Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Interpretation of Neural Networks and Advanced Image Augmentation for Visual Control of Drones in Human Proximity

Supervisor: Prof. Alessandro Giusti
Co-supervisor: Dr. Dario Mantegazza

Master Thesis by:
Marco Ferri
Student ID 807130

Academic Year 2019-2020

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 my mother and father

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

Abstract

We consider the task of predicting the pose of a person moving in front of a drone, using the input images coming from an on-board camera. We aim to improve a machine learning model designed for this intent [33]. The approach relies on supervised learning to perform a regression on the user's pose through a Residual Neural Network. The training data is collected in a dedicated drone arena, using a Motion Capture system to acquire the ground truth. The prototype achieves good performance inside the arena but cannot fulfill its duty in unknown environments.

First, we understand the main issues of the learned task through network interpretation. Applying Grad-CAM [55], we observe that the model not only focus on the user who is actually facing the drone's camera. Instead, various portions of the input images are considered when the model makes its predictions. We assume that the neural network has undesirably learned some details about the drone arena in which the dataset has been collected.

As a solution, we develop an advanced data augmentation technique designed for enhancing the generalization capabilities of the model. The goal is to break the relationship between the model's learning and the training room. Our approach consists on modifying the original dataset through images' background replacement, allowing us to simulate data collected in many different environments. The implementation is done by using Mask R-CNN [19] to compute the user's mask from each sample in the original training set. Then, we use the computed masks for replacing the background of the corresponding images and retraining the neural network.

We run experiments that show that our proposal is successful both from quantitative and qualitative viewpoints. The new model, trained on the augmented dataset, produces satisfactory results in a large variety of real-world scenarios.

Acknowledgements

My most grateful thanks go to my supervisors Alessandro and Dario for having thoroughly assisted me, both technically and emotionally, during the entire development of this thesis.

Thanks to the University of Milano-Bicocca for the opportunity of participating in this amazing Double Degree Program and to USI Università della Svizzera Italiana for immediately making me feel at home. Thanks to all my classmates and friends, who have lightened my journey over all these years.

A special thanks to my lovely Nadia, for always being by my side, supporting me unconditionally, and making me laugh with every single breath.

Un ultimo e sentito grazie ai miei genitori, per aver sempre appoggiato le mie scelte e per avermi trasmesso i loro valori, rendendomi la persona che sono oggi.

Contents

Figures

Tables

Chapter 1

Introduction

Nowadays, the use of Artificial Intelligence (AI) is increasing in Robotics and Computer Vision. Research in the area aims to find innovative solutions, especially using Deep Learning (DL), for well-known yet complex goals such as autonomous navigation, human-robot interaction, and object detection. A trending approach in the last years is Imitation Learning [21]. It consists of training a Neural Network (NN) on a given task by observing the behavior of experienced agents in the same job.

Being a branch of DL, also Imitation Learning (IL) requires a large amount of data to train on. A fundamental aspect in research is the collection of real-world data to build datasets for Machine Learning (ML) applications. However, in many cases, data is not easy to retrieve. In Robotics, datasets acquired with physical robots are usually limited in size or do not provide a faithful representation of the real world. This happens because many Robotics applications require the robots to act in a well-controlled environment, due to physical, technical, or legal constraints. This is why, in order to provide enough training data to the ML model, Imitation Learning often relies on datasets generated in simulation. If the simulator is good enough in proposing the neural network a good variety of realistic data, then the real world may appear to the model as just another variation of the training environment. The technique is known as Domain Randomization [67], designed to transfer a learned task from one domain to another.

In this thesis, we consider an Imitation Learning technique for teaching a drone how to continuously hover in front of a moving person. We focus on the existing work from Mantegazza et al. [33], aiming to improve the generalization capabilities of the model. In the paper, the authors propose a reactive control procedure for controlling the quadcopter using the ML-inferred user's pose. They build a Residual Neural Network (ResNet) to predict the user's 3D coordinates with respect to the drone, using in input the images coming from an on-board camera. The network is modeled to perform regression through supervised learning, in which the ground truth is given by an external motion capture (MoCap) system.

The original approach is an interesting starting point for many other Robotics applications, as it provides a small but fast neural network capable of producing real-time inferences. Also, it explores a challenging task in the research area of human-drone interaction, which will most probably experience interesting growth in the next years [66]. The main issue of the proposed model is the inability of making proper predictions outside of the training room. Since the data collection requires a dedicated MoCap system to acquire the ground truth, the possibility of recording new data in different locations must be excluded. A related work [82] explored different kinds of data augmentation for the task, applying classical image transformations. Results are encouraging but still not enough to make the model able to generalize the task in unknown environments.

Our goal is to understand the underlying causes of the issue and find a solution to the generalization problem. First, we apply Gradient-weighted Class Activation Mapping (Grad-CAM) [55] for network interpretation. In a few words, the algorithm produces a heatmap on the given images which enables the visualization of the input regions actually responsible for predicting a certain model's output. Using Grad-CAM, we discover that the frontal drone model is not only considering the people in the input images for producing its outputs. Instead, many recurrent elements in the training set are attracting the network's attention. We want to reduce the model overfitting by eliminating the biases coming from the data.

As a solution, we propose an innovative pipeline for image augmentation, inspired by Domain Randomization [29]. Our approach consists of modifying the original dataset through the replacement of the camera's frames' background. We rely on Mask R-CNN [19], a state of the art deep learning algorithm for object detection and segmentation. We adopt the algorithm to pre-compute the users' masks, used during training to blend the input frames with other images, serving as the backgrounds. For the replacement, we use images from a public dataset for Indoor Scene Recognition [48]. The technique allows the simulation of various scenarios without the need to actually collecting new data. We also apply classic image augmentation before retraining the ResNet architecture on the background-replaced dataset. Several quantitative and qualitative experiments demonstrate the robustness of the solution, providing satisfactory results in a large variety of real-world scenarios, both indoor and outdoor.

The solution we propose is reasonably applicable to other Computer Vision tasks. In particular, the usage of a computationally expensive method such as Mask R-CNN for pre-processing data enables a severe expansion of a given dataset. Hence, the augmented dataset can be used to train lighter models, which may require human or object recognition capabilities at a cheaper cost.

This work is the result of a collaboration with the Robotics research team at Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), in Lugano (Switzerland). The thesis has been submitted to the University of Milano-Bicocca (Unimib) and Università della Svizzera Italiana (USI) in the context of a Double Degree Program for the Master of Science in Informatics.

The entire source code and any additional resources, presentations, or videos are publicly available at `https://github.com/mferri17/cnn-drone-befree`.

Thesis Outline

The document is composed of seven chapters, briefly described here:

- This chapter provides an overview of our work and its structure.
- Chapter 2 gives a theoretical introduction on the concepts used in the thesis and explores the main literature related with the thesis.
- Chapter 3 illustrates the composition of the existing system and lists the main frameworks used during the development of our software.
- Chapter 4 summarizes our design process and choices by presenting the experiments conducted for shaping our final solution.
- Chapter 5 carefully describes practical details of our implementation from both methodology and technical perspectives.
- Chapter 6 shows the evaluation results for assessing our approach validity and robustness from a quantitative and a qualitative point of view.
- Chapter 7 concludes the thesis, summarizing what has been done with some final thoughts and future works.

Appendix A also contains additional images, not inserted in the main chapters, that add minor interesting details on various topics.

Chapter 2

Background and Related Work

This chapter introduces all the fundamental concepts that the reader should be aware of in order to fully understand the thesis. The first section focuses on some fundamentals of Machine Learning (ML) and Robotics, while the others explore the literature related to our work.

2.1 Theoretical Foundation

In this section, we provide some basic theoretical knowledge. We assume that the reader is already familiar with Deep Learning. Otherwise, we suggest this amazing introduction from the Massachusetts Institute of Technology (MIT): `https://www.youtube.com/watch?v=5tvmMX8r_OM`.

2.1.1 Robotics Introduction

A robot is a programmable system that exists in the physical world, can sense its environment, and can act on it to achieve some goals [34].

This formal definition gives a thorough view of what a robot is and does. In this section, we briefly explain some fundamental concepts of Robotics that are assumed to be well known in the next chapters.

Control A robot must be controlled to perform its task. The procedure is composed of multiple steps: sensing (taking raw inputs from the sensors), perception (extracting useful information from the previous phase), processing (reasoning and planning the actions to be taken), actuation (powering motors to act on the environment). Robots can have different levels of autonomy: teleoperated (fully controlled by humans), assisted (partially controlled by humans), total autonomous (no human control needed). The third category also includes robots capable of fully control themselves in a human-machine cooperation context.

3D Pose The position of a robot in space is defined by its pose. In case the robot can move in a 3D environment, the configuration is described by a 6-dimensional vector $(x, y, z, \alpha, \beta, \gamma)$. The first three (x, y, z) coordinates represent the position with respect to a cartesian 3D coordinate system while the others describe the robot's orientation. Coordinates (α, β, γ) respectively represent the rotations along the axis x (roll), y (pitch), and z (yaw). Figure 2.1a displays the three rotations relative to an aircraft.

Reference Frames Robots' poses have to be defined according to a specific reference coordinate frame. In other words, every pose describes the robot's position and orientation with respect to a given point in space. For example, considering robots moving inside a room, the main (fixed) reference frame can be the center (or a corner) of the room itself. An environment usually includes multiple reference frames, since each robot or component (e.g., a camera) can define its own reference frame. Mathematical computation allows the composition of poses, to transform coordinates from one reference frame to another. Figure 2.1b shows a sequential composition of multiple reference frames.

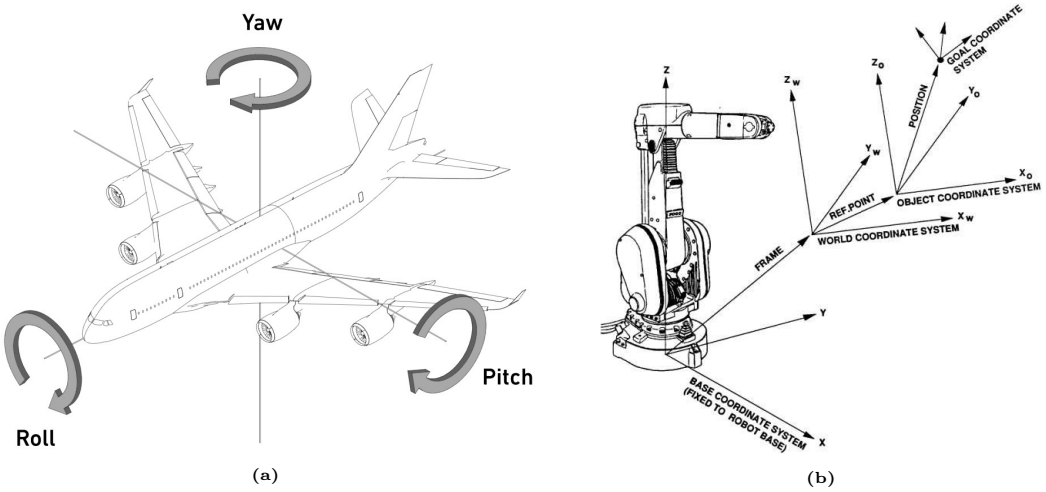


Figure 2.1: Robotics 3D pose fundamentals. On the left, roll, pitch and yaw rotations [1]. On the right, reference frames examples through multiple objects pose composition [60].

2.1.2 Convolutional and Residual Neural Networks

A Convolutional Neural Network (CNN) is a Deep Learning (DL) architecture suited for working with images, thus widely used in Computer Vision [25]. Like other Neural Networks, CNNs are inspired by the human brain and specifically by the visual cortex. Neurons in this area respond to stimuli in sub-regions of the visual field, known as the Receptive Field. A set of these fields is designed to cover the entire visible area.

In the same way, a CNN starts considering small fractions of an image to find spatial relation between pixels. The network learns local features and iteratively combines them together to progressively acquire a general understanding of the entire image. In a CNN, the network neurons are organized in filters of a given size, usually from 3×3 to 7×7 pixels. These filters slide over the image (convolution) to apply a matrix operation on the input and produce an output named feature map. Each feature map aims to recognize a particular input pattern. Feature maps from multiple filters constitute the output of a network's layer. During training, an optimization algorithm adjusts the weights of the filters over time to achieve the network's intent. A CNN is composed of multiple convolutional and pooling layers (to reduce the input size) ending with a set of fully connected layers.

In 2015, researchers took inspiration from cerebral pyramidal cells to define an evolution of CNNs: Residual Neural Network (ResNet) [18]. A ResNet is built from a CNN by adding a shortcut connection between a certain layer and another in the architecture, creating a residual block. Typically, ResNet models use to skip two or three convolutional layers with ReLU activation function [4] and batch normalization [5] in between. Sometimes, the shortcut connection must also include a convolutional layer for adapting the input shape for the receiver layer. Figure 2.2 shows an example of convolutional layers transformed into a ResNet block.

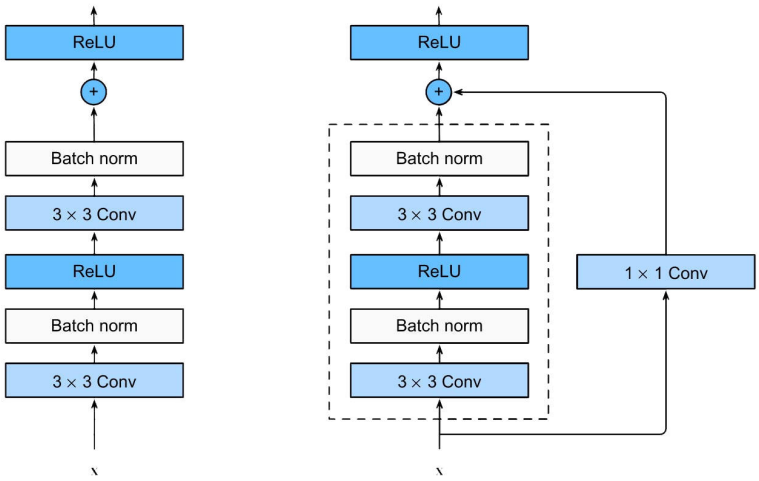


Figure 2.2: Classic convolutional layers (on the left) becomes a ResNet block (on the right) [12]

The reason behind the technique is that adding more layers to an existing Deep Neural Network (DNN) is not always a solution. After a certain limit, the network starts suffering from accuracy degradation, not because of overfitting, but for the vanishing gradient problem [20].

Very deep networks might not be able to learn simple functions like the identity function. In [18], the authors demonstrate that the issue is mitigated when using

residual connections. They propagate the input image as-is, acting as an identity map. For this reason, the network's layer which receives the identity is able to make a comparison with the output of the residual block. By doing so, the network not only learns the input features but also the residuals, giving the model more flexibility. For example, a ResNet is able to learn the identity function by setting the residuals to zero [54]. The shortcut connections allow the creation of smaller networks, accordingly easier to optimize.

2.1.3 Regression Metrics

The following list presents an overview of the principal metrics used in machine learning for evaluating a regression model. These have been used in chapter 6.

- 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. Assuming y_i as the predicted value and $\hat{y_i}$ as the ground truth, the MAE is defined as follows:

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

- 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:

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

2.2 Artificial Intelligence for Drones

Unmanned aerial vehicle (UAV), commonly known as drones, attracted a lot of interest in the last years. Their usage is expected to raise due to decreasing costs and emerging applications for personal, commercial, and social use. As the sector grows, many methodologies are under development for the control and coordination of drones. Most of the recent works in the field propose artificial intelligence solutions for a large variety of tasks.

In this section, we first present a general overview on recent studies for autonomous flight. Then, we shift our attention on the topic of human-drone interaction and we focus on related works by Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA).

2.2.1 Autonomous Flight

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 [21]. The approach consist in making a neural network learn a certain task by imitating humans' or robots' behavior on the given assignment.

In 2016, a conjoint team from IDSIA and University of Zürich (UZH) proposes a machine learning approach for the perception of forest or mountain trails [16]. They use a DNN as a supervised image classifier for understanding the direction of a trail, viewed from a camera placed on top of a robot following the path itself. The solution outperforms previous alternatives, making the drone able to autonomously drive following a tipical forest trail. Quantitative results show an accuracy which is comparable to the accuracy of humans on the same classification task.

Some years later, researchers from UZH have demonstrated that ResNets can achieve satisfactory performance for drones' autonomous driving in indoor and outdoor scenarios. They developed DroNet [31], a forked CNN that predicts a steering angle and a collision probability from given a gray-scale image. The model learns to steer and avoid obstacles from forward-looking videos recorded by cars and bikes while driving in real contexts. DroNet requires very little computational power, allowing real-time performance, even on a CPU. In this case, the predictions run off-drone on a dedicated computer, remotely connected through WiFi.

2.2.2 The State of the Art of 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.

An article published in November 2019 for the IEEE Access [66] explores literature and state of the art in the field of human-drone interaction. It reports that United States Federal Aviation Administration (FAA) expects drones' 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 on the subject. The main studies in human-drone interaction treat drones' control and human-machine communication. 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.

2.2.3 Vision-based Control of a Quadrotor in User Proximity

Our work is based on the master thesis [32] and paper [33] 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 while 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 the person's positions. Images produced by the front-facing camera of the drone are used as input for a custom-designed ResNet architecture that infers the relative user's position with respect to (wrt) the drone (Section 2.1.1). 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 compares the mediated approach described above with 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 our choice is that the learned model can be adapted to other tasks by simply designing a custom controller for the drone. Furthermore, 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. We will address this topic in Section 2.2.4.

Our entire project makes use of the original network architecture and dataset defined in [33], respectively explored in Sections 2.2.3.2 and 3.2. For a better understanding, the original code repository[2] and a descriptive video[3] are also available. To enhance readability of this thesis, having no official name, the custom ResNet architecture proposed by the author will be called *FrontalNet*.

2.2.3.1 FrontalNet Performance

The network is trained using the MAE loss function with the Adaptive Moment Estimation (ADAM) optimizer [26] 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, using 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), whose interpretation is described in Section 2.1.3. The author also experiments with finding the minimum cardinality of the dataset to obtain acceptable performance. Results are available in Figure 2.3, 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.

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

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

[3] `https://drive.switch.ch/index.php/s/MlEDrsuHcSl5Aw5`

These considerations on the variables are confirmed by the qualitative evaluation, obtained by comparing ground truth and predictions during a short simulation. Figure 2.4 shows that `X` and `Y` predictions are considerably worse than results achieved by `Z` and `W` when compared with the ground truth.

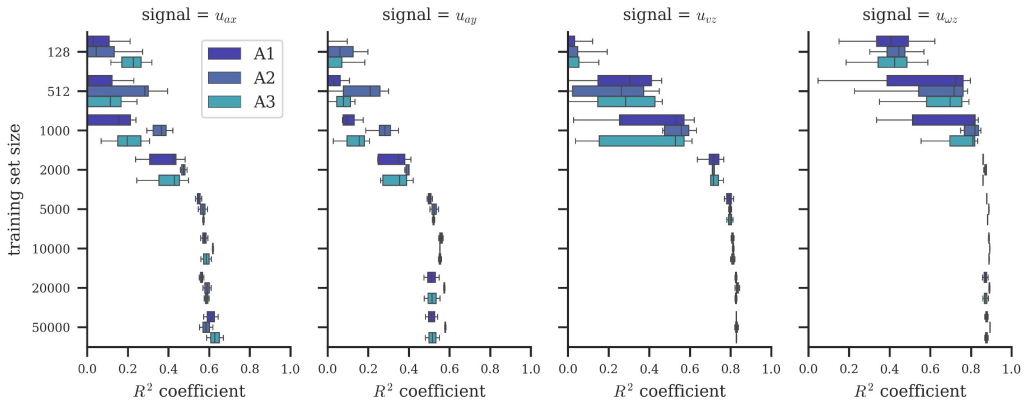


Figure 2.3: FrontalNet quantitative evaluation: R^2 results [33]. A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

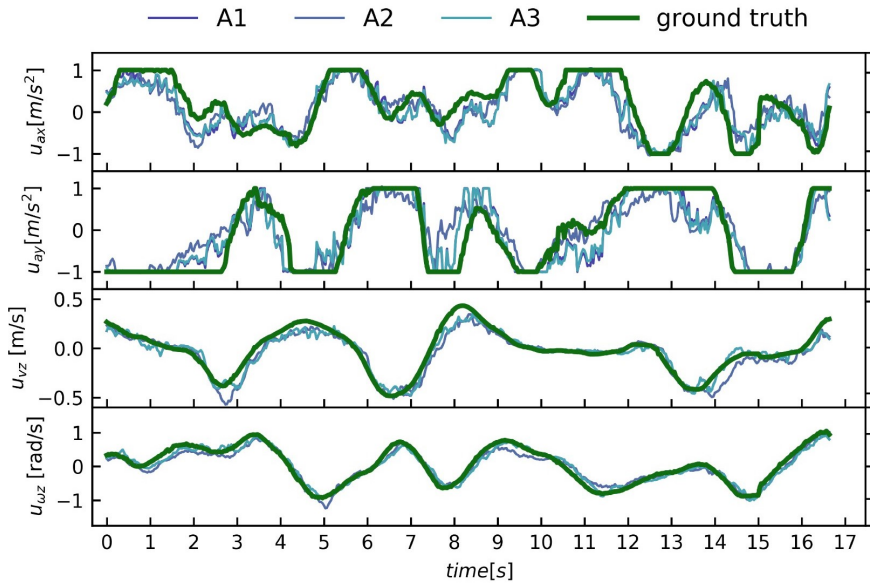


Figure 2.4: FrontalNet qualitative evaluation: GT vs. prediction results [33]. A1, A2 and A3 in the chart stands for different models, but they anyway achieve almost the same results.

2.2.3.2 FrontalNet Architecture

The ResNet 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 [5]. ReLU activations [4] are used for all layers except for the output neurons, which are associated with a linear activation function [77]. Figure 2.5 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.

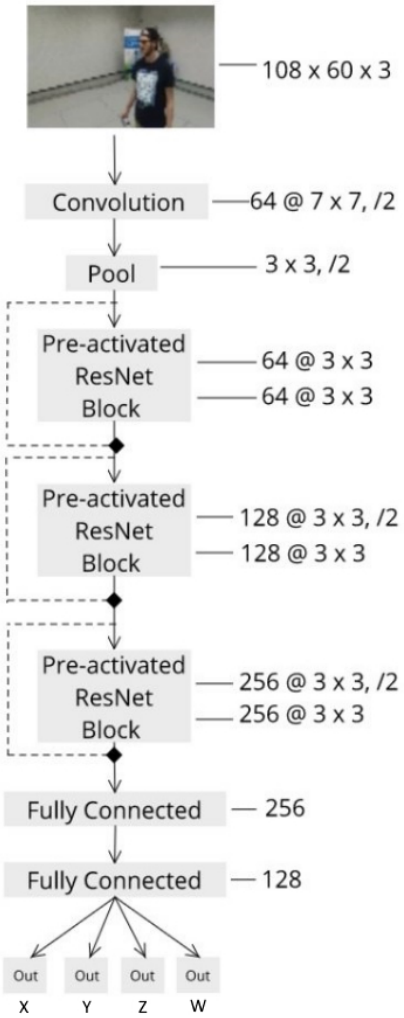


Figure 2.5: Schematic FrontalNet architecture [33]

[4]take a look at Figure A.1 for details about the architectures related to the other approaches

2.2.4 Embedded Implementation of Controller for Nano-Drones

In recent years, both industry and research are starting to work with nano-drones. Their minimal size facilitates indoor flights, even in close proximity to humans. For this reason, they are particularly suited for the exploration of innovative cases on human-drone interaction.

In 2019, researchers from ETH Zürich (ETH) developed PULP-DroNet [45] by porting the DroNet [31] model (explained in Section 2.2.1) on the Crazyflie[5]. This is a nano-drone with a size of only 3×3 centimeters for a weight of 27 grams. The authors 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 [33] to work on-board the Crazyflie with encouraging results [82]. The nano-drone can achieve good quantitative and qualitative performance, regardless of the issues deriving from working with such low-end devices. The main challenges highlighted by [82] are found in the need of managing low computational capabilities, energy consumption, and low-fidelity camera images acquired with no video stabilization.

2.3 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. Nevertheless, unlike other Machine Learning (ML) techniques, NNs 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.

However, when working with real-world problems, it is crucial to produce explainable results. A thorough understanding of ML methods builds trust in algorithms, and makes sure there are no undesirable biases in the models. Otherwise, serious problems can arise especially in critical fields such as medicine and law. Explainable AI (XAI) is the field of study which tries to make ML, and its underlying basis for decision-making, properly understandable to humans [73].

In Computer Vision, researchers have developed some techniques for understanding what a CNN considers for producing an output based on an input image. Primary efforts regard feature visualization and attribution. This section briefly explains these two major areas for CNN interpretability, with a particular focus on spatial attribution, the chosen methodology for our work.

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

2.3.1 Feature Visualization

Feature visualization aims to represent the way a Convolutional Neural Network (CNN) encode its underlying knowledge through a graphical representation of its features. This is done by generating abstract images that explain what the model is really looking for in the input.

A common technique for implementing feature visualization is by maximizing the activation functions of the network through an optimization algorithm [13] [37]. Such method can be applied on many parts of a CNN for understanding individual features: neurons, channels, layers, or even class probabilities. As a result, the algorithm finds a set of input images that is able to effectively maximize the network's output. For example, in a classification problem, feature visualization is able to produce extremely positive examples for a certain class. Please also note that the same features can be associated with different examples in the dataset, as shown in Figure 2.6. Furthermore, through this technique researchers have demonstrated [38] that the layers of a CNN build their reasoning in a hierarchical organization. Early layers learn basic visual properties such as edges or textures, while final layers respond to more abstract properties such as patterns, parts, or objects.

Figure 2.6: Feature visualization: layer-level features (top) for certain dataset examples (bottom).

Despite being an interesting area of research, feature visualization alone is not enough to produce a satisfactory understanding. Instead, the technique is more prone to be effectively used in the field of XAI when combined with other tools to explain the system as a whole [84] [39].

2.3.2 Spatial Attribution: Grad-CAM

Feature visualization gives an abstract representation on the feature learned by a CNN, hence what it is looking for inside an image. However, it does not provide evidence on how or why individual pieces of the network make a final decision. Instead, spatial attribution is capable of explaining the relationship between neurons by highlighting the specific parts of an input image that are most responsible for a certain response of the model [39]. These methods usually produce a heatmap that overlayed on the input images gives an easy interpretable human-ready visualization.

The methodology is younger yet more powerful than feature visualization. First studies in the field started in 2014, trying to determine most relevant parts in the input images by partially and iteratively occluding portions of them, in search of significant changes in the network output [78]. Later, researchers started to approach the problem from a mathematical perspective by computing saliency maps through backpropagation, back to the input images [58] or just until the last convolutional layers [80]. Among these techniques, we select Grad-CAM [55] for its clean and understandable visualizations.

We explain the methodology by first focusing on Class Activation Mapping (CAM) [80], which applies on classification tasks. CAM requires of a specific CNN architecture, composed as follows: a Global Average Pooling (GAP) layer[6] placed after convolution layers outputs the spatial average of the feature maps; the GAP output goes into a fully-connected layer; this computes the final classification output through a weighted sum of features values.

Taking inspiration from the forward pass, CAM projects back the weights of the output layer and computes a weighted sum between them and the feature maps of the last convolutional layer. The operation practically combines multiple activations in a single heatmap that identifies the importance of different parts of an input image for a certain classification output. Figure 2.7 illustrates the algorithm.

Gradient-weighted Class Activation Mapping (Grad-CAM) is a generalization of CAM which works for any network architecture. The main innovation is that Grad-CAM projects back to the network, instead of the last layer's weights, the gradients of the score (for a certain class) wrt the feature map activations of a convolutional layer. These gradients are averaged pooled - that is why the GAP layer is not required anymore - and, as before, weighted sum with the convolutional activation maps. If Grad-CAM is applied on a CAM-compatible network architecture, than it reduces to CAM itself. The result of both the algorithms is a heatmap that overlays the regions of an input image responsible for producing a certain model's output. Figure 2.8 clearly show the result on a cats vs. dogs classifier.

[6]The GAP layer is used for dimensionality reduction, by transforming $h \times w \times d$ feature maps to a single-dimension array of $1 \times 1 \times d$ features [9]

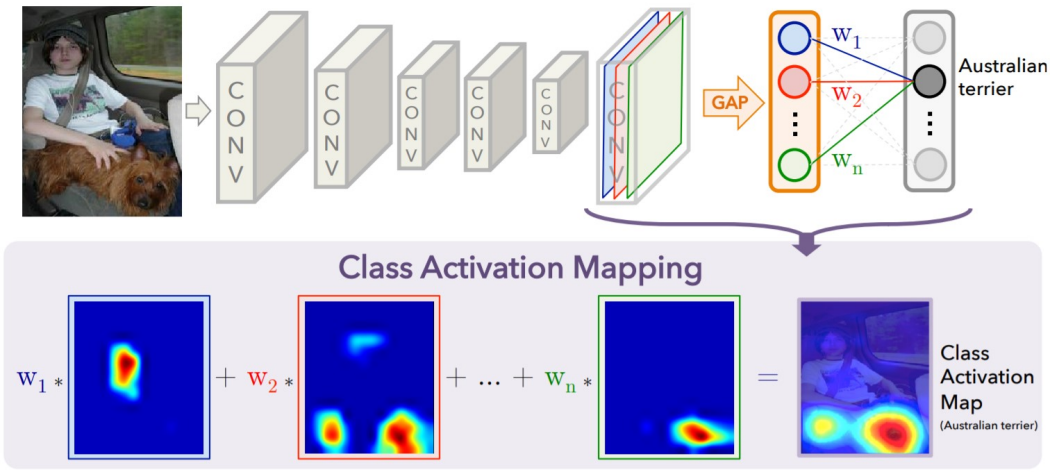


Figure 2.7: CAM schematic functioning [80]

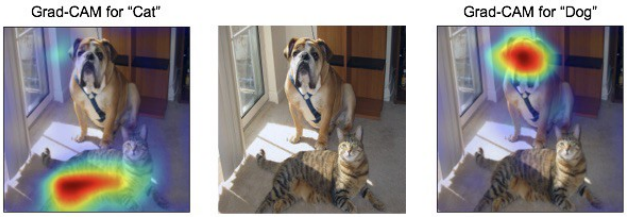


Figure 2.8: Grad-CAM example on dog vs. cat classification [55]

2.4 Network Generalization

This section explores various methodologies for improving neural networks' generalization capabilities. We first provide an overview of the most important methods. Then, introduce our generalization strategy and present alternative solutions, evaluated and discarded during the development of the thesis.

2.4.1 Data Augmentation

One of the most common approaches for reducing overfitting of a ML model is data augmentation. When working with datasets for Computer Vision, we speak about image augmentation. It aims to generate new images from the existing training samples. The technique is ubiquitously used for training CNNs, especially when the available data is limited or difficult to obtain. A huge variety of approaches have been proposed by researchers for achieving image augmentation [57].

The most common solutions apply geometric transformations [75], which can be divided in two types:

- Spatial-level augmentations: also called affine transformations, relocate pixels by cropping, scaling, rotating, translating, mirroring, and shearing the images. These techniques require 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.

Another simple technique is inspired by the NNs mechanism of dropout regularization. The method consists of erasing parts of the training images to contrast samples in which the subject is not totally visible. Also, it makes sure that the model actually looks at the entire image while learning its task [68] [79].

Fancier methods consider the possibility of mixing multiple samples to create a new image. In [61] and [59], training images are created as a collage of other randomly transformed samples. In [28], the authors propose to create a combination of images from the same classification category; the result is similar to the intermediate outputs obtained during the application of a morphing effect [72]. In the last years, the field is experiencing the development of ML approaches for automatically learning image augmentation strategies [83] [10].

Recent techniques of image augmentation are used to transfer a learned task from one domain into another; in other words, a model trained on images created or augmented in a simulation environment can be later used for achieving the same task in the real world. These techniques exploit the concept of Domain Randomization.

2.4.2 Domain Randomization

Domain Randomization is a trending technique for enabling domain transfer. It is useful when the real-world data is not easy to access and collect [36]. The main implementations make use of a simulated environment for representing a virtual world in which the ML model is trained. If the simulator is good enough in providing a satisfying variety of realistic data, then the real world may appear to the model as just another variation of the training environment. The approach is particularly used for reinforcement learning [22] and object recognition [29] [67].

Entire tasks, including environment and moving agents, can be simulated through a dedicated graphical engine. Some of the most important are Gazebo[7], Unreal Engine[8], and Unity[9]. Regardless of being designed specifically for Robotics or for generic purposes, such as game-development or Virtual Reality (VR), all of them

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

give developers unlimited possibilities. However, a complete and adaptable simulation of a machine learning task with physics engines requires a lot of effort.

Recent studies achiev Domain Randomization through advanced image augmentation. In [76], the authors propose an interesting approach for augmenting a dataset using artistic style transfer [15]. They modify an input sample by styling it according to another image taken from a different database. The results produce curious images which give to the training model enough variability to learn a generalized self-driving task. Figure 2.9 shows an example of this application.

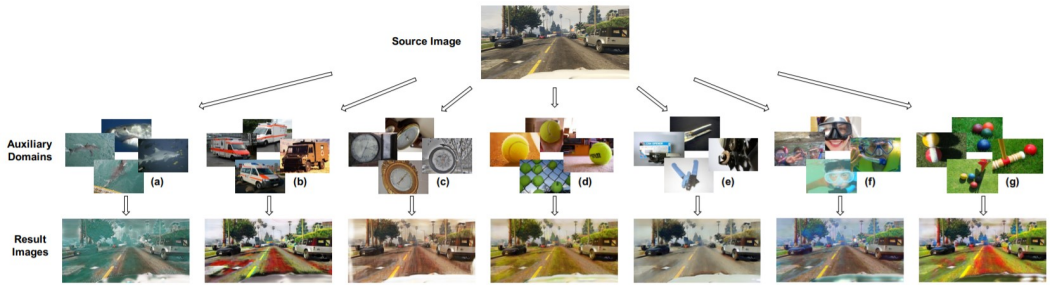


Figure 2.9: An example of artistic style transfer for Domain Randomization [76]

2.4.3 Background Replacement: Chroma Key

In Chapter 3 we will provide an overview of the system by Mantegazza et al. [33], presented in Section 2.2.3. As previously mentioned, the images for training the model have been recorded in a single room. Inspired by Domain Randomization, we focus on designing a custom image augmentation technique to simulate a larger variety of environments. More specifically, we want to replace the background behind each user in the dataset.

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 creating a mask that distinguishes between actual content and background. Usually, a chroma key is implemented using a blue or green screen placed behind the subject, making sure that such color is not present in the foreground image. Then, a post-production software takes care of creating the mask of the subject, enabling the background replacement.

Building up a proper chroma key setup can be challenging, especially in large environments. An experiment in this direction has been conducted during the development of [33], by placing a green screen on a portion of the drone arena walls. Masking results were actually satisfying but the limited size of the screen was suffering from limitations in terms of user's movements and background coverage. Figure 2.10 displays the setup, revealing a non-ignorable portion of the original background still appearing in the images. Besides, assuming the capability of building a well-

designed chroma key environment, the resulting solution would be highly dependent on the setup's geographical location.

For these reasons, we decide to focus on solutions that only involve software methodologies. They are much more flexible and open the possibility of extending the same approach for other tasks.

Figure 2.10: Experimental green screen setup in the drone arena

2.4.4 Human Detection and Segmentation: Mask R-CNN

Mask R-CNN [19] is a state of the art deep learning framework for object detection and instance segmentation. In other words, it detects and categorizes every single object inside an image and also creates the corresponding mask.

The method extends Faster R-CNN [50]. R-CNN stays for Regional Convolutional Neural Network, commonly used for achieving object detection. Faster R-CNN consists of two steps. The first is based on a Region Proposal Network, a Fully Convolutional Network [56] which proposes candidate object bounding boxes. The second fundamental stage extracts squared features from each candidate to perform classification and bounding-box regression.

Mask R-CNN takes the existing model and adds a branch in the network for simultaneously predicting the object mask while the other branch peforms bounding box recognition. Essentially, the first step remains the same as Faster R-CNN, but the second stage also includes a parallel prediction for achieving instance segmentation. The creation of a binary mask is of an object is independent from the object classification, since the algorithm does not actually use any information about the type of the objects for computing their masks. According to [19], this technique allows Mask R-CNN to be faster and more accurate than other state of the art algorithms for instance segmentation.

Please note that Faster R-CNN is much slower [14] than other object detection algorithms, such as YOLO [49]. However, the strength of Mask R-CNN stays in its high-precision in providing both object detection and instance segmentation at once. Results are qualitatively incredible[10] and the method currently outperforms any other study in the field. Furthermore, the framework is easily adaptable for various segmentation and keypoint detection tasks [19] [35] [3]. A practical demo applied to our dataset is available in Section 4.3.3.

[10] `https://youtu.be/OOT3UIXZztE`, `https://youtu.be/Dhkd_bAwwMc`

Chapter 3

System Description

We focus on improving the original work from Mantegazza et al. [33], introduced in Section 2.2.3. 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 [46] (Figure 3.1), a lightweight drone (500 grams) with a size of 382 × 328 × 89 millimeters. A 2700 mAh swappable battery gives power to four brushless motors and a dual-core processor with a

[1] now relocated to Lugano

quad-core GPU, for a maximum flight time of 25 minutes. Connectivity is provided through 2.4 GHz 802.11a/b/n/ac 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 [43], 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 of 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

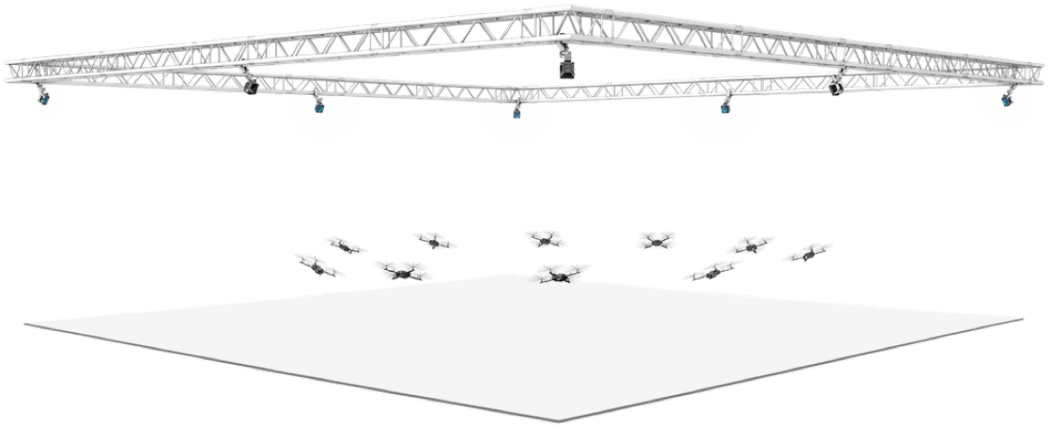
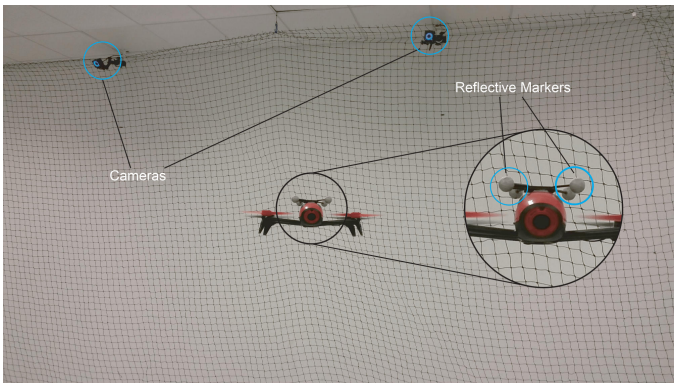


Figure 3.2: Schematic OptiTrack system with 12 OptiTrack Prime cameras

(a) Prime^{x}13 camera



(b) Drone with passive markers [32]

Figure 3.3: Drone arena at IDSIA

3.2 Dataset

This section refers to the dataset defined in Mantegazza et al. [33], 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 (Figure 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 [32]

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 thirteen 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 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 loses the user (Figure 3.8).

[5] mathematically computed by a script from [32]

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

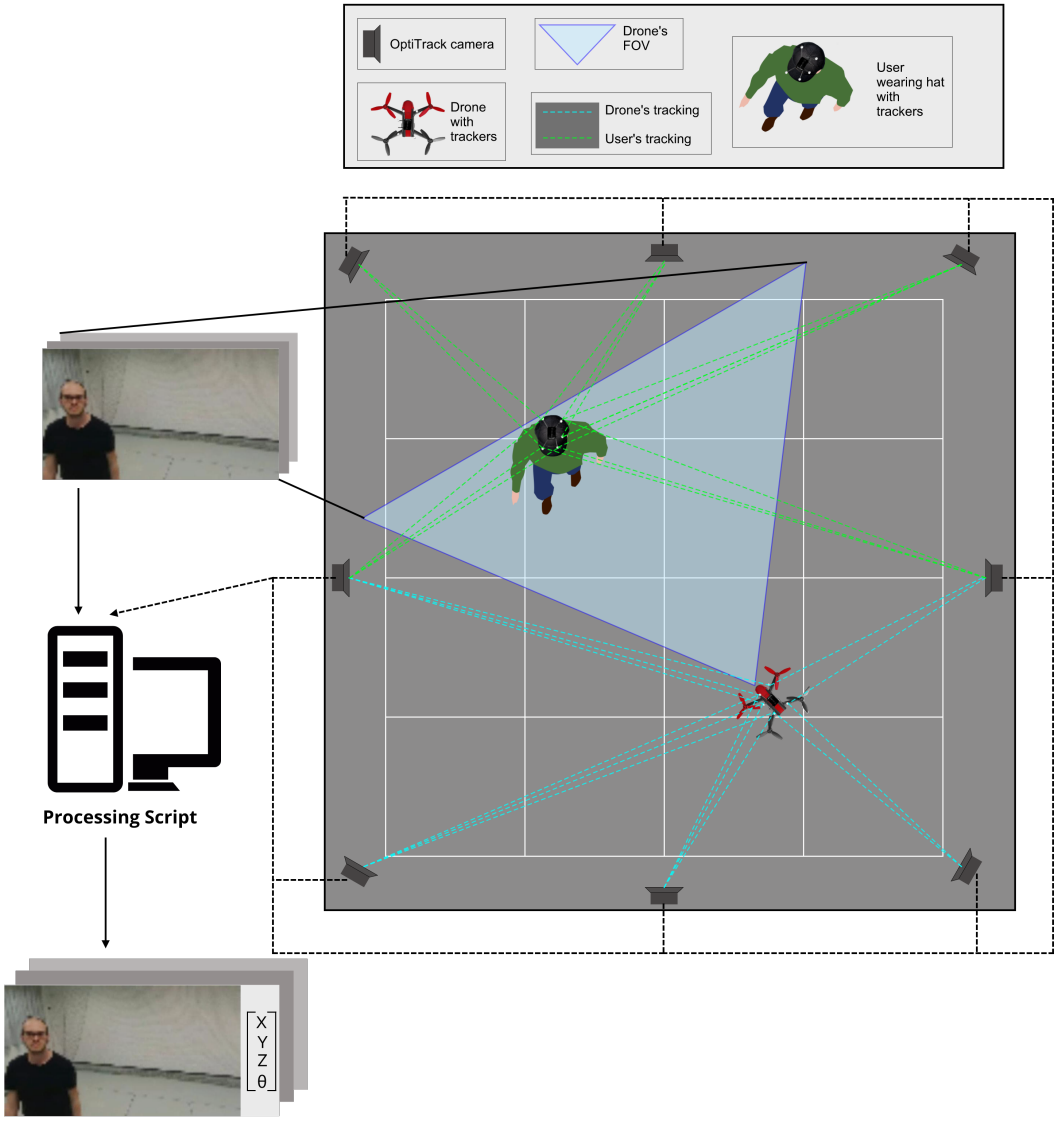


Figure 3.5: OptiTrack and data collection illustration [32]

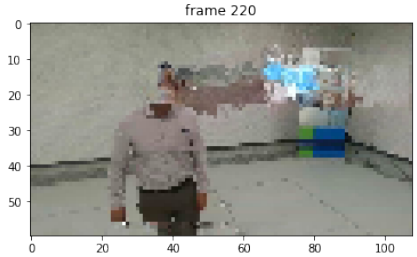


Figure 3.6: A frame with digital artifact caused by connection issues

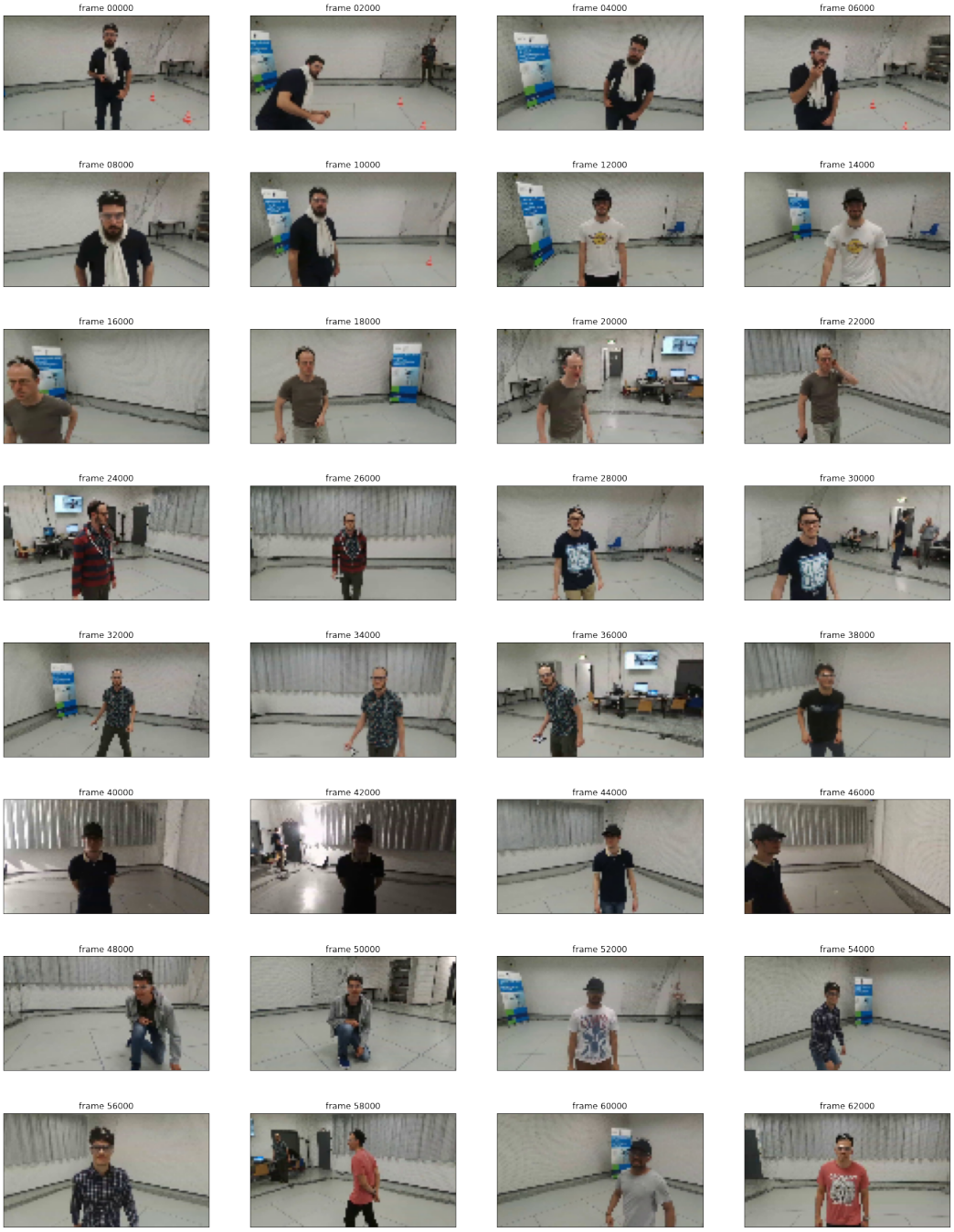


Figure 3.7: A complete overview of images in the training set

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

The ground truth is represented by 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 the 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.

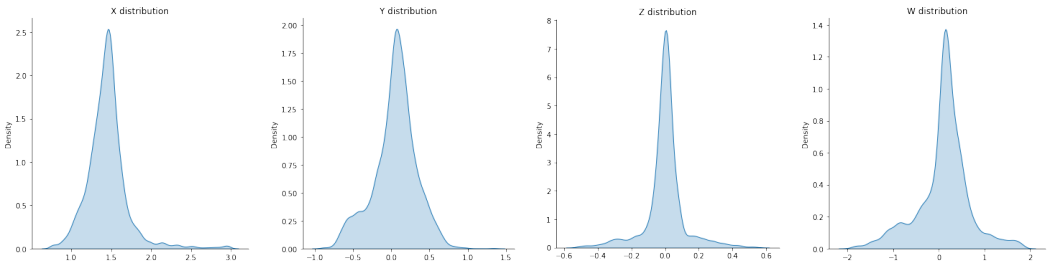


Figure 3.9: Target variables distribution for the regression task

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, 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. [33] to actually control the drone during flight. It will be used in the future for testing our model improvements on the real drone. ROS [52] is an open-source robotics middleware suite 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 [64] (TF) from start to end: network interpretation, person masking, training, and evaluation. Created by the Google Brain team, TensorFlow is an open-source library for numerical computation and large-scale machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. In version 2, TensorFlow 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 [24] 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 [63] 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 [27]. It is derived from the original `keras-vis` [17] 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 [23]. 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 2.2.3, 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. [33].

As explained in Section 2.2.3.1, FrontalNet achieves quite good performance on the test set. Nevertheless, its behavior must be proven on the real drone to certify the model's usability. [33] 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 address the topic, but during a direct discussion with the author, we discovered that flying performance outside of the drone arena was not consistent with the usual model behavior. The drone was not able to follow the user appropriately, and its movements were unpredictable. 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 2.3, 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 2.3.2, Grad-CAM is able to effectively visualize the 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` [62], 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 [27] for TensorFlow 2.

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 [70]), 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 2.8 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).

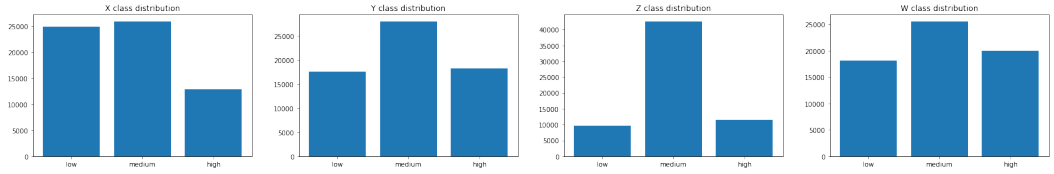


Figure 4.1: Target variables distribution for the classification task

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 Adaptive Moment Estimation (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`

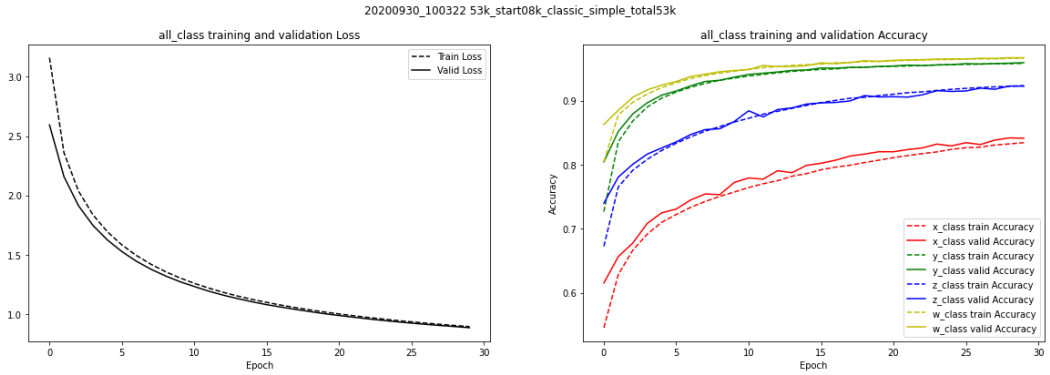


Figure 4.2: Grad-CAM: loss and accuracy of the new classification model

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).

However, our model 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, with a correctly trained model 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 A multi-output network introduces another complexity. Grad-CAM can be run separately on each variable, producing different heatmaps that can be challenging to understand. 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`).

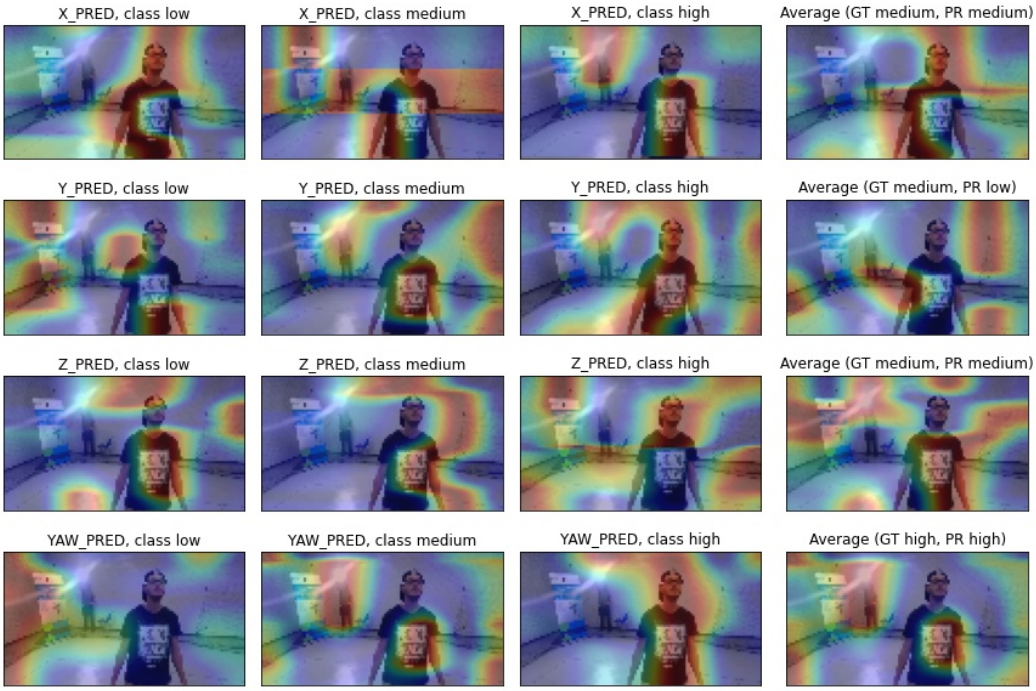


Figure 4.3: Grad-CAM: example of application for each variable and class

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.

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.

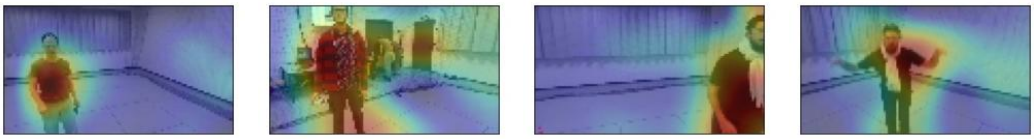


Figure 4.5: Grad-CAM: Correctly detected people

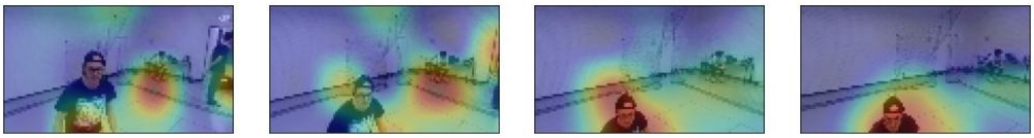


Figure 4.6: Grad-CAM: Sequence transitioning from wrong to correct detections

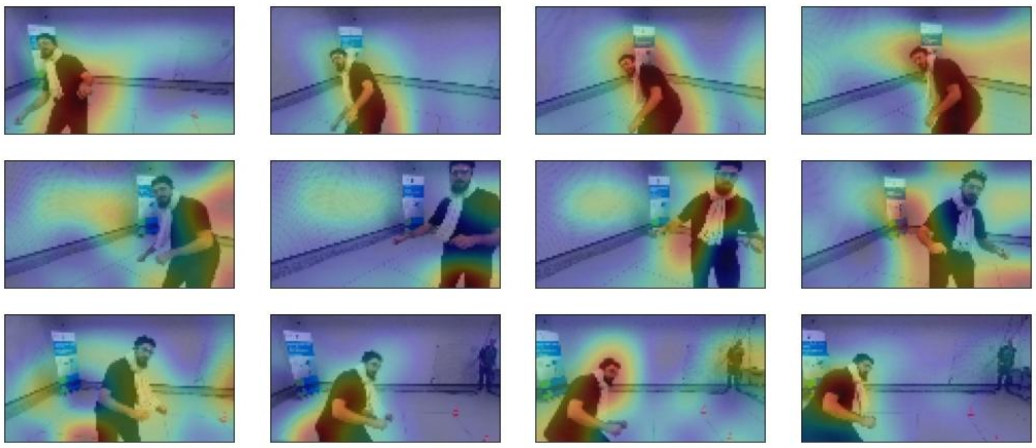


Figure 4.7: Grad-CAM: Sequence of unstable detections going in and out of the person

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)

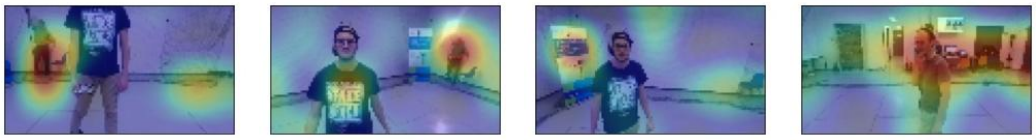


Figure 4.8: Grad-CAM: Objects in the background detected

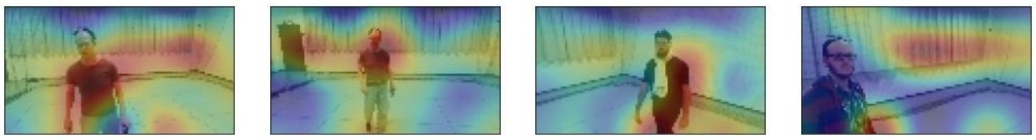


Figure 4.9: Grad-CAM: Curtains often distract the model

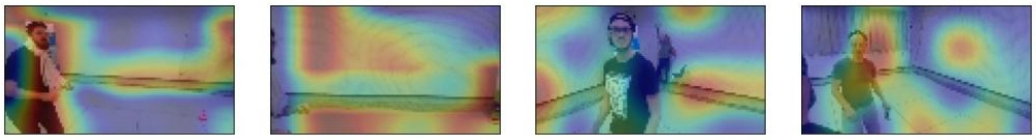


Figure 4.10: Grad-CAM: Model get easily distracted by various elements

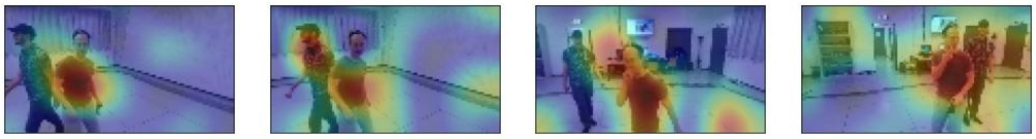


Figure 4.11: Grad-CAM: Detections when two people are present in the image

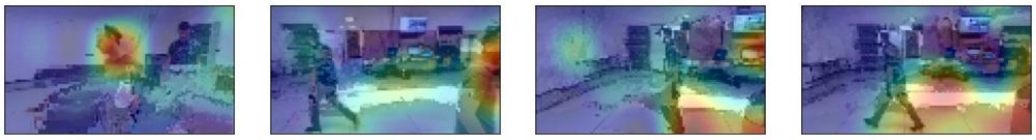


Figure 4.12: Grad-CAM: Model reactions to artificial glitches

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 Residual Neural Network (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 Machine Learning (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 [41] 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`

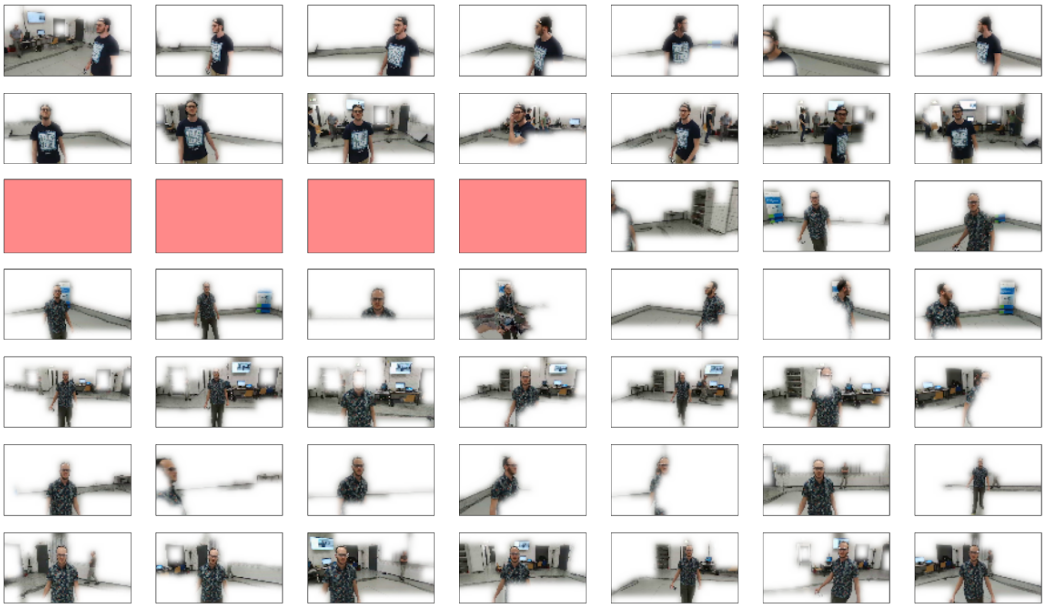


Figure 4.13: Canny edge detection overview on the training set

Figure 4.14: Canny edge enhanced algorithm demonstration

4.3.2 GrabCut

GrabCut [53] operates by using the subject position in the image and some statistical inference for labeling each pixel of the image as background or foreground.

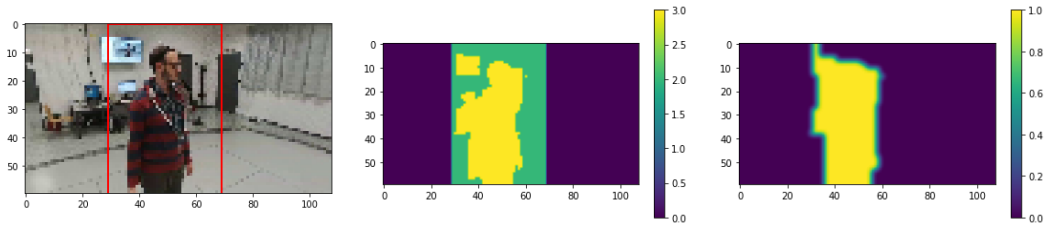
The base algorithm [42] requires a bounding box as an input, a rectangle to enclose the subject to be segmented. Everything outside this rectangle will be taken as *sure-background*; everything inside the rectangle is unknown. The image is first transformed into a graph, then iteratively processed using a Gaussian Mixture Model [51] 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.

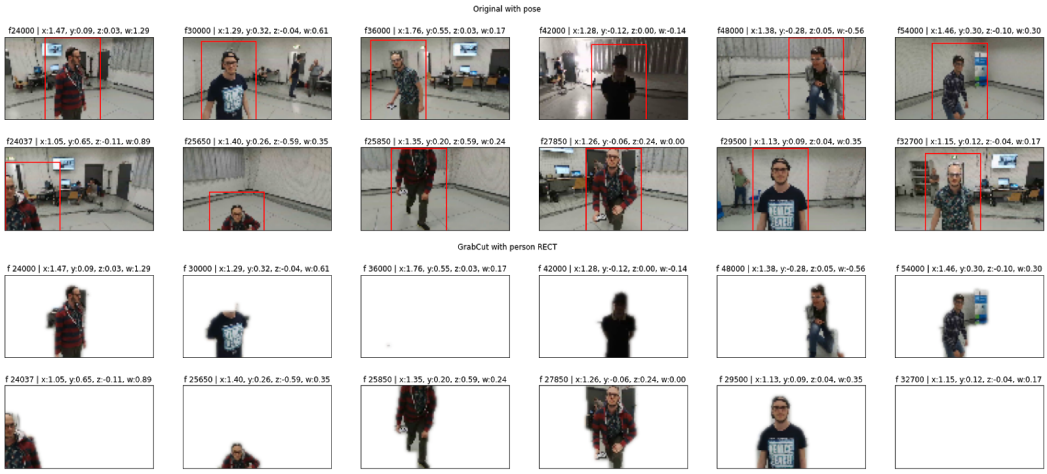
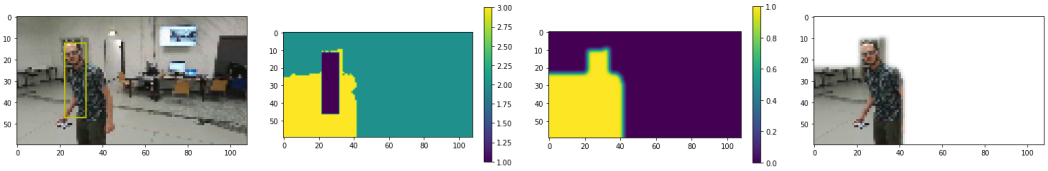


Figure 4.16: Grabcut demonstration: rectangle initialization

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

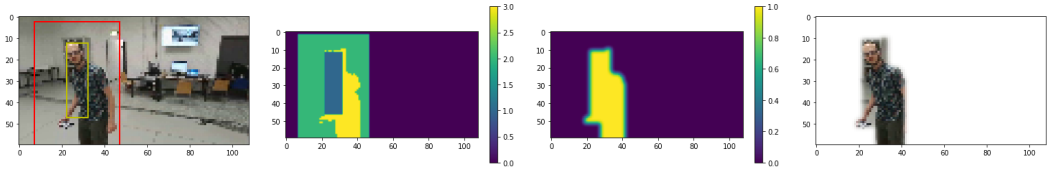
4.3.2.3 Hybrid initialization

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.

Figure 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

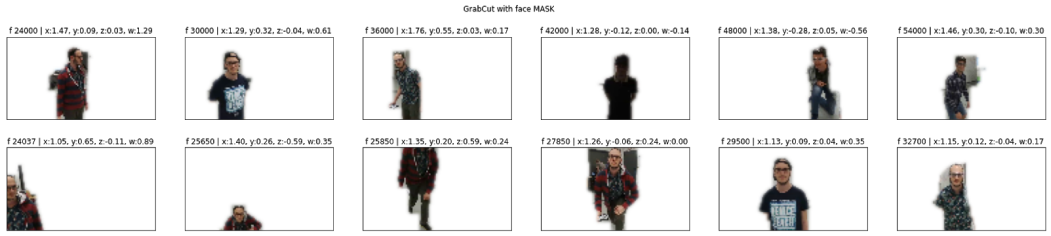


Figure 4.19: Grabcut demonstration: hybrid initialization

4.3.2.4 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 [49], a state of the art technique for object detection. We decide to adopt the `cvlib` library [11], which underneath uses the OpenCV dnn module [40] for implementing a YOLOv3 model trained on the Microsoft COCO dataset [30].

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.

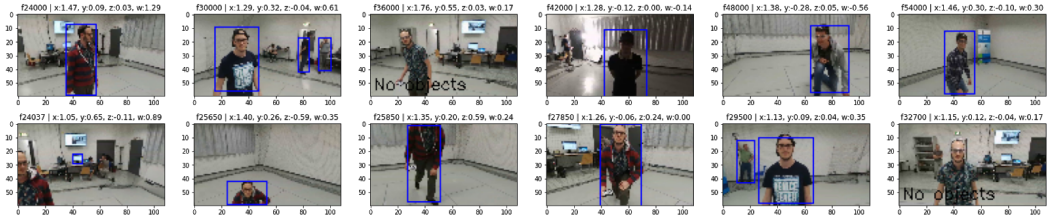


Figure 4.20: YOLO demonstration, which shows failures for 2 images

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 [19] is a state of the art deep learning framework for object detection and instance segmentation, whose technical details have been illustrated in Section 2.4.4. Initially developed by Facebook researchers in PyTorch [47], the algorithm has been ported on TensorFlow 1 [35] and later adapted for TensorFlow 2 [23].

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[6]. For reference, running Mask R-CNN on the test set - composed of about 11'000 images - requires a total computing time[7] of approximately 55 minutes on Google Colab using a GPU runtime[8].

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

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

[8] 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. [33] 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 2.4.2), 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 [48] for Indoor Scene Recognition presented during the 9th Conference on Computer Vision and Pattern Recognition (CVPR). 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. Even though the background replacement is done on the fly, the combination of each input image with a background forms a brand new dataset. Using this data, we train the network from scratch, starting to learn considerably different features than before. Figure 5.1 shows a demonstration of the background replacement technique applied to the samples already presented in Figure 3.7.

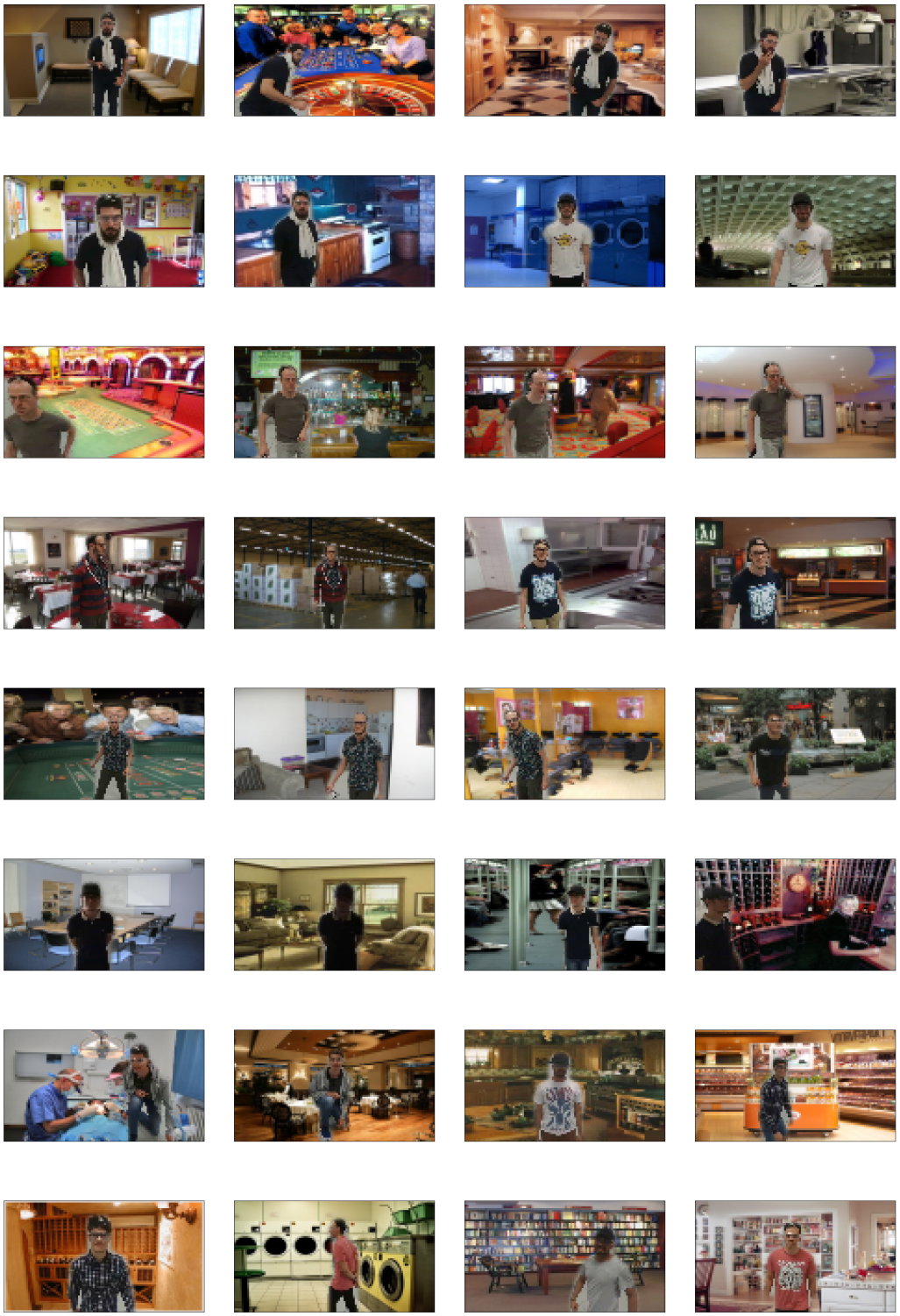


Figure 5.1: Example of background replacement on the training set

5.2 Image Augmentation

Data augmentation is a common technique for improving neural networks' ability to generalize their task on unknown data. Introduced in Section 2.4.1, image augmentation is ubiquitously used with CNNs for reducing overfitting on the training images by applying random transformations.

For implementing image augmentation, we choose Albumentations [6] due to its benchmarking performance[1]. Albumentations is a state of the art Python library that allows the definition of custom sequences composed of many different geometric transformations. These are applied according to user-specified probabilities and intensities. The possibilities are limitless, and the resulting images might be very different from the originals.

In our case, it is not possible to modify the ground truth according to affine transformations because the relationship between the person's position in the image and the ground is not known a priori. For this reason, spatial-level augmentations (see Section 2.4.1) are not an option. Instead, we mainly apply pixel-level transformations. 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%.

Figure 5.2 provides an example of image augmentation applied both on original and background-augmented samples. Those results combine pixel-level transformations on brightness and contrast, multiplicative noise, channels manipulation, and rectangular dropouts.

Figure 5.2: Example of image augmentation with Albumentations

The combination of different transformations composes a pipeline, whose time performance depends on custom choices and probabilities. Among all tested augmentations, most of them only require a maximum of 0.7 seconds to be applied on a

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

set of 10'000 60×108 images, while the most expensive ones take up to 12 seconds under the same conditions[2]. Tests have been performed on a notebook equipped with an Intel Core i7-6700HQ CPU @ 2.60 GHz.

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, adding noise to images can greatly help CNNs on avoiding overfitting [57]. For this reason, at the end of the pipeline, we also apply Perlin noise [71] with a probability of 20%. Since noise generation is highly time-consuming, we pre-compute a set of textures to be used later. During training, one of the generated Perlin noises for each augmented sample is randomly chosen, cropped, flipped, and finally applied to the input image. Half of the time, the noise is multiplied uniformly on all channels to produce a grayscale texture, while the other half of the time is differentiated over the RGB channels. Figure 5.4 illustrates both cases.

[2]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

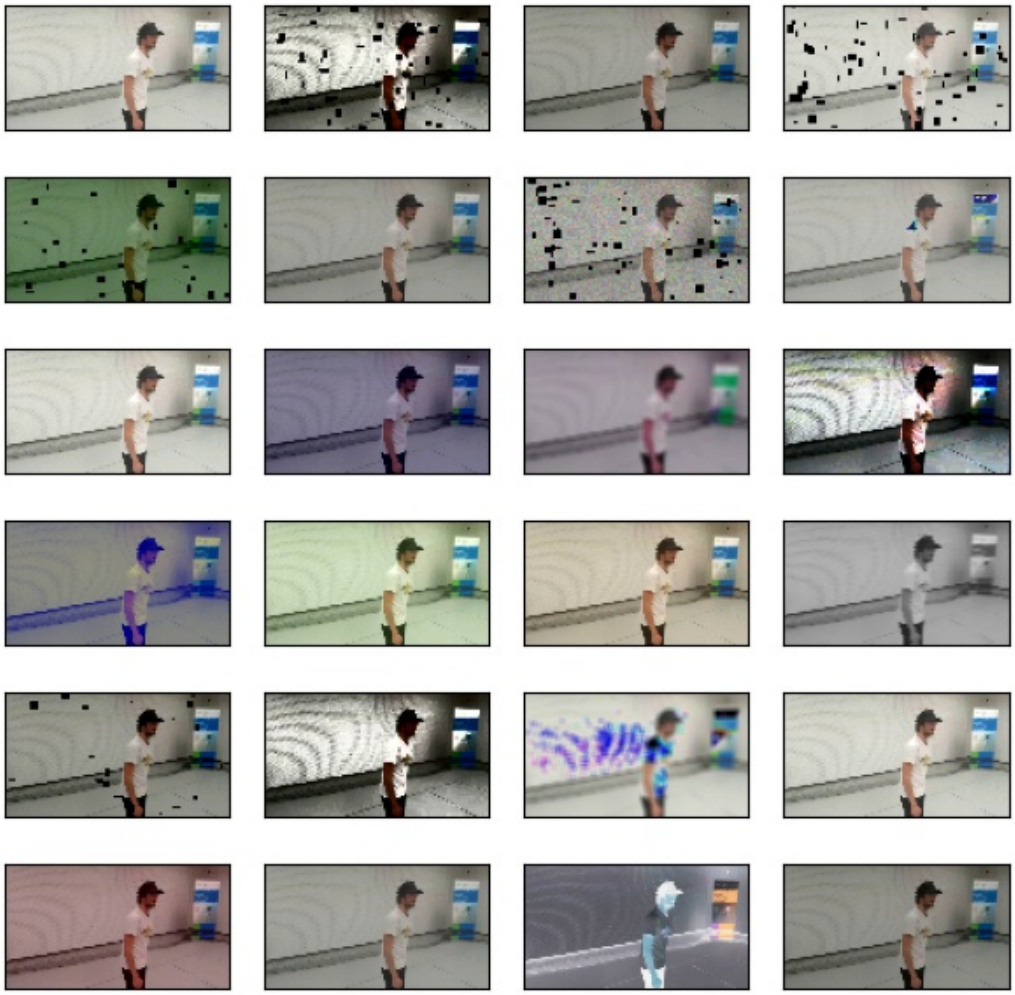


Figure 5.3: Examples of the chosen image augmentation pipeline

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 effects on generalization caused by background replacement. These effects can be amplified when background replacement is 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. [33]. Since it is trained with data coming directly from the drone arena, we call it the `Arena` model.

- A first variant of the model is defined by enabling background replacement only, without any other type of image augmentation. 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`.

- A second variant of the model includes 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 the three models is described 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 before passing it to the CNN.

For the implementation of the generator, we rely on the `tf.data` API [3] provided by TensorFlow. The library 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. This optimization allows achieving the best possible performance by minimizing the total idle time[4].

[3] `https://www.tensorflow.org/guide/data`

[4] Idle time refers to the time wasted on waiting for input or resources, thus without actual training activities

5.4.1 Basic Functioning

The input pipeline always starts from the retrieval of the dataset. When working with a massive amount of data or when there is the need of augmenting a dataset online[5], a common practice is to save each sample separately on disk. This method allows keeping in memory only the data required for a training step rather than the entire dataset. Otherwise, working with an extensive training set can lead to memory issues when using 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 sample contains 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 this image augmentation, the pipeline goes back to TensorFlow, performing horizontal flipping and adding Perlin noise[6].

Finally, the resulting image and its ground truth are processed to be passed as input to the neural network Keras model.

5.4.2 Optimization

Deep learning is incredibly time-consuming because many operations must be repeated a large number of times. Most of these operations regard the actual learning of the NN parameters. However, also the input pipeline defined above can be responsible for a bottleneck in the process. In fact, during input operations, the model is not actually learning but only waiting for resources to be prepared.

[5] In some cases, data augmentation can be previously performed to create a static augmented dataset. However, most of the time, augmentation is done during training.

[6] textures previously generated and randomly transformed, as explained in Section 2.4.1

If the input processing takes longer than actual training-related mathematical computations, 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[7] and a large majority of the time learning.

The main causes of this issue are found in complex input pipelines. This due to pre-processing operations that are usually done on CPU rather than on GPU. This significantly increases the total time required for training. 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 greatly reduces performance.

`tf.data` natively takes into account this 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, for better results, it is required to explicitly activate other optimizations provided by 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 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.

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

5.4.3 Source Code

Listing 5.2 shows the pseudo-code for implementing the data generator as explained. It takes into account appropriate parameters for controlling data processing and performance optimization.

```
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 [63] to profile the training procedure and evaluate the actual improvements provided by our choices. The tool produces accurate statistics on CPU and GPU usage.

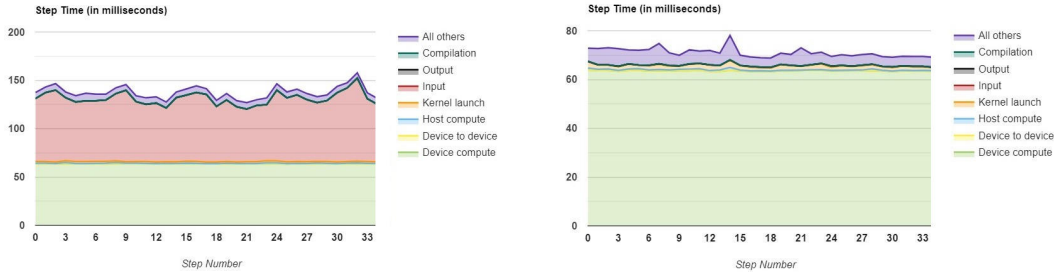
Table 5.1 reports the performance summary for our three models, both with and without enabled optimizations. As expected, the improvement is clearly visible with the `CVPR` and the `CVPR Aug` models, but absent for the `Arena` model. In general, GPU Compute Time is equal to 63.5 ms in all cases. The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

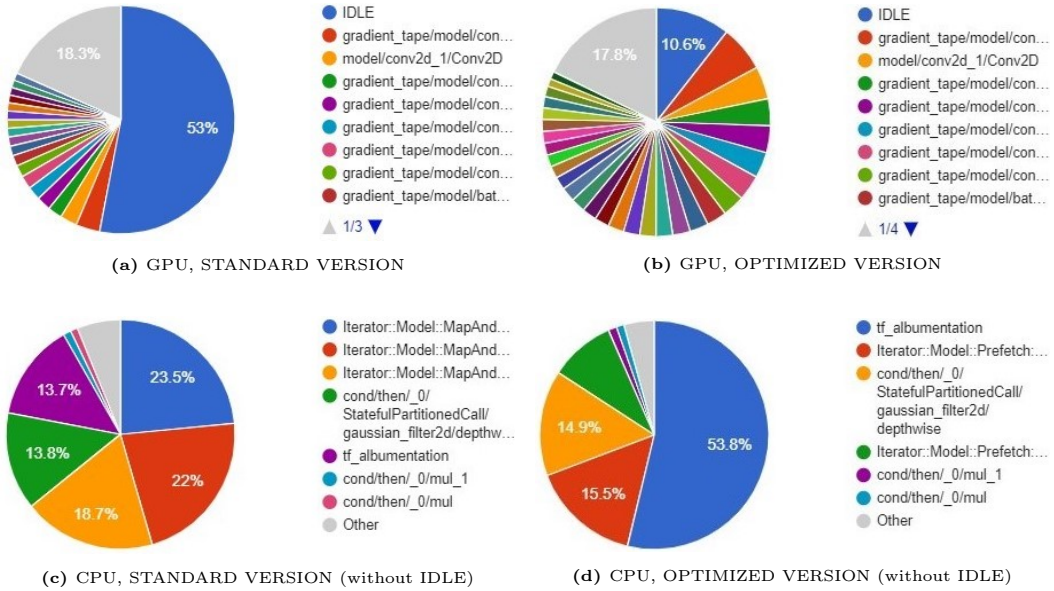
For a better understanding, we also display TensorBoard visualizations for standard and optimized versions of the `CVPR Aug` model. Figure 5.5 presents an overview of the performance over 35 training steps. The two charts clearly show how the non-optimized version is highly input-bound (46.9%).

Also, Figure 5.6 shows the main tasks for which GPU and CPU are used during training. On GPU, the main difference regards the IDLE time. On the other hand, CPU computation of the optimized version mainly focuses on Albumentations while spreads over different tasks when no parallelization is used. This behavior is probably due to sequential non-prefetched operations that need a greater number of context switches between GPU and CPU, resulting in a waste of time.



(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 6.1 in the next chapter for consulting training results.

5.5.1 Settings

As in the original paper from Mantegazza et al. [33], we use the Mean Absolute Error (MAE) loss function and the Adaptive Moment Estimation (ADAM) optimizer [26] 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 to simulate an oversampling strategy, especially suited for the `CVPR` and the `CVPR Aug` models. The latter is trained with a prior augmentation probability of 95%, as detailed in Section 5.2. 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 local 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
- GPU Nvidia GeForce GTX 950M (2 GB of dedicated memory)
- RAM 8 GB 2400MHz

The actual training is performed on a dedicated workstation available at IDSIA:

- OS Ubuntu 18.04 64 bit
- 2 CPUs Intel Xeon Gold 5217 octa-core @ 3 GHz
- 4 GPUs[8] Nvidia GeForce RTX 2080 Ti (11 GB of dedicated memory)
- RAM 128 GB 2666MHz

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.

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

Chapter 6

Evaluation

This chapter illustrates the evaluation of our generalization strategy by comparing the model variants proposed in Section 5.3. The measurement includes 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.

6.1 Training Results

This section considers the models' performance during the training procedure presented in Section 5.5. In this phase, the loss function (Mean Absolute Error (MAE)) and the R Squared (R^2) score are taken 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, least 30 epochs are needed to reach convergence, after which the performance stabilizes. 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.

`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. On the other hand, data augmentation seems to cause a significant decrease in the gap between training and validation loss, as shown in Figure 6.3. Even tough further inspection is needed, this effect may be a symptom of better generalization capabilities, driven by the possibly learned task of actually recognizing a human in the image.

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 [33] could not properly operate 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.

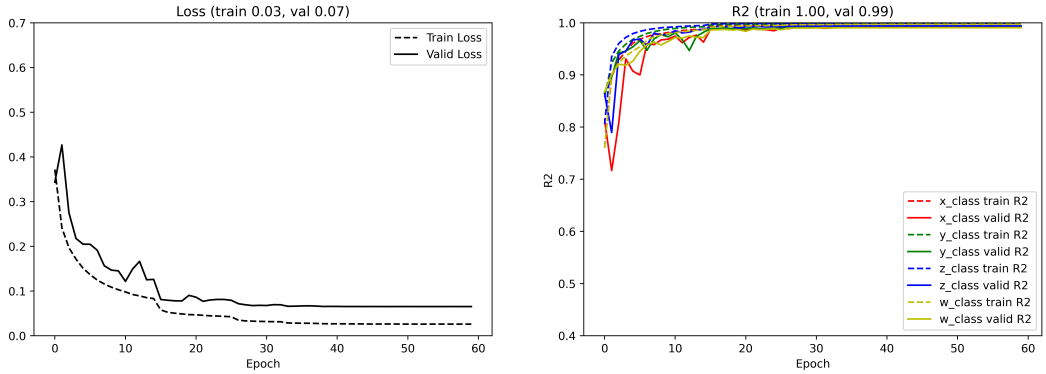


Figure 6.1: `Arena` model's performance during training. Loss and R^2 on training and validation sets

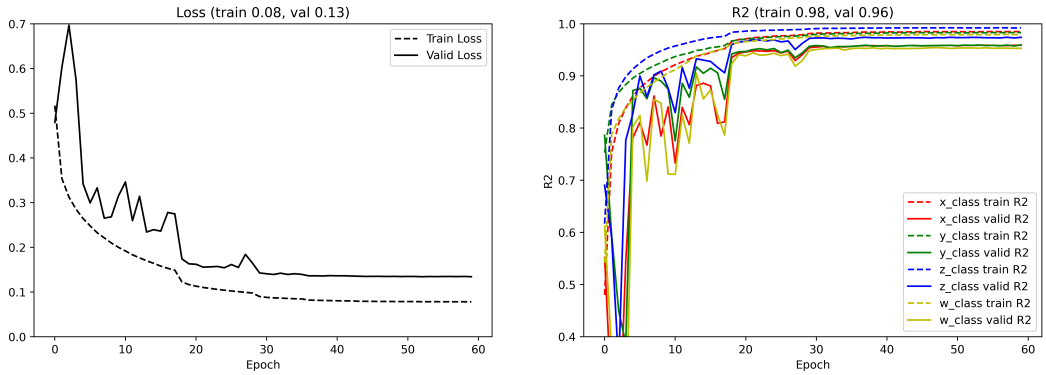


Figure 6.2: `CVPR` model's performance during training. Loss and R^2 on training and validation sets

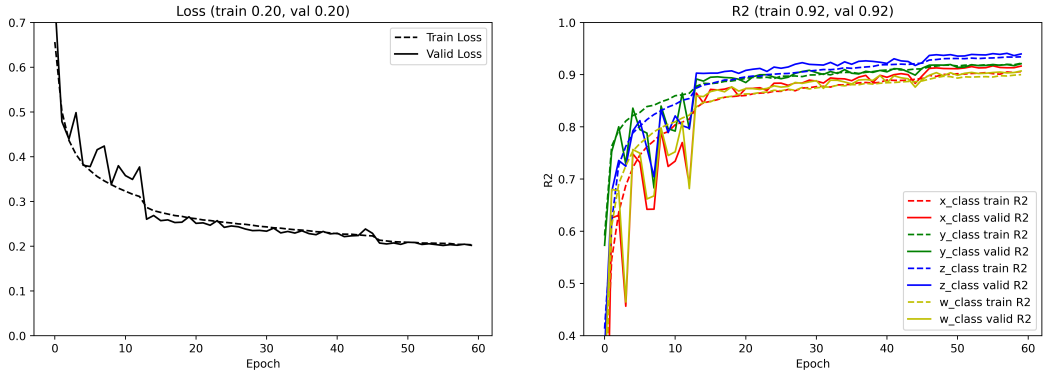


Figure 6.3: `CVPR Aug` model's performance during training. Loss and R^2 on training and validation sets

6.2 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 [33]. 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 explained in Section 2.1.3, 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 Rooted Mean Squared Error (RMSE), particularly useful for understanding the errors' magnitude.

Results are shown in Table 6.1 and summarized below. The table is completed with colorful annotations to improve general understanding. For R^2, higher is better. For loss (MAE) and RMSE, lower is better.

Arena model 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, the model behavior on background-replaced test sets 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 The model trained on the background-replaced dataset is 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 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, registering no particular differences among different test sets.

Notes on variables Overall, all the models report consistent trends wrt their variables. In all cases, the RMSE associated with `W` is considerably higher than the others. Accordingly, the respective R^2 is lower. The table also shows that the `Z` RMSE is particularly low. This is probably because of the variable distribution, rather than actual good predictions made by the model. In fact, the `Z` R^2 is worse than the `Y` R^2, even if the RMSE of the former seems better.

According to the R^2 score, `Y` and `Z` are the best predicted variables, followed by `X`. Also, `X` receives a huge advantage from the image augmentation (`CVPR Aug`).

Final considerations on quantitative evaluation

According to the resulting metrics, 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 network. The `CVPR Aug` model even registers better metrics than the original inside the drone arena. The reason is that such an environment is relatively more comfortable to handle than the CVPR backgrounds used during training, which are usually complex and sometimes challenging to interpret even by humans.

DATASET	METRIC		MODEL		
			Arena	CVPR	CVPR Aug
Training	LOSS	train	0,03	0,08	0,20
		valid	0,07	0,13	0,20
	R2	train	1,00	0,98	0,92
		valid	0,99	0,96	0,92
Test Arena	TEST LOSS		**0,41**	**0,42**	**0,36**
	R2	x	**0,81**	**0,69**	**0,83**
		y	**0,86**	**0,82**	**0,87**
		z	**0,79**	**0,79**	**0,86**
		w	**0,74**	**0,74**	**0,78**
	RMSE	x	0,12	0,15	0,11
		y	0,10	0,12	0,10
		z	0,06	0,06	0,05
		w	0,29	0,29	0,26
Test Indoor1	TEST LOSS		**0,86**	**0,44**	**0,37**
	R2	x	**0,16**	**0,58**	**0,83**
		y	**0,30**	**0,81**	**0,83**
		z	**0,21**	**0,78**	**0,85**
		w	**0,08**	**0,70**	**0,76**
	RMSE	x	0,24	0,17	0,11
		y	0,23	0,12	0,12
		z	0,11	0,06	0,05
		w	0,54	0,31	0,28
Test Indoor2	TEST LOSS		**1,00**	**0,45**	**0,37**
	R2	x	**-0,86**	**0,64**	**0,81**
		y	**0,51**	**0,82**	**0,84**
		z	**-2,78**	**0,77**	**0,86**
		w	**0,24**	**0,67**	**0,75**
	RMSE	x	0,36	0,16	0,11
		y	0,20	0,12	0,11
		z	0,24	0,06	0,05
		w	0,50	0,33	0,29

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

6.3 Qualitative Evaluation

In this section, we do not consider precise metrics for assessing the model's fitness. Instead, we extend the reasoning over numbers and rely on simulations to create various visualizations. These are used to produce a behavioral evaluation of the results. In practice, we confront the models' predictions with the ground truth from a qualitative point of view.

A first evaluation, detailed in Section 6.3.1, presents a comparison with timeline charts. They investigate the general trend of each model on different test sets. Next, in Section 6.3.2, we make use of interactive visualizations to interpret the models' results from a human perspective.

6.3.1 Timeline Analysis

In this section we carefully analyze our three models' behavior on different test sets. This is done through a simulation of about 19 seconds[1] for comparing the ground truth and the values predicted by each considered model.

For the evaluation, we take into consideration the three test sets presented in the previous section (`arena`, `indoor1` and `indoor2`). Moreover, we add another test set created again through background replacement, as the other two. The difference is that this new test set is composed of a set of multiple backgrounds, rather than a single one. Thus, we call it the `mixed` test set, composed of a total of 27 backgrounds, which include the 2 indoor scenes of before (Figure 6.4), 3 pictures from the drone arena, 5 textures, and 17 outdoor photos coming from diverse scenarios. Most of the backgrounds in the `mixed` test set (some are shown in Figure 6.5) represent real-world scenes. Hence, we expect them to be particularly challenging for the `Arena` model. Note that consecutive images in the dataset are randomly assigned to one of the available backgrounds, hence there is no relation between a certain part of the test set and its background.

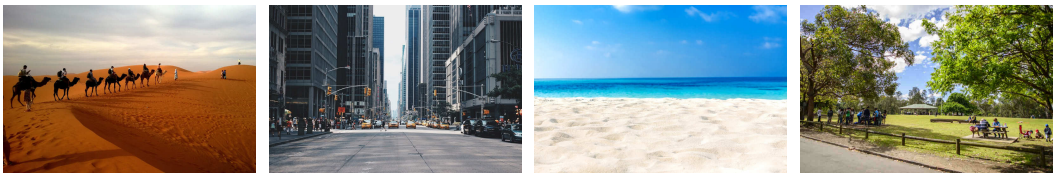


Figure 6.5: Examples of backgrounds from the `mixed` test set

Test set `arena` (Figure 6.6) All the models achieve good results on the original test set. As in quantitative evaluation, there is no significant difference between the three models. For all variables, the predictions follow the GT reasonably accurately.

[1] the dataset is considered to be collected at 30 FPS

It sometimes happen that one of the models registers slightly better results than the others, but this is a non-systematic behavior. Performance is very similar to the ones presented in Figure 2.4, obtained by the original FrontalNet model according to Mantegazza et al. [33].

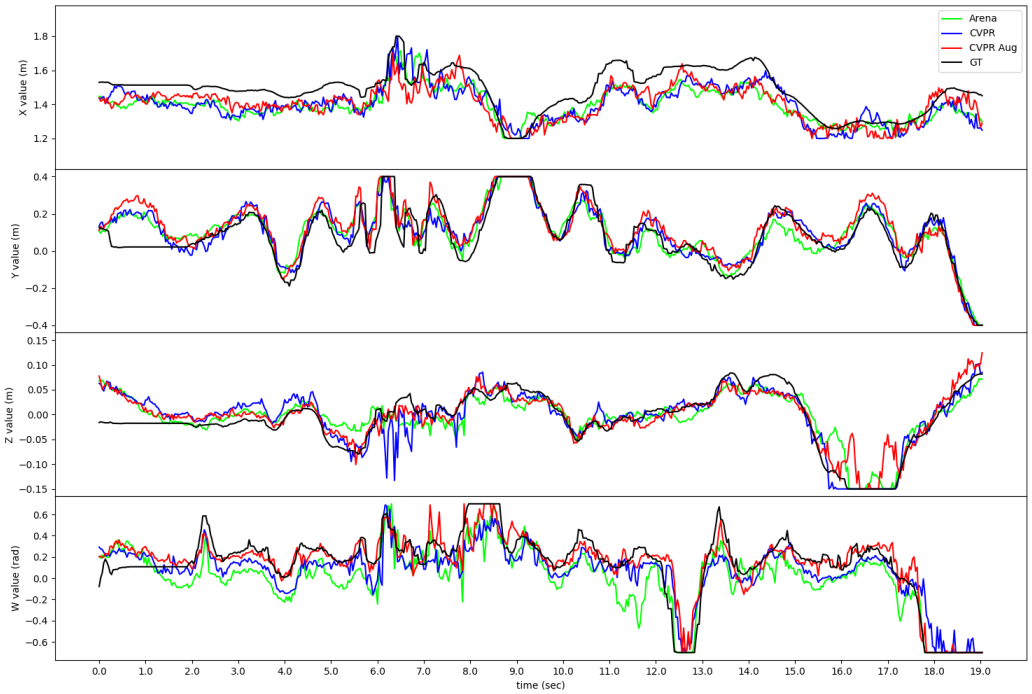


Figure 6.6: Qualitative evaluation: GT vs. predictions results on the `arena` test set

Test sets `indoor1` and `indoor2` (Figures 6.7 and 6.8) With artificial test sets, things start to change. None of the networks already know the respective backgrounds[2], but severe differences arise between the models.

`CVPR` and `CVPR Aug` are both able to produce predictions very similar to the GT, for all the variables. Overall, the model trained on augmented images (the red one in the graphs) appears to be more robust than the other.

On the contrary, the `Arena` model only follows the ground truth during a tiny percentage of the time, and only for the variables `Y` and `W`. Instead, predictions produced for the `X` and the `Z` variables seem always irrelevant and totally uncorrelated with the desired value. For `indoor2`, the variable `Z` even assumes a value higher than 0.15 for the majority of the time.

[2] which means that the three models have not been trained on these new backgrounds

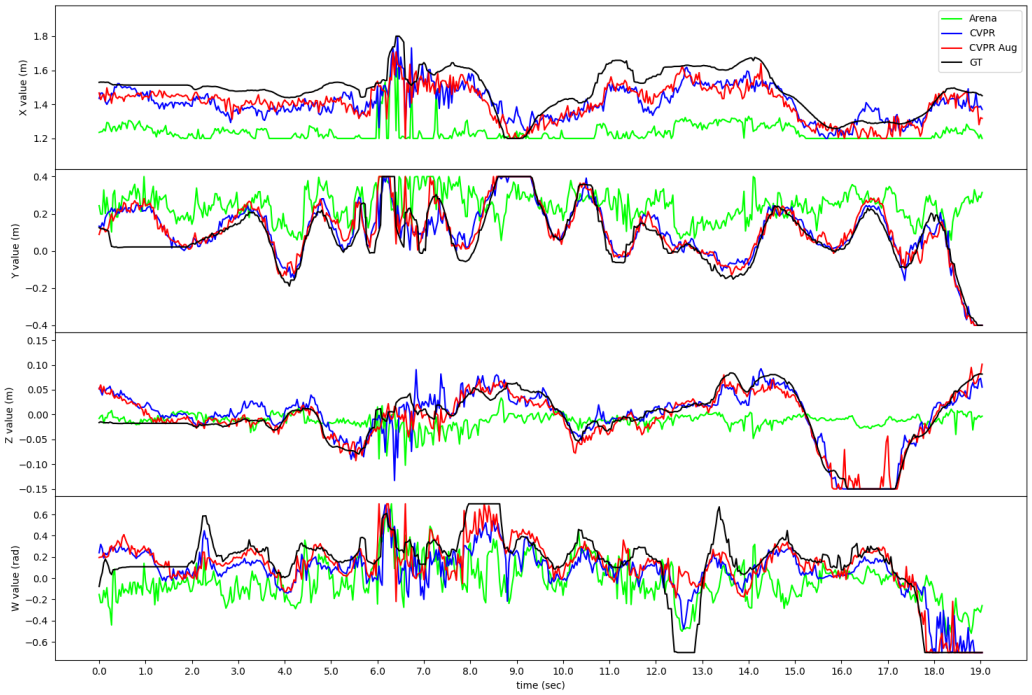


Figure 6.7: Qualitative evaluation: GT vs. predictions results on the `indoor1` test set

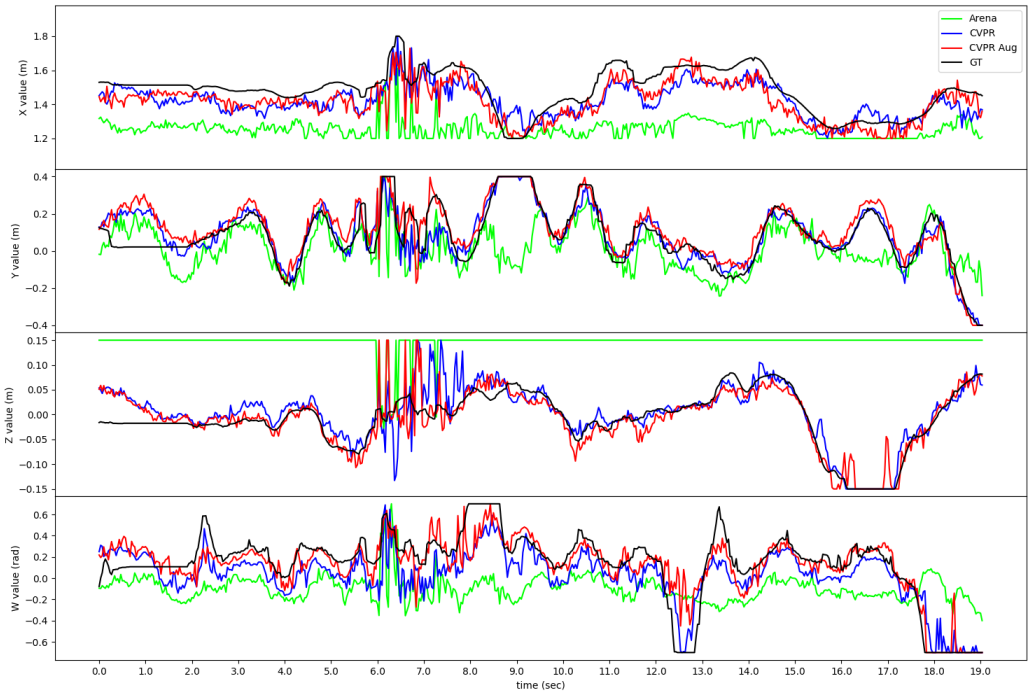


Figure 6.8: Qualitative evaluation: GT vs. predictions results on the `indoor2` test set

Test set `mixed` (Figure 6.9) Evaluation on the `mixed` test set further provides evidence that the `Arena` model is not properly able to predict the user's pose in an unseen scenario. Even worse than before, here the model predictions seem random since they continuously go up and down arbitrarily. This effect, not observed for previous test sets, is due to the composition of the dataset. The `mixed` test set has different backgrounds for subsequent samples. Thus, the oscillations occur as a symptom of the model sensitivity to the background, as demonstrated through Grad-CAM in Section 4.2.

Conversely, the models trained on background-replaced training sets can perform their task as they did on the original test set, reasonably following the GT.

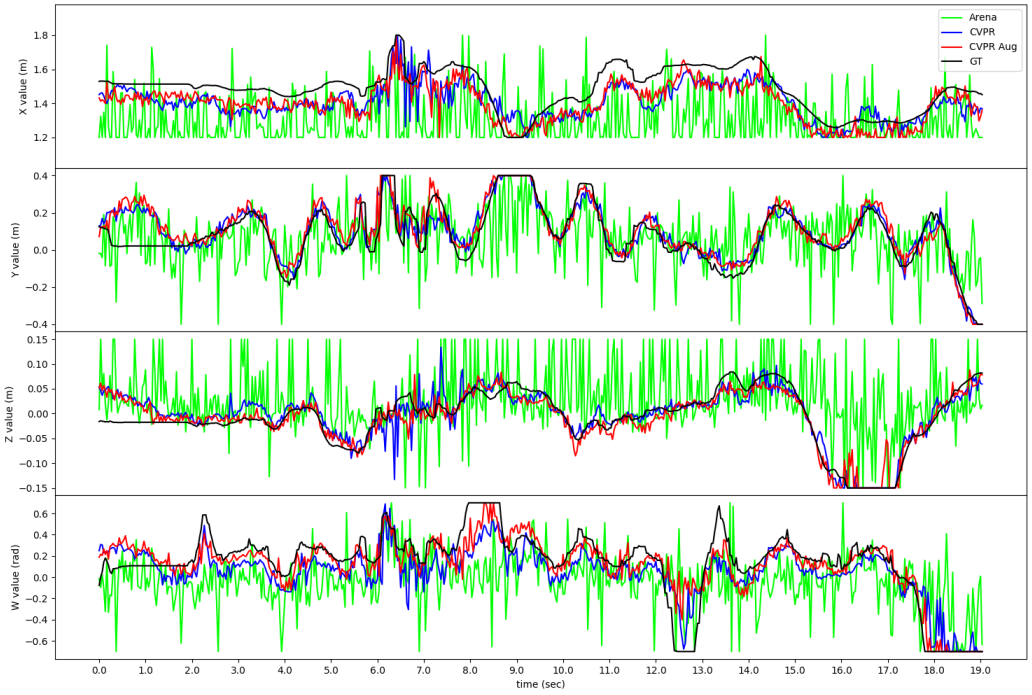


Figure 6.9: Qualitative evaluation: GT vs. predictions results on the `mixed` test set

Additional analysis on predictions variance

According to the previous charts, the `CVPR` and the `CVPR Aug` models should be able to predict the user's pose in unknown environments. However, the predictions present much more oscillations than the ground truth, even on the `arena` test set. This phenomenon can lead to undesired consequences when using the model to control the real drone. Strong oscillation can cause the controller to produce unfeasible movements. For this reason, we want to ensure that the variance of our predictions is not much greater than the variance of the ground truth.

We consider different time windows for computing the variance. Each of them bring to the same conclusion. For simplicity, in this thesis we focus on a four seconds window, meaning that we compute the variance every 120 frames. The visualization is done through a timeline, shown in Figure 6.10 for the `arena` and the `mixed` test sets. From the charts, we observe that the variances over time of both `CVPR` and `CVPR Aug` are aligned with the variance of the GT. For this reason, we conclude that such models can most likely be used to control the real drone.

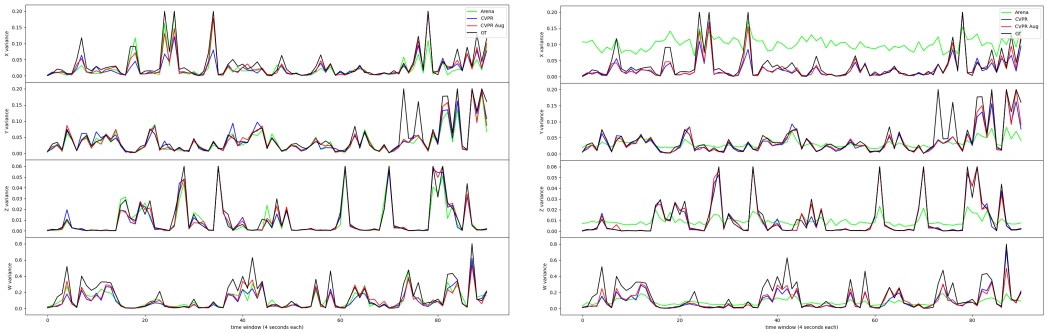


Figure 6.10: Qualitative evaluation: variance on `arena` and `mixed` tests set with a 4 seconds time window

6.3.2 Per-frame Analysis

To complete the evaluation, we test the model on new images. Being not able to experiment with a MoCap system in an unknown environment, we collect new data without having the corresponding ground truth. For this reason, the following assessment is only based on human interpretation.

For simulating the drone's camera, we use a low/medium-tier smartphone from 2017. The device is equipped with a 12 MP f/2.2 camera, able to shot 1080p videos at 30 FPS. Data have been acquired with three different subjects in a large variety of situations, both indoor and outdoor.

We rely on a visualization that simultaneously shows the considered frame and all the necessary information about the variables' predictions. The camera image is in the center, surrounded by the predicted values. Models `Arena`, `CVPR`, `CVPR Aug` are shown with markers in green, blue, and red, respectively. For the analysis, we evaluate the markers' position over the variables axis to check whether they comply with the image. A summary of the variables' interpretation is shown below.

- `X`, on the left, is the distance from the camera. When the user is 1.5 meters distant, the marker is in the center. If the person is further, the marker goes up; otherwise, it goes down.
- `Y`, on the bottom, represents the horizontal alignment. Usually, the relative marker exactly maps the user's position in the image over the x-axis. In other

words, when the prediction is correct, the `Y` marker is precisely under the person in the frame.

- `Z`, on the right, represents the vertical alignment. When the user's face is vertically aligned with the camera, the marker is centered. If the user lowers, the mark goes down; otherwise, it goes up.

- `W`, on the top, is the orientation of the head. When the user is perfectly facing the camera, the marker will be centered; otherwise, it will follow the user's sight. For example, if the person turns the head on his right, the marker will go left, following the user's gaze.

Figure 6.11 presents a complete overview of the models' behavior. The examples show many different scenarios: a park, an empty room, a street, different light conditions, weird user's positions, and even a face partially covered by a medical mask. The `CVPR Aug` model (in red) overall achieves the best results in tracking the user's pose in the image. Variables `X`, `Y` and `Z` generally follow the user properly, without many estimation errors. The `W` is more prone to fail on its task.

The main differences between the three models are observed for the `X` and the `Z` variables, coherently with the metrics reported in Table 6.1. This is especially visible when considering tough situations, such as people jumping (Figure 6.12) or running (Figure 6.13). The last case also demonstrate that the `CVPR Aug` model can surprisingly work with a person who is not actually facing the camera.

Some experiments also consider multiple people in the camera's frame. In such situations, `CVPR Aug` sometimes correctly tracks the foreground person (figure 6.14) while other times does not explicitly choose one to follow (Figure 6.15). Variables `X` and `Y` are the most suited to be considered in such situations, as easily understandable by humans. Many reasons can be the cause of this problem, and further investigation is still needed. We also notice, from Figure 6.16, that if multiple people are present in the image and one of them moves out of the scene, then the model instantly gives its attention on the remaining person.

The main issue which arises from the qualitative evaluation regards the `W` variable. A large percentage of wrong predictions are made when the user is at the margins of the image. The variable seems affected by the horizontal position of the person, regardless of the head orientation. If the user is in the leftmost part of the frame, `W` is most likely predicted as if the user is watching on his left (thus, the marker goes on the right part of the axis). Otherwise, when the person is on the right, the `W` marker often appears on the left. Figure 6.17 clearly illustrates this phenomenon, which is also observable in other examples. The problem is probably deriving from a bias in the training data, as also stated in [32].

Figure 6.11: Qualitative evaluation: overview of the models' behavior on newly recorded images

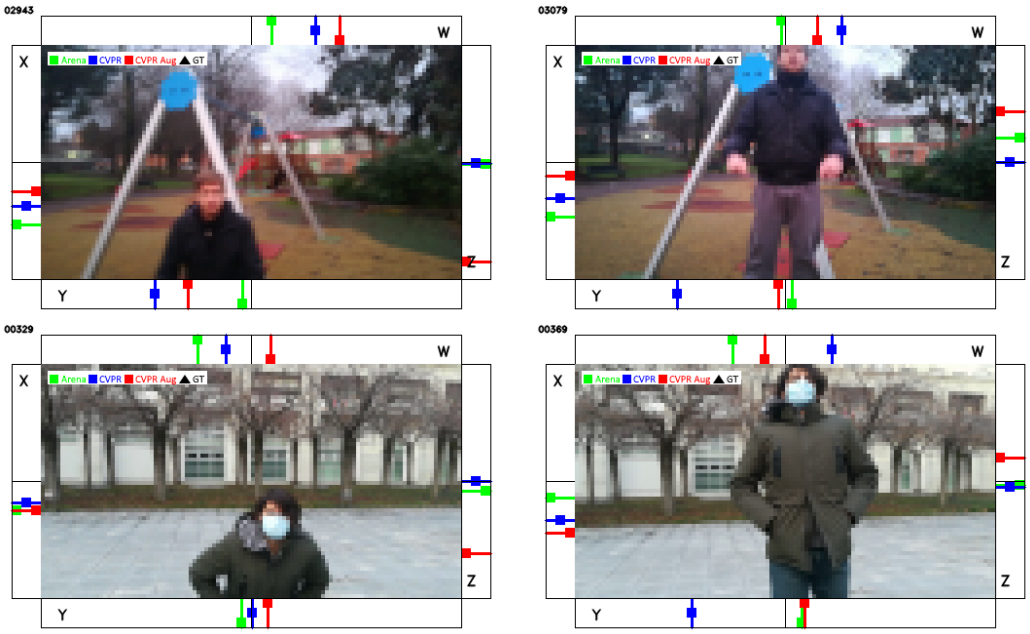


Figure 6.12: Qualitative evaluation: models' behavior on jumping people. The `Z` variable is correctly predicted only by the `CVPR Aug` (red) model.

Figure 6.13: Qualitative evaluation: models' behavior on running people. `CVPR` is even worse than the `Arena` model. `CVPR Aug` is mostly superior for the `X` and the `W` variables.

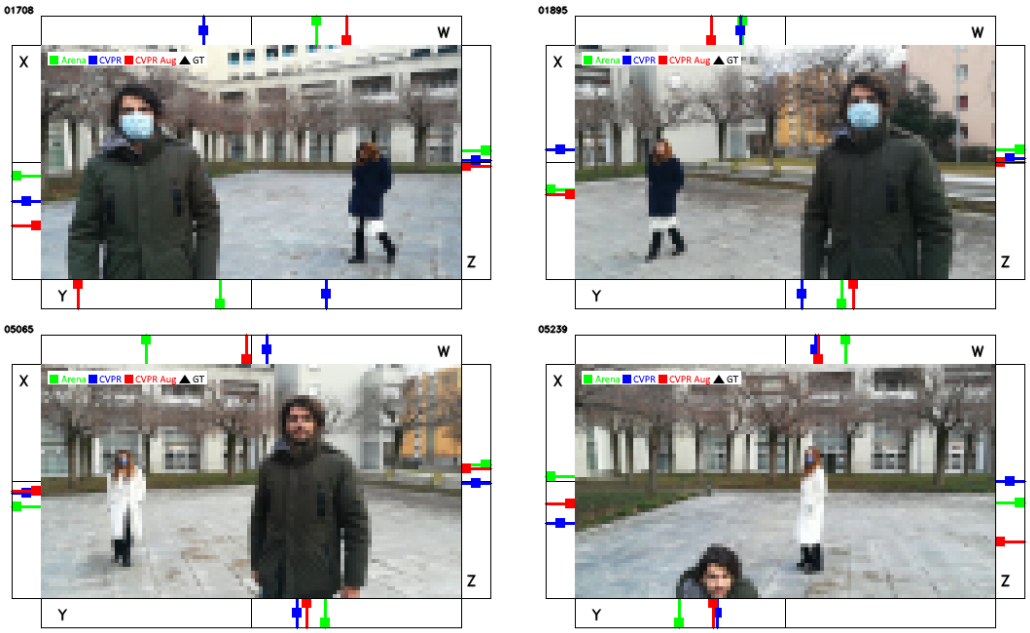


Figure 6.14: Qualitative evaluation: models' behavior on multiple people. In all the images, the models track the person in the foreground, as desired. Focus on `X` and `Y` for briefly understanding what is going on behind the models' reasoning. The reason behind such precision is probably found on the far distance of the background user and even on its clothes color. The most surprising result is observed in the 4th frame, for which the foreground user is detected even if it appears at the bottom of the image.

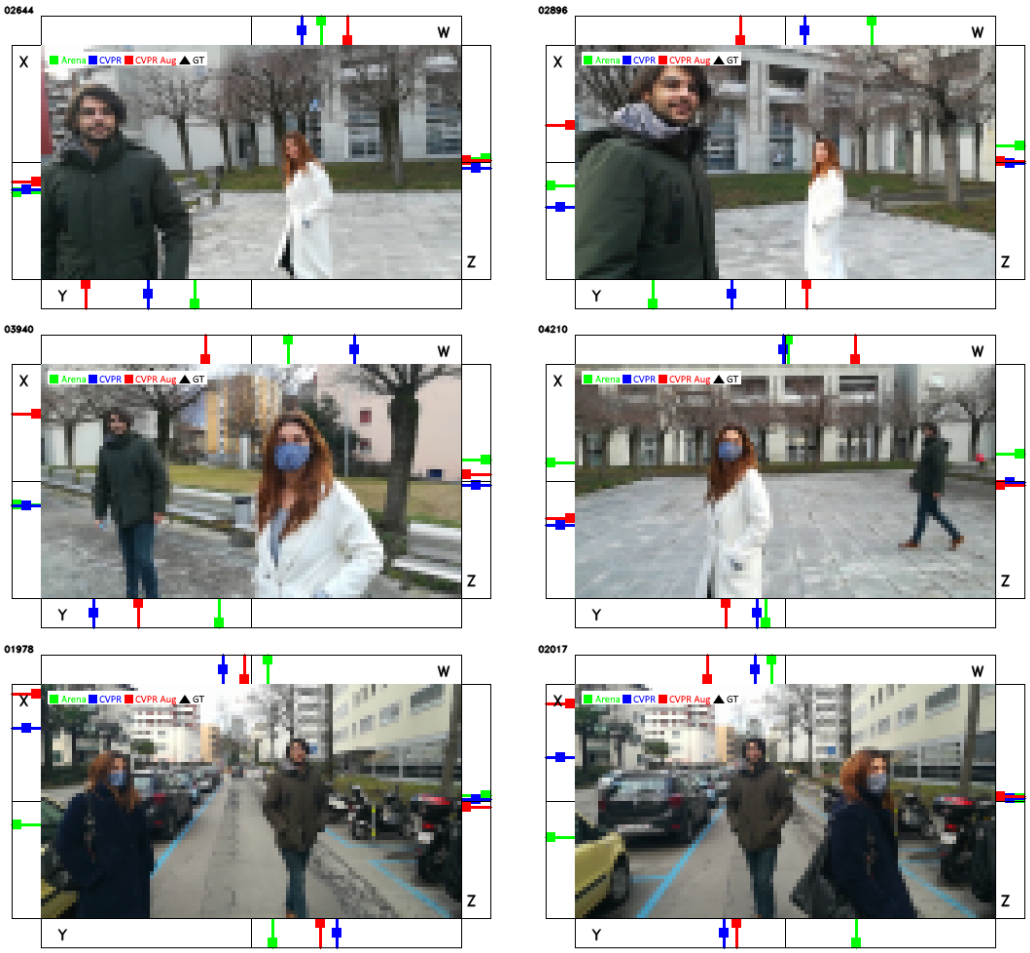


Figure 6.15: Qualitative evaluation: models' behavior on multiple people. The models is highly uncertain on the person who must track. Focus on `X` and `Y` for briefly understanding what is going on behind the models' reasoning. From left to right, top to bottom, the foreground person is tracked in images 1 and 4. In the others, the background person is taking the model attention. This can be due to specific distances, clothes colors, and also faces partially covered by a mask.

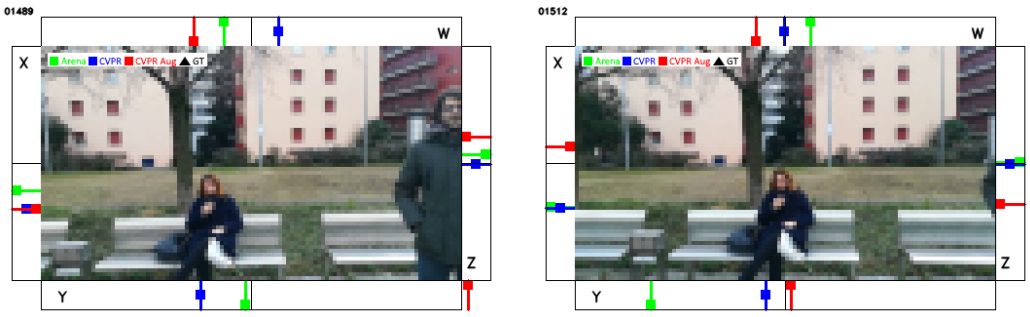


Figure 6.16: Qualitative evaluation: models' behavior on multiple people. As soon as the person in the right moves out, the seated user becomes tracked by the model.

Figure 6.17: Qualitative evaluation: `CVPR Aug` issue on `W` variable. When the user is on one side, most of the time, the `W` is predicted on the opposite side. Only the last image is showing a correct behavior, which is anyway caused by the same bias that affects the wrong predictions.

Chapter 7

Conclusion

In this thesis, we presented a methodology for generalizing the neural network defined in [33]. Designed to predict a user's pose from an image, the model was not able to achieve its task out of the training environment. We applied Grad-CAM to understand what the model was actually considering while computing its prediction, and we discovered that the network was suffering from many biases coming from the training data.

As a solution, we proposed a modification of the dataset to reduce overfitting. Our approach consists of applying background replacement and image augmentation to the original images for the creation of new data. Retraining the original neural network on the data obtained after our augmentation pipeline, we created an enhanced version of the model.

Our quantitative and qualitative evaluation experiments demonstrated that our improved model is capable of solving the task in unseen environments. Background replacement is not sufficient if used alone, but combined with image augmentation obtains surprising results. The final model produces encouraging predictions both indoor and outdoor, even from images recorded with a smartphone camera. According to the initial objective, we succeeded in enhancing the original model's generalization capabilities.

Future Works

We suggest a couple of future milestones for certifying the potential of our work and expand it towards new research possibilities.

- Until now, our model's results have been only verified by simulation. A complete test with the real drone is required to confirm the success of the solution in the task. We want to make sure that our approach can predict a user's pose in any environment through the actual drone's camera.

- More recent interpretability techniques can be applied on the new generalized model, such as Score-CAM [69]. In case spatial attribution provides evidence of the model's ability to recognize people, also feature visualization (Section 2.3.1) can be tried for a better understanding of what convolutional layers of the model actually learn.

- As detailed in Section 2.2.4, a recent work in IDSIA [82] adapted FrontalNet to run on-board an ultra-low-power nano-drone. By applying our augmentation pipeline on that work, we would be able to prove the effectiveness of our solution on different platforms designed to achieve the same task.

- The augmentation pipeline we propose can actually be used for various tasks. Our background replacement approach could become a good alternative to more standard domain randomization techniques (Section 2.4.2). Running high-expensive inferences offline, as we did with Mask R-CNN, we enable the opportunity to aggressively augment a dataset. That dataset can be used to train a simpler model that may be able to provide faster and real-time predictions, even on low-end devices.

Appendix A

Extra Figures

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

A.1 FrontalNet

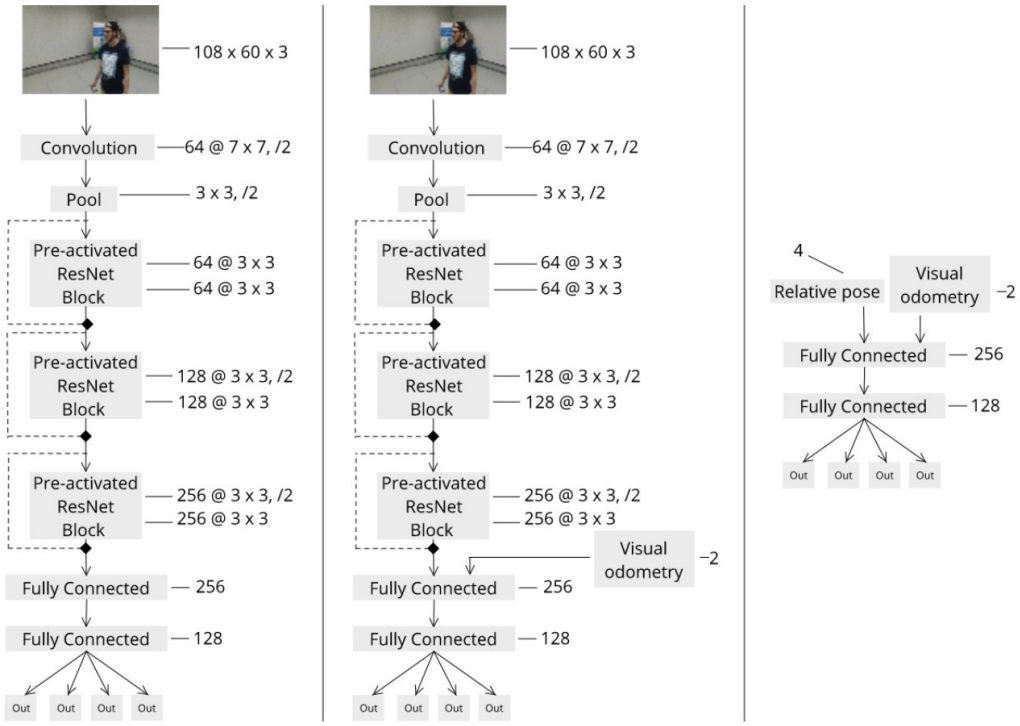


Figure A.1: Models by [33]: mediated, end-to-end, learned controlled

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

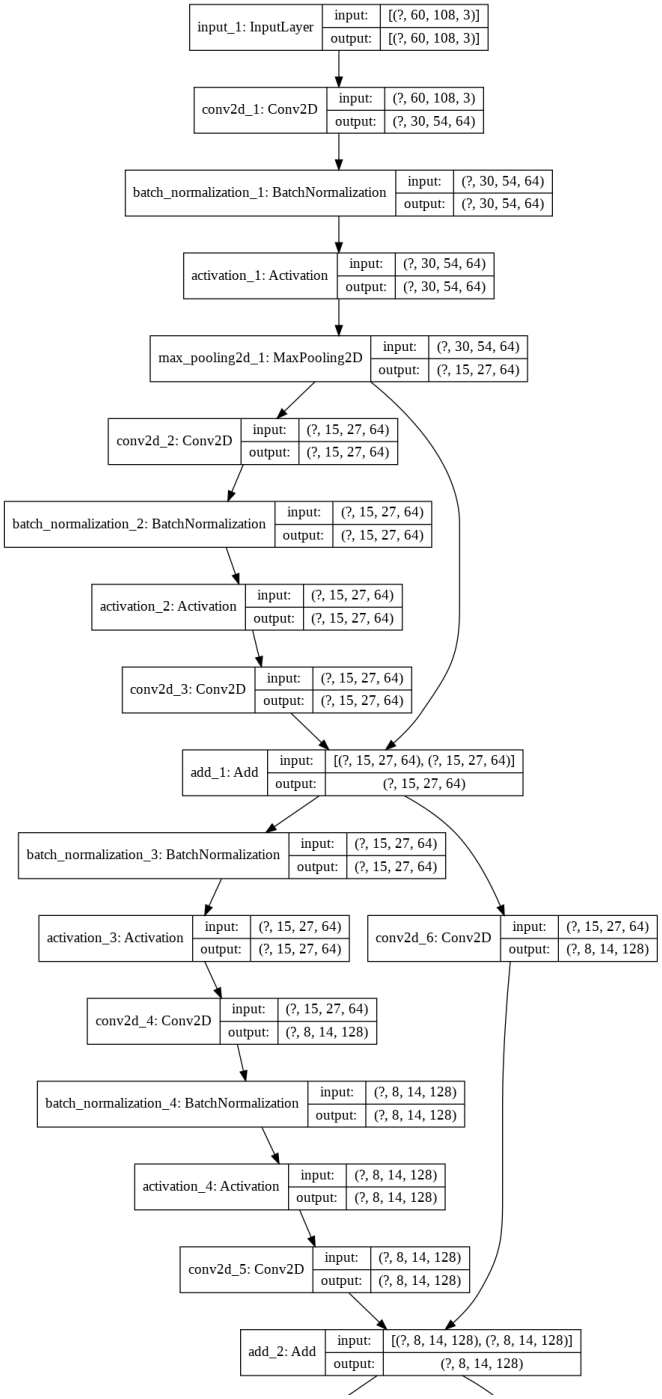


Figure A.2: FrontalNet complete architecture (part 1, from input to layer 18)

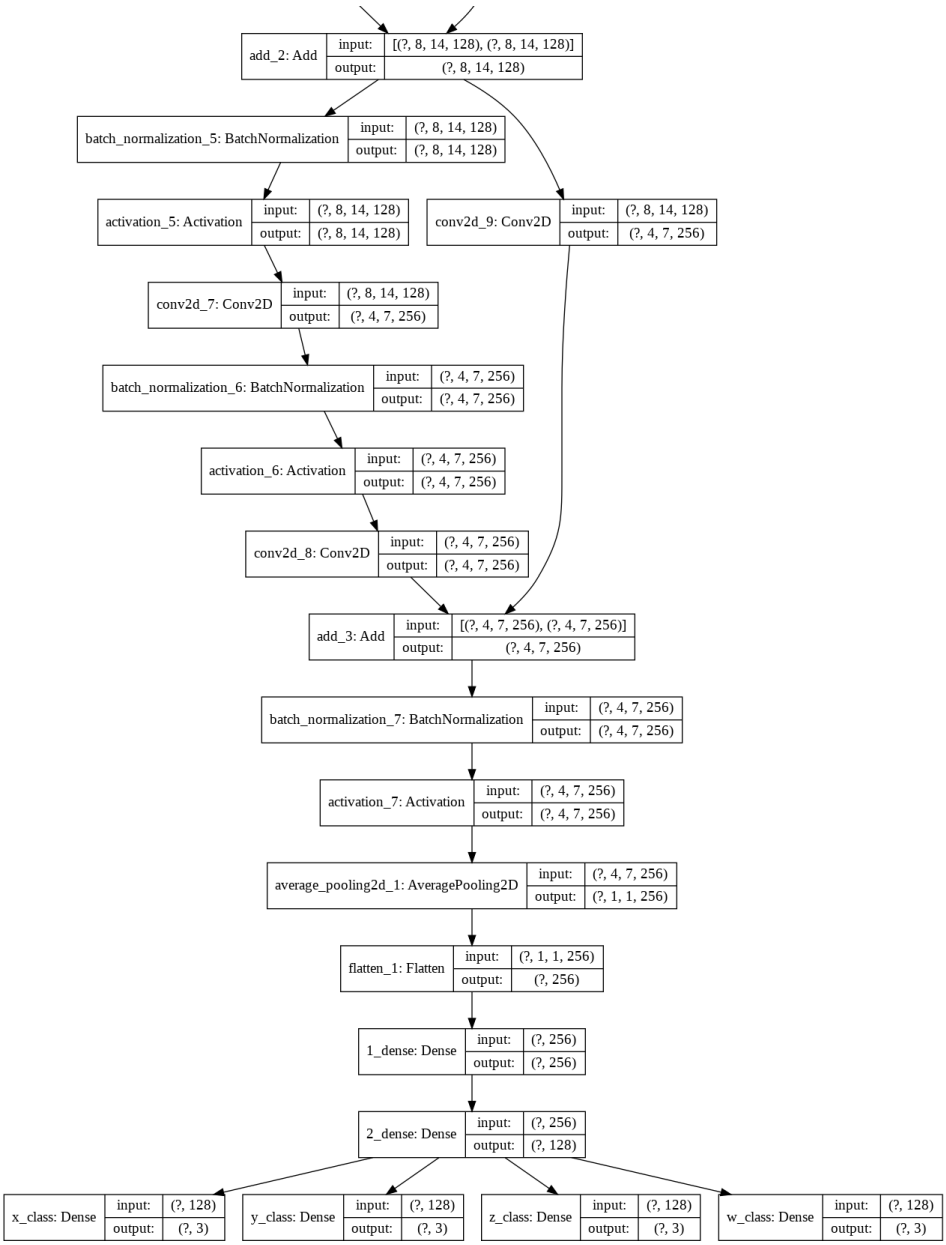


Figure A.3: FrontalNet complete architecture (part 2, from layer 18 to outputs)

Figure A.4 presents qualitative results from [33]. It shows the trajectories followed by the drone during five consecutive flights in the drone arena. The quadrotor has to face a user initially rotated by 90 degrees. Although the paths (obtained by 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.

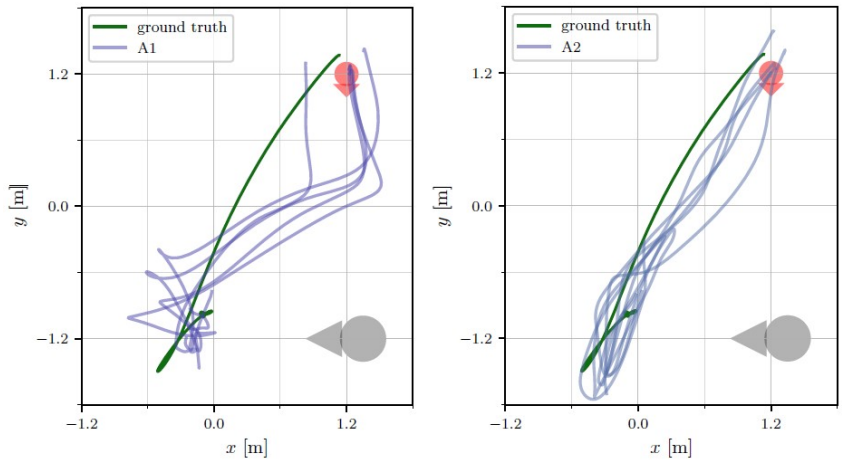


Figure A.4: FrontalNet trajectories for positioning in front of the user initially rotated by 90 degrees [33]

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.

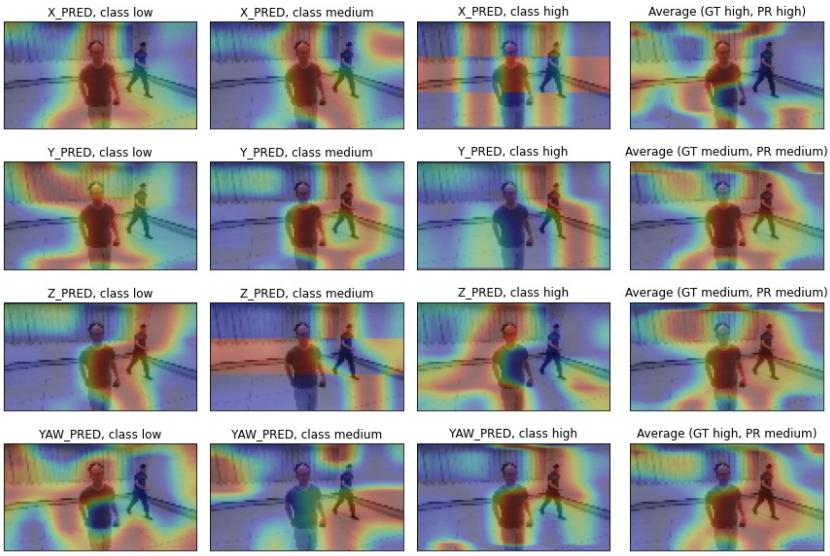


Figure A.5: Full Grad-CAM: two people in the frame

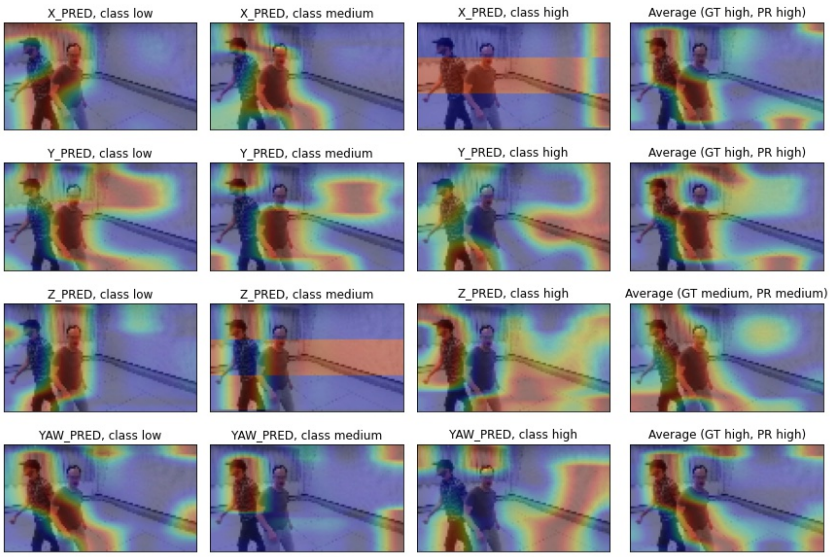


Figure A.6: Full Grad-CAM: two people in the frame

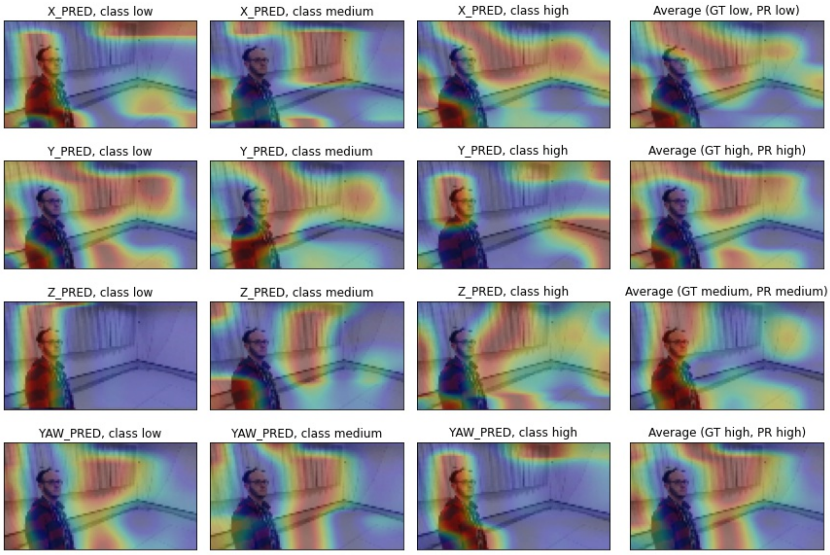


Figure A.7: Full Grad-CAM: model is attracted by curtains

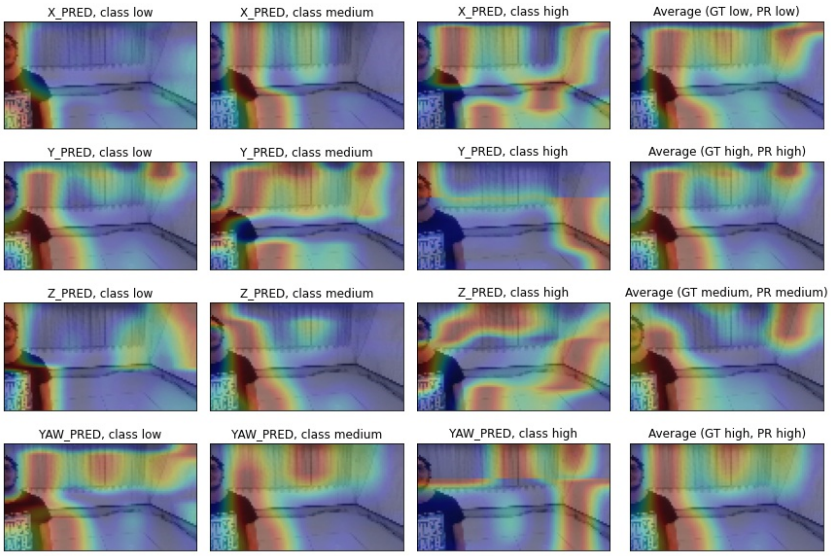


Figure A.8: Full Grad-CAM: model is attracted by curtains

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

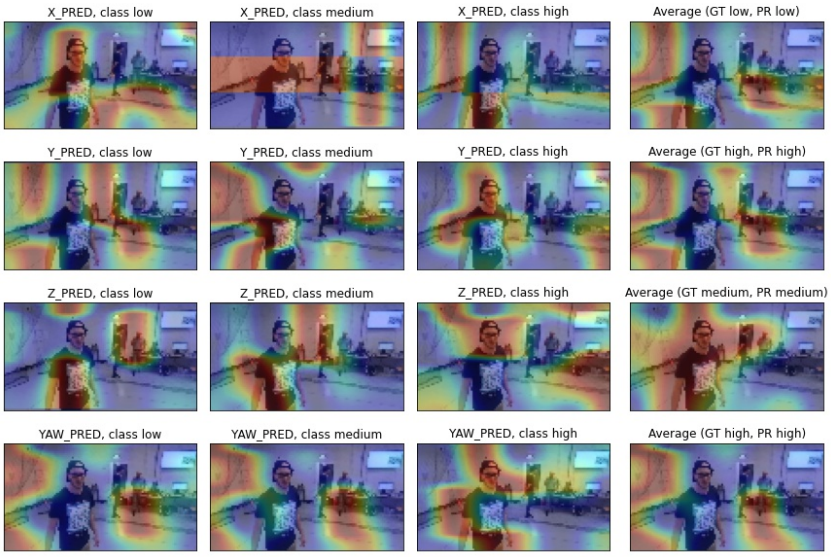


Figure A.10: Full Grad-CAM: model is attracted by background objects

Appendix B

Acronyms

ADAM Adaptive Moment Estimation

AI Artificial Intelligence

CAM Class Activation Mapping

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

DoF degrees of freedom

ETH ETH Zürich

FAA United States Federal Aviation Administration

FOV field of view

FPS frames per second

GAP Global Average Pooling

Grad-CAM Gradient-weighted Class Activation Mapping

GT ground truth

IDSIA Istituto Dalle Molle di Studi sull'Intelligenza Artificiale

IL Imitation Learning

IR infrared

MAE Mean Absolute Error

MIT Massachusetts Institute of Technology

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

UAV Unmanned aerial vehicle

Unimib University of Milano-Bicocca

USI Università della Svizzera Italiana

UZH University of Zürich

VR Virtual Reality

wrt with respect to

XAI Explainable AI

References

[1] Aashmango4793. Flight dynamics: roll, pitch and yaw, 2019. URL https://commons.wikimedia.org/w/index.php?curid=81688701. Visited on Jan 2021.

[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] 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.

[5] Jason Brownlee. A gentle introduction to batch normalization for deep neural networks, 2019. URL https://machinelearningmastery.com/batch-normalization-for-training-of-deep-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] Alexis Cook. Global average pooling layers for object localization, 2017. URL https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/. Visited on Jan 2021.

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

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

[12] Dive into Deep Learning. Residual networks (resnet) explanation, 2020. URL `https://d2l.ai/chapter_convolutional-modern/resnet.html`. Visited on Jan 2021.

[13] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 01 2009.

[14] Rohith Gandhi. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms, 2018. URL `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`. Visited on Jan 2021.

[15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[16] Alessandro Giusti, Jerome Guzzi, Dan Ciresan, Fang-Lin He, Juan Pablo Rodriguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jurgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, and Luca Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 2016.

[17] 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.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

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

[20] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

[21] 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`.

[22] 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.

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

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

[25] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, Apr 2020. ISSN 1573-7462. doi: 10.1007/s10462-020-09825-6. URL `http://dx.doi.org/10.1007/s10462-020-09825-6`.

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

[27] Yasuhiro Kubota. tf-keras-vis is a visualization toolkit for debugging tf.keras models in tensorflow 2. `https://github.com/keisen/tf-keras-vis`, 2020. Visited on May 2020.

[28] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017. ISSN 2169-3536. doi: 10.1109/access.2017.2696121. URL `http://dx.doi.org/10.1109/ACCESS.2017.2696121`.

[29] 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.

[30] 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/`.

[31] 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.

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

[33] 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`.

[34] Maja J Matarić et al. *The robotics primer*. Mit Press, 2007.

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

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

[37] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks, 2016.

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

[39] 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.

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

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

[42] 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.

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

[44] 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.

[45] 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.

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

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

[48] 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`.

[49] 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/`.

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[51] 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`.

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

[53] 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.

[54] Sabyasachi Sahoo. Residual blocks — building blocks of resnet, 2018. URL `https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec`. Visited on Jan 2021.

[55] 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`.

[56] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2016.

[57] 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`.

[58] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

[59] Cecilia Summers and Michael J. Dinneen. Improved mixed-example data augmentation, 2019.

[60] Gabor Sziebig. Interactive vision-based robot path planning. 01 2007.

[61] 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`.

[62] 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.

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

[64] 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/`.

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

[66] 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`.

[67] 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.

[68] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL `http://proceedings.mlr.press/v28/wan13.html`.

[69] Haofan Wang, Zifan Wang, Mengnan Du, Fan Yang, Zijian Zhang, Sirui Ding, Piotr Mardziel, and Xia Hu. Score-cam: Score-weighted visual explanations for convolutional neural networks, 2020.

[70] 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.

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

[72] Wikipedia. Morphing, . URL `https://en.wikipedia.org/wiki/Morphing`.

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

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

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

[76] 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.

[77] 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.

[78] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013.

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

[80] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization, 2015.

[81] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

[82] Nicky Zimmerman. Embedded implementation of reactive end-to-end visual controller for nano-drones, 2020. URL `https://github.com/FullMetalNicky/FrontNetPorting`. Master Thesis.

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

[84] Martin Zurowietz and Tim W. Nattkemper. An interactive visualization for feature localization in deep neural networks. *Frontiers in Artificial Intelligence*, 3(49), July 2020. doi: doi:10.3389/frai.2020.00049. URL `https://doi.org/10.3389/frai.2020.00049`.