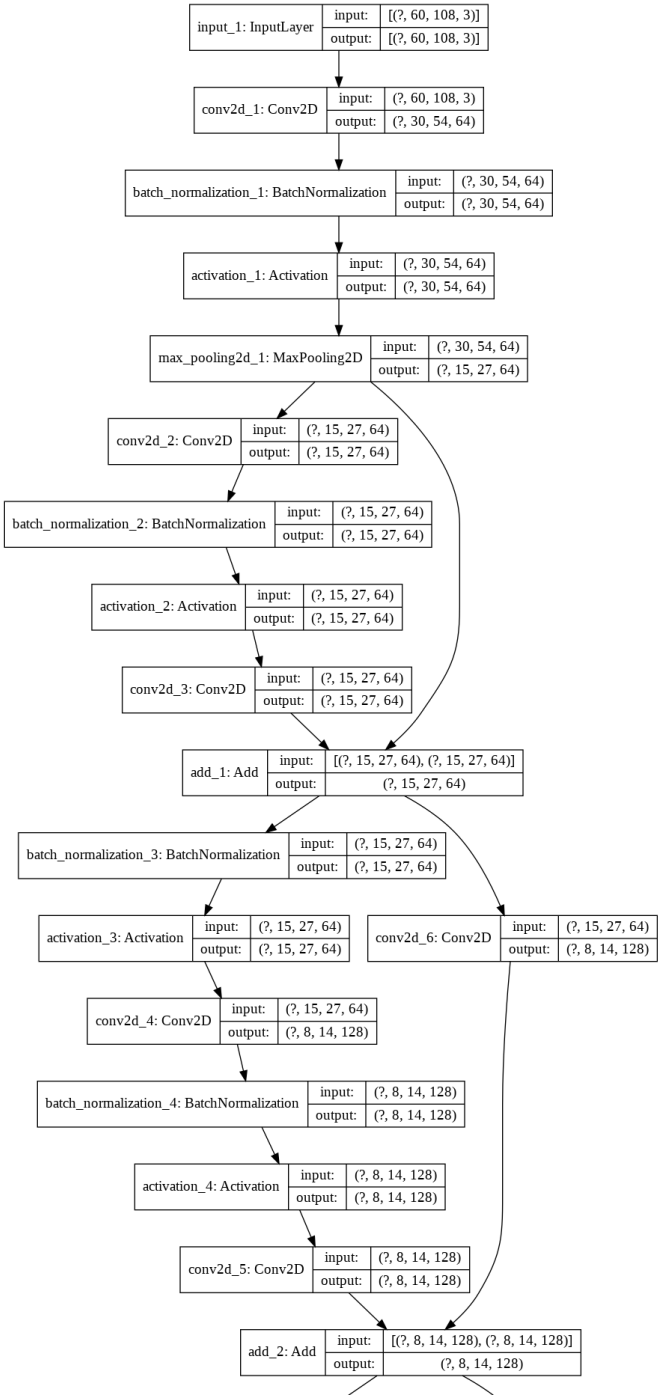


Figure A.2: FrontalNet complete architecture (part 1, from input to layer 18)

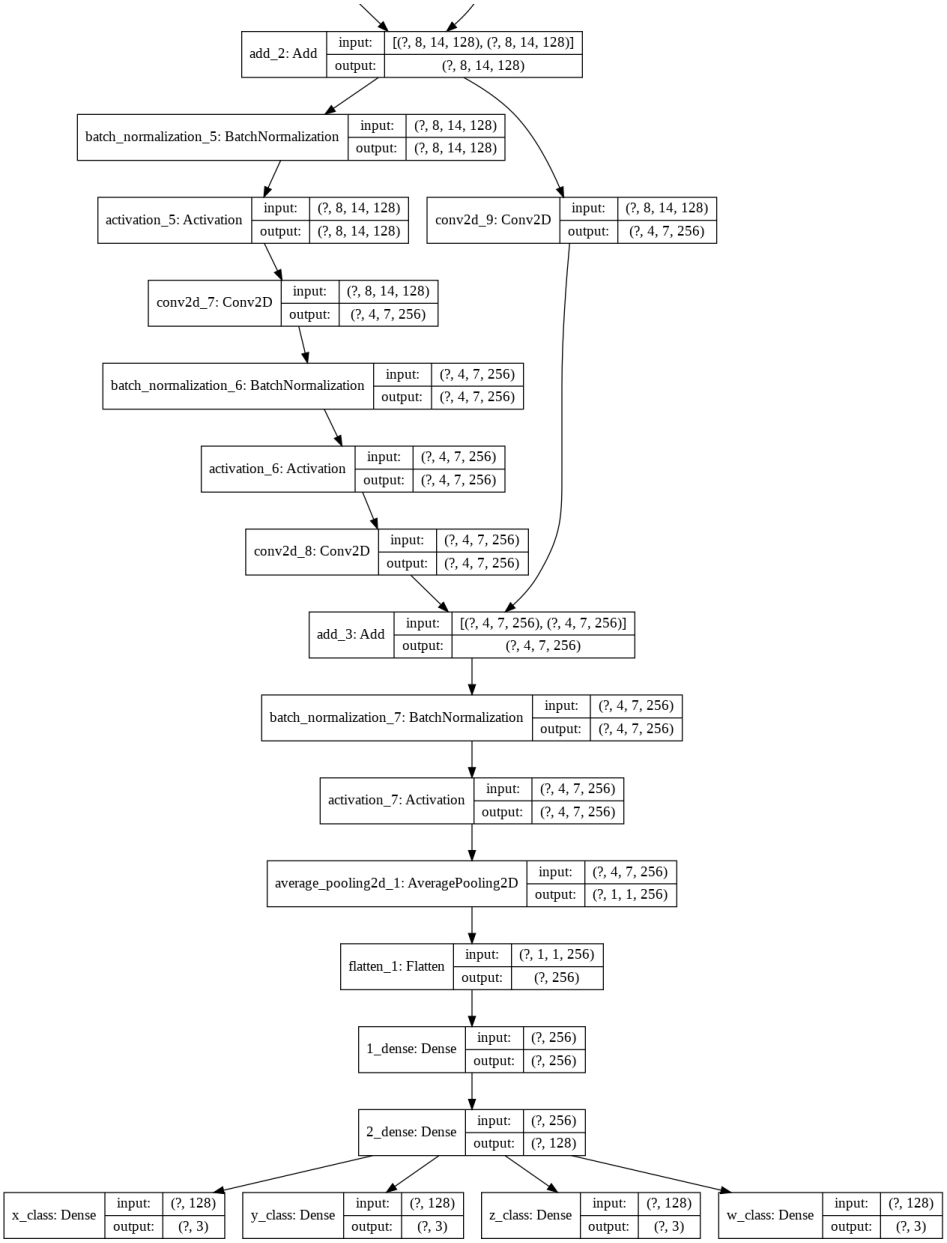


Figure A.3: FrontalNet complete architecture (part 2, from layer 18 to outputs)

Figure A.4, taken from Mantegazza et al. [2019], presents trajectories followed by the drone during five consecutive trials in the drone arena. The quadrotor has to face a user initially rotated by 90 degrees. Although the paths (obtained two distinct models A1 and A2) are sometimes different from what designed by the omniscient controller (the ground truth), they are still reasonable, and flying capabilities inside the drone arena seem very promising.

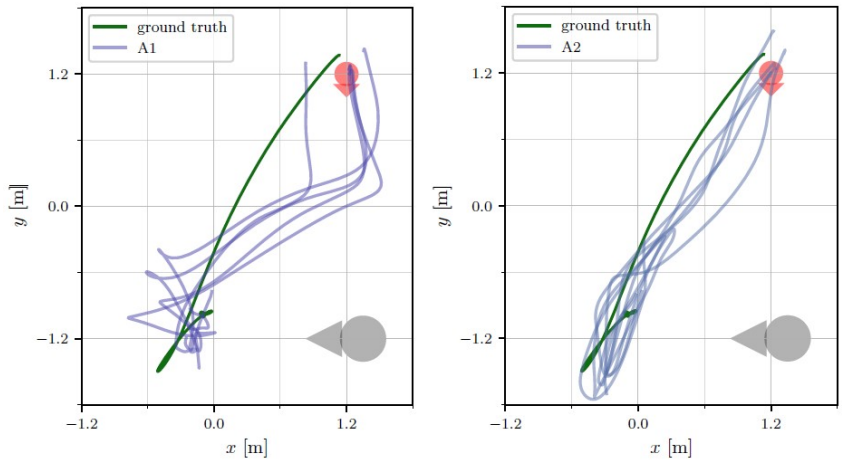


Figure A.4: FrontalNet trajectories for positioning in front of the user initially rotated by 90 degrees (Mantegazza et al. [2019])

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 4.2.4.

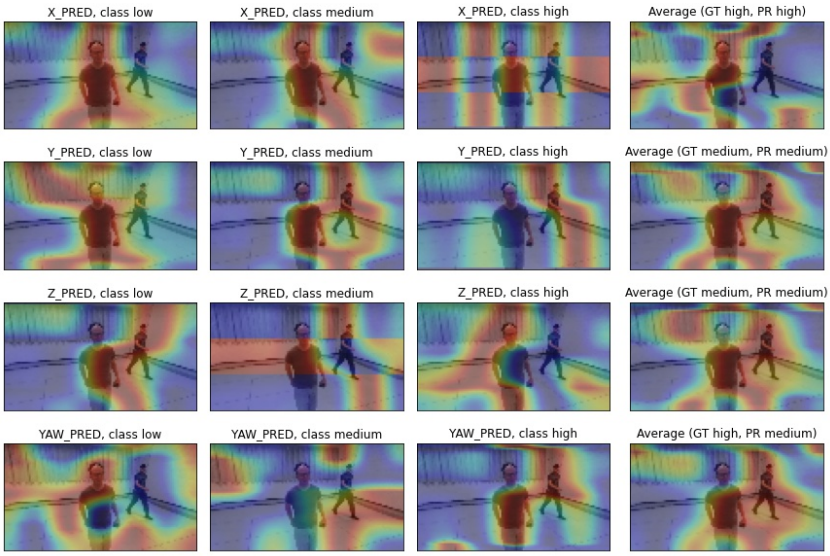


Figure A.5: Full Grad-CAM: two people in the frame

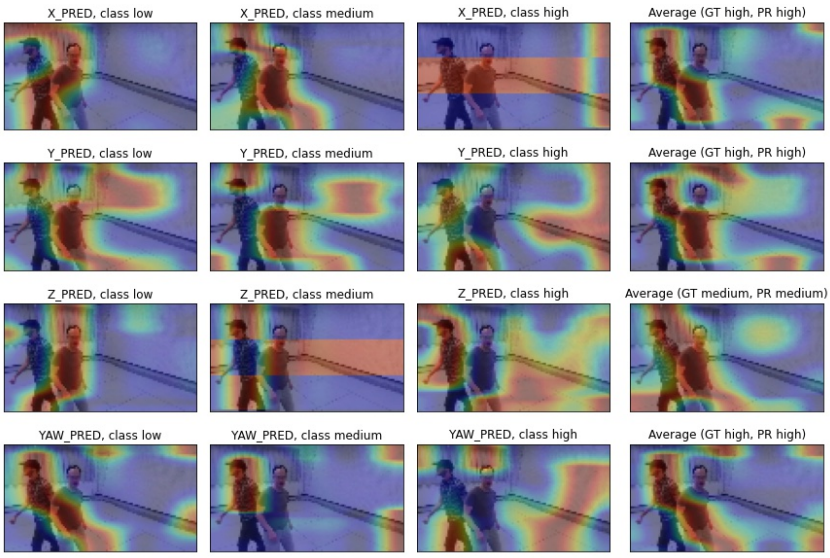


Figure A.6: Full Grad-CAM: two people in the frame

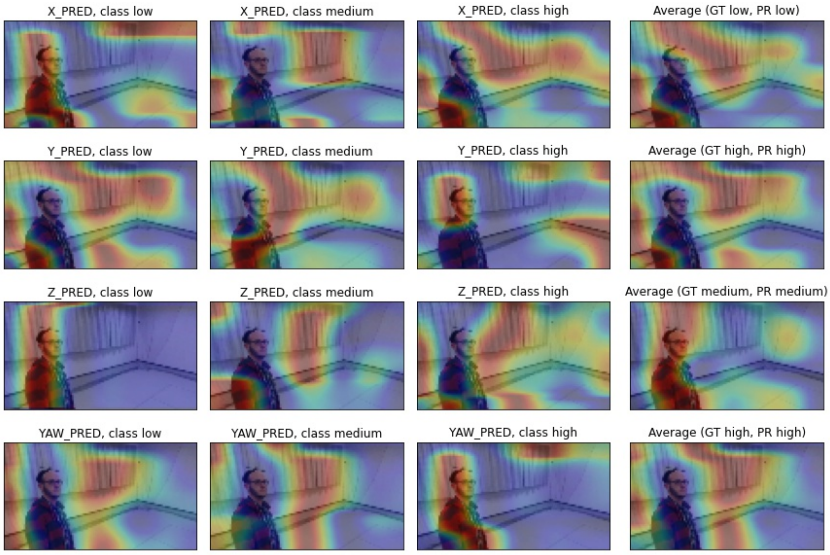


Figure A.7: Full Grad-CAM: model is attracted by curtains

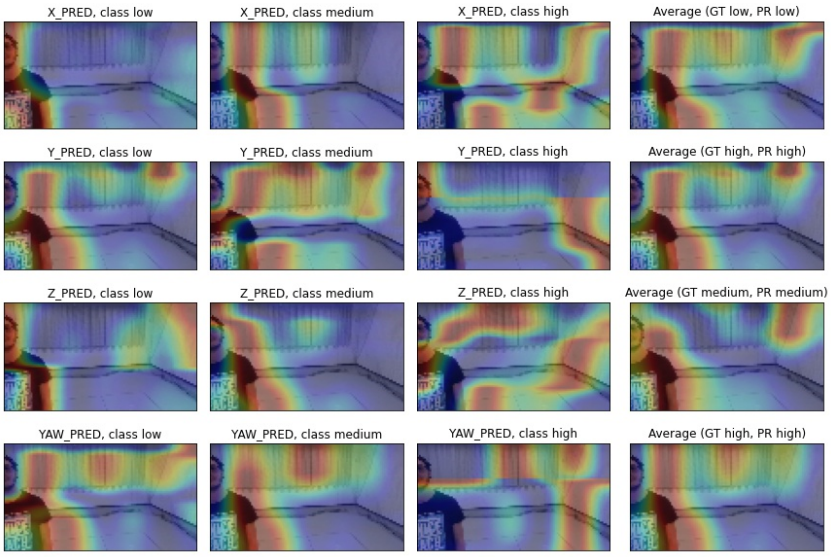


Figure A.8: Full Grad-CAM: model is attracted by curtains

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

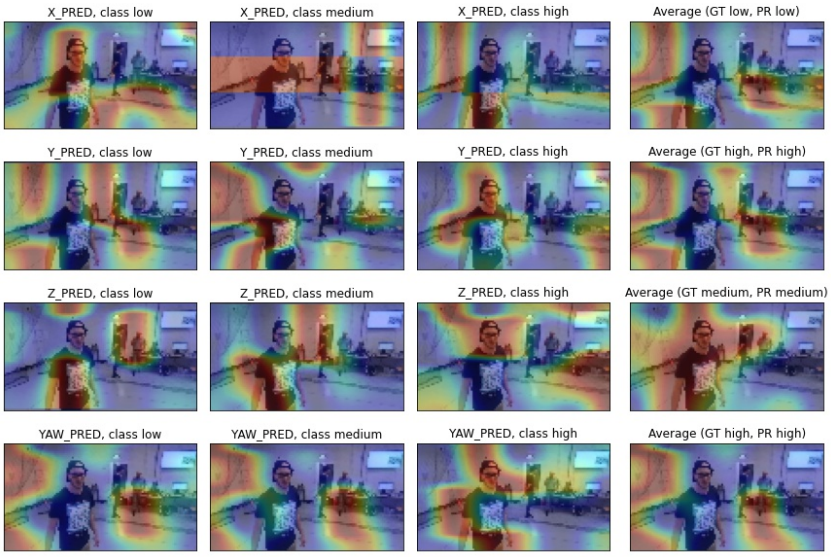


Figure A.10: Full Grad-CAM: model is attracted by background objects

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.

Albumentations. Fast and flexible image augmentation library. URL https://albumentations.ai/.

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. Visited on Jan 2021.

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

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. Visited on Jan 2021.

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2):125, Feb 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL http://dx.doi.org/10.3390/info11020125.

Mohamed Chetoui. Explanation of gradient-weighted class activation mapping, 2019. URL https://medium.com/@mohamedchetoui/grad-cam-gradient-weighted-class-activation-mapping-ffd72742243a. Visited on Jan 2021.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data, 2019.

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. Visited on May 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. Visited on Oct 2020.

Keras. Industry-strength deep learning framework built on top of tensorflow 2. URL https://keras.io/.

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. Visited on May 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. Visited on Jan 2021.

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015. URL https://cocodataset.org/.

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. Defining 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. Visited on Jan 2021.

Dario Mantegazza, Jérôme Guzzi, Luca Maria Gambardella, and Alessandro Giusti. Vision-based control of a quadrotor in user proximity: Mediated vs end-to-end learning approaches. *2019 IEEE International Conference on Robotics and Automation (ICRA)*, May 2019. doi: 10.1109/ICRA.2019.8794377. URL `https://github.com/idsia-robotics/proximity-quadrotor-learning`.

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.

Canny OpenCV. Canny edge detection tutorial, a. URL `https://docs.opencv.org/master/da/d22/tutorial_py_canny.html`. Visited on Sep 2020.

Grabcut OpenCV. Interactive foreground extraction using the grabcut algorithm, b. URL `https://docs.opencv.org/4.1.2/d8/d83/tutorial_py_grabcut.html`. Visited on Oct 2020.

OptiTrack. 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`.

PyTorch. Open source machine learning framework that accelerates the path from research prototyping to production deployment. URL https://pytorch.org/.

A. Quattoni and A. Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009. doi: 10.1109/CVPR.2009.5206537. URL http://web.mit.edu/torralba/www/indoor.html.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL https://pjreddie.com/darknet/yolo/.

ROS. 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.

Connor Shorten and Taghi Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019. doi: 10.1186/s40537-019-0197-0. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

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.

TensorBoard. Tensorflow's visualization toolkit. URL https://www.tensorflow.org/tensorboard.

TensorFlow. End-to-end open source platform for machine learning composed of a flexible ecosystem of tools, libraries and community resources. URL https://www.tensorflow.org/.

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. URL `https://github.com/cauchyturing/kaggle_diabetic_RAM`. Visited on Jan 2021.

Wikipedia. Perlin noise. URL `https://en.wikipedia.org/wiki/Perlin_noise`.

Wikipedia. XAI, 2021. URL `https://en.wikipedia.org/wiki/Explainable_artificial_intelligence`. Visited on Jan 2021.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019. Visited on Jan 2021.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training, 2020.

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`. Visited on Jan 2021.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017.

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

Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection, 2019.