A

shipout/backgroundshipout/foreground

B

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.

1.1 Objective

Lorem ipsum dolor sit amet.

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 first present a general overview of the topic, and then we focus on related works by IDSIA.

2.3.1 The State of the Art of Human–Drone Interaction

Drones attracted a lot of attention in the last years, and their usage is expected to keep growing, thanks to decreasing costs and the powerful features they can provide for personal, commercial, and social usage.

An article published in Nov 2019 for the IEEE Access [48] explores literature and state of the art for human-drone interaction. It reports that United States Federal Aviation Administration (FAA) expects total drone registrations to increase by more than 60% between 2018 and 2022. The main increment will be seen in the commercial sector, rather than the hobbyist one, even though the latter still counts the large majority of units. Nowadays, almost half of drone usage is for aerial photography (48%), followed by industrial inspection (28%) and agriculture (17%). In the next decade, the authors argue that drones will become ubiquitous to society and extensively used in advertising, shipping, sports, emergency, and many other fields for augmenting human capabilities.

Providing a nice overview of the latest works, the article highlights the most important challenges of today's research in human-drone interaction. The main studies in the area treat drones' control and human-machine communication, while the principal concerns regard the users' perception of safety during flight.

These topics are strictly related to our task, concerning autonomous flight in human proximity.

2.3.2 Vision-based Control of a Quadrotor in User Proximity

Our work is built upon the original master thesis [23] and paper [24] from Dario Mantegazza, named *Vision-based Control of a Quadrotor in User Proximity: Mediated vs. End-to-End Learning Approaches*, developed at IDSIA between 2018 and 2019.

In his thesis, the author proposes a machine learning technique for teaching a model to control a drone interacting with a person. In a few words, the goal is to continuously fly the drone to hover in front of a moving user. The problem is approached as a reactive control procedure and addressed with supervised learning. Thus, it provides an interesting starting point for many other robotics applications.

Training data is acquired by programmatically flying the drone to face a user frontally. In this phase, the quadrotor is controlled through an omniscient controller that knows both the drone's and person'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 that infers the relative user's position with respect to (wrt) the drone. In practice, 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 using spatial information in the input images.

In the paper, the author also compares the mediated approach described above and another end-to-end approach based on imitation learning. This second solution directly learns control signals[1] for the drone, instead of the user's pose. A third experiment also considers the task of automatically learning a controller that drives the drone based on the given user's position.

All the solutions provide very similar results, but for this thesis, we will only take into consideration the mediated approach. The reason behind this choice is that the learned model can be adapted to other tasks by simply designing a custom controller for the drone. It also provides a more transparent and analyzable solution.

Even though this kind of problem 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.

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

Our entire project makes use of the original network architecture and dataset defined in [24], respectively explored in sections 2.3.2.1 and 3.2, For a better understanding, the original code repository[2] and a descriptive video[3] are also available. To enhance readability, having no official name, the custom ResNet architecture proposed by the author will be called *FrontalNet*.

2.3.2.1 FrontalNet Architecture

The network comprises 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 [5] are used for all layers except for the output neurons, which are associated with a linear activation function [56].

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.

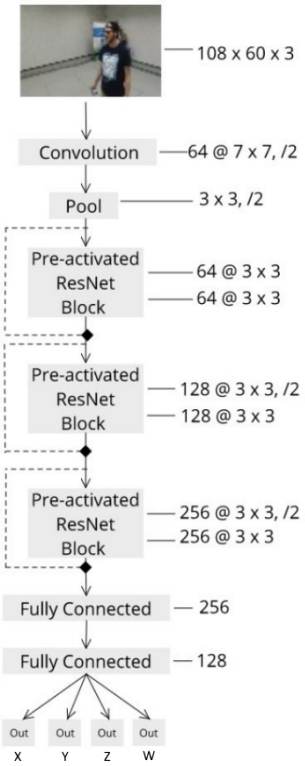


Figure 2.1: Schematic FrontalNet architecture [24]

[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 [17] 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 a patience of 10 epochs on the validation loss.

Performance is evaluated 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 $[-\infty, 1]$, where 1 represents the optimality.

The author also experiments with finding the minimum cardinality of the dataset to obtain acceptable performance. Results are available in figure 2.2, directly taken from the paper. As shown, at least 5,000 samples are required to achieve decent performance, which improves as the size of the training set increases.

Specifically, predictions seem more accurate for variables `Z` and `W` with an R^2 score of 0.82 and 0.88, respectively. Quite different the results for `X` and `Y` which only reach an R^2 of 0.59 and 0.57, respectively.

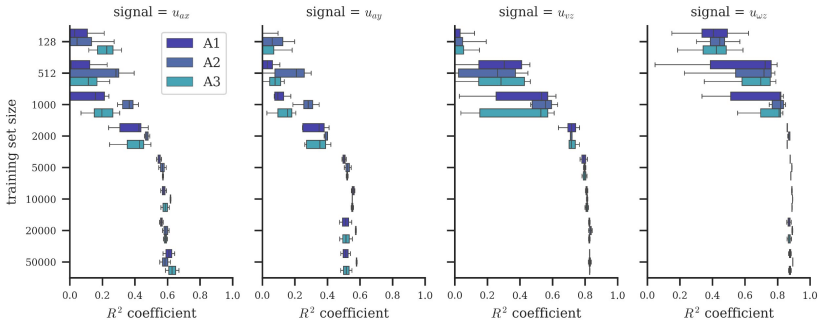


Figure 2.2: FrontalNet R^2 results [24]. A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

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

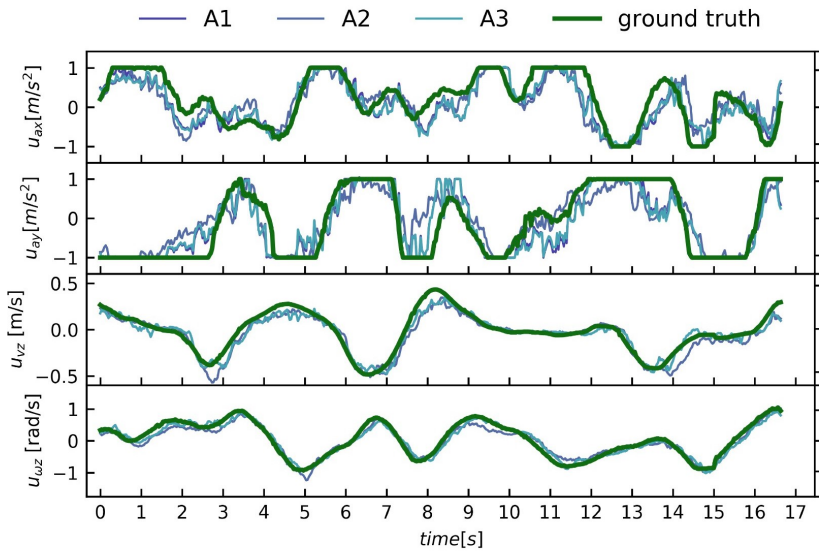


Figure 2.3: FrontalNet GT vs prediction results [24]. A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

2.3.3 Embedded Implementation of Controller for Nano-Drones

Autonomous navigation is an important and well-known research area 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 learning [13], where neural networks learn by imitating human behavior in specific tasks.

In 2018, researchers at the UZH University of Zürich have demonstrated that ResNets can provide satisfactory performance in the field. They developed DroNet [22], 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 [33], 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 without needing an external laptop to run the software.

Inspired by PULP-DroNet, IDSIA adapted its FrontalNet to work on-board the

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

Crazyflie with excellent results [58]. The nano-drone can 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

A Neural Network (NN) learns abstract representations for finding a mapping between its 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. Their reasoning and comprehension are intrinsic in the network parameters, which are nothing but numbers, particularly hard to understand even from domain experts.

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 [52]. 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 [27]. 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

TODO

Sources: [26]

2.4.2 Spatial Attribution with GradCAM

TODO

[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 transferring.

Sources: [41], [8]

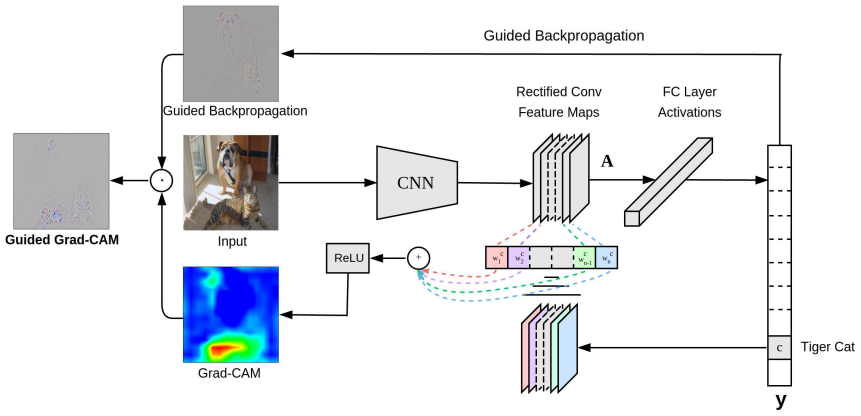


Figure 2.4: Grad-CAM schematic functioning [41]

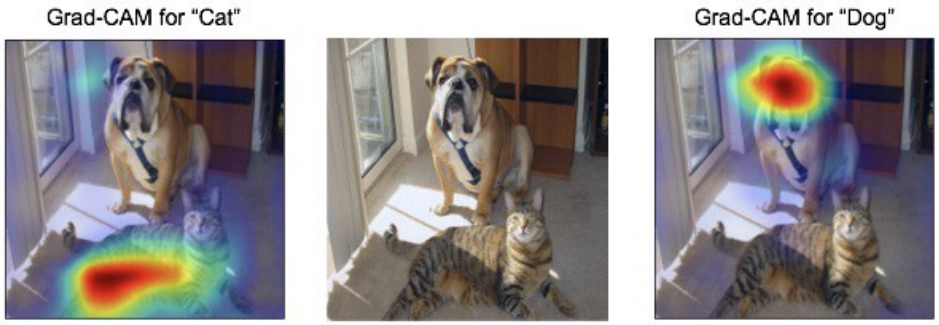


Figure 2.5: Grad-CAM example on dog-cat classification [41]

2.5 Network Generalization

TODO

2.5.1 Data Augmentation

TODO

Random Erasing Data Augmentation: [57]

Previous works on images: [55], [43], [54]

AutoAugment: [9] Learning Data Augmentation Strategies for Object Detection: [59]

Image transformations can be divided into two types:

- Spatial-level augmentations: also called affine transformations, they relocate 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 key points.

- Pixel-level augmentations: algorithms that 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 key points.

2.5.2 Domain Randomization

TODO

Summary from [25]

See [20], [49]

"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 [14] and object recognition [49] [19].

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. Even general-purpose graphic engines can be used, which includes 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, to collect the data very efficiently.

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

The virtual simulation gives both the opportunity to reproduce real indoor/outdoor scenes and also to randomize the background with artificially generated textures.

Even though the approach obtains auspicious 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 to work with an easier generalization pipeline that only concerns machine learning and can be easily reused for other tasks.

2.6 Human Detection and Segmentation

TODO

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. A post-production software then takes care of creating the appropriate mask, which separates the two parts and enables background replacement.

Although this technique is trendy 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 the development of Mantegazza et al. [24], by placing a green screen on a portion of the arena walls. Masking results were actually satisfying, but such a small green screen's limitations are huge both in terms of user's movements and background coverage. Figure 2.6 displays the setup, revealing a non-ignorable portion of the original background still appearing in the images.

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

2.6.2 Mask R-CNN

TODO

MaskRCNN: [12], [3], [4]

Figure 2.6: Experimental green screen setup in the drone arena

Chapter 3

System Description

We focus on improving the original work from Mantegazza et al. [24], 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 to enhance the model generalization capabilities.

3.1 Environment

Working from a machine learning perspective, our task does not require interaction 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 2 [34] (figure 3.1), a lightweight drone (500 grams) with a size of $382 \times 328 \times 89$ millimeters. A 2700 mAh swappable

[1] now relocated to Lugano

battery gives power to four brushless motors and a dual-core processor with a quad-core GPU, for a maximum flight time of 25 minutes. Connectivity is provided through 2.4 GHz 802.11a/b/n/ac WiFi 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 to ensure maximum stability during the whole flight. For this project, we mainly focus on its camera, which can 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°. Parrot's 3-axis digital stabilization technique is able to compensate for the drone's pitch and roll to provide correct-oriented horizontal images and stable videos regardless of the drone's movements.

3.1.2 OptiTrack

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

IDSIA adopt OptiTrack [31], 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 passive markers' movements, 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 can 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

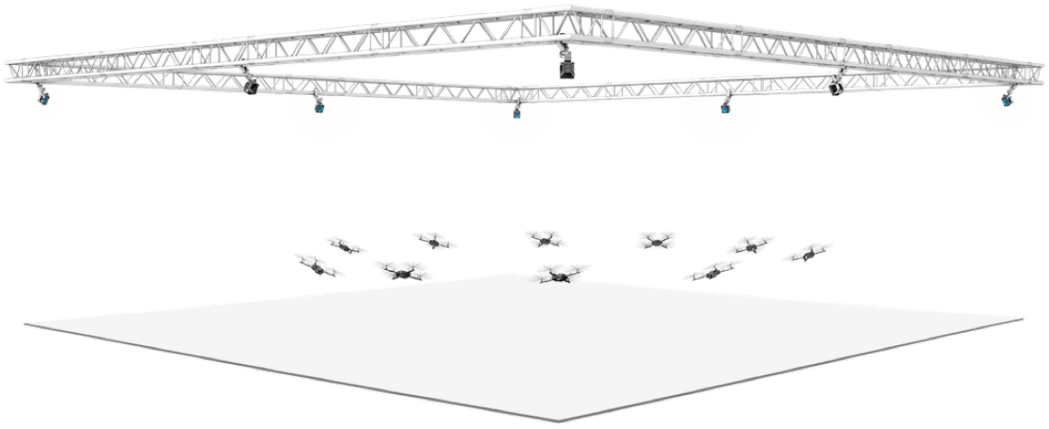
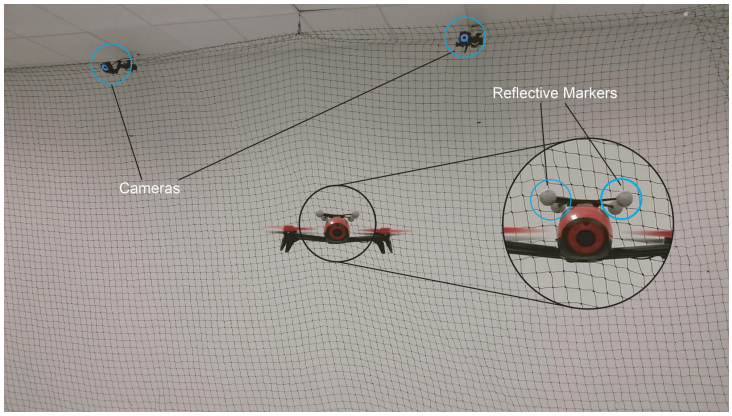


Figure 3.2: Schematic OptiTrack system with 12 OptiTrack Prime cameras

(a) Prime^{x}13 camera



(b) Drone with passive markers [23]

Figure 3.3: Drone arena at IDSIA

3.2 Dataset

This section refers to the dataset defined in Mantegazza et al. [24], which is also used for our research.

3.2.1 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 Robot Operating System (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 compute acceleration commands for the drone. The script is responsible for 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 the person's head (picture 3.4). The target poses over time[5], 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 [23]

3.2.2 Composition

Data collected inside the arena have been used to build the dataset for training a machine learning model 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 13 different people, which differ 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[6]. In total, 45 minutes of usable videos were used to compose the dataset, which counts about 63'000 and 11'000 frames for training and testing sets, respectively.

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); moreover, in some frames, no person is present because of particular movements sequences during which the drone actually lose the user (figure 3.8).

[5] mathematically computed by a script from [23]

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

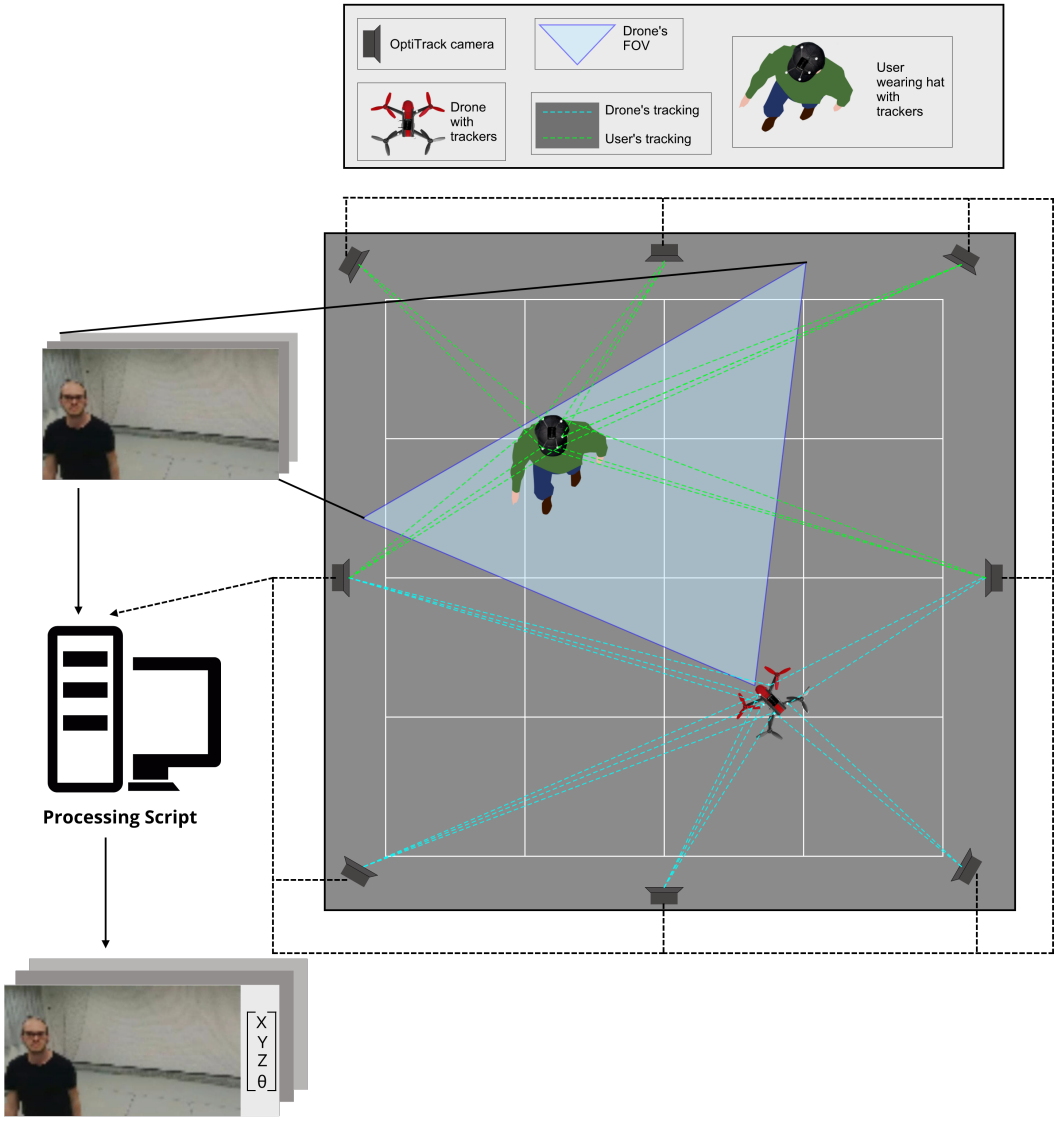


Figure 3.5: OptiTrack and data collection illustration [23]

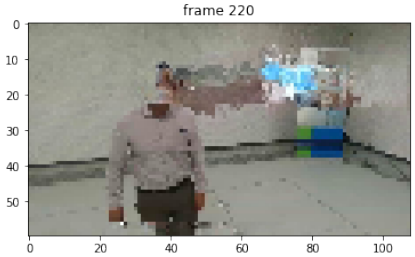


Figure 3.6: A frame with digital artifact caused by connection issues

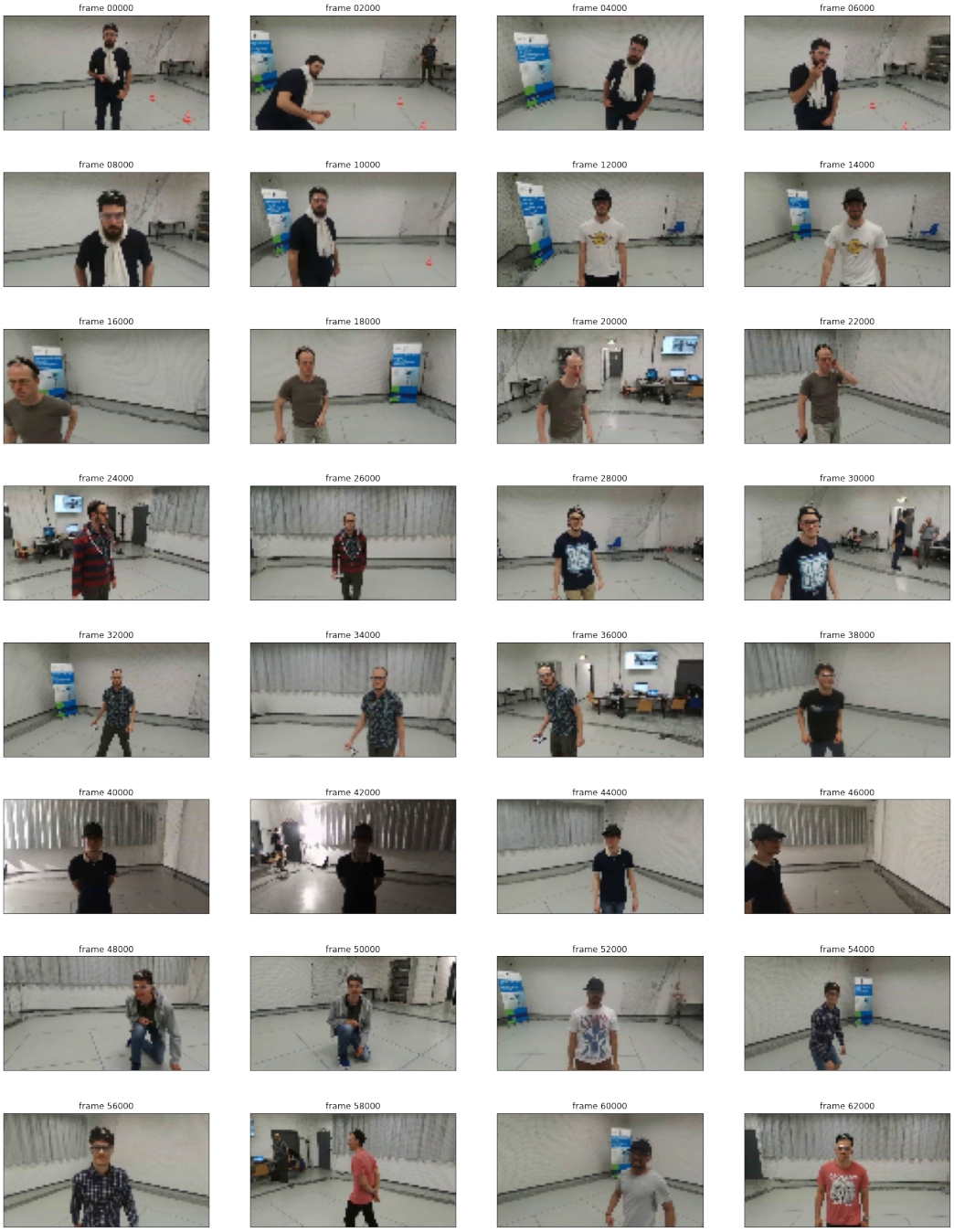


Figure 3.7: A complete overview of images in the training set

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 the 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.

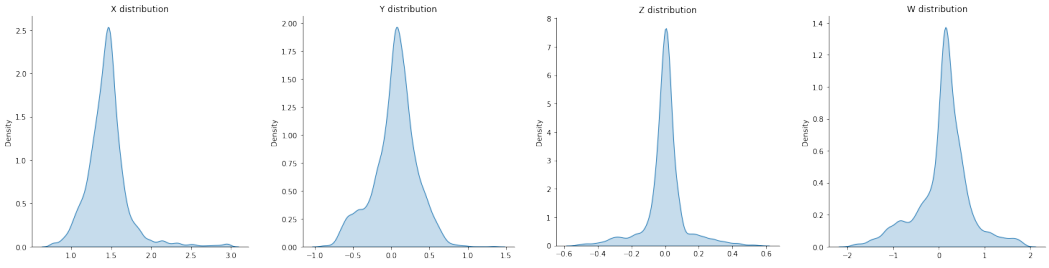


Figure 3.9: Target variables distribution for the regression task

3.3 Frameworks

This section presents tools and software used to conduct our research and improve the existing system. 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. Specific details about the hardware used for training the model are available in section 5.5.2.

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

`ROS` Even though not used in our work, it is crucial in Mantegazza et al. [24] to actually control the drone during flight. It will be used in the future for testing our model improvements on the real drone. ROS [39] is an open-source robotics middleware suite[7] 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 for real-time 3D visualizations and simulations.

`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 various kinds of figures. 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 [46] (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

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

networks. In version 2, it introduces many comforts for easier development with a less steep learning curve.

`Keras` Used for defining the network architecture, training, and evaluating the model. Keras [16] is the high-level API of TensorFlow 2: an approachable, highly-productive interface for solving machine learning problems, focusing 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 [45] 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 [2] efficiently implements a wide variety of image transform operations 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 with TensorFlow 2.0+, available on GitHub [18]. It is derived from the original `keras-vis` [11] developed by Google engineers, which is a toolkit for visualizing and debugging trained Keras neural network models.

`akTwelve Mask_RCNN` Used for human detection and segmentation in background replacement. It is an open-source implementation of Mask R-CNN built on Python 3, Keras, and TensorFlow 2 available on GitHub [15]. The model generates bounding boxes, segmentation masks, and categorization labels for each instance of an object in the image.

Chapter 4

Solution Design

This chapter explores the existing model's issues, 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 environment composition and presents the dataset used to train the machine learning model, as declared in Mantegazza et al. [24].

As explained in section 2.3.2.2, FrontalNet achieved quite good performance on the test set. Nevertheless, its behavior must be proven on the real drone to certify the model's usability. [24] 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 excellent, with the drone actually performing its task without any issues[1].

However, both quantitative and qualitative evaluations have been carried out using a set of images similar to the training one. Even though the model predictions were good with unknown users, they still move in the same drone arena where the training data have been collected. For a complete analysis, we must consider model performance in unknown environments.

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 appropriately, and its movements were unexplainable. We conclude that the Convolutional Neural Network (CNN) is not able to generalize the task when outside of the environment it already knows.

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

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

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 generalization capabilities by understanding what the model is actually learning.

Convolutional Neural Networks (CNN) are suited for computer vision because of their ability to extract spatial-related insights from images. As any other Neural Network (NN), even CNNs are "black-boxes". This means that their internal behavior is particularly challenging for humans to understand.

Among network interpretability techniques 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 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` [44], from the official TensorFlow team. Since Lucid does not provide native support with Keras models, we prefer to use instead the `tf-keras-vis` library [18] for TensorFlow 2.0+.

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 [50]), it appears to be an isolated case. 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. Figure 3.9 in the previous chapter shows their distribution. Every variable has a particular "central" value obtained when the user is centered in the image.

We decide to split continuous values into three 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

- `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 intervals have been defined to have a uniform class distribution over the training set (figure 4.1).

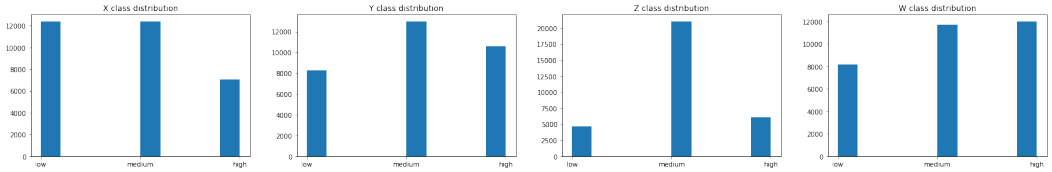


Figure 4.1: Target variables distribution for the classification task

4.2.2 Re-training

Considering the new ground truth, we define a new model architecture by replacing regression outputs with classification ones. We completely re-train the new model, by using the `categorical_crossentropy` loss and the `accuracy` metric, both suited for multi-class problems. As the original FrontalNet model, we opt for an ADAM optimizer and a base learning rate of 0.001, progressively reduced on validation loss plateaus.

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

4.2.3 Interpretation Issues

A proper understanding of Grad-CAM results requires a thorough ability on reading the following charts[3].

To run Grad-CAM, it is required to specify a class for which to compute the respective activation mapping. In our case, the class would be one of the three defined in section 4.2.1: `low`, `medium` or `high`.

[3]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`

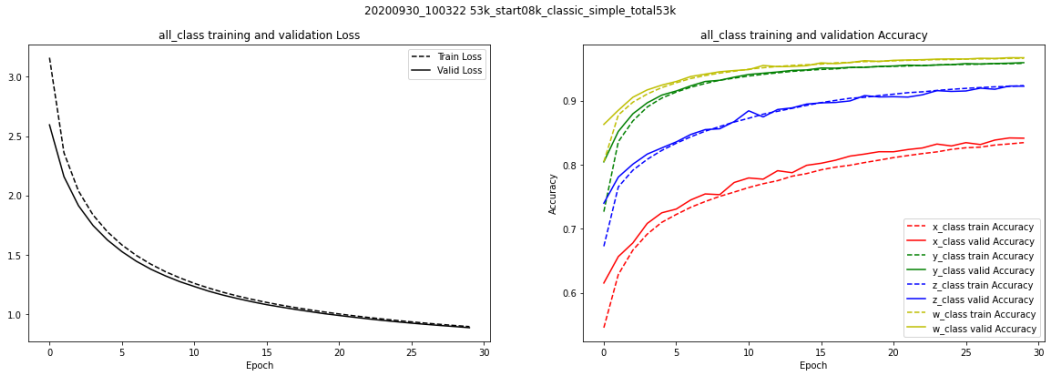


Figure 4.2: Grad-CAM: loss and accuracy of the new classification model

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 overlays 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 since it has been adapted from a multi-output regression task. Classes only serve as categorical values for variables that are actually numerical, and this can introduce some issues when analyzing Grad-CAM visualizations.

A For a certain input image, there is no particular portion of the image that can be strictly associated with one or another class in particular. Therefore, assuming to have a correctly trained model and considering a single image, we do not expect the heatmap generated by Grad-CAM for a certain class (e.g., `low`) to be different from heatmaps generated for the other classes (e.g., `medium` and `high`). The discriminator for the model's predictions must always be the user; thus, also Grad-CAM should only focus on the user.

B As the regression task, also the classification one predicts four variables that correspond to the user's coordinates. A multi-output network further introduces another complexity on reading Grad-CAM results since the algorithm can be run differently on each variable, producing different heatmaps. Once again, if the model is correct, we expect that Grad-CAM will produce the same output regardless of the inspected variable (i.e., it always highlights the user in the images).

C In some cases, the network predictions for one or more variables are different from the actual value (i.e., the ground truth). When examining Grad-CAM results corresponding to erroneously estimated values, both the predicted and

the actual class have to be considered to understand better what is going wrong with the model.

Analyzing Grad-CAM results, we want to confirm the hypothesis made in A and B. If heatmaps mainly overlay people in the images, we can conclude that the CNN can effectively understand the concept of a person and produce its outputs based on the user's position only. Otherwise, if Grad-CAM reveals that other parts of the images are considered important by the model, then we realize that the predictions are driven by undesired factors (e.g., objects in the background).

Figure 4.3 displays a typical example on Grad-CAM application with a thorough division in variables (`X, Y, Z, W`) and classes (`low, medium, high`).

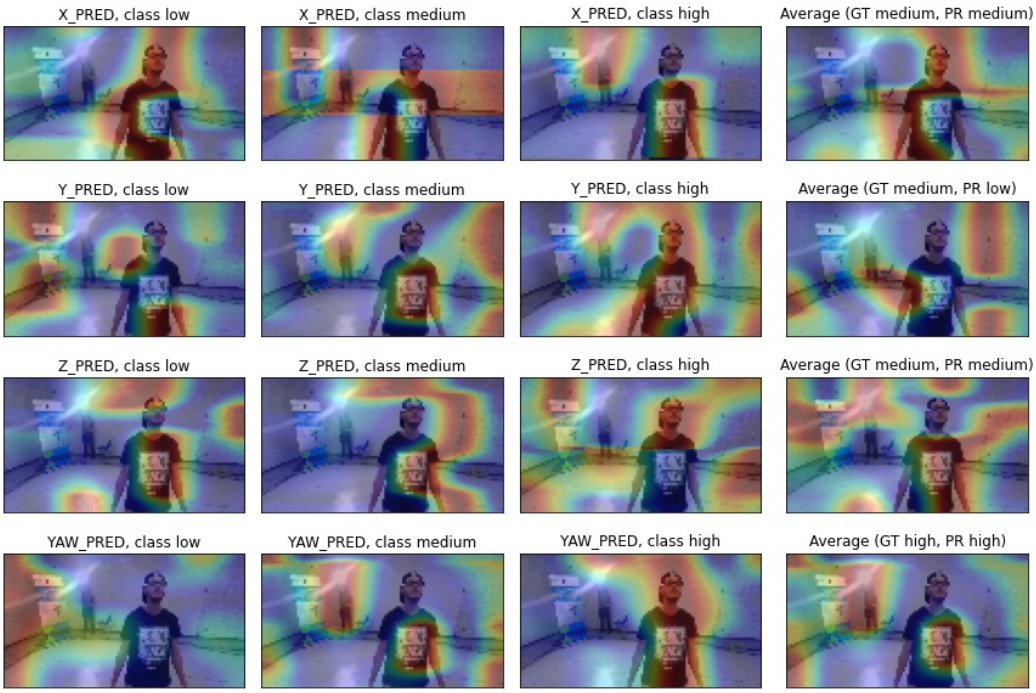


Figure 4.3: Grad-CAM: example of application for each variable and class

Heatmaps are not easy to interpret due to a large variety of parameters to consider. As a guideline, it is possible to only consider, for each variable, the column which corresponds to the predicted class (or the actual, if it differs). Actual 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 in Grad-CAM results is available between variables, classes, and computed predictions.

By calling Grad-CAM without specifying a particular class, it is also possible to obtain an average result on all classes, shown in the last column. Applying

similar reasoning also on variables, we can produce a single meaningful image. This represents a Grad-CAM global average, which is computed on every variable and class at once (figure 4.4).

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

4.2.4 Results

As already shown in figure 4.3, it seems that 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 plot the global average Grad-CAM instead. For simplicity, this section will only focus on single-image results. However, full Grad-CAM visualizations are available in the appendix A.2 for further inspection.

Reasonable detections

Figure 4.5 displays examples of correctly working scenarios in which the person is well-detected by Grad-CAM. In these situations, it often happens that the entire user is highlighted, but sometimes only the body or the head gets proper attention.

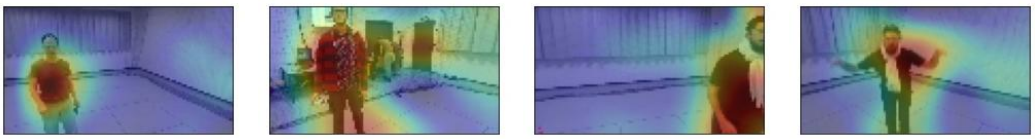


Figure 4.5: Grad-CAM: Correctly detected people

Such precise results are not the standard. In many cases, the network focus is unstable, and the heatmaps frequently go in and out of the target person. The two sequences of frames shown in figures 4.6 and 4.7 fairly describe the usual behavior of the model seen by Grad-CAM.

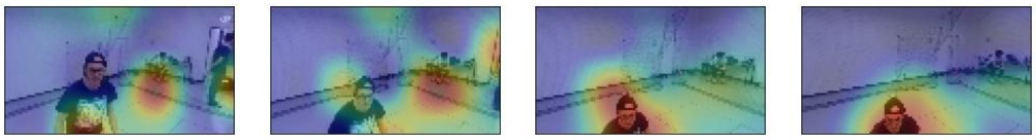


Figure 4.6: Grad-CAM: Sequence transitioning from wrong to correct detections

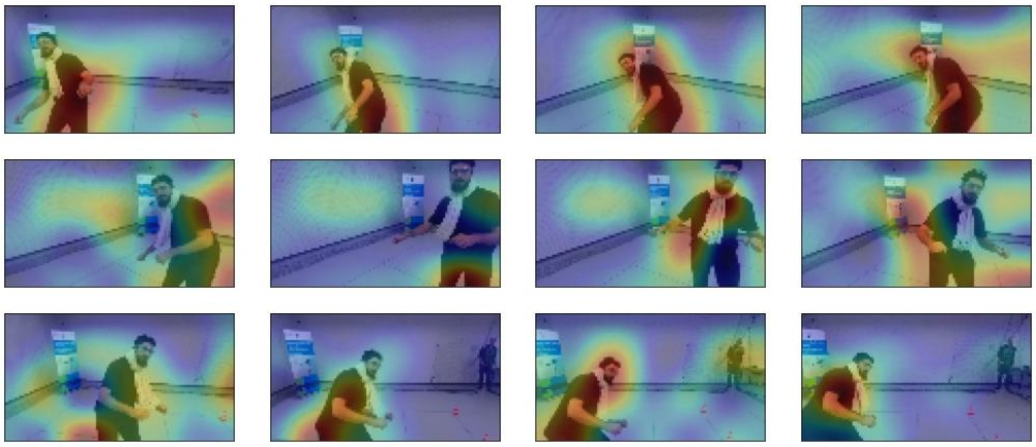


Figure 4.7: Grad-CAM: Sequence of unstable detections going in and out of the person

Problematic detections

The examples presented above mostly reflect the model expected behavior. However, our network interpretation also reveals many flaws in the prediction task. Grad-CAM exhibits several situations in which the model output is affected by recurrent elements in the dataset.

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

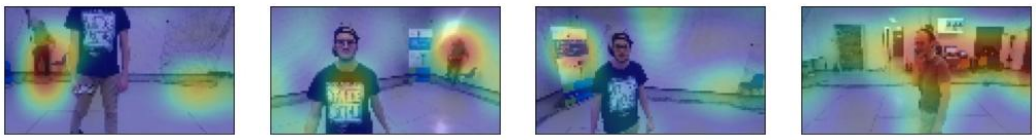


Figure 4.8: Grad-CAM: Objects in the background detected

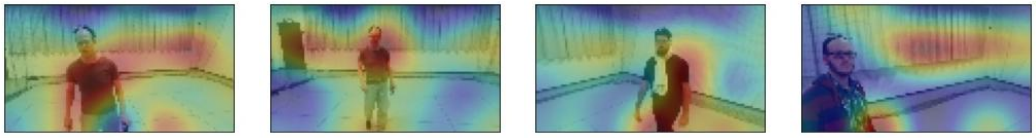


Figure 4.9: Grad-CAM: Curtains often distract the model

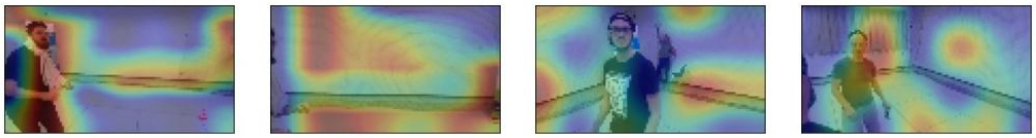


Figure 4.10: Grad-CAM: Model get easily distracted by various elements

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

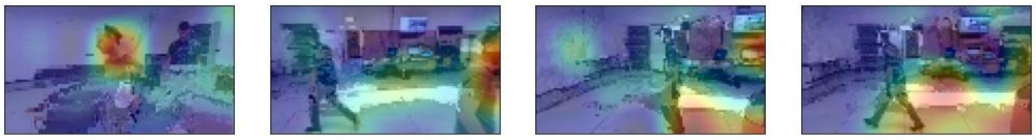


Figure 4.12: Grad-CAM: Model reactions to artificial glitches

4.2.5 Summary

Grad-CAM 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 has undesirably learned some details about the drone arena in which the dataset has been collected. The issue is common with supervised learning, and it is called overfitting. It happens when a ML model learns much information that only belongs to the training set, being not able to properly work later on previously unseen data.

This is most likely the reason why the model is unable to control the drone outside of the arena, as previously 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 would like to remove this factor (i.e., the room itself) from the equation.

We propose a solution, inspired by domain randomization, that consists of performing advanced data augmentation on the training data. Considering the images that compose our dataset, we only keep the user facing the drone and remove the background by randomly replacing it with something else. This section explores various algorithms for creating the mask of a person in an image. After several experiments, the solution we have decided to adopt is Mask R-CNN (section 4.3.3).

4.3.1 Canny

The first experiments are based on a classic computer vision technique called Canny Edge Detection [7]. A custom algorithm[4] applies the related function from OpenCV [29] 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, in order to find appropriate contours, is the choice of its parameters `minVal` and `maxVal`. These are used by the algorithm for distinguishing between *sure-edges*, *probable-edges* and *no-edges*. Several experiments have been done 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.

In some cases, it even happens that the person's body is present in the image while the face disappears. For mitigating this problem, an enhanced version of the algorithm has been considered. This version is 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 illustrates the problem and demonstrates that results obtained from the enhanced version are still not acceptable since we cannot appropriately remove the scene's background.

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

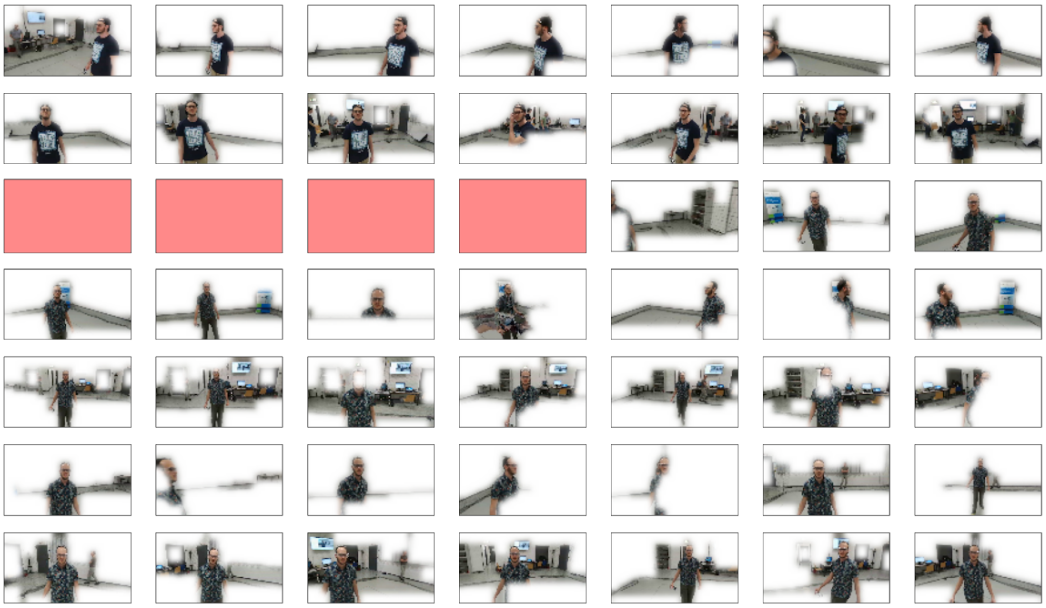


Figure 4.13: Canny edge detection overview on the training set

Figure 4.14: Canny edge enhanced algorithm demonstration

4.3.2 GrabCut

GrabCut [40] operates by using the subject position in the image and some statistical inference for labeling each pixel of the image as background or foreground.

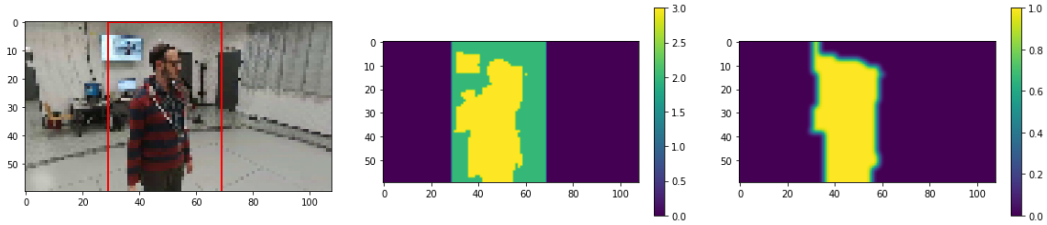
The base algorithm [30] requires a bounding box as an input, a rectangle to enclose the subject to be segmented. Everything outside this rectangle will be taken as *sure-background*; everything inside the rectangle is unknown. The image is first transformed into a graph, then iteratively processed using a Gaussian Mixture Model [38] for color-based statistical labeling. A min-cut algorithm is used to segment the graph until convergence.

OpenCV GrabCut implementation has two initialization modalities. It is possible to pass only the rectangle, as explained before, or a mask of the image which further specifies whether a certain pixel is *sure-background*, *probable-background*, *probable-foreground* or *sure-foreground*. The library also uses this categorization during the algorithm itself.

Both approaches require previous knowledge about the subject position in the image. Our initial experiments assume that such information is given. In section 4.3.2.4, we will consider an automatic human detection algorithm.

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 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.

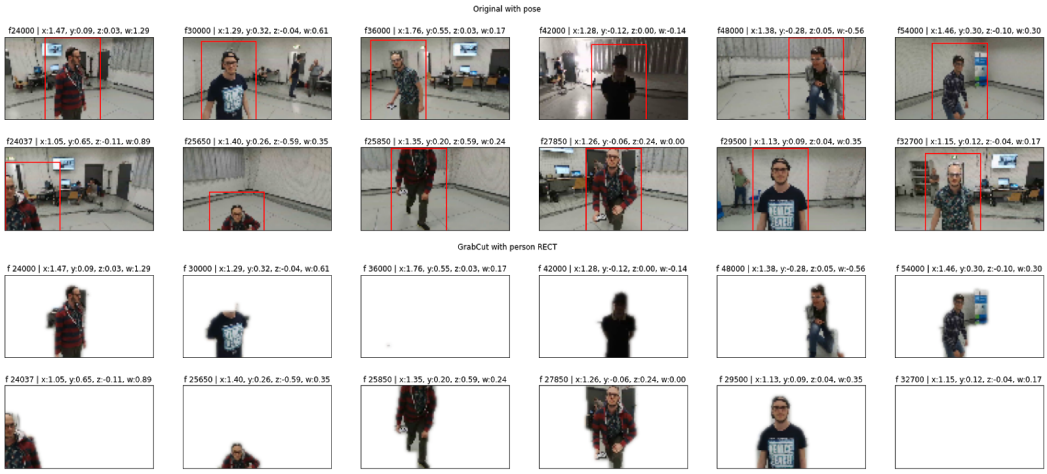
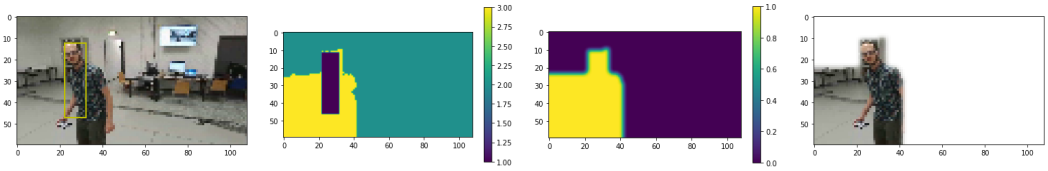


Figure 4.16: Grabcut demonstration: rectangle initialization

4.3.2.2 Mask initialization

For advanced purposes, OpenCV also gives the possibility to initially classify the pixels with a mask. We choose this methodology for setting the face and part of the body as *sure-foreground* from the beginning.

For demonstration, we consider an image for which the rectangle initialization above was completely missing the person in the result. Procedure is shown in figure 4.17. We start by specifying *sure-foreground* pixels (blue), then GrabCut automatically infers *probable-foreground* (yellow) and *probable-background* (green) areas. Results are undoubtedly better, but we notice that the left-most background has been kept in the final image. However, this part of the image could have been easily identified as *sure-background* by using the person pose we assume as known, as we did for rectangle initialization.



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

Figure 4.17: Grabcut algorithm explained: mask initialization

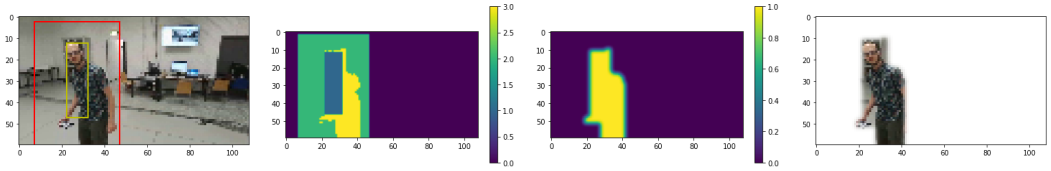
4.3.2.3 Hybrid initialization

Finally, a mixed approach between rectangle and mask initializations has been tried. It allows to specify both the person and the face positions in the image,

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

In figure 4.18 is available a step-by-step explanation like the ones presented above. In there, *sure-background* is dark blue, *probable-background* is green, *probable-foreground* is yellow and *sure-foreground* is light blue.

Image 4.19 shows the results of hybrid initialization applied to the same samples introduced in figure 4.16. The segmentation is very precise, and the approach reveals to be almost optimal.



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

Figure 4.18: Grabcut algorithm explained: hybrid initialization

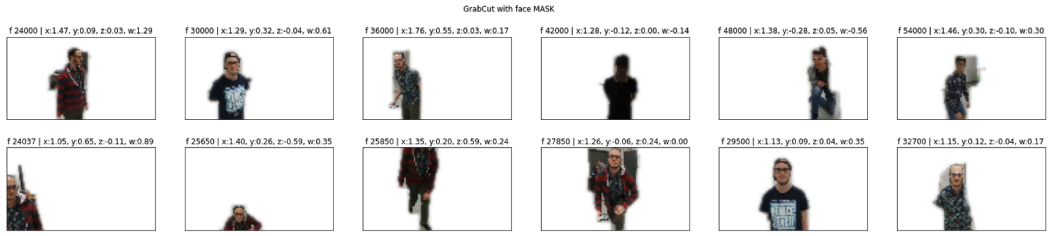


Figure 4.19: Grabcut demonstration: hybrid initialization

4.3.2.4 Automatic Human Detection

The approaches presented until now are used for doing human segmentation, which is an essential step for performing background removal. To work, GrabCut hybrid initialization requires two pieces of information: the bounding boxes associated with both the entire person and its head.

Given the available data, the mapping between the user's pose (the ground truth (GT)) and its appearing in an image is unknown. Thus, proper detection algorithms are required on the top of the segmentation task to find both the person and its face in an image.

For detecting the user we try YOLO [37], a state of the art technique for object detection. We decide to adopt the `cvlib` library [10], which underneath uses the OpenCV dnn module [28] for implementing a YOLOv3 model trained on the Microsoft COCO dataset [21].

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.

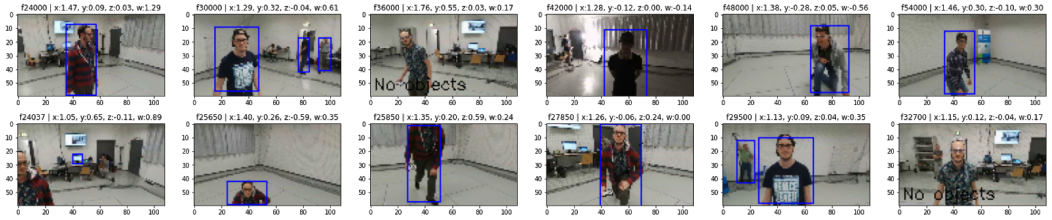


Figure 4.20: YOLO demonstration, which shows failures for 2 images

The same problem also affects face detection, which is needed for providing the related mask to the GrabCut algorithm. An open-source head detector has been tried[5]. Results are poor, once again due to the small size of our images.

4.3.3 Mask R-CNN

Mask R-CNN [12] 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. Initially developed by Facebook researchers in PyTorch [35], the algorithm has been ported on TensorFlow 1 [1] and later adapted for TensorFlow 2 [15][6].

Mask R-CNN results on our dataset are incredibly precise, and the method undoubtedly outperforms any other previously experimented since it provides 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, multiple people or objects in the background are correctly detected, even if they are tiny in the images.

This high-level of accuracy in detection and segmentation comes with an extremely-high computing power requirement[7]. For reference, running Mask R-CNN on the test set - composed of about 11'000 images - requires a total computing time[8] of approximately 55 minutes on Google Colab using a GPU runtime[9].

Because of this, the inference on the images must be made offline. Together with each input image, the training procedure will only receive the previously computed user's mask. This will be used to perform background replacement.

[5] `https://github.com/AVAuco/ssd_head_keras`

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

[7] according to the original paper, Mask R-CNN runs at 5 FPS on Nvidia Tesla M40 GPU

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

[9] equipped with Nvidia T4 GPU

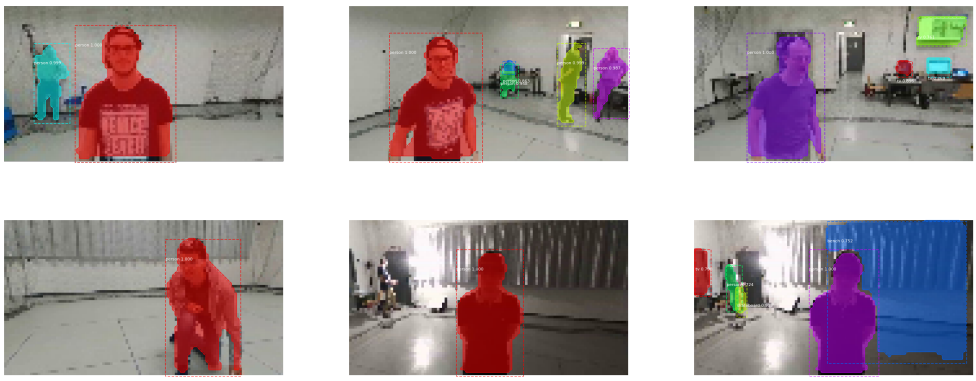


Figure 4.21: Mask R-CNN applied to our training set

Chapter 5

Model Implementation

In this chapter, we explain how the proposed solution has been implemented.

Firstly, we focus on the implementation of our generalization strategy, mainly concerning the way the dataset is treated and processed to reduce overfitting. Then, section 5.3 provides three model alternatives that will be considered for evaluation and comparison in chapter 6. Finally, we provide technical details about the training procedure, with a particular focus on timing performance.

5.1 Background Replacement

As demonstrated in section 4.2.4, the approach defined by Mantegazza et al. [24] 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. 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 pre-process 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 belongs (e.g., person, TV, bike, car, etc.). However, for our purposes, we are only interested in the mask corresponding to the user who is actually facing the drone. Since it 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, for each sample, also the corresponding user's mask. This is used to distinguish the subject from the rest of the image and 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 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 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 [36], 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`

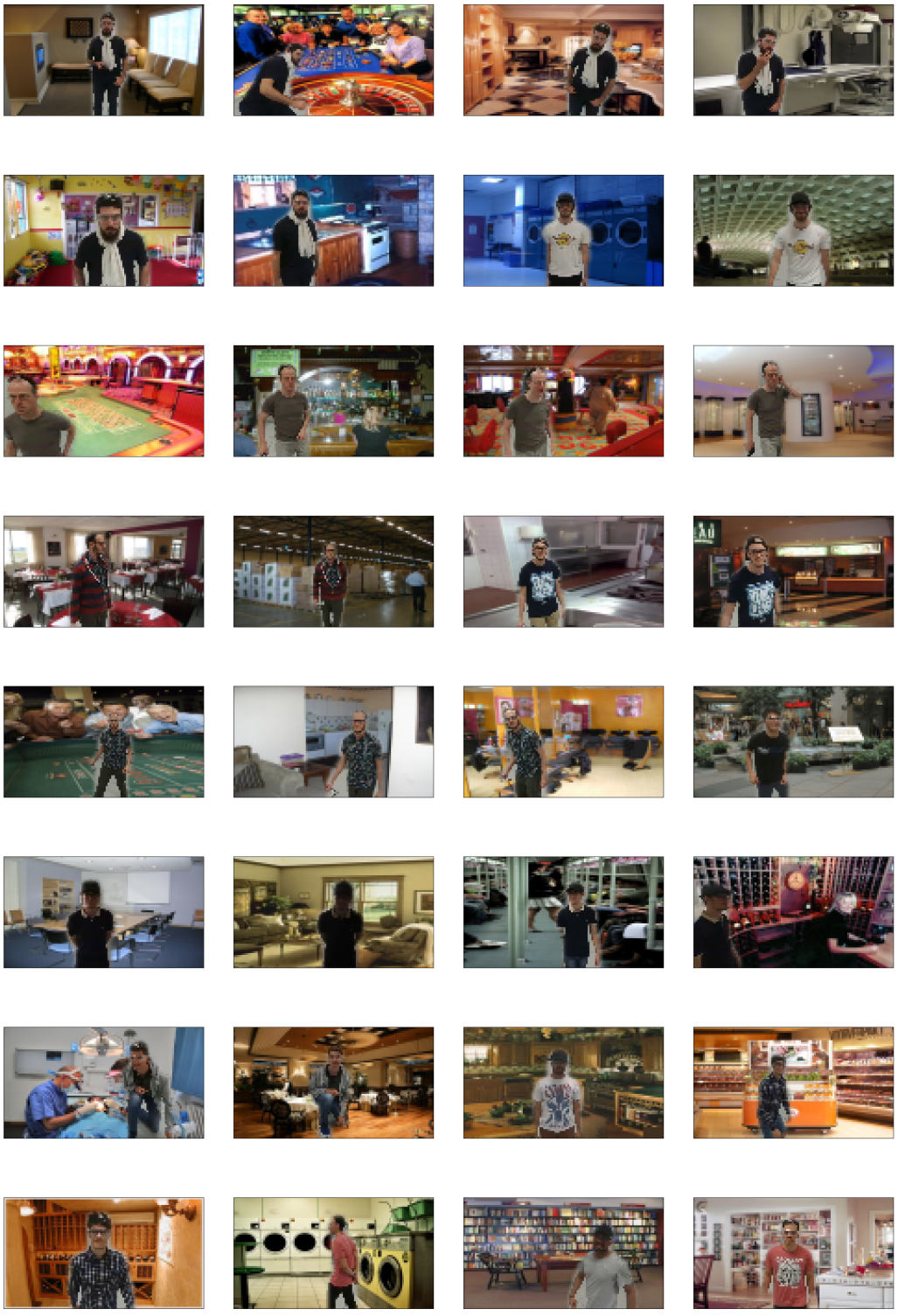


Figure 5.1: Example of background replacement on the training set

5.2 Image Augmentation

Data augmentation is a common regularization technique for improving neural networks' ability to generalize their task on unknown data. 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 [6], a state of the art Python library that provides a huge variety of image augmentations. Constantly updated and well-documented, Albumentations can boast the best benchmarking performance in the field[2].

FrontalNet learns to predict the user's pose, but the relationship 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 (see section 2.5.1) is not an option in our case. Instead, we mainly apply pixel-level augmentations.

The only exception is horizontal flipping, which only requires inverting the `Y` coordinate in the ground truth. As explained in section 3.2.2, the `Y` coordinate represents the horizontal alignment of the user wrt the drone. Figure 4.1, in the previous chapter, shows a propensity for the user to be on the right-part of the images. For this reason, we decide to mirror the camera's frames horizontally with a probability of 50%.

According to user-defined probabilities, Albumentations allows applying 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. Those results combine transformations on brightness and contrast, multiplicative noise, channels manipulation, and rectangular dropouts.

Figure 5.2: Example of image augmentation with Albumentations

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

The combination of different transformations composes a pipeline, whose time performance depends on custom choices and probabilities. Among all tested[3] augmentations, most of them only require a maximum of 0.7 seconds to be applied on a set of 10'000 60×108 images, while the most expensive ones take up to 12 seconds under the same conditions[4].

The Albumentations pipeline adopted for our work is shown in listing 5.1. It takes about 4 seconds to process 10'000 images and accepts a parameter `aug_prob` to define the prior probability of actually applying the augmentations to an input image. Some examples of resulting images are available in figure 5.3.

```
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, elementwise, 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

Furthermore, injecting noise into images can greatly help CNNs on avoiding overfitting [42]. For this reason, at the end of the pipeline, we also apply Perlin noise [51] with a probability of 20%. Its generation is highly time-consuming, then a set of textures has been pre-computed. During training, for each augmented sample, one of the generated Perlin noises is randomly chosen, cropped, flipped, and finally applied to the input image. Half of the time, the noise is multiplied

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

[4] Benchmarks (in seconds): InvertImg 0.2; ToGray 0.2; ChannelShuffle 0.3; ChannelDropout 0.4; Blur 0.5; RandomGamma 0.5; RandomBrightnessContrast 0.6; Solarize 0.6; Equalize 0.7; MotionBlur: 0.7; HueSaturation 1.5 sec; RGBShift 1.5 sec; CLAHE 3.5 sec; CoarseDropout 3.5 sec; MultiplicativeNoise 7 sec; GaussNoise 10 sec; ISONoise 12 sec

uniformly on all channels to produce a grayscale texture, while the other half is differentiated over RGB channels. Picture 5.4 illustrates both cases.

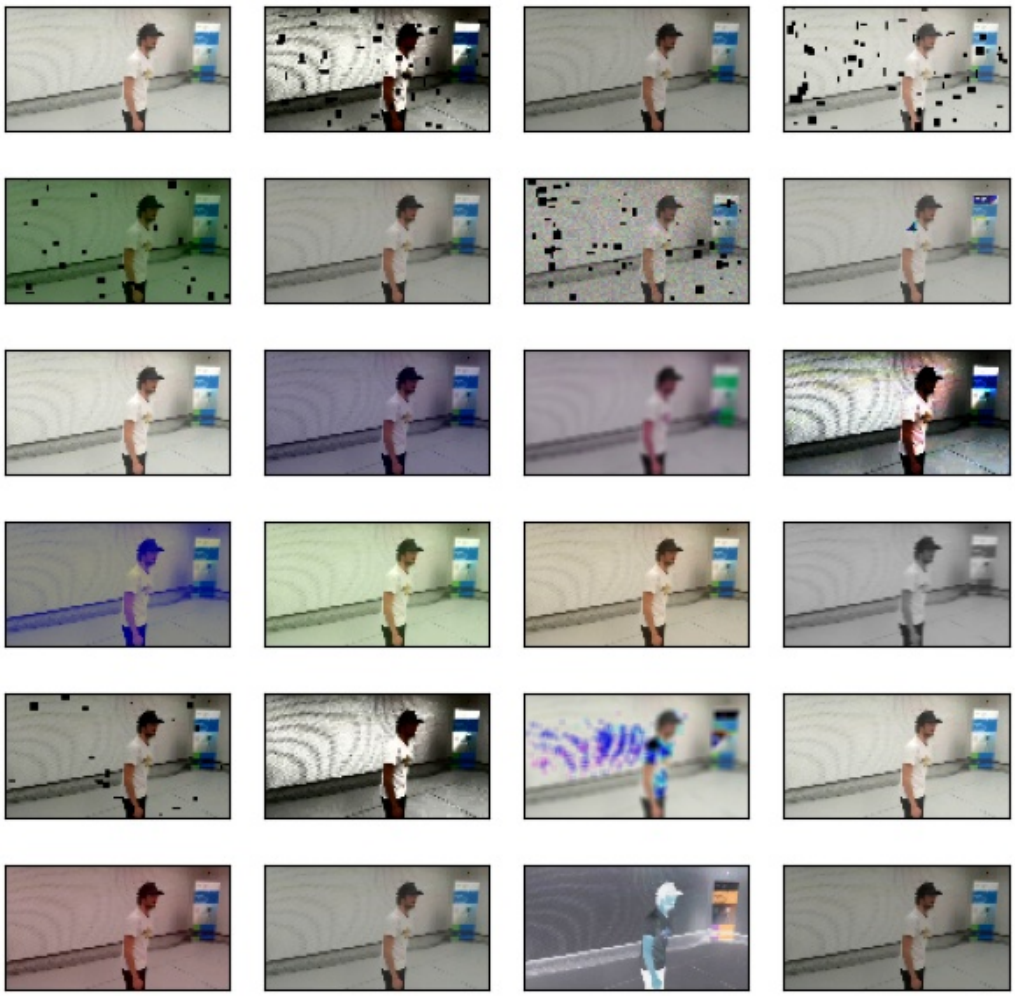


Figure 5.3: Examples of the chosen image augmentation pipeline

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

5.3 Models Definition

We have defined the methodologies to perform background replacement and data augmentation on the original images. According to our work's objective, we are interested in the beneficial effect on generalization caused by the usage of background replacement, which can even increase when combined with image augmentation. For this reason, we define three model alternatives to evaluate the validity of our solution.

- A baseline model, which is trained on the original dataset as-is (section 3.2), and corresponds to the exact approach proposed by Mantegazza et al. [24]. Since it is trained with data coming directly from the drone arena, we call it the `Arena` model.
- The first variant is defined by enabling background replacement. In this case, we replace original backgrounds (i.e., the drone arena) with randomly chosen images. For this purpose, we use the CVPR dataset presented in section 5.1, therefore we call this model `CVPR`.
- Finally, we apply both background replacement and image augmentation. This combination constitutes the most advanced model proposed in our thesis. For simplicity, it is called `CVPR Aug`.

The evaluation of these three models is available in chapter 6.

5.4 Data Generator

This section focuses on the design and optimization of the data generator, which consists of retrieving and pre-processing the dataset to prepare its injection into the CNN.

For its implementation, we rely on the `tf.data` API [47] provided by TensorFlow. It allows the development of complex and modular input pipelines for transforming and iterate over the data in a highly-customizable way. Compared with other methods for managing the model's dataset, `tf.data` gives the possibility of optimizing its operations through asynchronous execution on the GPU[5]. This allows achieving the best possible performance by minimizing the total idle time[6].

5.4.1 Basic functioning

The input pipeline always starts from the retrieval of the dataset. When working with a huge amount of data or there is the need of augmenting it online[7], a common

[5] `https://www.tensorflow.org/guide/data_performance`

[6] wasted on waiting for input or resources, thus without actual running activities

[7] perform the augmentation during training, not relying on a static one previously computed

practice is to save each sample separately on disk. This method allows to only keep in memory the current batch during training rather than the entire dataset, which would otherwise easily raise out of memory issues when working with modest hardware.

We opt for this solution, and we split both training and test sets in 63'720 and 11'030 `pickle` files, respectively. Each accounts for a single sample, containing the original image from the drone's camera, the ground truth according to the MoCap system, and the user's mask previously computed with Mask R-CNN.

The generator only receives the entire list of file paths as input. On-demand, the samples are loaded using the `pickle` library, which unfortunately does not benefit from GPU acceleration.

After a sample is loaded, it passes through the augmentation pipeline. In our case, it includes both background replacement and classic image augmentation.

The former is done on every sample, as long as a dataset for backgrounds has been provided to the data generator. The replacement is implemented with TensorFlow functions by blending the camera frame and the randomly chosen background using the relative user's mask. To speed up the process, all the background images from the given dataset are previously loaded in memory, already pre-processed to be compliant with input image shapes and types.

Data augmentation is applied according to a specified prior probability. It firstly calls the proper Albumentations pipeline defined in listing 5.1. Being not implemented in native TensorFlow, it does not benefit from GPU optimization. After it, the pipeline goes back to TensorFlow, performing horizontal flipping and adding Perlin noise[8].

Finally, the resulting image and its ground truth are easily processed to be injected into the neural network Keras model, which requires specific input shapes.

`tf.data` comes in help for merging samples into mini-batches of a given size. Also, for each epoch, the dataset can be repeated multiple times to simulate an oversampling strategy.

5.4.2 Optimization

Deep learning is particularly time-consuming because many operations must be repeated a huge number of times. Not only network parameters learning affects time performance, layer by layer, but also the input pipeline defined above.

A common problem when dealing with a ML model is the bottleneck caused by the input time. During this kind of operations, the training procedure is not actually learning but, instead, only waiting for resources to be ready. If the input processing takes longer than actual training-related mathematical computations,

[8] textures previously generated and randomly transformed, as explained in section 2.5.1

then it is said that the program is highly input-bound. A good model should spend a tiny percentage of its total time waiting for input[9].

The main causes of the issue are found in complex input pipelines. This is because pre-processing operations are usually done on CPU rather than GPU, significantly increasing the time required. As described above, in our case, both data loading and image augmentation are implemented on the CPU. Furthermore, standard sequential computation requires the GPU to stop and wait for the input at every single mini-batch. This easily kills performance.

`tf.data` natively takes into account the problem and applies some automatic optimizations. This explains why the framework is preferred over the implementation of a data generator made in pure-Python. However, satisfying results require explicit optimizations to be activated for the library.

The following list provides a complete overview of the strategies adopted to reduce the GPU idle time. All of them are available through specific parameters shown in section 5.4.3.

- *Prefetching* decouples the moment when a data is produced from the time when it is actually used for computation. It is done by reading data ahead of the time they are actually requested, enabling the overlapping between input processing and training steps. The result is that total training time is the maximum time between input and computing time, rather than the sum.

  In our case, prefetching affects the entire input pipeline.

- *Parallelization* allows parallelizing data transformation across multiple CPU cores, as much as the hardware is capable of. For example, a dual-core CPU can process two batches simultaneously, halving the total input time.

  In our case, parallelization is enabled for the pre-processing part, mainly including background replacement and image augmentation.

- *Caching* is used to store the dataset in memory or on local storage to prevent some operations from being executed for each epoch. Caching is particularly useful to avoid expensive data transformations or file opening.

  In our case, caching is applied on capable hardware just after the data loading procedure, which is responsible for reading files from the disk.

5.4.3 Source code

Listing 5.2 shows the pseudo-code for implementing the data generator as explained, taking care of both augmentation and optimization parameters.

[9] a maximum of 5-10% input time is considered acceptable

```
def tfdata_generator(files, batch_size,
                     bgs, aug_prob, noises,
                     prefetch, parallelize, deterministic,
                     cache, repeat) :

    # Dataset from files list
    gen = tf.data.Dataset.from_tensor_slices(files)

    # Data loading
    gen = gen.map(lambda filename :
                  map_parse_input(filename),
                  parallelize, deterministic)

    # Caching
    if cache :
        gen = gen.cache()

    # Preprocessing
    gen = gen.shuffle(len(files), reshuffle_each_iteration = True)
    gen = gen.map(lambda img, mask, gt :
                  map_replace_background(img, mask, gt, bgs),
                  parallelize, deterministic)
    gen = gen.map(lambda img, gt :
                  map_augmentation(img, gt, aug_prob, noises),
                  parallelize, deterministic)
    gen = gen.map(lambda img, gt :
                  map_preprocessing(img, gt),
                  parallelize, deterministic)

    # Batching
    gen = gen.batch(batch_size, drop_remainder = True)

    # Oversampling
    gen = gen.repeat(repeat)

    # Prefetching
    if prefetch :
        gen = gen.prefetch(tf.data.experimental.AUTOTUNE)

    # Result
    return gen
```

Listing 5.2: Chosen `tf.data` input pipeline

5.4.4 Profiling

As we have discussed, optimizations are crucial when working with complex input pipelines. We use TensorBoard [45] to profile the training procedure and evaluate the actual improvements provided by our choices. The tool produces accurate statistics

on CPU and GPU usage.

Table 5.1 reports the performance summary for our three models, both with and without enabled optimizations. As expected, the improvement is clearly visible with the `CVPR` and the `CVPR Aug` models, but absent for the `Arena` model. In general, GPU Compute Time is equal to 63.5 ms in all cases. The main difference is observed for the Input Time between optimized and non-optimized alternatives.

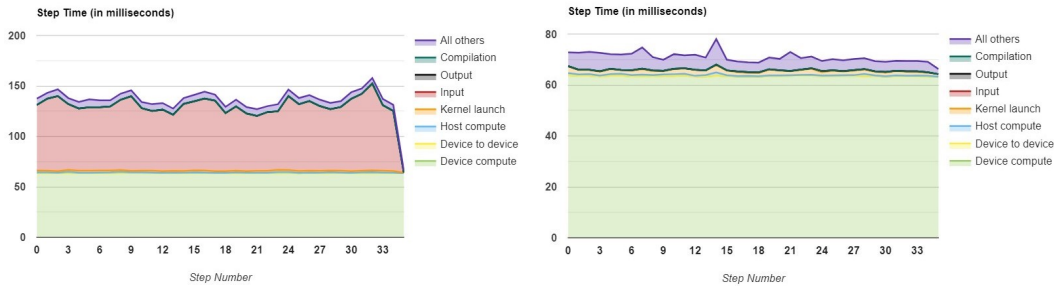
Table 5.1: Training step time performance, profiled by TensorBoard

Model	**Arena**		**CVPR**		**CVPR Aug**	
Optimized	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**
Input Bound Percentage	0.3%	0.4%	35.7%	0.4%	46.9%	0.2%
GPU IDLE Percentage	12.5%	12.8%	42.7%	14.5%	53.0%	10.6%
Total Step Time (ms)	72.8	72.4	110.8	74.1	135.6	70.9
Input Time (ms)	0.2	0.3	39.6	0.3	63.6	0.2
GPU Compute Time (ms)	63.8	63.2	63.5	63.5	63.7	63.5

For a better understanding, we also display TensorBoard visualizations for standard and optimized versions of the `CVPR Aug` model.

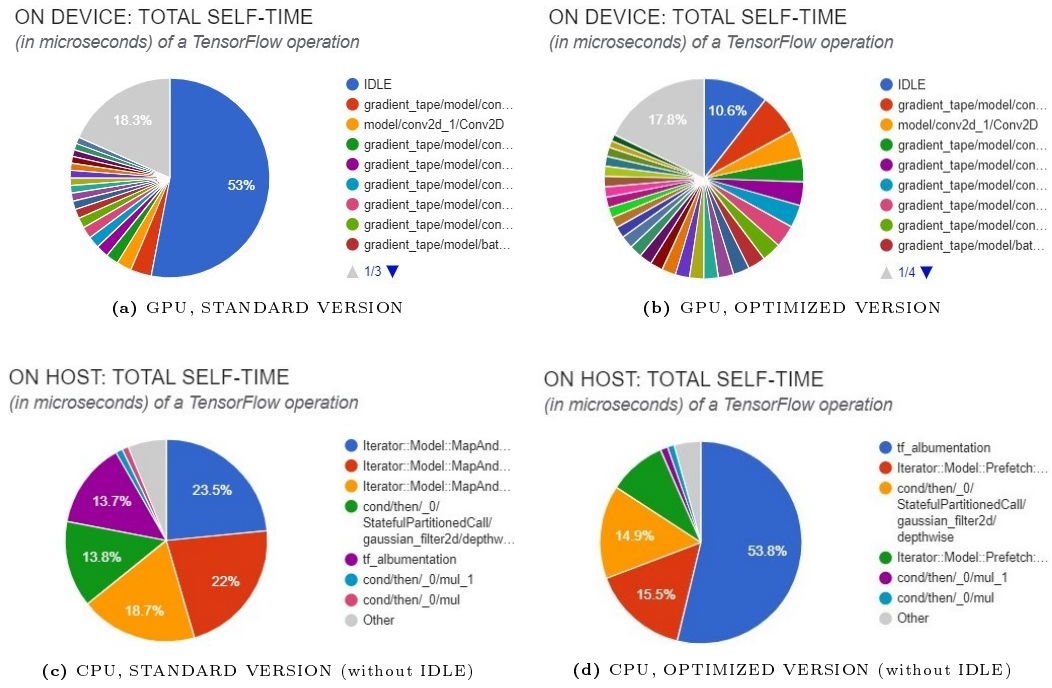
Figure 5.5 presents an overview of the performance over 35 training steps. The two charts clearly show how the non-optimized version is highly input-bound (46.9%).

Figure 5.6 shows the main tasks for which GPU and CPU are used during training. On GPU, the main difference is observed regarding the IDLE time. On the other hand, CPU computation of the optimized version mainly focus on Albumentations, while spreads over different tasks when no parallelization is used. This is probably because sequential non-prefetched operations probably need a greater number of context switches between GPU and CPU, which results in more time wasted for both of them.



(a) Standard version **(b)** Optimized version

Figure 5.5: Profiling: training performance summary of the `CVPR Aug` model



(a) GPU, STANDARD VERSION

(b) GPU, OPTIMIZED VERSION

(c) CPU, STANDARD VERSION (without IDLE)

(d) CPU, OPTIMIZED VERSION (without IDLE)

Figure 5.6: Profiling: GPU and CPU main tasks utilization during `CVPR Aug` training

5.5 Training

This section illustrates technical details about the training procedure. We firstly present chosen parameters, then provide time performance on the selected hardware. Please refer to section 6.2 in the next chapter for consulting training results.

5.5.1 Settings

As in the original paper from Mantegazza et al. [24], we use the Mean Absolute Error (MAE) loss function and the Adaptive Moment Estimation (ADAM) optimizer [17] with the default learning rate of 0.001 and automatic reducer on plateaus (4 epochs patience). The batch size is 64, and the last 30% of the training data (not shuffled) is used as a validation set.

The training runs for 60 epochs, without early stopping. The dataset is shuffled and repeated three times for each epoch. This imitates an oversampling strategy, especially suited for the `CVPR` and the `CVPR Aug` models. The latter is trained with a prior augmentation probability of 95%.

Prefetching and parallelization `tf.data` parameters are automatically tuned by TensorFlow.

5.5.2 Timing

While our first experiments on network interpretation and person masking have been conducted using Jupyter Notebooks on Google Colab, the training task has been carried out through classic Python scripts and executed on custom machines.

For debugging, we use a laptop with the following specifications: OS Windows 10 Pro 64 bit, CPU Intel Core i7-6700HQ quad-core @ 2.60 GHz, RAM 8GB 2400MHz, GPU Nvidia GeForce GTX 950M with 2GB of dedicated memory.

The actual training is performed on a dedicated workstation available at IDSIA: OS Ubuntu 18.04, 2 CPUs Intel Xeon Gold 5217 octa-core @ 3 GHz, RAM 128GB, 4 GPUs[10] Nvidia GeForce RTX 2080 Ti with 12GB of dedicated memory.

On the specified hardware, the training procedure requires a different amount of time according to the model which is being trained. Considering the training parameters stated in 5.5.1 and the three models alternatives presented in 5.3, we observe the following average performance.

- `Arena` model takes 20 minutes with a 65-85% of GPU utilization.
- `CVPR` model takes 40 minutes with a 50% of GPU utilization.
- `CVPR Aug` model takes 120 minutes with a 20% of GPU utilization.

All the models require 1724MiB/11019MiB of constant GPU memory usage.

Their training time is inversely proportional to GPU utilization since more data pre-processing operations, usually performed on CPU, also require a greater amount of time. As explained in section 5.4.1, background replacement is implemented with TensorFlow and executed on GPU, thus only doubles the training time (compare the `Arena` and the `CVPR` models). On the contrary, Albumentations (hence the `CVPR Aug` model) does not benefit from GPU usage and has to be run on CPU, which is significantly slower than the former.

[10] please note that GPUs are used singularly, not for multi-GPU computing

Chapter 6

Evaluation

This chapter thoroughly evaluates our generalization strategy by comparing the three model variants proposed in section 5.3.

6.1 Methodologies

The comparison includes both quantitative and qualitative evaluations. On one side, a quantitative valuation provides numerical results, which are crucial for scientifically assessing the network's capabilities through a set of chosen metrics. On the other side, qualitative observations rely on human interpretation to evaluate models' possible behaviors in some real contexts.

The first part of this chapter focuses on the application of statistical metrics for quantitatively examine the three models both on the training (section 6.2) and the test set (section 6.3). The former is used to briefly understand overall models' learning, while the latter aims to submit unseen examples for assessing the networks' capability in generalization.

An official test set is made available by Mantegazza et al. [24]. For enhancing testing capabilities, we take advantage of background replacement to generate artificial datasets from the test set. Even though this technique can lead to potential biases during the evaluation, caused by unrealistic patterns possibly created during the image manipulation, it allows performing quantitative evaluations without the need of recording new data in different environments.

Qualitative evaluation mostly concerns direct comparisons between the models' predictions and the ground truth. On the contrary, quantitative results require statistical computations suited for regression, thus commonly used in machine learning.

Metrics

The following list presents an overview of all the principal metrics adopted for our evaluation and used in the next sections.

- Mean Absolute Error (MAE) measures the average residuals in the dataset, which means the absolute difference between the actual and predicted values. The lower the MAE, the better the model. Considering y_i the predicted value and $\hat{y}_i$ the actual value:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- Rooted Mean Squared Error (RMSE) measures the standard deviation of residuals. As for MAE, RMSE should be as lower as possible. It penalizes large prediction errors, thus is not particularly suited to be used on a dataset with many outliers. The main advantage and reason for its usage in evaluating regression models is because RMSE has the same unit as the dependent variable, so it is easy to interpret.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

- R Squared (R^2) (or coefficient of determination) measures the robustness of the regression. It represents the proportion of the variance in the dependent variable which is explained by the independent variable. R^2 is often the best choice for evaluating regression performance because it provides a measure of fitness for the model to predict unseen data correctly. Also, its domain is easily understandable since it goes from $-\infty$ to 1: an optimal model (with no prediction errors) will have an R^2 of 1; a model that always predicts the average value will have an R^2 of 0; a model worse than the average predictor will have a negative R^2. Considering $\overline{y}$ the mean value:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$$

6.2 Training Results

As for the entire chapter, this section considers the `Arena`, `CVPR` and `CVPR Aug` models defined in section 5.3. Here, we introduce their performance during the training procedure presented in section 5.5.

In this phase, the loss function (MAE) and the R^2 score are take into consideration, both for the train and the validation set. Figures 6.1, 6.2 and 6.3 show metrics evolution over the 60 training epochs. Overall, it can be said that at least 30 epochs are needed to reach the convergence, after which the performance stabilizes.

`Arena`, `CVPR`, and `CVPR Aug` achieve a validation loss of 0.07, 0.13 and 0.20, respectively. As the model complexity increases, the loss does. However, with data

augmentation, the gap between training and validation loss significantly decreases, as shown in figure 6.3. This may be a symptom of better generalization capabilities, although we do not yet have enough information to draw a conclusion on the subject. Even though the `CVPR Aug` model appears to improve by further increasing the number of epochs, experiments with 200 epochs have substantially the same results as 60.

The `Arena` model registers an R^2 of 0.99, which can be attributable to overfitting. In fact, such a score would be obtained by an optimal model, but we already know that the original FrontalNet by [24] could not properly work outside of the drone arena. In the next sections, we will examine models `CVPR` and `CVPR Aug` on various test sets to certify their robustness against the `Arena` one.

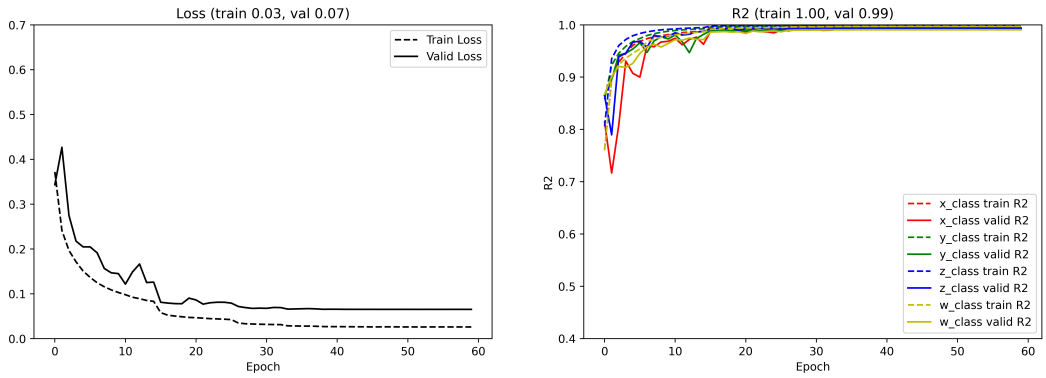


Figure 6.1: `Arena` model's performance during training. Loss and R^2 on training and validation sets

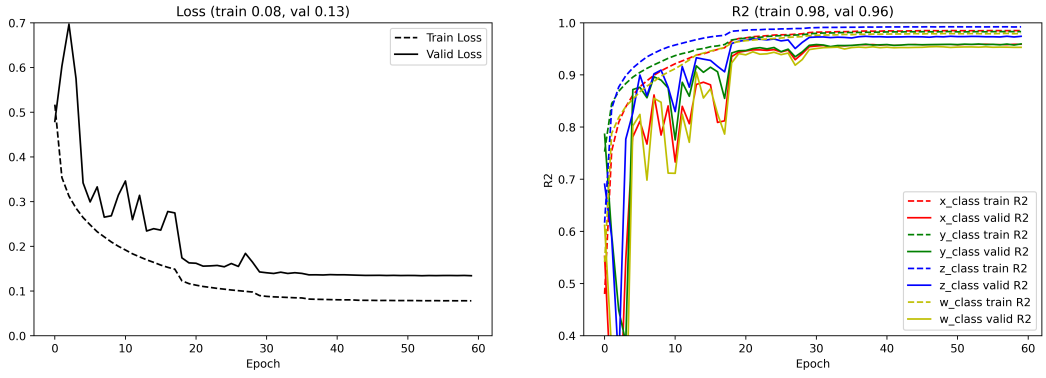


Figure 6.2: `CVPR` model's performance during training. Loss and R^2 on training and validation sets

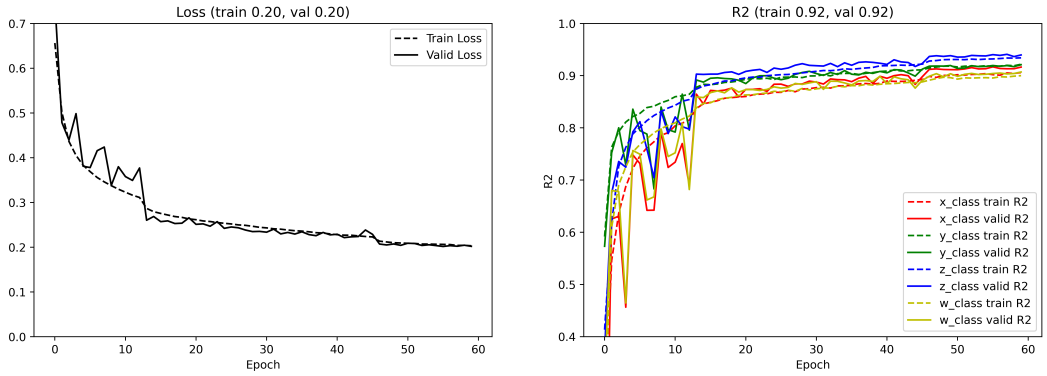


Figure 6.3: `CVPR Aug` model's performance during training. Loss and R^2 on training and validation sets

6.3 Quantitative Evaluation

Performance reported during training is not sufficient to evaluate the model. Its strengths must be measured on previously unseen data, but the validation set is very similar to the training one.

For testing, we first rely on the official test set [24]. Then, we expand it through background replacement, choosing the two indoor scenarios shown in figure 6.4. They do not belong to the CVPR dataset and have been accurately selected to be challenging enough for the model while somehow allowing a person in them to be easily recognized. The respective datasets, created by replacing all the test images backgrounds, are called `indoor1` and `indoor2`. Instead, we call `arena` the original test set.

Figure 6.4: Indoor backgrounds for quantitative evaluation

For a complete analysis, we test our three model variants (section 5.3) on the three test sets defined above. Again, we choose to compare the loss (MAE) and the R^2 score. As previously said, R^2 represents the proportion of variance in the target (i.e., the user's pose) that is explained by the model features (i.e., the input image). As such variance is dataset dependent, R^2 may not be meaningfully comparable

across different datasets. Thus, the metric can only be compared among different models for the same dataset, but not on different data for the same model.

Unlike for training, during testing, we evaluate the R^2 score separately on each variable, allowing us to inspect models' behavior further. Besides, we compute RMSE, particularly useful for understanding the errors' magnitude.

Results are shown in table 6.1 and summarized below.

Arena model evaluation This represents the baseline performance on the test set. On its native dataset (i.e., the `arena` test set) the loss is 0.41 and R^2 lies between 0.74 and 0.86. As expected according to our previous knowledge on model performance out of the drone arena, also on background-replaced test sets the model behavior is very poor. On the `indoor1` dataset, the loss is 0.86 and the R^2 scores ranges between 0.08 and 0.30. Even worse, on `indoor2` the loss is 1 and R^2 registers negative values for variables `X` and `Z`.

CVPR model evaluation The model trained on the background-replaced dataset is much better than the first when predicting the user's pose with unknown backgrounds. `CVPR` performance on `arena` test set are very similar to the one obtained on it by the `Arena` model. The loss is just 0.01 higher and the only big difference is found for the `X` R^2, which decreases from 0.81 to 0.69. Howbeit, the `indoor1` and `indoor2` test sets both registers similar performance for this model, with a loss of 0.44 and 0.45, respectively.

CVPR Aug model evaluation Image augmentation seems to provide not only general improvements but also leverages the outcome for all variables. The `CVPR Aug` model achieves an R^2 score greater of 0.80 for each variable on every test set. Moreover, its performance on the `arena` test set is the best so far, with a loss of 0.36. Relative R^2 scores go from 0.75 to 0.87, encountering no particular differences among different test sets.

Notes on variables Overall, all the models have consistent trends wrt their variables. In all cases, the `W` RMSE is considerably highest than the others, and accordingly lower is its R^2. Also, `Z` RMSE is particularly low because of its distribution rather than actual good predictions. In fact, its R^2 is lower than `Y` R^2 even if the apparent squared error is better.

According to coefficient of determination, `Y` and `Z` are the best predicted variables, followed by `X`. On the latter, image augmentation gives a huge advantage to the `CVPR Aug` model.

Table 6.1: Quantitative evaluation: `Arena`, `CVPR`, `CVPR Aug` models on `arena`, `indoor1`, `indoor2` test sets.

Final considerations on quantitative evaluation

Considering the resulting metrics, it appears that our strategy undoubtedly improves the original model from a numerical perspective. The approach seems able to uncouple the training images from the generalized task of recognizing the user's position. Data augmentation plays a fundamental role in enhancing the robustness of the model. The `CVPR Aug` model, tested inside the drone arena, even produces better theoretical results than the original model. The reason is probably that such an environment is fairly easy to handle compared to the CVPR backgrounds, used during training, which are usually complex and sometimes tough to interpret even by humans.

To validate the results and extend the reasoning over numbers, a qualitative evaluation must be performed.

6.4 Qualitative Evaluation

Charts: video frames with predictions comparison; time series GT & pred **model** comparison; time series GT & pred **background** comparison for a given model; variance considerations

- arena dataset
- bg replaced dataset
- phone videos dataset

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

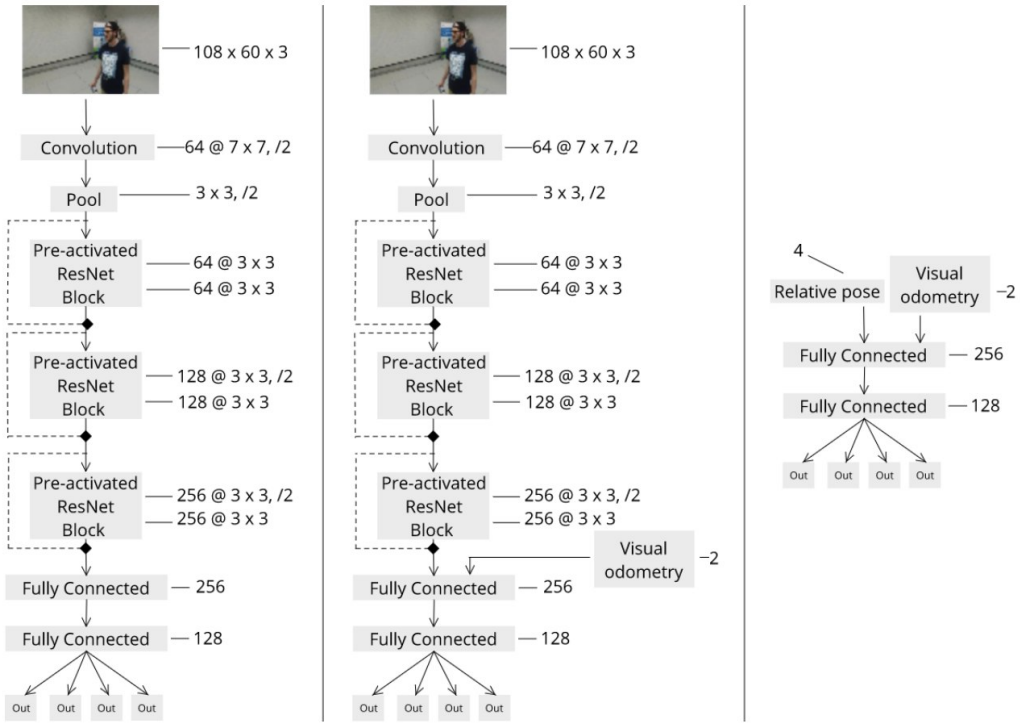


Figure A.1: Models by [24]: mediated, end-to-end, learned controlled

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

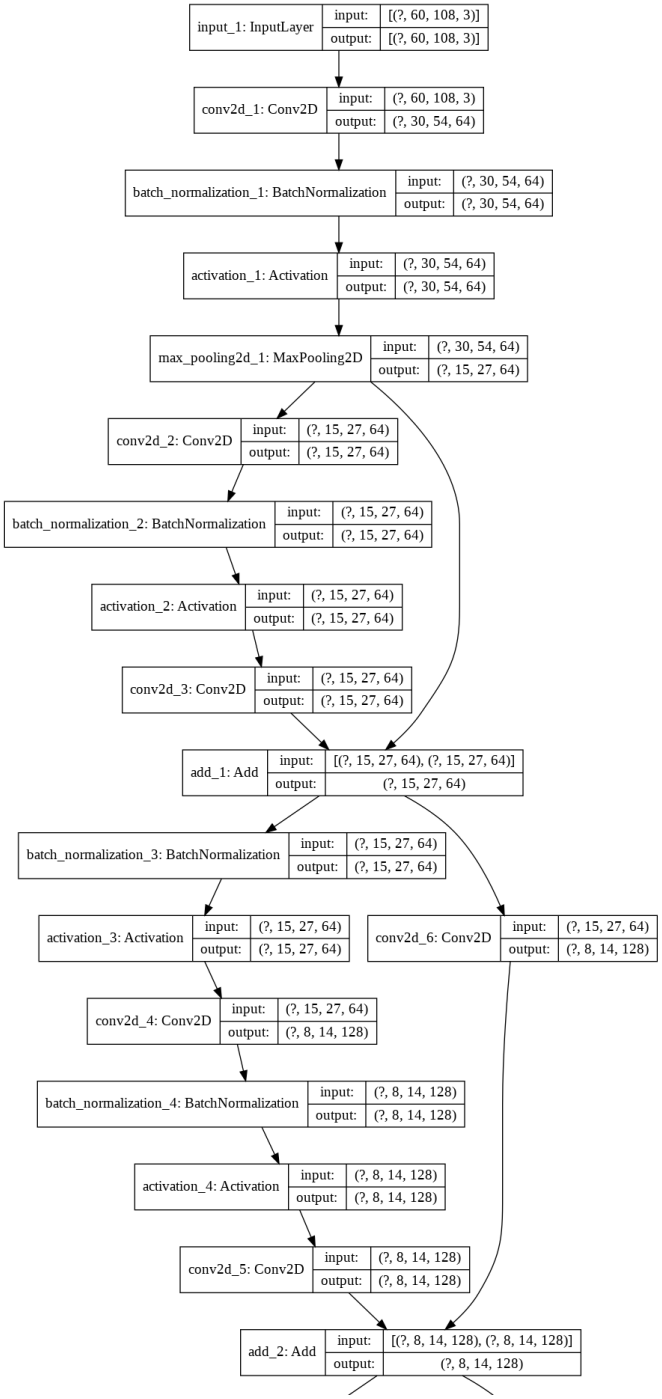


Figure A.2: FrontalNet complete architecture (part 1, from input to layer 18)

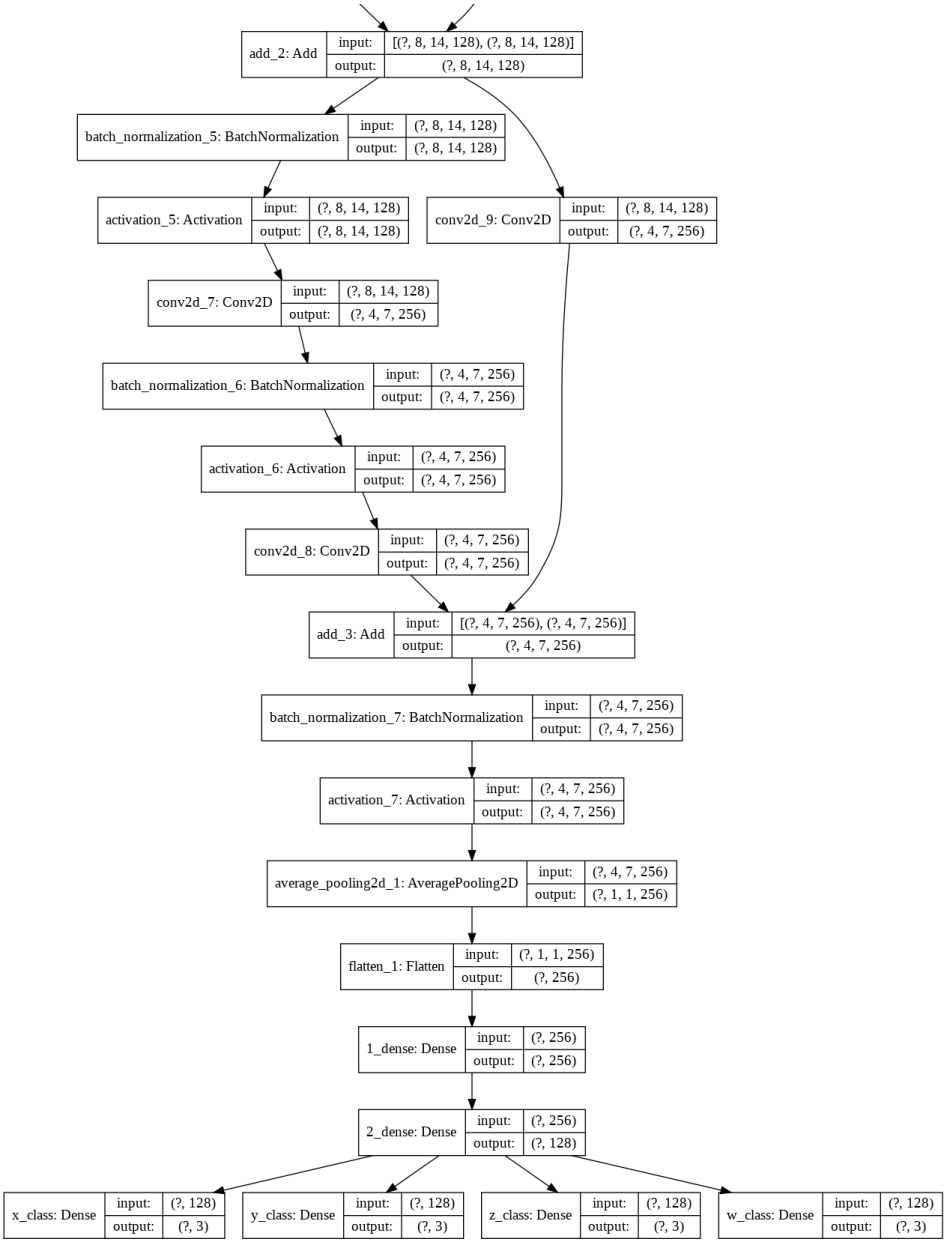


Figure A.3: FrontalNet complete architecture (part 2, from layer 18 to outputs)

Figure A.4, taken from [24], 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.

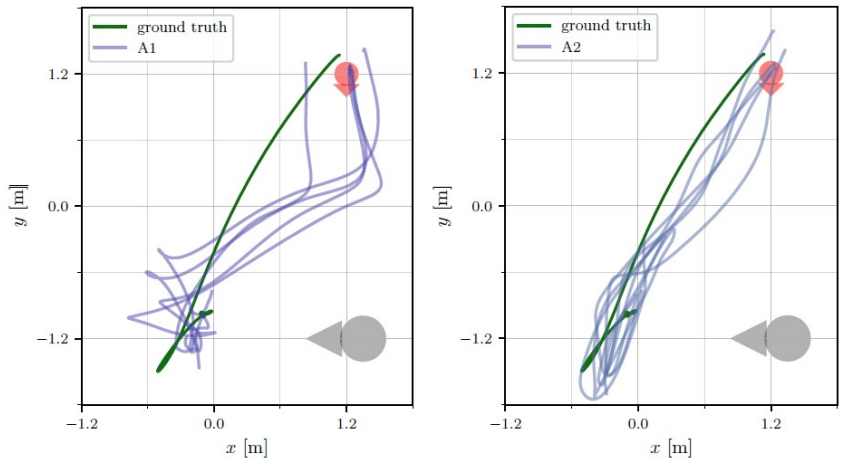


Figure A.4: FrontalNet trajectories for positioning in front of the user initially rotated by 90 degrees [24]

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.

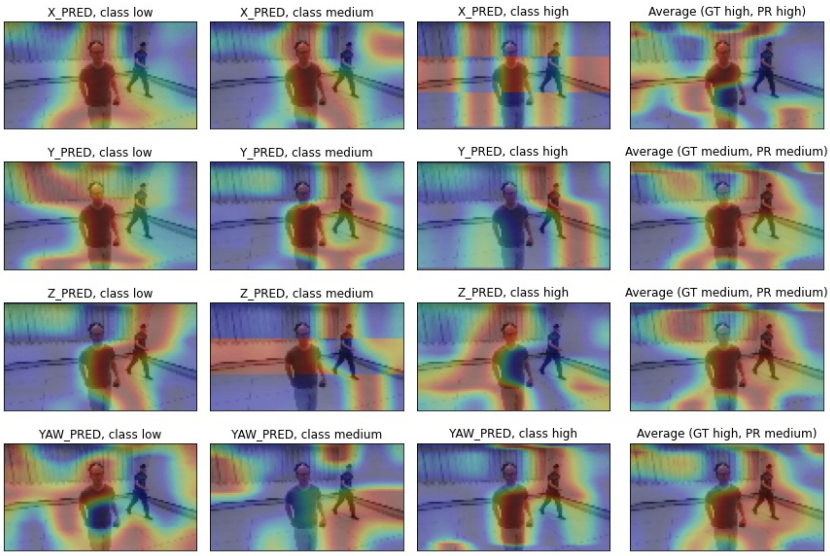


Figure A.5: Full Grad-CAM: two people in the frame

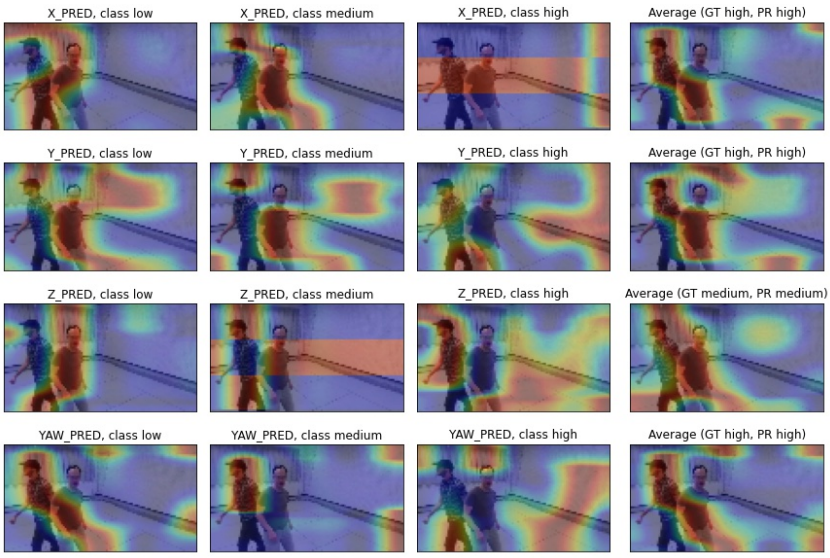


Figure A.6: Full Grad-CAM: two people in the frame

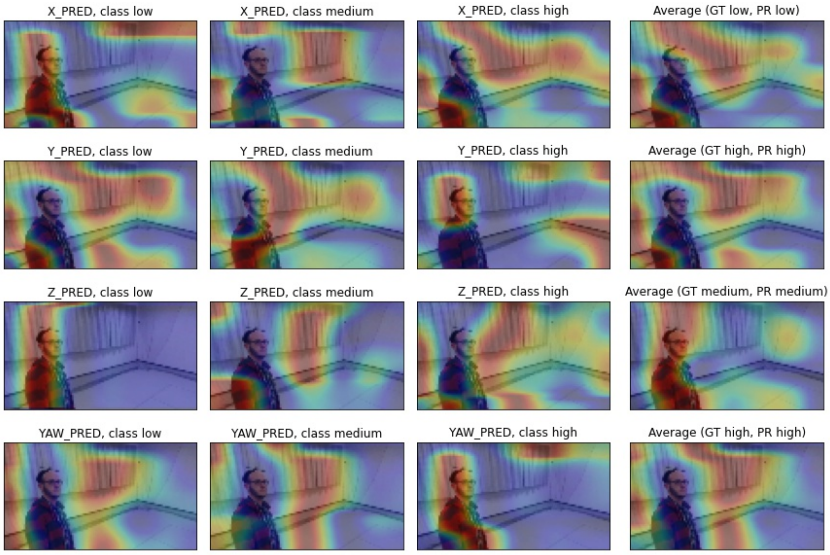


Figure A.7: Full Grad-CAM: model is attracted by curtains

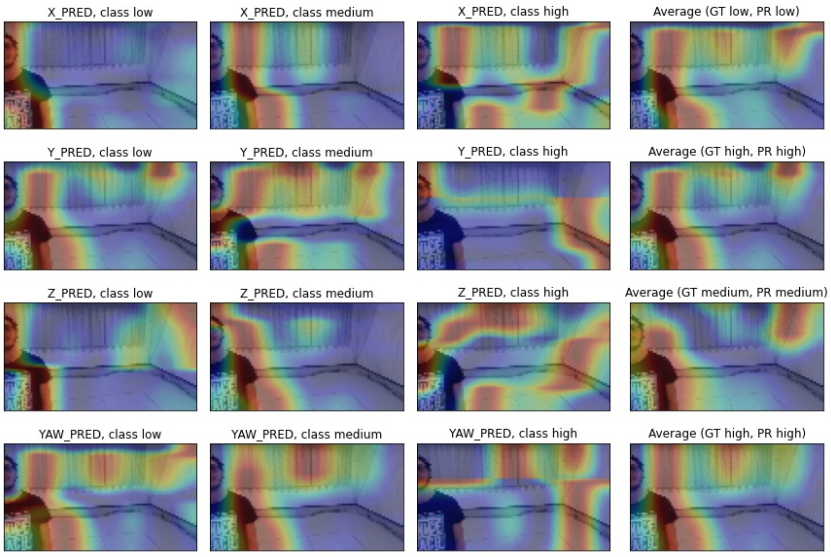


Figure A.8: Full Grad-CAM: model is attracted by curtains

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

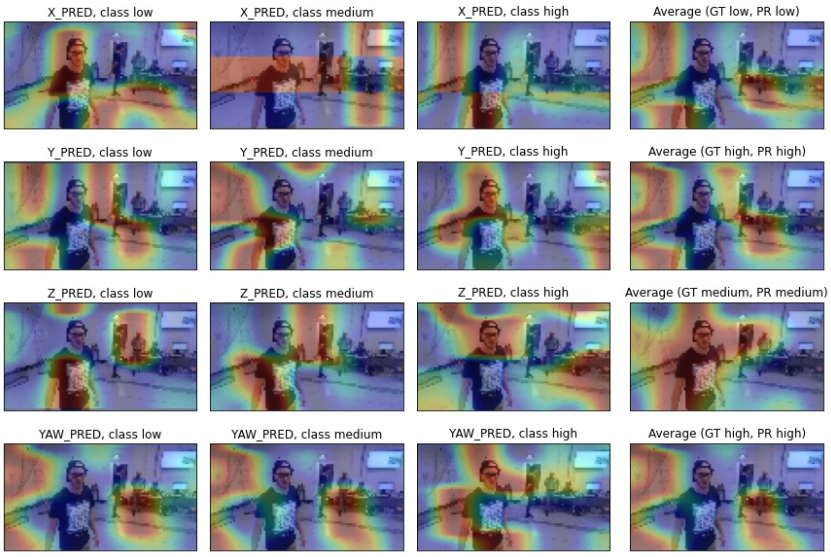


Figure A.10: Full Grad-CAM: model is attracted by background objects

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

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

RMSE Rooted Mean Squared Error

ROS Robot Operating System

VR Virtual Reality

wrt with respect to

XAI Explainable AI

References

[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017. Visited on Oct 2020.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

[8] 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.

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

[10] CVlib. A simple, high level, easy-to-use open source computer vision library for python. URL https://www.cvlib.net/. Visited on Oct 2020.

[11] 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.

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

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

[14] 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.

[15] Adam Kelly. Mask r-cnn in tensorflow 2. https://github.com/akTwelve/Mask_RCNN, 2020. Visited on Oct 2020.

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

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

[18] 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.

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

[20] Weng Lilian. Paper explanation: Domain randomization for sim2real transfer, 2019. URL https://lilianweng.github.io/lil-log/2019/05/05/domain-randomization.html. Visited on Jan 2021.

[21] 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/.

[22] 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.

[23] Dario Mantegazza. Defining a new controller for a drone with user partial pose estimation, 2018.

[24] 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`.

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

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

[27] 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.

[28] OpenCV. Deep neural networks module (dnn), . URL `https://docs.opencv.org/master/d2/d58/tutorial_table_of_content_dnn.html`. Visited on Oct 2020.

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

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

[31] OptiTrack. Professional motion capture systems for robotics. URL `https://optitrack.com/applications/robotics/`.

[32] 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 2019. doi: 10.1109/DCOSS.2019.00111.

[33] 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*, 2019. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2917066.

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

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

[36] 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.

[37] 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/.

[38] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009. ISBN 978-0-387-73003-5. doi: 10.1007/978-0-387-73003-5_196. URL https://doi.org/10.1007/978-0-387-73003-5_196.

[39] ROS. Professional motion capture systems for robotics. URL https://www.ros.org/.

[40] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23: 309–314, 08 2004. doi: 10.1145/1186562.1015720.

[41] 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.

[42] 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.

[43] 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.

[44] TensorFlow Team. Lucid, a collection of infrastructure and tools for research in neural network interpretability. `https://github.com/tensorflow/lucid`, 2020. Visited on May 2020.

[45] TensorBoard. Tensorflow's visualization toolkit. URL `https://www.tensorflow.org/tensorboard`.

[46] 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/`.

[47] TensorFlow. `tf.data` api, for building complex input pipelines from simple and reusable pieces. URL `https://www.tensorflow.org/guide/data`.

[48] 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`.

[49] 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.

[50] 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.

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

[52] Wikipedia. Explainable artificial intelligence, 2021. URL `https://en.wikipedia.org/wiki/Explainable_artificial_intelligence`. Visited on Jan 2021.

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

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

[55] 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.

[56] 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.

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

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

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