POLITECNICO DI TORINO

Master's Degree in Automation and cyber intelligent physical systems

Master's Degree Thesis

Drone to human pose estimation with deep neural networks

Supervisors

Prof. DANIELE JAHIER PAGLIARI

Prof. DANIELE PALOSSI

Prof. CHRISTIAN PILATO

Candidate

LUCA CRUPI

10/2022

Summary

Nano-drones are capable of performing a vast amount of tasks that are not doable in any other, comparably versatile, way, including: indoor surveillance, safe and rescue and inspection.
The reduced size and cost, as well as the limitation in power supply for computational purposes (under 100 mW) and computational capabilities make running deep learning models on these devices particularly challenging.
The aim of this work is to study automated ways to design and deploy sufficiently shrank Neural Network (NN) architectures, reducing the number of parameters, number of operations of an input seed network. In order to address this task we employed a novel Network Architecture Search (NAS) technique called, Pruning In Time (PIT) [1]. PIT was previously design and tested on 1D networks and TCNs, but with this thesis we extended its use to 2D models and demonstrate its capabilities on a Drone-to-human pose estimation task on the Crazyflie 2.1 nano drone where, the prediction variables are x, y, z and ϕ (angle of rotation around z). The main interest for this NAS relies on the fact that its search time is approximately equal to the time of training the actual network.
Since the GAP8 System-on-Chip (SoC), mounted onboard the Crazyflie, has only 512 KB of L2 memory, model size reduction was crucial in order to avoid an increase of the latency due to accesses in the off-chip DRAM memory. The architectures obtained from the NAS, that belong to the Pareto front of the cycle-performance analysis, have been carefully tested on testset images and in on-field experiments. Starting from two different seeds, FrontNet and MobileNet v1, we were able to obtain up to 5.6x size reduction that helped in providing up to 50% faster inference. Performance improved by 5% with respect to FrontNet [2], evaluated through the Mean Absolute Error (MAE) of the distance between predictions and ground truth relative pose of the x, y, z and ϕ variables.
Training and testing images and, corresponding labels, were acquired in the Manno (CH) laboratory and used for the selection of the various networks as well as a first attempt performance analysis. In field tests instead were performed in a never seen before environment, namely, the Lugano (CH) laboratory. Our NAS technology was able to design networks that perform up to 48% better with respect to FrontNet in

the new environment, in terms of MAE on the most challenging variable to predict (ϕ).
In the on-field experiments the control performance of the drone improved by 32% in terms of absolute distance and 24% for what concerns the yaw control angle. It is worth noting that the path walked for testing, in the never seen before environment, was completed only by the architectures obtained by PIT starting from a MobileNet seed. FrontNet seed and derived models were not able to conclude the path and, on a three experiments average, they completed at most 85% of it.
In summary, the thesis demonstrates that efficient NAS techniques can be successfully employed to optimize deep learning models on constrained robotic platforms, reducing the size and complexity of networks while simultaneously improving their predictive performance.

Acknowledgements

This project was realized thanks to the efforts of many people. I express my gratitude to my supervisors Daniele Jahier Pagliari, Daniele Palossi, Christian Pilato, Alessio Burrello, Matteo Risso and Elia Cereda, who devoted much time to guide my work for the sake of the best possible results.

Thanks to the Polytechnic University of Turin and Dalle Molle Institute for Artificial Intelligence that provided me with this challenging opportunity.

And, last but not least, thanks to my family and godfather, my girlfriend, my best friends and all the close people, who have always supported me and were ready to help in every moment through this complicated path.

Sincerely, thank you.

"A person who never made a mistake never tried anything new".
Albert Einstein

Table of Contents

List of Tables

List of Figures

Acronyms

AI

artificial intelligence

IoT

Internet of Things

ML

Machine Learning

SoC

System on Chip

CL

Cluster

FC

Fabric Controller

fps

frame per second

NAS

Network Architecture Search

MACs

Multiply and accumulate operations

SoA

state-of-the-art

MAE	Mean Average Error
MSE	Mean Squared Error
ToF	Time of Flight
FoV	Field of View
ROI	Region of interest
FPU	Floating Point Unit
PULP	parallel ultra low power
DMA	Direct Memory Access
NN	Neural Network
TCN	Temporal Convolutional Network
PIT	Pruning in Time
GT	Ground Truth
RELU	REctified Linear Unit

NEMO	NEural Minimization for pytorch
DORY	Deployment Oriented to memoRY
CNN	Convolutional Neural Network
SGD	Stochastic gradient descent
BN	Batch Normalization
DNAS	Differentiable NAS
FLOPS	Floating Point Operation Per Second
MLP	Multilayer Perceptron
FQ	Fake quantized

Chapter 1

Introduction

1.1 Background and Motivation

Robotics platforms and Internet of Things (IoT) devices are pervasive instruments nowadays since people around the world are continuously helped by them. In particular, considering how simple is to be surrounded by these objects, smartphones, watches, cars, home assistant robots and many more types of machines are constant parts of our lives.
With the advent of Machine Learning (ML), the focus switched from the creation of the aforementioned systems to the automatization of them, with the final aim of creating autonomous machines, an evergreen dream of humankind.
In this regard, ML may be seen as a way to let machines improve their behavior (as measured by quantitative metrics) thanks to the use of labeled or unlabeled data. In order to perform ML tasks, these devices need a conspicuous power budget and in activities that require mobility, there are usually strict constraints of energy which is provided in general by battery systems. Furthermore, latency should be considered a crucial aspect for systems that perform real-time control and tracking as in the case of this thesis. On the other side, if energy is not a constraint, memory size may be the limit and efficiency should be pursued anyway to be economically attractive for industrial applications.
Considering these crucial needs Network Architecture Search (NAS) poses a broad solution to obtain an energy-efficient system, a sufficiently tiny network, or whatever optimization metrics may be needed without reducing performance and providing an automatic way to perform the network design.
Memory and energy optimization is fundamental in the case the deployment platform is a nano-drone or an IoT device that has to perform inference for controlling purposes. These kinds of devices have in general a reduced amount of memory and computational power. This is the case also for the platform

used for this thesis, the crazyflie 2.1, and the AI-deck based on GAP8 system on chip (SoC). In fact with only 512 kB of L2 memory and less than 1 GOPs of computational power a NAS system may be particularly interesting for this kind of device and as a consequence for the entire field of research, allowing the deployment of more and more complex networks without sacrificing performance. Furthermore, considering that drones are gaining interest and more applications arise continuously a toolchain to develop and deploy efficient Neural Networks (NN) may be a significant breakthrough allowing more and more development. In particular, nano-drones are needed for indoor applications such as indoor security, safety and rescue as well as inspection of high buildings and complex systems. In this environment, considering the absence of GPS-based tracking services, NN visual navigation systems are a convenient and portable solution, independent from the environment and the subject.
This thesis propose an integrated solution employing NAS to develop a NN that can solve a pose estimation task between the crazyflie 2.1 nano-drone and a human using only an Himax camera and the computational capabilities present onboard. The aim of the work is to reduce the originally designed architecture Frontnet and provide an improved version of it while also testing two Mobilenet-based architectures.
Furthermore, every network has been tested on the GAP8 SoC and deployed in an 8-bit quantized fashion in order to limit as much as possible the computational overhead. As a result, a full pipeline is provided from the design of the neural network with the NAS to the deployment onboard, passing through quantization and tiling.
Several NAS techniques have been explored in detail, as reported in chapter 2 and, from this analysis Pruning in Time (PIT) have been chosen and employed for this work. Since PIT was originally designed for temporal convolutional networks (TCNs) some adaptations have been performed, including a 2D domain adaptation in order to perform operations and inferences on images.
The focus was then brought to the NAS research using PIT and to the exploration of the solution space changing the strength value λ. This parameter allows the balancing of the algorithm between task performance and the hardware cost metric. λ is inversely proportional, in this case, to the number of parameter in the network thus larger values of the parameter corresponding to tighter networks. With all the results obtained from the solution space exploration, a pareto analysis has been done and the architectures to be tested onfield were chosen from the front according to the methodology described in chapter 3.
Every NN selected has been tested onfield with a defined and reproducible path, described in section 3.3.4. A position tracking system has been employed that allowed the precise 3D tracking of the person and the drone in order to understand the capabilities of each network. The results that included the streaming of the

camera have been also tested in post-processing mode as described in section 4.3.5. In order to perform every test and acquire data the pipeline detailed in section 3.3.2 has been used, providing a reliable and consistent way of testing the networks both in closed loop with the controller and post-processing images.
The method proposed by this thesis was able to perform the drone-to-human pose estimation task with a reduction of 32% of the horizontal displacement error while providing a network that is up to 1.5× faster and up to 5.6× smaller with respect to the original Frontnet network. Furthermore, the networks reached about 50 Hz using only 90 mW of average power consumption.
The results obtained in this work have been submitted in a paper at the International Conference on Robotics and Automation (ICRA).

Chapter 2

Related works

2.1 Introduction to NAS

Since the beginning designing neural networks was more an art than a science, most of the development tools should be learned with practice and applied according to trial and error or previous experience. Even if the procedure may be successful, it may take an extended amount of time and resources to develop a working model. In the early days, manual research was the main choice since Neural Networks were not well established. Since the neural network proved to be very successful in their ability to adapt to a plethora of tasks, several automatic techniques have been developed in order to explore the spaces of hyper-parameters. The necessity of such automation lead to the development of an emergent research field, the so-called Neural Architecture Search (NAS).
A NAS typically works on a search space, which contains all the possible NNs variants that we want to explore.
Several strategies have been proposed in order to perform automatic searches and optimization of networks. Although it is not a proper NAS search, one of the first methods to shrink networks was an L1 term (reported in equation 2.1). This term when summed to the task loss function promotes sparsity. It was introduced in [3] and in [4].

$$L1_{Loss} = \sum |y_{true} - y_{predicted}| \tag{2.1}$$

2.1: L1 loss function

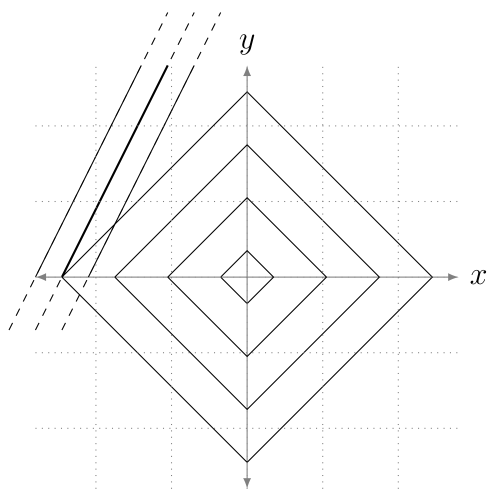


Figure 2.1: Representation of the L1 Loss in $\mathbb{R}^2$

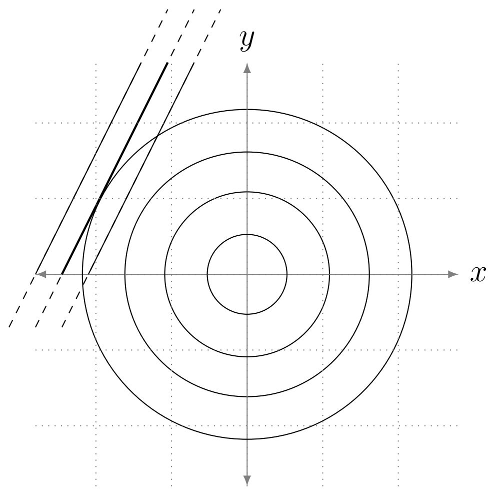


Figure 2.2: Representation of the L2 Loss in $\mathbb{R}^2$

As an example, if the optimization over the line depicted in figure 2.1 is considered, a solution that has at least one zero component is obtained. Instead in the case of L2 loss, reported in equation 2.2, is practically impossible to have a sparse solution as depicted in figure 2.2. The L1 term induces less relevant weights to decrease in the norm. However, this term does not target a specific resource to optimize and in general is not guaranteed to provide an efficient solution for modern accelerated hardware such as GPUs. The introduction of such a complexity thus does not provide reasonable assurance of a speedup in practice.
Other than weights as proposed by Optimal Brain Damage [5], also entire neurons have been an object of optimization and reduction with dedicated techniques. Even

in this case, there was not a method to target specifically a resource, such as FLOPs or size, so other methods have been explored.
The broad topic of Neural Architecture Search (NAS) has the main aim of creating

$$L2_{Loss} = \sum (y_{true} - y_{predicted})^2 \tag{2.2}$$

2.2: L2 loss function

a NN automatically taking into account constraints on size, FLOPs, or any metric properly modeled.
As a first attempt, NAS was designed considering them as a meta-learning process and guiding the exploration thanks to a meta-controller (for example a Recurrent Neural Network) that is therefore trained at each iteration.
On the one hand, this approach should lead to a complete exploration of space, and as so, the best-performing network may be found for the task.
On the other side, such a method need several thousand hours of GPUs in order to train a single model, this may be a problem since the time to get a reliable solution increase with the dimension of the task.
As a consequence, the need for automatically designed NN models and the objective of saving computational resources in the training phase is leading to the development of novel techniques for the exploration of architectural spaces.
Some effort has been devoted to the improvement of the meta-learning process in order to reduce the time required to obtain a network. A solution that shares the weights has been proposed in Pham et al. (2018) [6], another solution realized in [7] obtains such a goal with a hyper network to generate the weights, avoiding also the training phase.
Orthogonally to the NAS techniques, several nonarchitectural reduction methods have been proposed. As an example, low-bit quantization reduces the size of the network representing weights with 8 bits. Other techniques may include methods that design the entire network instead of pruning an already built one.
Several solutions to the problem started to be proposed in 2018. The most interesting for the sake of this thesis are MnasNet, ProxylessNAS, MorphNet, FBNetV2, and PIT. These NASes are respectively explored in more detail in section 2.1.1, section 2.1.2, section 2.1.3, section 2.1.4 and section 2.1.5.

2.1.1 MnasNet

MnasNet formulates the design problem as a multi-objective optimization, as described in figure 2.3 that considers both the accuracy and inference latency of CNN models. Unlike previous work [36, 26, 21] that uses FLOPS to approximate inference latency, for MnasNet direct latency measures on real-world devices have

been done, overcoming the problem that considering FLOPS as a proxy of the latency is consistently inaccurate as the case of MobileNet and NASNet, where that have similar FLOPS (575M vs. 564M) even if they have quite different latencies (113ms vs. 183ms).
Another important aspect is the fact that techniques previously developed search for a few types of cells and then stack them through the network, resulting in a simplification of the search space and consequently a restriction that prevents the research of specifical computationally efficient solutions.
This NAS technique explores a novel factorized hierarchical search space that enables layer diversity. In particular, the CNN is factorized into unique blocks, as depicted in figure 2.4, and then analyzed per block separately in order to find different architectures in different blocks. The search space is thus reduced by several orders of magnitude depending on the number of blocks B, the number of layers per block N, and search space size S. A typical case reported in [8] is a reduction of 10^{26} in the search space dimension obtained with a search space size S=432, B=5, and N=3. In the case of a complete search with a per-layer search the size should be S^{B*N} and using MnasNet it becomes S^{B}. Constraining the target accuracy, then a model obtained through MnasNet was 1.8× faster than MobileNetV2 and 2.3× faster than NASNet [36] with better accuracy. Compared to the widely used ResNet-50 [9], MnasNet-based model achieves slightly higher accuracy with 4.8× fewer parameters and 10× fewer multiply-add operations.

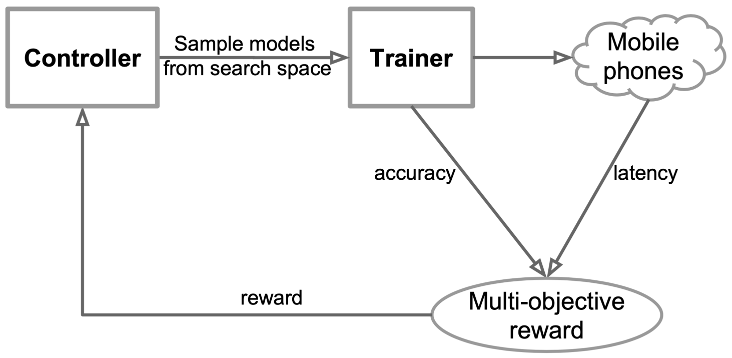


Figure 2.3: MnasNet high level schema with multi-objective reward.

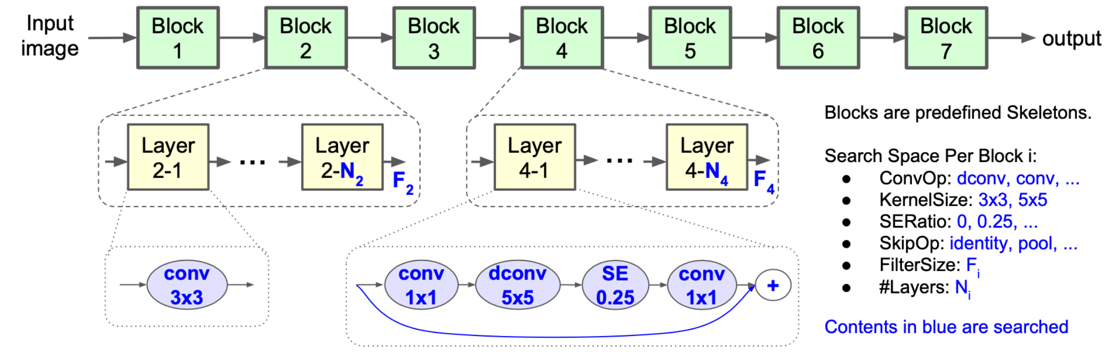


Figure 2.4: Factorized Hierarchical Search Space. Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N, then the same layer is repeated N times (e.g., Layer 4-1 to 4-N4 are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

2.1.2 ProxylessNAS

In order to solve the search task in a reduced amount of time, several proxy techniques have been introduced limiting for example the number of training samples, the number of blocks in the exploration, or searching for NN with reduced datasets. These approaches produce novel and possibly more accurate architectures but they are not guaranteed to be optimal since the space explored is limited. ProxylessNAS [9], a DNAS, achieves the same goal without using any type of proxy, exploring the full search space. In fact, it directly learns the architectures on the target task and hardware as depicted in figure 2.5.

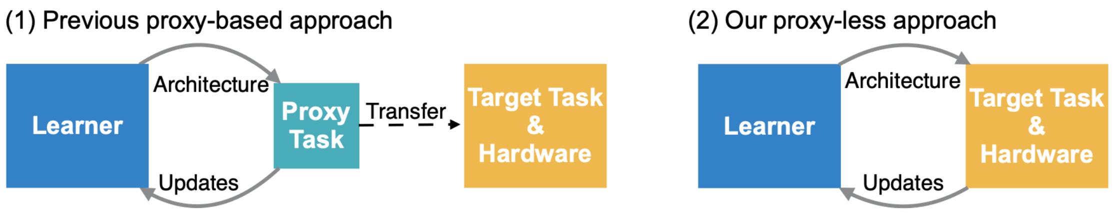


Figure 2.5: Proxy NAS and ProxylessNAS.

This NAS obtained better results than previous proxy-based approaches while

also reducing to 200 hours of GPU the computational cost for training a network (200× fewer than MnasNet for ImageNet task). Instead, it achieved the same performance as a MobileNetV2 while being 1.8× faster. The search is performed as a path-level pruning process inspired by [10] and ???[11] as reported in figure 2.6, training an over-parameterized network that contains all paths, with different layers (diff convolutional kernels, depthwise convolution, pooling, skip connections, ...), and are then pruned at the end of the training phase.
The experiments performed on CIFAR-10 and ImageNet achieved respectively 2.08% error with only 5.7M parameters and 75.01% top-1 accuracy which is 3.1% better if compared with MobileNetV2 while being 1.2×faster.

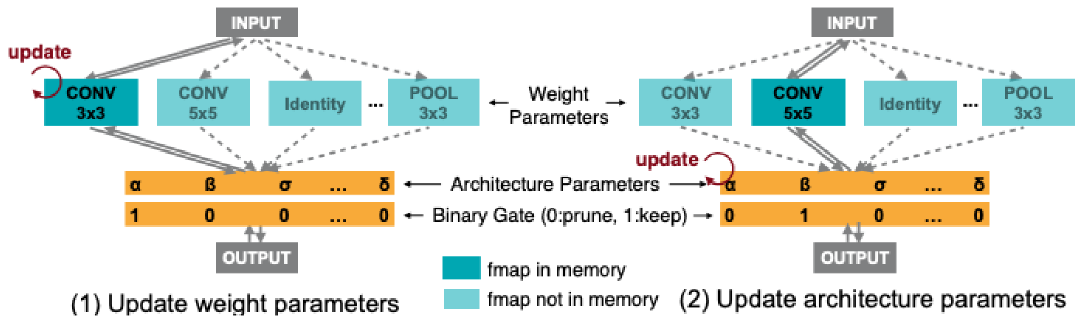


Figure 2.6: ProxylessNAS over parameterized network training.

2.1.3 MorphNet

MorphNet is a NAS that iteratively shrinks and expands a seed NN. In fact, if the NASes previously described exploring a predefined search space defined by the designer, MorphNet searches for architectures in a subspace of the NN.
The shrinkage phase happens thanks to a resource-weighted sparsifying regularizer that acts on the γ parameter of the batch normalization that consequently produces a modification in the activations. The expansions happen by means of a uniform multiplicative factor on all layers.
For embedded tasks as well as for industrial applications, power consumption and inference speed become of crucial importance. As a consequence, the need of targeting specific resources, such as FLOPs per inference, arises. The main aim was the design of NN autonomously and that achieve comparable or improved results from the accuracy point of view while reducing specific metrics.
The solution proposed with MorphNet [12] is able to scale to large models and large datasets.
Scalability and NAS performance may be seen in the case of the JFT dataset, with

more than 350M images and 20K classes, where the NAS is able to accomplish the task with a 2.1% improvement in evaluation MAP while obtaining the same number of FLOPs per inference. The training and the research of the new architecture require slightly greater resources if compared to the single model training.
The approach proposed is extremely scalable since it usually requires only a small constant number, 2 in general, of automated trial and error attempts.
The approach uses batch normalization parameters to selectively eliminate channels from a convolutional layer. The NAS employs a regularization loss to force γ in the batch normalization reported in equation 2.7 to small values and then prune them if they are under a certain threshold T as reported in figure 2.7. The crucial limitation that this tool has is the requirement of a batch normalization layer that is not always present. Furthermore, it applies the masks to the output activations.

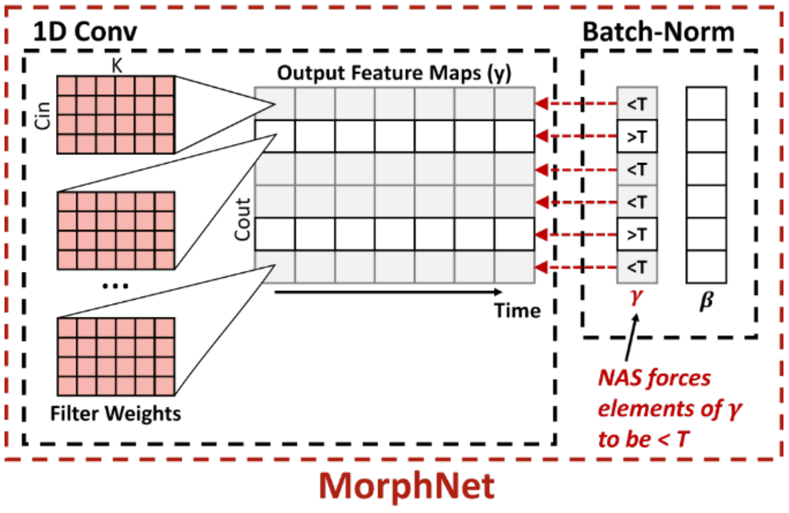


Figure 2.7: MorphNet channels pruning

The experiments performed on ImageNet achieved a 1.1% test accuracy error with respect to Inception V2 and provies 1% improvement also with respect to MobileNet targeting FLOPS as a constraint. This is worth of notice since MobileNet was already designed targetting computational resources.

2.1.4 FBNetV2

FBNetV2 is a DMaskingNAS that was designed with the specific aim of enlarging the constrained search space that commonly used DNASs have, considering that all candidate networks should be listed explicitly. As an example, in order to search for channels between 1 and 32 in a layer, all the possibilities should be listed resulting

in 32 different paths in the search space for just one layer.
The algorithm proposed by FBNetV2 [13] expands the search space by up to $10^{14}\times$ over conventional DNAS approaches supporting searches over spatial and channel dimensions that are otherwise prohibitively expensive.
The NAS obtained up to 421$\times$ less search cost while obtaining higher accuracy if compared to MobileNet V3.
In order to increase the search space, the method proposes a representation through masks that allow compact storage in memory. In particular, this method is used for channels and input resolutions. The algorithm jointly optimizes over multiple input resolutions and channel options simultaneously, increasing memory cost only negligibly as the number of options grows.

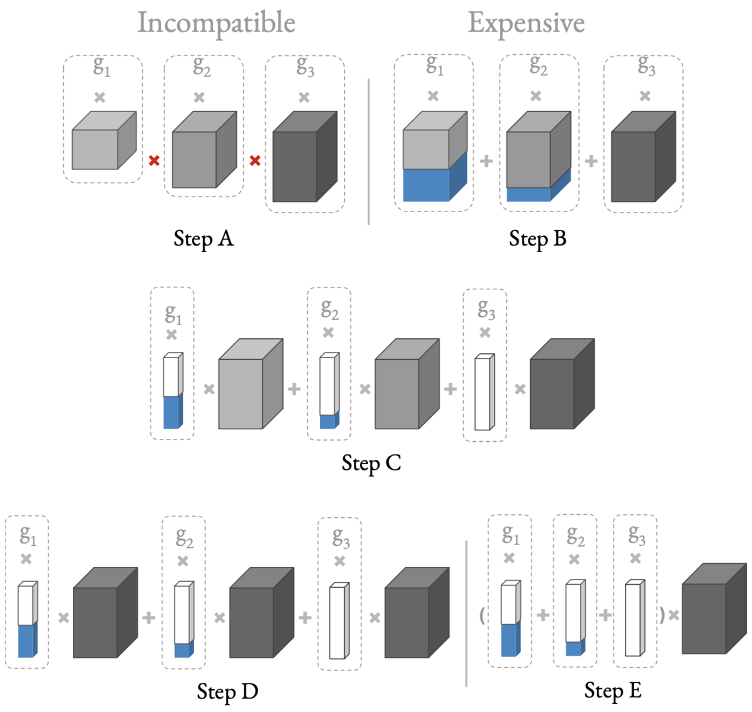


Figure 2.8: FBNetV2 channels pruning.

Previously developed DNAS approaches instantiate a block for every channel option in the supergraph.
These approaches present two main issues, the output of each block should have the same number of channels otherwise the combination is not possible and the DNAS could not perform the weighted sum. The second issue is slower training with growing options, which should be run separately thus resulting in $O(k)$ increase in

FLOPS.
In order to address the incompatibility problem, a zero padding is introduced in order to reach the maximum number of channels k as reported in figure 2.8 Step B. This is equivalent to performing an increase in the number of filters for all the convolutions to k as reported in 2.8 Step C.
Since all blocks have the same number of filters we can approximate by sharing weights as depicted in Step D of figure 2.8. Finally, this is equivalent to computing the aggregate mask and running the block b only once as reported in figure 2.8 Step E.
This allows the NAS to perform the channel search for any block including related architectural decisions.

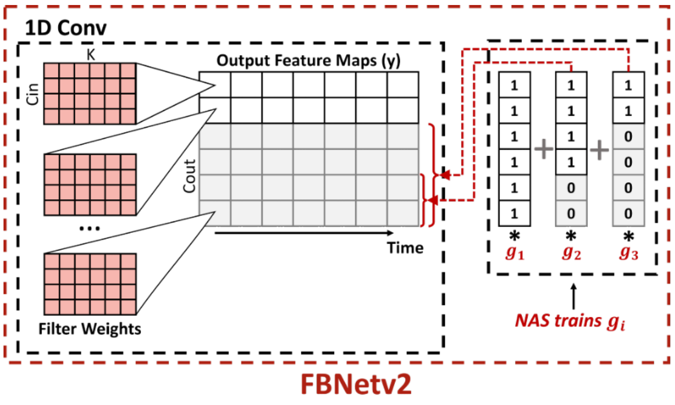


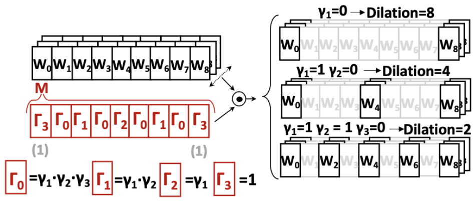
Figure 2.9: FBNetV2 channels pruning

Also in the case of FBNetV2 the pruning is done on the output activations and, as shown by figure 2.9, it requires a set of predefined masks to explore the full granularity can become noisy due to the Gumbel softmax operator [14] and generate suboptimal solutions. Furthermore, FBNetV2 is able to eliminate only a portion of the final output channels of a layer.

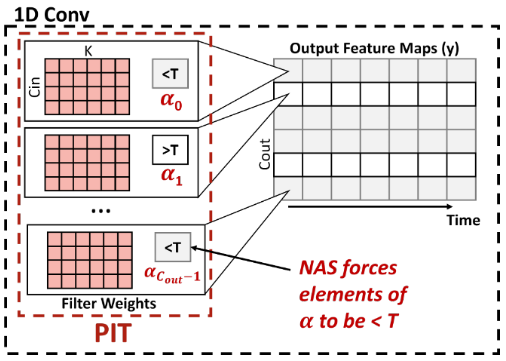
2.1.5 Pruning in Time

PIT is the first architecture optimizer that targets dilation as a hyperparameter. Also, this NAS is a DMaskingNAS, even if the application is different, targetting mainly TCNs with a focus mainly on 1D convolutions and fully connected layers since they are the most demanding in terms of OPs and Memory.
The method's first issue to assess is how to embed the dilation factors in metrics that is differentiable and thus can be optimized during training. The additional trainable parameter added by the tool is named γ.

The exploration done by PIT for what concerns the dilation factor and the receptive field allows only the use of dilations which are the power of 2 (2, 4, 8, etc.).
For each temporal convolution PIT, being rf_{max} the maximum receptive field, defines a vector of binary parameters γ containing $L = \lfloor log_2(rf_{max} - 1) \rfloor + 1$ elements.
In order to perform the search only with regular dilation patterns, the trainable parameters in γ are combined as reported in figure 2.10a and according to 2.3. Γ parameters act as a selector for the timeslice to be included or not. The mask vector has L elements and each element is then multiplied with all filter weights to perform the actual masking.


γ₁=0 → Dilation=8
γ₁=1 γ₂=0 → Dilation=4
γ₁=1 γ₂ = 1 γ₃=0 → Dilation=2
M
Γ₀=γ₁·γ₂·γ₃ Γ₁=γ₁·γ₂ Γ₂=γ₁ Γ₃=1

(a) Dilation



(b) Channels

Figure 2.10: Filtering dilation and channels.

$$\Gamma_i = \prod_{k=0}^{L-1-i} \gamma_k \tag{2.3}$$

The NAS explores the channels' search space with a novel masking approach compared to the previously described methods. The main difference is the position in which the masks are applied, in fact, PIT apply them on the weights tensor allowing a seamless implementation of the research of dilation and the receptive field.
The searches done with the tool may provide a completly customized channels structure allowing the presence or the elimination of any channel in any layer of the network and thus achieving maximum flexibility. In order to perform the selection, each element of the output tensor is binarized and then multiplied with one convolutional filter.
PIT then performs a standard training, as reported in figure 2.11, where the loss function is augmented with a L1 regularization term to promote sparsity of γ.

In this case, the training loss is the sum of a loss given by the accuracy of the actual prediction and a term described in 2.4 that accounts for the dimension of the network found. $C_{in}^{(l)}$ and $C_{out}^{(l)}$ are the input and output channel dimension for the layer l. The equation reported in 2.5 reports the full training loss.

$$\mathcal{L}_R^{size}(\gamma) = \lambda \sum_{l=1}^{layers} C_{in}^{(l)} \cdot C_{out}^{(l)} \sum_{i=1}^{L-1} \textbf{round}(\frac{rf_{max} - 1}{2^{L-i}})|\hat{\gamma}_i^{(l)}| \tag{2.4}$$

$$\mathcal{L}_{PIT}(\mathbf{W},\gamma) = \mathcal{L}^{perf}(\mathbf{W}) + \mathcal{L}_R^{size}(\gamma) \tag{2.5}$$

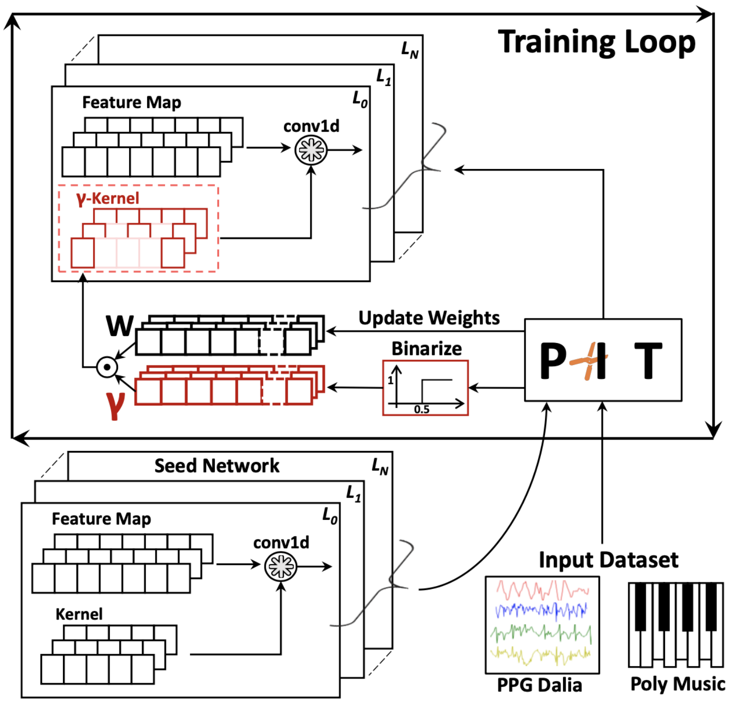


Figure 2.11: Training PIT

Even if Pruning in Time (PIT) was originally designed in order to perform architecture search on TCNs, in this work it was adapted in order to assess exploration of the architecture space for 2D convolutional networks employed for image processing tasks.
Although the NAS is able to perform searches on all the parameters' space the focus was clustered on channels exploration as reported in figure 2.10b.
In the experiments reported, PIT was able to reduce the training time of more

than 10.4× in the case of TEMPONet medium size resulting in training only 1.4× slower than without the NAS research.
With TCNs based networks, PIT is able to produce Pareto optimal solutions and reaches a reduction in the number of parameters up to 54% without affecting performance. Furthermore, it reduced the inference latency (on the GAP8) and the model size up to 3× and 7.4× respectively.
The tool here analyzed may be integrated with other Network Architecture Search based on DMaskingNAS techniques that affect different hyperparameters.

2.1.6 Main takeaways

The NAS used for this thesis is PIT since it explores the widest solution space but still maintains a reduced computational cost and requires only a fraction of the total computations needed by a traditional, not masked NAS. Furthermore, PIT does not need a Batch normalization layer to perform the optimization and thus is far more general and has further levels of application.
PIT maintains the possibility to prune any channel within the layer and it masks the weights and not the feature maps as it is for MorphNet and FBNetV2.

2.2 Networks building blocks

In this section are described the building blocks of the architectures that are analyzed in section 3.1.3 and 3.1.3.

2.2.1 Convolution

The convolution layer represents the main portion in the computational load of a CNN.
This layer performs a product between a sliding matrix and another matrix. The sliding matrix, namely the kernel, may have more than one channel in order to extract different features in one step. In order to perform inference, the kernel slides all over the fixed matrix, an image in the case of this thesis providing an activation map. The steps between one position and the next one of the convolutions is called stride.
The output width dimension, in the case of convolutions, is reported in equation 2.6. The complete output dimension for squared input and kernel is $W_{out} \cdot W_{out} \cdot D_{out}$ where D_{out} is the number of channels.

$$W_{out} = \frac{W - F + 2P}{S} + 1 \tag{2.6}$$

where

W is the width and height of the input matrix assuming it is squared
F is the size of the kernel
P is the padding
S is the stride

The fundamental reasons behind the use of convolutions are sparse interaction, parameter sharing, and equivariant representation.
Sparse interaction allows CNN to be statistically effective in storing fewer parameters and as such reducing also memory requirement. This is obtained using kernels that are smaller than the input.
Multilayer Perceptron MLP, networks realized stacking fully connected layers, use the weights only once, instead CNNs reuse the same parameters and, weights applied to inputs are always the same and so, in this case, parameters are shared. Since the weights are shared changing the input means consequently modifying the output resulting in a property named equivariance to translation.

2.2.2 Batch normalization

Batch normalization takes a step towards reducing internal covariance shift, and in doing so dramatically accelerates the training of the deep neural networks [15]. This layer is able to do so thanks to a normalization shift that fixes the means and the variances of a layer's input. Doing this, also the Stochastic Gradient Descent (SGD), the algorithm used to optimize the weights of the network, has significant advantages since the gradient flow is less likely to saturate and less dependent by the scale of values involved hence, the value of the learning rate may be significantly higher.
For each activation x_i a shift and scale parameters are obtained and so, the activation is transformed according to 2.7 over a mini-batch of dimension m applying Batch Normalization (BN).
ϵ is added to equation 2.7 in the computation of $\hat{x_i}$ for numerical stability.

$$y_i = \gamma \hat{x}_i + \beta \tag{2.7}$$

where

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$
$$\sigma_B^2 = \frac{1}{m}\sum(x_i - \mu_B)^2$$
$$\mu_b = \frac{1}{m}\sum x_i$$

2.2.3 Depthwise separable convolution

Depthwise separable convolutions may be factorized in two parts: a depthwise convolution and a pointwise convolution. The depthwise convolution has the effect of filtering the input data and the pointwise convolution act as a combining layer allowing the extraction of new features. The effect is an extremely reduced computational footprint and model size with respect to standard convolutions. In particular, standard convolutions have a computational cost proportional to $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$, where M represents the number of input channels, N the number of output channels, $D_K \cdot D_K$ is the kernel size and $D_F \cdot D_F$ is the feature map size.
Depthwise convolutions instead have a computational cost of $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$ but it performs only the filtering operation, so in order to perform also the combining operation a further $M \cdot N \cdot D_F \cdot D_F$ need to be employed for pointwise convolutions. The total cost for depthwise separable convolutions is $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$.
By comparing the cost obtained for depthwise separable convolutions with the cost of standard convolutions a reduction in cost proportional to $\frac{1}{N} + \frac{1}{D_K^2}$ may be observed, and in particular for the case of MobileNet v1 with 3 x 3 conv the reduction in computational cost is around 8 times.

MobileNet v1 has two parameters that can be tuned by the designer in order to reduce the size and the computational cost of the networks depending on the application.
The first one, namely the width multiplier α is devoted to reducing the thickness of the model uniformly at each layer. The second one is the resolution multiplier, represented as ρ, and is set to change the size of the input image. The complete computational cost is $D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$. By taking into account the two parameters α and ρ the complete cost is proportional to α^2 and ρ^2.

2.2.4 Pooling blocks

In the standard structure, three further blocks may be observed: an average pooling layer, a fully connected layer, and a softmax.
Average pooling blocks are used in order to reduce the features considered in subsequent layers. This layer takes as input a feature map (in the case of MobileNet 7 x 7) and provides as output an average of the various cells. In figure 2.12 an example of average pooling is reported.

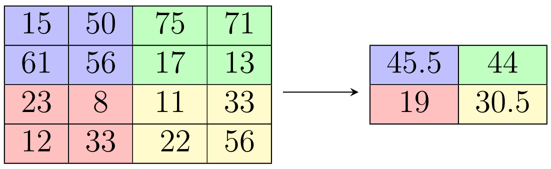


Figure 2.12: Average pooling example, 2 x 2 filter case

On the other hand, max pooling, extracts the maximum value in each window of the filter. An example of this type of pooling is reported in figure 2.13.

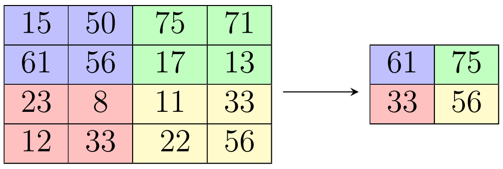


Figure 2.13: Max pooling example, 2 x 2 filter case

Both, max and average pooling are used in order to reduce the number of features and simplify the representation by providing a reduced size output according to equation 2.8.

$$W_{out} = \frac{W - F}{S} + 1 \tag{2.8}$$

where

> W is the width and height of the input matrix assuming it is squared
> F is the size of the kernel
> S is the stride

2.2.5 Fully connected layer

Fully connected layers are composed of a linear transformation and a nonlinear transformation. The linear transformation is applied to the input vector via a dot product between the vector and a matrix of weights according to 2.9.

$$y_{jk}(x) = \sigma(\sum_{i=1}^{n_H} w_{jk}x_i + w_{j0}) \tag{2.9}$$

where

> σ is the non linear function also named activation function.

2.2.6 Activation functions

All the layers previously reported are exploited at first with a forward pass in order to perform inference and, then, with a back propagation step in order to modify the weights and perform the learning activity of networks.
Given $f(\theta, x)$ a function that describes a fully connected network where x is the input and θ are the learnable weights. Then the backpropagation algorithm computes $\frac{\partial f}{\partial \theta}$.
The function σ applied to the value obtained with the linear combination is called an activation function and, in particular, is a nonlinear function. Even though many activation functions exist, two common examples are the Sigmoid and the Rectified Linear Unit (ReLU).
The function for the Sigmoidal function is reported in 2.10

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

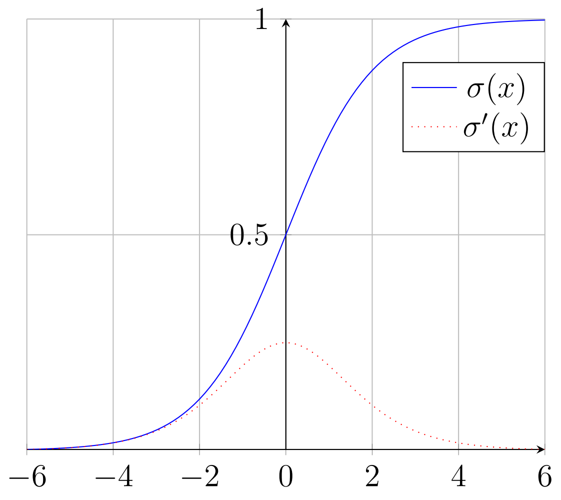


Figure 2.14: Sigmoidal function and its derivative

From figure 2.14 some peculiar characteristic of the Sigmoidal functions may be observed, in particular, the function θ is prone to saturation outside the [-4, 4] range and, as so, its derivative is equal to the derivative of a constant function (almost 0) outside that range. This can generate problems when the network has to learn with backpropagation. The presence of a derivative equal to 0 results in the vanishing gradient problem. The network in this case has no ability to learn. In order to solve this issue, many other activation functions have been proposed, an example is the ReLU and its variant, the leaky ReLU.
The ReLU function is reported in equation 2.11.

$$relu(x) = max(0, x). \tag{2.11}$$

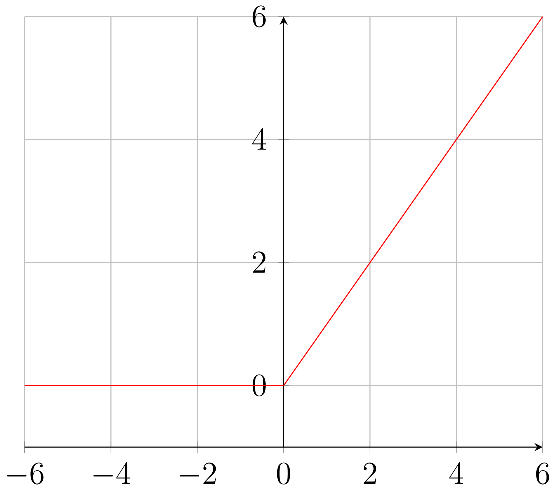


Figure 2.15: ReLU function

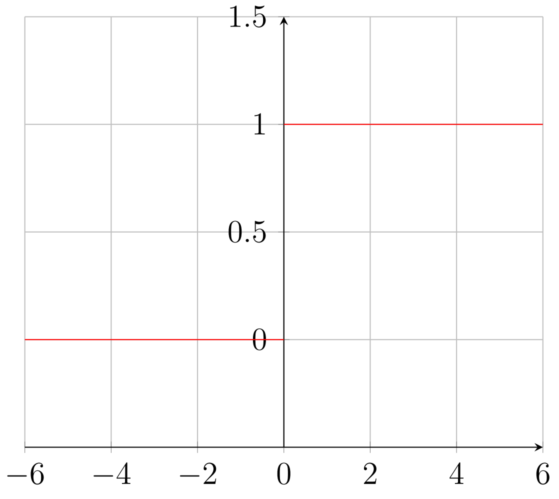


Figure 2.16: ReLU function derivative

The ReLU function may create some issues whenever $x < 0$ resulting in the dead neuron phenomenon. In order to solve the problem reported before the leaky ReLU has been proposed. In most of its part, the leaky ReLU is equal to the ReLU but, in the case of $x < 0$ it allows a slight upgrade of the value of the weight.

$$relu(x) = max(0.01x, x). \tag{2.12}$$

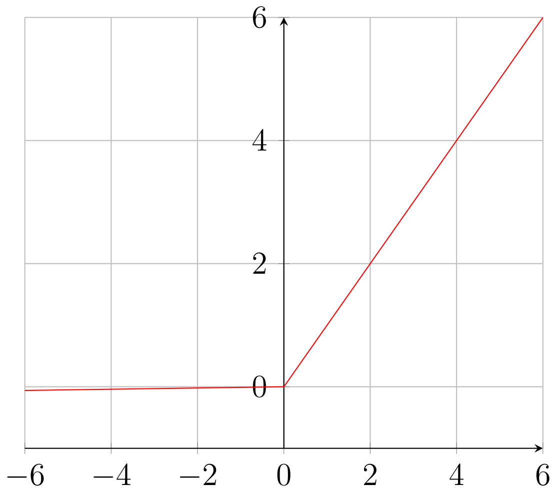


Figure 2.17: Leaky ReLU function

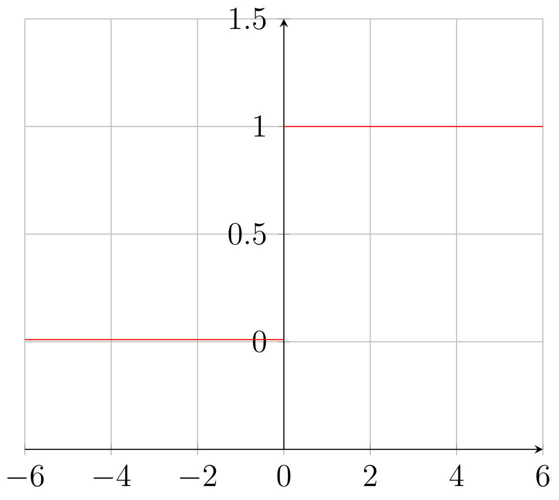


Figure 2.18: Leaky ReLU function derivative

In general, MobileNet networks may end with a softmax but, in this case, since the purpose of the network is not classification, this layer has been deleted. The last fully connected layer has been modified in order to output 4 values namely x, y, z, ϕ

2.2.7 Dropout

Dropout is a technique that solves two main issues: model combination and data scarcity. On the one hand it asses the problem of model combination, in fact creating different architectures by means of the dropping of units in the NN. On the other side skipping the irrelevant part of the network allows reducing the overfitting when the data are insufficient.
Dropping a unit means removing it from the architecture with a probability p, p is chosen using a validation set or a set by default at 0.5. Applying dropout to a neural network amounts to sampling a "thinned" network from it [16].

Chapter 3

Methodology

The main aim of this work is to prove the functionalities of PIT in reducing the seed networks in the number of parameter without affecting the accuracy.
In order to do so, three seed NNs have been explored: one based on Frontnet network and two based on MobileNet. The first has a reduced depth and it does not contain any depthwise separable convolution. The latter instead have been explored in two fashions, considering a 1.0 and a 0.25 width multiplier.
The networks have been reduced using the technique described above and tested on the drone detailed in section 3.3.1.
In section 3.2 the tools used for deployment have been detailed while, section 3.3.2 reports the software pipeline used for all the tests.
Finally, sections 3.3.3 and 3.3.4 respectively report how the training dataset has been acquired and how the experiment has been run.

3.1 NAS application to the visual pose estimation task

3.1.1 Task

Nano drones are extremely useful for indoor applications and, given their reduced size, they can interact easily and without any risk with humans. In order to allow an easier and more consistent interaction, the correct estimation of the distance between the human and the drone is of crucial interest.
In order to assess the visual pose estimation problem, Frontnet was created and tested in 2021.
The presence of grayscale images that are 2D streams of data, requires the use of 2D convolutional networks to perform the pose estimation.

3.1.2 NAS integration for 2D convolutions

Given that the NAS selected in chapter 2, namely PIT, does not perform the exploration for 2D layers in the current implementation some adaptations have been performed.
The extension proposed by this work accounts only for channel-wise exploration and allows the research of the architecture channels space in 2D convolutions. This is possible thanks to the development of a novel searchable layer that performs 2D convolutions instead of the previous one that allowed only 1D convolutions.
The searches are done with particular attention to the coherence of networks, in fact, the channel number is a tunable parameter, but it needs to be constant between one layer and the next one in the same network.

3.1.3 Seed networks and searches with NAS

Frontnet

PULP-FrontNet was introduced in 2021 with the aim of estimating the distance between a drone and a human in order to allow the drone to follow the person. It extracts 4 different real values that represent x, y, z, and ϕ estimates. The original network features 7 convolutional layers with a total of 304k parameters.
The main blocks included in this architecture, as displayed in figure 3.1, are the convolution layer, max pooling, batch normalization, RELU, Dropout, and a fully connected layer.
In order to perform the searches with NAS the parameter λ, namely the strength has been changed with values between λ =0 and $\lambda = 10^{-10}$.

MobileNet

MobileNet, designed in 2017 by Google, was conceived as a lightweight convolutional neural network capable of performing visual tasks with reduced latency.
As embedded in the name, the network is devoted to mobile applications and as a consequence is particularly suited for robotics use, in particular in the case of constrained resources environments.
A vast amount of experiments have been conducted as reported in [17]. In each of those cases, remarkable performance has been obtained, in particular, on the ImageNet task, the result has been slightly worse than VGG16 but using a 32× smaller network and 27× fewer operations.
Since our task doesn't need classification the standard network reported in table 3.1 has been modified to obtain 4 real numbers describing the relative position between the drone and the person in x, y, z, and phi.

The base structure is summarized in table 3.1.

Table 3.1: MobileNet v1 base architecture

Type / Stride	**Filter Shape**	**Input Size**
Conv / s2	3×3×3×32	224×224×3
Conv dw / s1	3×3×32 dw	112×112×32
Conv / s1	1×1×32×64	112×112×32
Conv dw / s2	3×3×64 dw	112×112×64
Conv / s1	1×1×64×128	56×56×64
Conv dw / s1	3×3×128 dw	56×56×128
Conv / s1	1×1×128×128	56×56×128
Conv dw / s2	3×3×128 dw	56×56×128
Conv / s1	1×1×128×256	28×28×128
Conv dw / s1	3×3×256 dw	28×28×256
Conv / s1	1×1×256×256	28×28×256
Conv dw / s2	3×3×256 dw	28×28×256
Conv / s1	1×1×256×512	14×14×256
5× **Conv dw / s1**	3 × 3 × 512 dw	14×14×512
5× **Conv / s1**	1×1×512×512	14×14×512
Conv dw /s2	3×3×512 dw	14×14×512
Conv / s1	1×1×512×1024	7×7×512
Conv dw /s2	3×3×1024 dw	7×7×1024
Conv / s1	1×1×1024×1024	7×7×1024
Avg Pool / s1	Pool 7 × 7	7×7×1024
FC / s1	1024 × 1000	1×1×1024
Softmax / s1	Classifier	1×1×1000

This model is based on depthwise separable convolutions, a particular type of convolutions further analyzed in section 2.2.3.

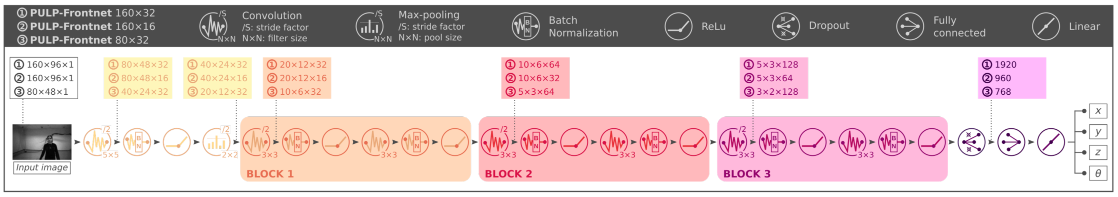


Figure 3.1: Frontnet building blocks.

For the sake of this thesis two values of α have been employed in order to perform a wider number of searches and provide a more accurate definition of NAS performance. In particular experiments with $\alpha = 1.0$ and $\alpha = 0.25$ have been reported, resulting in a couple of staring MobileNet, the first with 3 million parameters and the second one with 300 thousand parameters. With this remarkable gap in the number of parameters the first network is more prone to overfitting but gives extended flexibility to PIT exploration.
In all the cases considered the value of ρ has been kept constant, namely $\rho = 0.714$ which results in using an input image with a 160-pixel width instead of the standard 224-pixel of MobileNet. The size of the images (160x96) is the same as the other network architecture considered in this work, namely Frontnet. In these cases, the searches have been done with parameter λ ranging between 0 and 10^{-5} for MobileNet with width multiplier 1.0 while, for MobileNet with width multiplier 0.25, the parameter ranges between 0 and $\lambda = 5 \cdot 10^{-5}$.

3.2 Network implementation on SoC

The conversion of DNNs to low-level instruction, directly utilizable on the drone, is still a challenge. Considering the on-chip memory there is a size constraint since the L2 memory space is smaller than 1MB. Furthermore, the AI deck illustrated in section 3.3.1 has only an Arithmetic Logic Unit that performs integer computations, as a consequence, the networks with weights and operations have to be converted from floating point to INT8. If the conversion is not performed, there is the possibility of emulating the floating point unit but, in this case, almost 100 times more computations need to be done resulting in an extremely slow inference throughput that is practically not employable for the sake of controlling the drone. To perform the conversion and integrate the generated code into the control pipeline the UniBo flow is adopted. It consists of 3 main parts: NEural Minimization for PyTorch (NEMO), Deployment Oriented to memoRY (DORY), and integration in the Parallel Ultra Low Power (PULP) architecture.

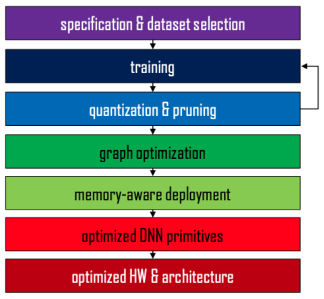


Figure 3.2: UniBo flow for DNNs deployment.

3.2.1 NEMO: NEural Minimization for pytorch

NEMO operations in the UniBo pipeline consist of the quantization of networks and the consequent generation of architectures that are employable by DORY for the generation of a code that manages the memory in a proper manner.
The software provides commands for the quantization of already trained networks and for quantization-aware training. It takes as input a PyTorch model or an onnx model and it quantizes the graph and provides the tools to reduce the precision of different layers to a different number of bits ranging from 2 to 16 bits. Although NEMO provides such flexibility, DORY requires as input an 8-bit network and so, the quantization has been done considering 8 bits weights and operations.
NEMO also provides the functions to do the entire retraining of the network after the quantization.
The tool is able to perform the linear quantization of all the tensors inside the graph, but the structure needs to respect particular characteristics. In fact, it uses batch normalization and the RELU to quantize all the graphs dividing them into multiplications additions, and shifts. A single tensor is first translated into a fixed point tensor and then to an integer tensor that is multiplied by a quantum (smallest value representable).
DORY needs an integer deployable network in order to perform the tiling and so no fixed point or floating point architectures may be used as input for this end tool. During inferences, the only operations allowed are multiplications and shifts.
The quants around the network are automatically changed during the training and, after the training of the network has been performed, NEMO is employed in order to provide the final code to be used onboard. The operations done by NEMO are split into two parts: the first is a quantization from FP32 to fake quantized (reported also as FQ), and the second one is a quantization from Fake quantized to UINT8. The tables and figures in sections 4.1.1, 4.1.2, 4.1.2, 4.1.1 and 4.1.2, 4.1.2 report the results relative to these two different steps.

3.2.2 DORY: Deployment Oriented to memoRY

DORY starting from the INT8 quantized networks performs the tiling and the code generation with primitives contained in the PULP-NN library.
The software starts from an ONNX file generated by NEMO or Quantlab for example and provides the final c-code that needs to be deployed. In order to do so, it analyzes the ONNX and provides the graphs with the basic blocks if they are contained in the backend tool. Subsequently, each node is analyzed. If a block is recognized there are two possibilities, create a new node or stick it to the previous one, the choice is performed in order to create the most efficient block on a computational cost level. Operators introduced by the software that does not have the corresponding version in hardware are not considered. This is the case

with casts.
An example of unsupported operations may be found looking at the maxpool layer, in fact, it cannot be fused in the previous convolution (in the current implementation) since in the current backend there is not a node that does the work together with convolutions.
From the ONNX a set of properties of the various nodes is extracted and stored, then inputs and outputs are read in order to link to other nodes and support branches and residual connections useful in specifical architecture, such as MobileNet v2, but not for the sake of this thesis.
As of September 2022, the tiling is done by the software layer by layer looking inside each node and understanding how much memory is necessary for the node and how to allocate it, respecting the 64kB L1 maximum size, the 512kB L2 memory constraint and the 64MB L3 memory constraint.
DORY takes from the external (L3) memory a tile and copies it to the internal (L1) one then it processes it and put it back in the external one. Weights instead are just copied in and used for the computations, they are not copied back since is not of interest and they do not change in the computations.
The tool is able to do L3/L2 tiling as well as L2/L1 tiling. The former enables the execution of big networks but slows down the execution substantially since the memory bandwidth between the external Hyperram and the internal chip memory is lower compared to the bandwidth between the two internal L2 and L1 memories. Fortunately it is not always necessary as in the cases considered here where all the networks fit in L2 memory. The latter type of tiling is always used since architectures need to be very tight to fit in L1 and it is used for this thesis since the occupation in memory of the architectures is in the order of the hundreds of kB.
As stated above the L2/L1 tiling is almost always necessary since just 64kB of L1 memory are present on the chip and the tiles need to be switched from the L2 memory of 512 kB to the 64 kB one. For this tiling since is much smaller, the 3D transfers are enabled. The tiles movement is performed solving an optimization problem, solved using the ORTools library, instead of using a greedy scheme. The equations to be optimized and the constraints to be respected are reported here in equations 3.1, 3.2, 3.3 and 3.4, optimized in order to occupy as much as possible of the L1 memory.

$$cost = maxSize(W_{tile}) + Size(x_{tile}) + Size(y_{tile}) \tag{3.1}$$

3.1: Cost function

$$Size(W_{tile}) + Size(x_{tile}) + Size(y_{tile}) < L1_{size} \tag{3.2}$$

3.2: Constraint on memory dimension

$$\{y_{tile}[ch_{out}] = W_{tile}[ch_{out}], ...\} \tag{3.3}$$

3.3: Geometry constraint

$$cost' = cost + \{y_{tile}[ch_{out}] \textbf{ divisible by } \mathbf{4}, ...\} \tag{3.4}$$

3.4: Geometry constraint

PULP-nn parallelizes its workload on the height dimension of the image so one of the heuristics to maximize is to always have the height a multiple of 8 so that the workload is balanced between cores.

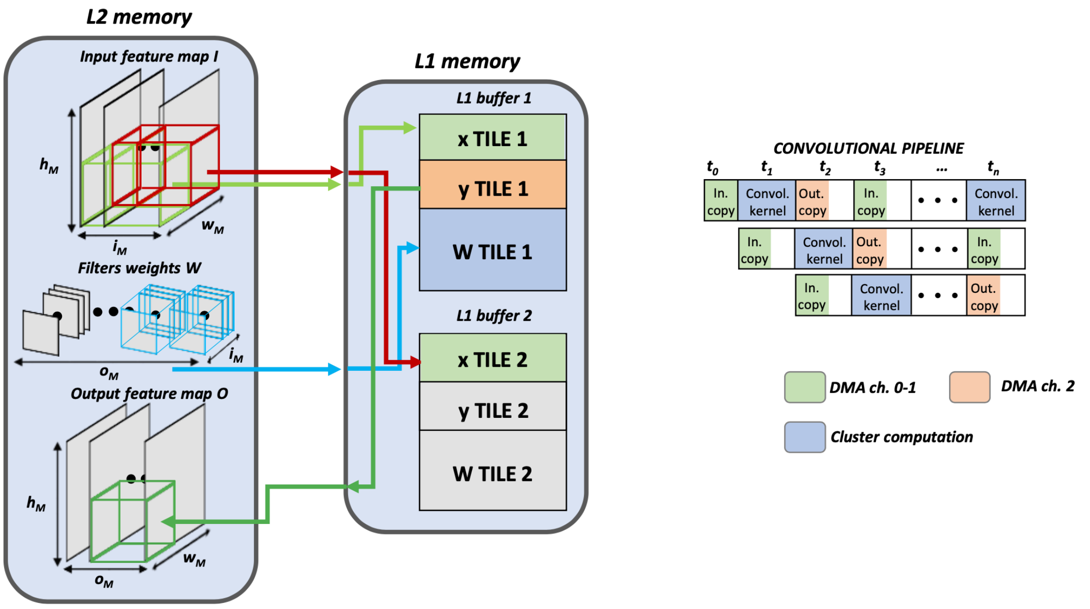


Figure 3.3: DORY data movement between L2 and L1 memory. Credits Alessio Burrello

As described in figure 3.3 at t0 we transfer input tiles and weights tiles. While the computation is performed an async call to the DMA is made in order to copy

the next tile and use the double buffering (i.e., doing in-parallel computation and data transfers through two different IPs) to hide the cost of transfers behind the computations. In fact, as described in the t1 part, the convolutions completely hide the in copy and the out copy. Out tiles are then transferred to L2 memory and in the meanwhile computations of the next tiles are performed. DORY knows all the nodes and the various memory dimensions of every block and is able to compute how to allocate the memory in the three levels. The next step consists of the compilation of a layer template, this template is filled with the parameters in the pulp graphs generated previously. The first tile performs just acquisition and then starts the actual tiles' loop that is the core of the execution where the in and out memory transfers are hidden by the convolutions.
Finally, the network generation is performed. It starts with a loop over the layers of the graph and the copy from the L3 to the L2 memory of all the weights. This is done in general for larger networks than the ones considered in this thesis, in the cases studied here all the architectures fit in L2. At this step, it is independent of the network size since DORY assumes that the first time it compiles the architecture it is stored in the flash of the chip. Also, this copy, even if it is not part of layers, is double buffered. L2 memory management then allocates and deallocates the buffers of the networks in order to minimize the usage of memory.
The main target in this case is the maximization of the data in the register file with the exploitation of the 8 cores at its maximum.

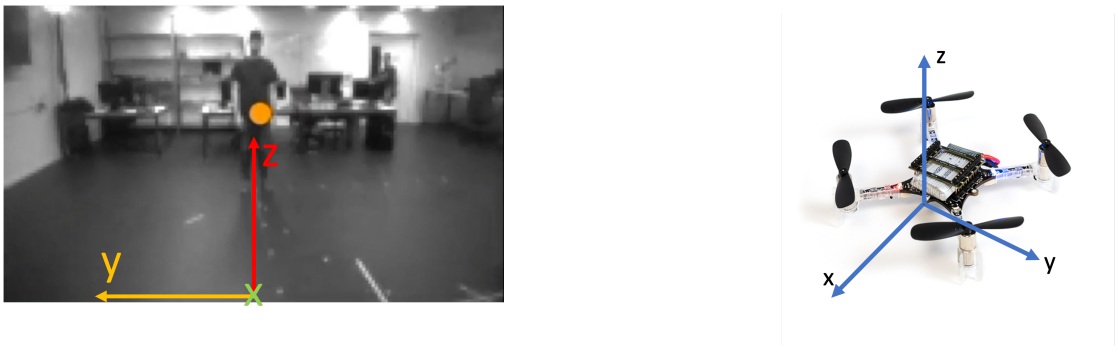
3.3 Setup

3.3.1 Drone

The Crazyflie is an open-source nano-quadrotor drone (figure 3.4) produced by bitcraze with a weight of only 27 g. The hardware is fully expandable and for the purpose of this work three main parts are used: the crazyflie drone structure that features an STM32, the flow deck, and the AI deck.

Figure 3.4: Crazyflie 2.1 nano-drone (Credits bitcraze.io)

In order to track the position of the drone, several markers have been applied to the structure in an asymmetric manner. As a consequence of the asymmetry, the optitrack system is able to solve an optimization problem and provide the accurate 3D positioning of the drone in the room and the orientation angles. Figure 3.5 is present a representation of the system of reference in the camera frame and in the drone odometry.



(a) **(b)**

Figure 3.5: Drone and camera frame with references

Crazyflie 2.1

The Crazyflie is the structure of the drone, it features four motors, a battery, an antenna for radio transmission, and several connections. On board, it is equipped with an STM32F405 with a maximum frequency of 168 MHz that takes charge of the control system computations. It has a flash memory of 1 Mb and 192 Kb SRAM. Furthermore, a BMI088 IMU and a high-precision pressure sensor are mounted onboard. Thanks to several pins a number of expansion boards can be plugged into the drone, they range from the Flow deck to the AI deck but, also proximity sensors or other kinds of sensors.

Flow deck

The flowdeck is an expansion board that includes a Time of Flight (ToF) sensor and an optical flow for visual odometry navigation.
The ToF VL53L1x integrates a receiving array as well as a 940nm invisible laser emitter that thanks to the novel technology included allows the measurement of the absolute distance whatever the target color and reflectance. It features a customizable ROI on the receiving array that reflects in the sensor's FoV reduction. It provides a stable and reliable estimation of the height from the ground.
On the other side, the PMW3901 optical flow allows the recognition of the movement in any direction as long as the distance from the ground is at least 80mm. The behavior of this sensor is similar to the ones mounted on computer's mouses.

Figure 3.6: Flowdeck v2 (Credits bitcraze.io)

AI deck

Figure 3.7: AI deck (Credits bitcraze.io)

The AI deck is an expansion board that allows the deployment of neural network models. It features a camera and a WiFi radio, and the actual heart is the GAP8 SoC.
In the case of this application, the camera employed is a grayscale Himax HM01B0 that provides images up to a 320x320 pixels resolution. Applying bounding techniques the camera is able to produce 160x160 pixels images that are then used by the networks cropped to 160x96 pixels. The Himax HM01B0 is an ultra-low power camera that only accounts for 3.9 mW in the total power budget.
The ESP32 WiFi provides WiFi connectivity and is used in this work for what concerns the streaming of the images.
Another crucial part of the AI deck is the GAP8, a SoC that provides the capabilities of running reduced-size neural networks (The biggest model tested in this thesis is in the order of 300k parameters) without using the off-chip RAM and using only the L1 and L2 memories. This SoC has two main blocks, a Fabric Controller (FC) and a Cluster (CL) which have two completely different power domains as depicted in figure 3.8. The FC has the objective to orchestrate computations and provides access to the actual CL with specific instructions that allow operations on it. The basic core unit that composes the FC and the CL is the same but, in the case of FC there is a single core architecture and, in the case of CL there are 8 cores.
GAP8 has only an Integer Unit for computation and, so arises the necessity of

quantizing the weights and biases of the networks to INT8 in order to maintain high performance and avoid the emulation of the floating point unit (FPU) that requires 100 times the computational time required by integer computations.
The idea behind the development of the SoC found its ground in the fact that allowing computation at the edge and transmit just the result, may reduce the power consumption and as such enhance the capabilities of such IoT devices. The processor is realized following the parallel ultra-low power (PULP) paradigm that fosters the low frequencies and low voltage chips. As a result, the image processing in the case of a MobileNet employed for image processing that has been analyzed by greenwaves reports only 1.5mJ@66fps. Tensions in fact are close to the threshold voltage of transistors in order to limit them as much as possible the power used. Further details may be seen in figure 3.8, such as the cluster DMA and the μDMA that provides a convenient mechanism to access memory also in the case of peripherals without providing further load to the processor.
The instructions' set has been evolved from RI5CY to RISC-V and the next model of GAP processor will include an FPU.

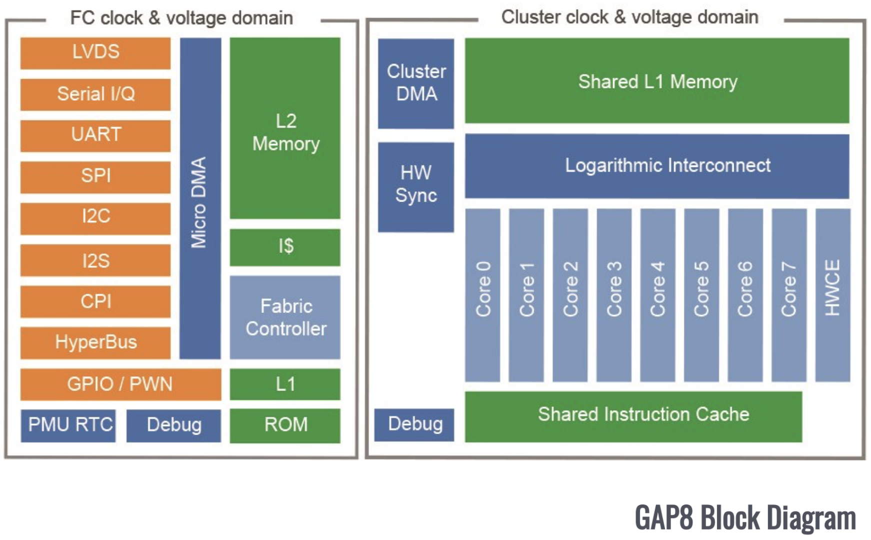


Figure 3.8: GAP8 components schema (Credits Greenwaves technology.)

3.3.2 Software pipeline

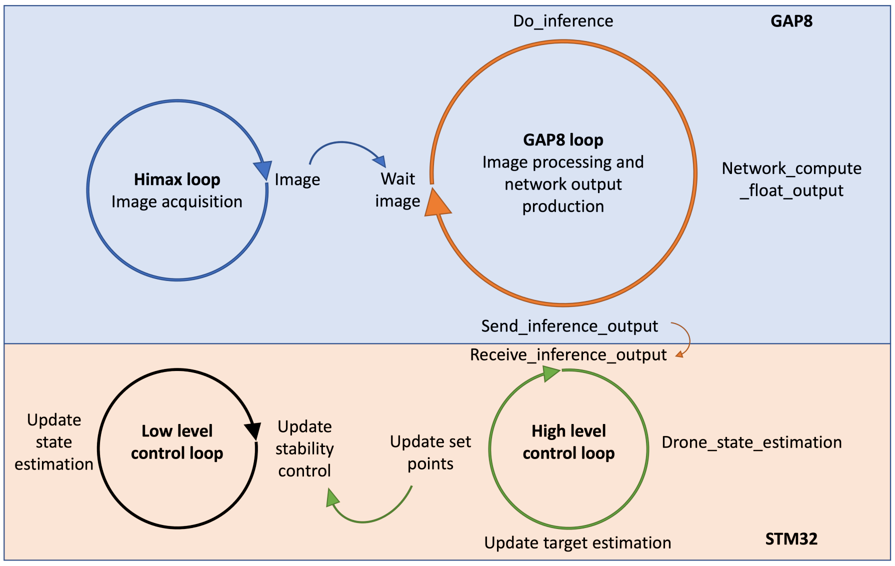


Figure 3.9: GAP8 and STM32 building blocks

The Software is split in two main parts, the first and, the one mostly interesting for this work, lives in the GAP8 while the other part resides in the STM32 as reported in figure 3.9.
For what concerns the GAP8, two loops may be observed. The first acquires images from the Himax camera and crops it to 160x96 pixels. The second instead waits for the image acquisition loop to provide a figure and then passes it to the cluster in order to process it and provide the inference output. These two cycles work simultaneously and as such, they have synchronized thanks to waiting procedures performed by the GAP8 loop.
On the other side, the STM32 waits for the inferences on the UART port, and whenever a message with the proper header arrives, it updates the target estimation and the set points. The set points are subsequently used by the low-level control loop in order to update the stability control. Finally, the state estimation is updated.
The decomposition of the entire pipeline in these logical blocks provides a fully customizable and adaptable infrastructure that allows the implementation of any type of network. It is in fact sufficient to change the inference part in the GAP8 loop in order to perform a different task or use a different NN.
As a result of the isolated loops, even the controller can be replaced easily providing access to any sort of customization and needs that may arise. As the work of this thesis concerns more the NN part, the controller has been maintained as the

standard one and used with the functions, provided by bitcraze, in order to set a target position or a target speed.

3.3.3 Environment for recordings and dataset

The dataset employed for the task of this thesis is composed of more than 66k images paired with the relative position between a person and the drone. It was developed at the "Dalle Molle Institute for Artificial Intelligence" by means of a tracking system composed by 12 Optitrack PM13 cameras, disposed in a square room of 10m x 10m, each able to locate a target with a specific marker in its field of view.
The Optitrack PM13 camera is a 1.3 MP camera able to produce frames up to 1000 FPS with a 3D accuracy of ±0.20 mm. This system is used to perform the tracking of the drone and its target identifying markers previously attached to the drone and to the person. In the case of the FrontNet dataset the images were acquired thanks to the Himax, grayscale, camera mounted onboard the AI deck of the drone and streamed via WiFi to a base station. Some samples of figures recorded by the Himax camera are displayed in figure 3.10.
The images are then paired with the data of the relative position to form the dataset used.

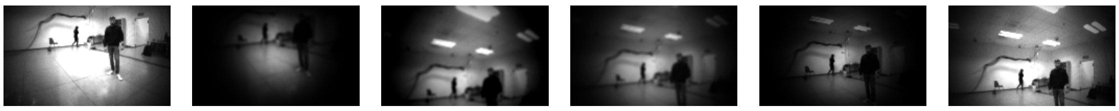


Figure 3.10: Sample image from the dataset

The dataset is composed of 6557 samples, acquired in roughly 25 minutes of recording at a 4.5 fps rate. For the sake of data variety, 10 different subjects were recorded with and without face masks and sunglasses. Each person was recorded for 150 s. From these ten recordings, six are used for training purposes and four for validation, this division ensures the correctness of the tests avoiding all overfitting problems that may occur considering the same person in the training and validation phase. A further 20% of the training set is separated and is used just to validate the performance while training and to apply the early stopping.
Each image is paired with a four values tuple, namely the coordinate x, y, and z in space and θ, the angle depicted in figure 3.11. In order to increase the plethora of data, several data augmentation techniques have been applied. In particular, for what concerns the pitch the augmentation consists in cropping the initial 160 x 160 pixels frame in more frame 160 x 96 pixels frames removing the upper and lower parts of the image, and simulating the pitch movement of the drone. A comparison of real pitch and synthetic pitch augmentation may be observed in figure 3.12. In

the figure below, the first two images on the left were recorded with physical pitch and cropped from a 160 x 160 pixels image removing the first 32 rows and the last 32. The two frames on the right instead were cropped from 160x160 removing the lower 64 rows of each image, simulating a pitch of 14°. Some differences may be perceived in the perspective but considering the resolution of the Himax camera as well as, the average distance between the drone and the subject, the effect was considered irrelevant. In order to further increase the strength of networks, augmentations on a photometric and optical basis have been performed introducing: contrast, brightness, gamma correction, synthetic vignetting, and blurring.
The set of images was increased by performing a horizontal flip, namely negating y and θ variables. The result is a dataset with more than 66000 training instances that has already proved sufficient for the case of FrontNet discussed in chapter 3.1.3.

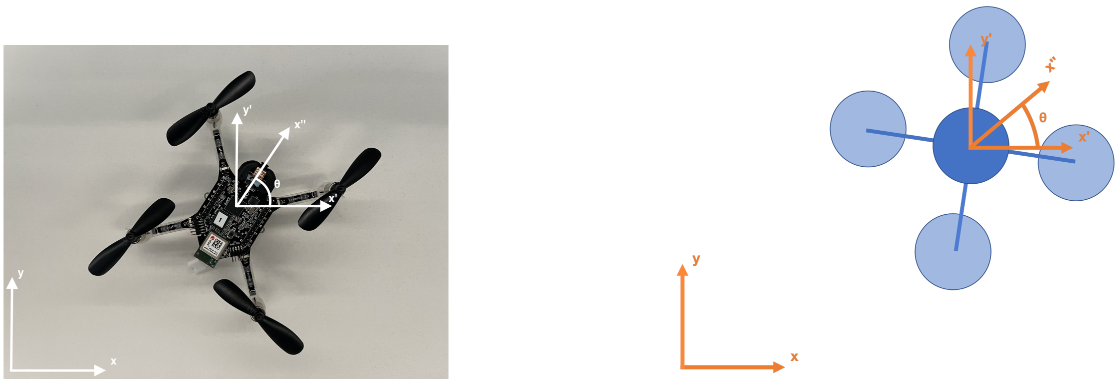


Figure 3.11: Definition of angle θ on the crazyflie

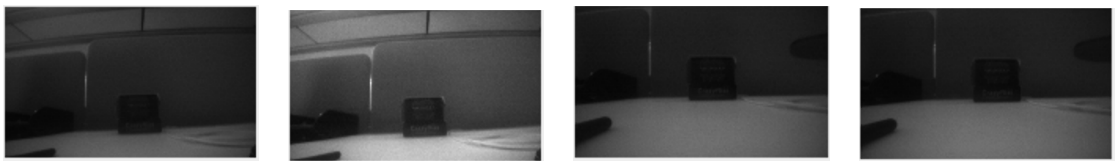


Figure 3.12: Physical vs synthetic pitch

All this dataset was acquired in the laboratory of Manno (CH), but for the tests, a new environment was used, the Lugano (CH) labs. As a consequence, novel conditions of lights, backgrounds, and disturbances are present.

3.3.4 Testing setup

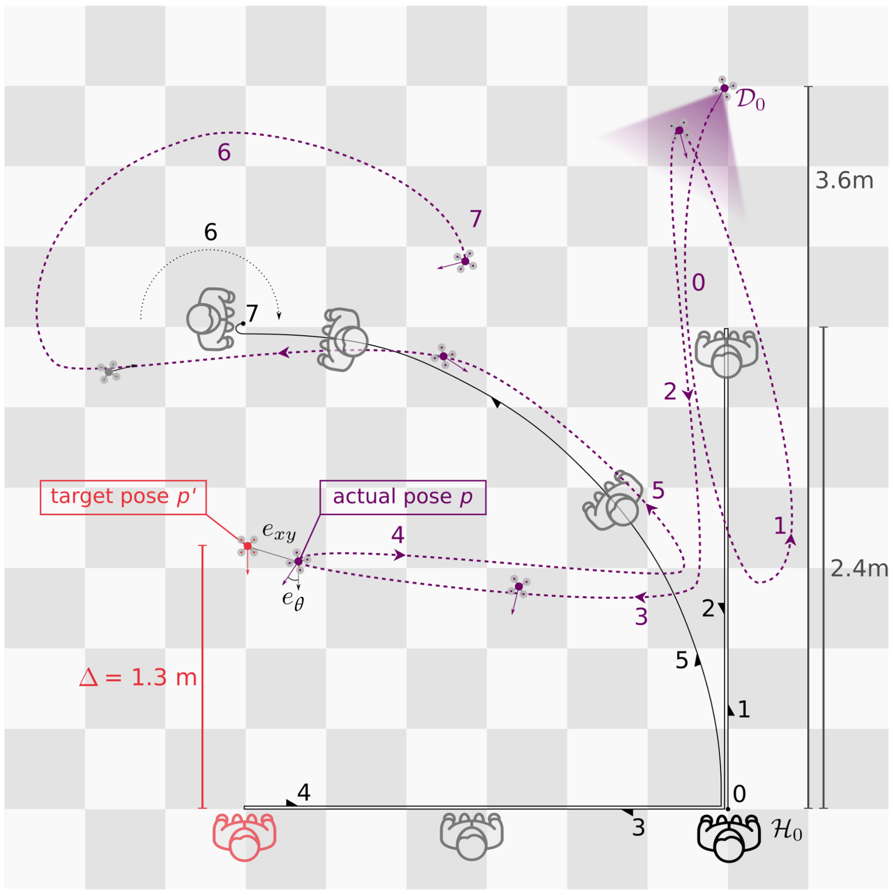


Figure 3.13: Path used during the experiments.

In order to perform reliable and reproducible experiments, a path has been designed. Since the one designed for Frontnet was complete and challenges several use cases, it was adapted for the sake of this thesis. The experiment starts with the drone in position D_0 at 30° of rotation and the person begins standing in H_0. In the beginning, the drone is able to see the subject in its field of view. The inference is started and the subject stands for 5s in position H_0 in order to let the drone position at the target distance (1.3 m in this case) facing the subject. Then the person performs a walk in the front direction composed of 6 steps, each of 1 second, to cover a total distance of 2.4 m. Subsequently, the individual walks back to position H_0. After this movement immediately begins a sidewalk on the left side of the person, even this time composed of 6 steps performed left side and right side back to position H_0. Also in this case the distance is 2.4 m and the steps are 6 for every direction, they are performed each in 1 second. In order to further challenge

the phi estimation, an arc of 90° is performed moving on a circumference with the center in the position previously reached on the left side (2.4 from H_0). The movement is performed rigidly in order to let the drone perform a translation and a rotation at the same time. Finally, after the person has reached the final point she rotates in place 180° and stands still for 5s in order to let the drone complete the maneuver.
The experimental setup is realized in a new environment never seen from the networks in order to verify how they generalize. In particular, the test field is reported in figure 3.13. The background in the test environment is dived into two parts, the first has several difficulties with various monitors and shelves in the back. The second on the other hand has a white background that is closer to the training one.

Chapter 4

Results

This chapter discusses the research with the NAS in section 4.1, section 4.2 presents the deployment on the GAPuino SoC with an analysis of the power consumption of each network and section 4.3 reports the infield results with the control task performed onboard the drone.

4.1 Models' search with PIT

This section reports the searches done with PIT in order to create the Pareto front for the selection of the networks to deploy infield. Section 4.1.1 presents the networks obtained using Frontnet as seed network. Section 4.1.2 presents the analysis on the two seed networks considered for MobileNet with changing width multiplier, respectively $\alpha = 0.25$ and $\alpha = 1.0$. Each section present also the analysis of the quantization performed on the networks.

4.1.1 Frontnet

The search space of Frontnet has been explored using PIT, explained in more detail in section 2.1.5 in the extended version useful for this work. Various values of the strength parameter λ have been used. In particular, starting from the seed network (equivalent to $\lambda = 0$) the exploration has gone through increasing strength values arriving at the maximum value of $\lambda = 1 \cdot 10^{-10}$. Tables 4.1 and 4.2 report the results in MSE, MAE and R2 for x, y, z and ϕ.

Table 4.1: Models with changing strength, x and y performance

Strength	**x**			**y**		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	**0.0621**	**0.1870**	**0.8035**	**0.0805**	**0.1837**	**0.6532**
$1 \cdot 10^{-20}$	0.0690	0.1987	0.7815	0.0941	0.1960	0.5950
$1 \cdot 10^{-15}$	0.0580	0.1853	0.8164	0.0896	0.1929	0.6141
$1 \cdot 10^{-12}$	0.0639	0.1976	0.7977	0.0727	0.1753	0.6870
$1 \cdot 10^{-11}$	0.0663	0.1968	0.7889	0.0882	0.1787	0.6202
$1.5 \cdot 10^{-11}$	0.0748	0.2090	0.7630	0.0959	0.2084	0.5871
$2 \cdot 10^{-11}$	0.0704	0.2049	0.7772	0.0911	0.1901	0.6078
$2.5 \cdot 10^{-11}$	**0.0668**	**0.1957**	**0.7885**	**0.0815**	**0.1943**	**0.6489**
$5 \cdot 10^{-11}$	0.0703	0.2043	0.7773	0.1035	0.2218	0.5545
$1 \cdot 10^{-10}$	0.1398	0.2779	0.5571	0.2999	0.3737	-0.2915

Table 4.2: Models with changing strength, z and ϕ performance

Strength	**z**			ϕ		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	**0.0166**	**0.0927**	**0.5481**	**0.3244**	**0.4367**	**0.2585**
$1 \cdot 10^{-20}$	0.0185	0.0931	0.4950	0.3144	0.4403	0.2813
$1 \cdot 10^{-15}$	0.0183	0.0974	0.5004	0.3543	0.4578	0.1900
$1 \cdot 10^{-12}$	0.0180	0.0966	0.5095	0.3591	0.4629	0.1791
$1 \cdot 10^{-11}$	0.0192	0.0959	0.4775	0.3212	0.4374	0.2657
$1.5 \cdot 10^{-11}$	0.0201	0.0988	0.4509	0.3266	0.4486	0.2534
$2 \cdot 10^{-11}$	0.0216	0.1005	0.4095	0.3604	0.4603	0.1763
$2.5 \cdot 10^{-11}$	**0.0179**	**0.0893**	**0.5128**	**0.3315**	**0.4437**	**0.2423**
$5 \cdot 10^{-11}$	0.0374	0.1360	-0.0210	0.4398	0.5233	-0.0054
$1 \cdot 10^{-10}$	0.0370	0.1263	-0.0103	0.4312	0.5355	0.0142

Quantization of FrontNet models: Fake quantized (FQ)

The results of the quantization with NEMO of FrontNet's models are reported in table 4.4 and further expanded in the graphs 4.1, 4.2 and 4.3. In this case, no downgrade of performance between FP32 and FQ can be observed for any of the networks. Also comparing the quantized networks with the plain FrontNet FP32

the results are satisfactory on the test set.
The use of the same data type as the original network provides a solid base for a comparison with the results reported in 'Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs'. Even more, using such data type allow the system to perform at the fastest rate possible, avoiding Floating point emulation which usually results in a slowdown in the order of 100x, unacceptable in the case of the power-constrained environment of the Crazyflie 2.1 nano drone in exam.

Table 4.3: Non quantized vs quantized FQ FrontNet models

Name	$\frac{1}{7}$ **FrontNet FP32**	$\frac{1}{7}$ **FrontNet UINT8 FQ**	$\frac{1}{3}$ **FrontNet FP32**	$\frac{1}{3}$ **FrontNet UINT8 FQ**	**FrontNet FP32**
Strength	**2.5E-11**	**2.5E-11**	**1E-11**	**1E-11**	**0**
FLOPS	$7.6 \cdot 10^6$	$7.6 \cdot 10^6$	$1.03 \cdot 10^7$	$1.03 \cdot 10^7$	$1.47 \cdot 10^7$
Par size [MB]	0.17	0.17	0.37	0.37	0
# par	44545	44545	95860	95860	304356
MSE x	0.0668	0.0721	0.0633	0.0637	0.0621
MSE y	0.0815	0.0776	0.0704	0.0685	0.0805
MSE z	0.0179	0.0178	0.0170	0.0172	0.0166
MSE phi	0.3315	0.3160	0.3565	0.3387	0.3244
MAE x	0.1957	0.2038	0.1970	0.1982	0.187
MAE y	0.1943	0.1867	0.1793	0.1740	0.1837
MAE z	0.0893	0.0905	0.0931	0.0931	0.0927
MAE phi	0.4437	0.4327	0.4598	0.4478	0.4367
R2 x	0.7885	0.7716	0.7995	0.7983	0.8035
R2 y	0.6489	0.6659	0.6969	0.7052	0.6532
R2 z	0.5128	0.5135	0.5352	0.5300	0.5481
R2 phi	0.2423	0.2776	0.1852	0.2257	0.2585

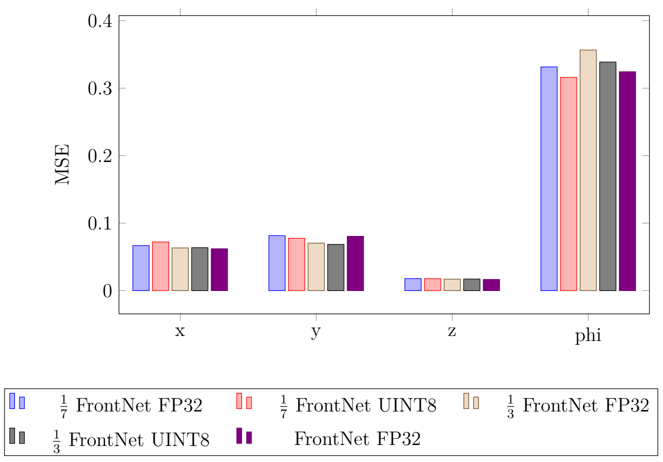


Figure 4.1: MSE of non-quantized vs quantized FQ FrontNet models

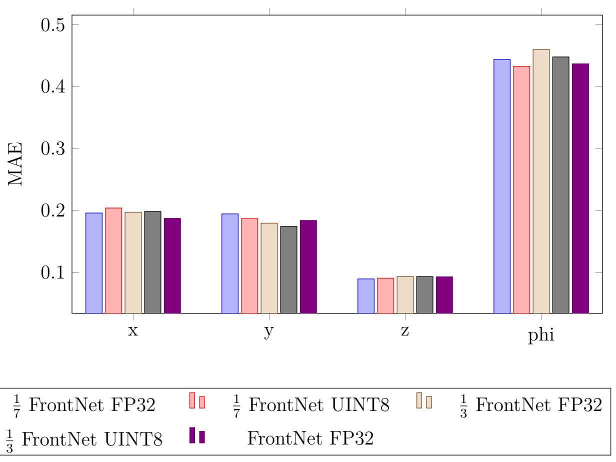
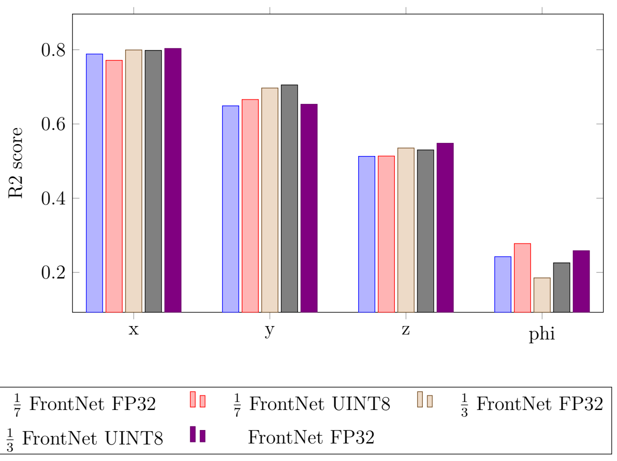


Figure 4.2: MAE of non-quantized vs quantized FQ FrontNet models



H

Figure 4.3: R2 score of non-quantized vs quantized FQ FrontNet models

Quantization of FrontNet models: Integer deployable

The next step in the quantization consists in the switch from fake quantized networks to integer deployable ones. In this case, there is no loss of performance in any of the cases reported below. Details may be found in table 4.4 and graphs 4.4, 4.5 and **??**.

Table 4.4: Non quantized vs quantized ID FrontNet models

Name	$\frac{1}{7}$ **FrontNet FP32**	$\frac{1}{7}$ **FrontNet UINT8**	$\frac{1}{3}$ **FrontNet FP32**	$\frac{1}{3}$ **FrontNet UINT8**	**FrontNet FP32**
Strength	**2.5E-11**	**2.5E-11**	**1E-11**	**1E-11**	**0**
FLOPS	$7.6 \cdot 10^6$	$7.6 \cdot 10^6$	$1.03 \cdot 10^7$	$1.03 \cdot 10^7$	$1.47 \cdot 10^7$
Par size [MB]	0.17	0.17	0.37	0.37	0
# par	44545	44545	95860	95860	304356
MSE x	0.0668	0.0702	0.0633	0.0644	0.0621
MSE y	0.0815	0.0794	0.0704	0.0654	0.0805
MSE z	0.0179	0.0173	0.0170	0.0174	0.0166
MSE phi	0.3315	0.3275	0.3565	0.3492	0.3244
MAE x	0.1957	0.2016	0.1970	0.1999	0.187
MAE y	0.1943	0.1901	0.1793	0.1699	0.1837
MAE z	0.0893	0.0891	0.0931	0.0955	0.0927
MAE phi	0.4437	0.4407	0.4598	0.4544	0.4367
R2 x	0.7885	0.7776	0.7995	0.7962	0.8035
R2 y	0.6489	0.6683	0.6969	0.7182	0.6532
R2 z	0.5128	0.5274	0.5352	0.5246	0.5481
R2 phi	0.2423	0.2514	0.1852	0.2018	0.2585

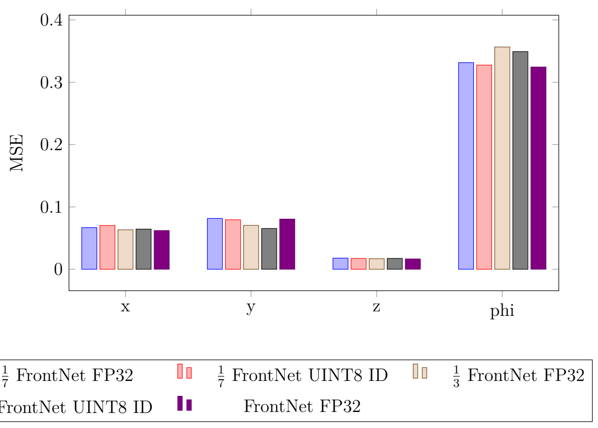


Figure 4.4: MSE of non-quantized vs quantized ID FrontNet models

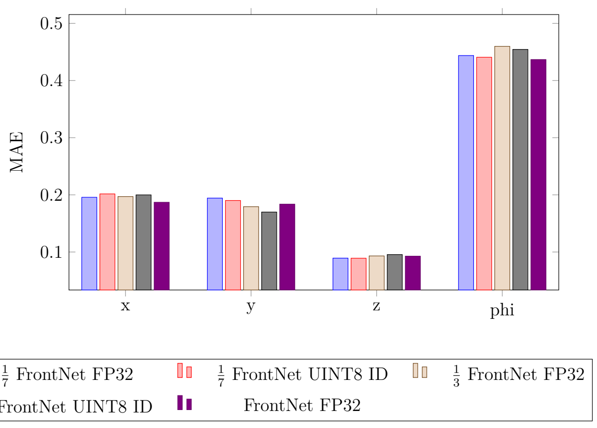


Figure 4.5: MAE of non-quantized vs quantized ID FrontNet models

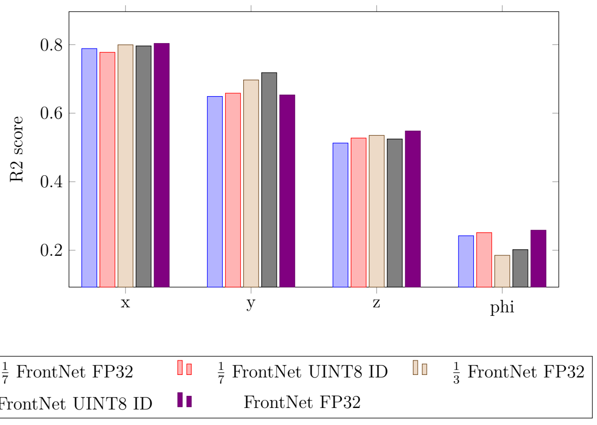


Figure 4.6: R2 score of non-quantized vs quantized ID FrontNet models

4.1.2 MobileNet

Employing PIT a plethora of models have been researched, and the objective of optimization was the number of parameters (size of the network). In order to discover various architectures the strength parameter λ has been modified. In the section 4.1.2 the experiments done with MobileNet v1 with $\alpha = 0.25$ have been reported, instead in section 4.1.2 are displayed the experiments with MobileNet v1 with $\alpha = 1.0$.

MobileNet v1 $\alpha = 0.25$

In the case of MobileNet v1 with $\alpha = 0.25$, the research with PIT has been done with 9 different values of strength (λ), in particular the values of λ employed and

the respective results are listed in table 4.5 for x and y while, table 4.6 for z and ϕ. The values of λ used have been chosen to start from $\lambda = 0$ and iteratively increased in order to reach smaller networks. Starting from a total number of parameters equal to 213956 for the original MobileNet v1 with $\alpha = 0.25$, the most compact reached has 5824 parameters.
In the following graphs, the networks with $\lambda = 1 \cdot 10^{-8}$, $\lambda = 5 \cdot 10^{-8}$,$\lambda = 1 \cdot 10^{-10}$ are referred respectively as $\frac{1}{7}$ MobileNet, $\frac{1}{5}$ MobileNet, $\frac{1}{4}$ MobileNet in order to further stress the reduction in dimension with respect to the original model.

Table 4.5: Models with changing strength, x and y performance

Strength	**x**			**y**		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	0.0459	0.1594	0.8546	0.1223	0.2125	0.4733
$5 \cdot 10^{-11}$	0.0529	0.1751	0.8326	0.0837	0.1916	0.6395
$1 \cdot 10^{-10}$	**0.0510**	**0.1709**	**0.8384**	**0.0850**	**0.2018**	**0.6340**
$1 \cdot 10^{-8}$	0.0604	0.1841	0.8086	0.0742	0.1822	0.6803
$5 \cdot 10^{-8}$	0.0373	0.1431	0.8819	0.0730	0.1917	0.6858
$1 \cdot 10^{-7}$	0.0566	0.1787	0.8209	0.0801	0.1858	0.6552
$5 \cdot 10^{-7}$	0.0404	0.1492	0.8720	0.1162	0.2007	0.4996
$5 \cdot 10^{-6}$	0.0539	0.1770	0.8294	0.0804	0.1923	0.6539
$1 \cdot 10^{-5}$	0.0459	0.1641	0.8548	0.0740	0.1904	0.6815
$5 \cdot 10^{-5}$	0.0689	0.1952	0.7817	0.0751	0.1886	0.6766

Table 4.6: Models with changing strength, z and ϕ performance

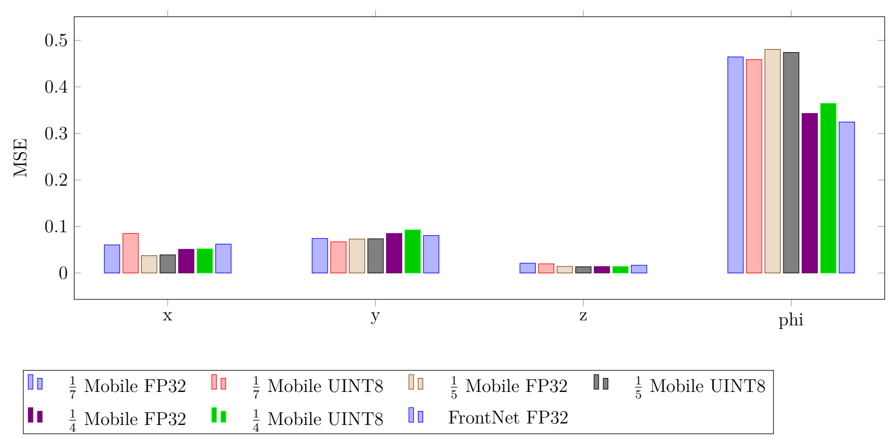
Strength	**z**			ϕ		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	0.0145	0.0838	0.6038	0.3547	0.4548	0.1892
$5 \cdot 10^{-11}$	0.0170	0.0895	0.5365	0.3788	0.4738	0.1342
$1 \cdot 10^{-10}$	**0.0139**	**0.0813**	**0.6198**	**0.3428**	**0.4401**	**0.2163**
$1 \cdot 10^{-8}$	0.0210	0.1015	0.4272	0.4643	0.5330	-0.0612
$5 \cdot 10^{-8}$	0.0140	0.0797	0.6192	0.3986	0.4835	0.0889
$1 \cdot 10^{-7}$	0.0388	0.1424	-0.0591	0.3986	0.4835	0.0889
$5 \cdot 10^{-7}$	0.0134	0.0785	0.6334	0.4944	0.5473	-0.1301
$5 \cdot 10^{-6}$	0.0391	0.1374	-0.0667	0.3898	0.4826	0.1089
$1 \cdot 10^{-5}$	0.0394	0.1388	-0.0743	0.3821	0.4721	0.1265
$5 \cdot 10^{-5}$	0.0389	0.1339	-0.0611	0.3317	0.4459	0.2418

Quantization of MobileNet v1 0.25 models: Fake quantized

Table 4.7 and figures 4.7, 4.8 and 4.9 report the results for MobileNet v1 0.25. Also, in this case, the lose in performance is sufficiently low and all the networks maintain the results of the FP32 test set.

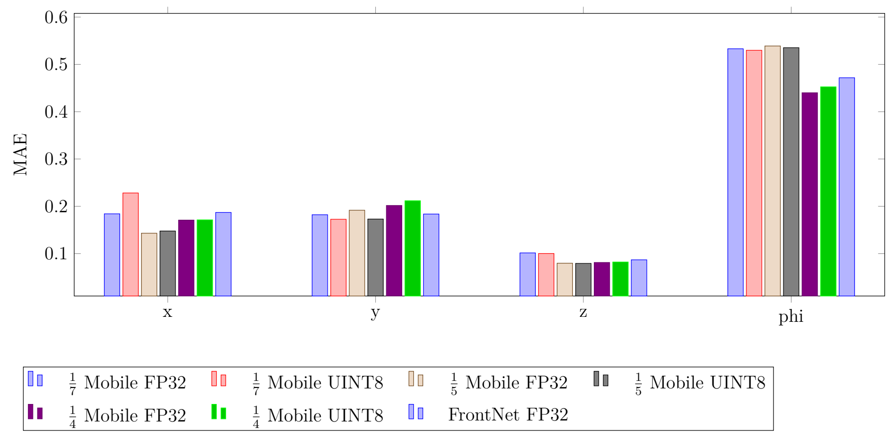
Table 4.7: Non quantized vs quantized FQ MobileNet v1 0.25 models

Name	$\frac{1}{7}$ **Mobile FP32**	$\frac{1}{7}$ **Mobile UINT8**	$\frac{1}{5}$ **Mobile FP32**	$\frac{1}{5}$ **Mobile UINT8**	$\frac{1}{4}$ **Mobile FP32**	$\frac{1}{4}$ **Mobile UINT8**	**FrontNet FP32**
Strength	**1E-8**	**1E-8**	**5E-8**	**5E-8**	**1E-10**	**1E-10**	**0**
FLOPS	$4.06 \cdot 10^6$	$4.06 \cdot 10^6$	$5.37 \cdot 10^6$	$5.37 \cdot 10^6$	$7.40 \cdot 10^6$	$7.40 \cdot 10^7$	$1.47 \cdot 10^7$
Par [MB]	0.11	0.11	0.14	0.14	0.21	0.21	1.14
# par	29281	29281	37709	37709	56302	56302	304356
MSE x	0.0604	0.0850	0.0373	0.0390	0.0510	0.0519	0.0621
MSE y	0.0742	0.0670	0.0730	0.0735	0.0850	0.0927	0.0805
MSE z	0.0210	0.0197	0.0140	0.0134	0.0139	0.0140	0.0166
MSE phi	0.4643	0.4587	0.4806	0.4739	0.3428	0.3644	0.3244
MAE x	0.1841	0.2282	0.1431	0.1478	0.1709	0.1714	0.187
MAE y	0.1822	0.1726	0.1917	0.1731	0.2018	0.2118	0.1837
MAE z	0.1015	0.1002	0.0797	0.0793	0.0813	0.0823	0.0927
MAE phi	0.5330	0.5298	0.5389	0.5353	0.4401	0.4526	0.4367
R2 x	0.8086	0.7309	0.8819	0.8765	0.8384	0.8356	0.8035
R2 y	0.6803	0.7117	0.6858	0.6835	0.6340	0.6010	0.6532
R2 z	0.4272	0.4628	0.6192	0.6351	0.6198	0.6192	0.5481
R2 phi	-0.0612	-0.0486	-0.0985	-0.0832	0.2163	0.1670	0.2585



H

Figure 4.7: MSE of non-quantized vs quantized FQ Mobilenet v1 0.25 models



H

Figure 4.8: MAE of non-quantized vs quantized FQ Mobilenet v1 0.25 models

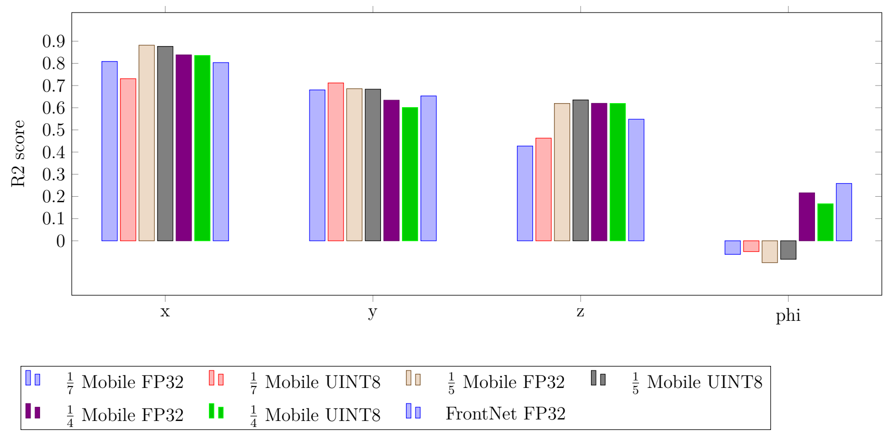


Figure 4.9: R2 score of non-quantized vs quantized FQ Mobilenet v1 0.25 models

Quantization of MobileNet v1 0.25 models: Integer deployable

Finally in the case of MobileNet v1 0.25 in the step from fake quantized to integer deployable there is no loss of performance in any of the cases reported below. Details may be found in table 4.8 and graphs 4.10, 4.11 and 4.12.

Table 4.8: MobileNet v1 models, width multiplier = 0.25, Integer Deployable performance

Name	$\frac{1}{7}$ **Mobile FP32**	$\frac{1}{7}$ **Mobile UINT8**	$\frac{1}{5}$ **Mobile FP32**	$\frac{1}{5}$ **Mobile UINT8**	$\frac{1}{4}$ **Mobile FP32**	$\frac{1}{4}$ **Mobile UINT8**	**FrontNet FP32**
Strength	**1E-8**	**1E-8**	**5E-8**	**5E-8**	**1E-10**	**1E-10**	**0**
FLOPS	$4.06 \cdot 10^6$	$4.06 \cdot 10^6$	$5.37 \cdot 10^6$	$5.37 \cdot 10^6$	$7.40 \cdot 10^6$	$7.40 \cdot 10^7$	$1.47 \cdot 10^7$
Par [MB]	0.11	0.11	0.14	0.14	0.21	0.21	1.14
# par	29281	29281	37709	37709	56302	56302	304356
MSE x	0.0604	0.0621	0.0373	0.0584	0.0510	0.0543	0.0621
MSE y	0.0742	0.0750	0.0730	0.0654	0.0850	0.0822	0.0805
MSE z	0.0210	0.0229	0.0140	0.0159	0.0139	0.0147	0.0166
MSE phi	0.4643	0.4633	0.4806	0.4759	0.3428	0.3428	0.3244
MAE x	0.1841	0.1900	0.1431	0.1838	0.1709	0.1778	0.187
MAE y	0.1822	0.1786	0.1917	0.1739	0.2018	0.1904	0.1837
MAE z	0.1015	0.1071	0.0797	0.0839	0.0813	0.0840	0.0927
MAE phi	0.5330	0.5332	0.5389	0.5369	0.4401	0.4415	0.4367
R2 x	0.8086	0.8032	0.8819	0.8152	0.8384	0.8280	0.8035
R2 y	0.6803	0.6769	0.6858	0.7184	0.6340	0.6460	0.6532
R2 z	0.4272	0.3763	0.6192	0.5661	0.6198	0.5990	0.5481
R2 phi	-0.0612	-0.0591	-0.0985	-0.0878	0.2163	0.2163	0.2585

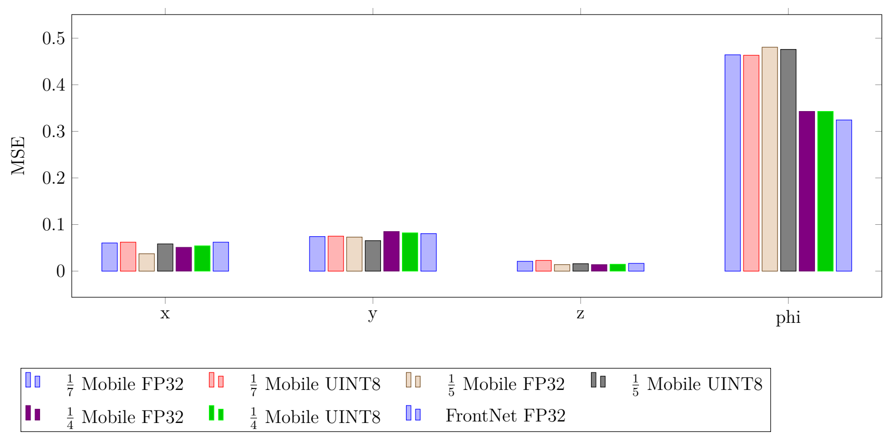


Figure 4.10: MSE of non quantized vs quantized ID Mobilenet v1 0.25 models

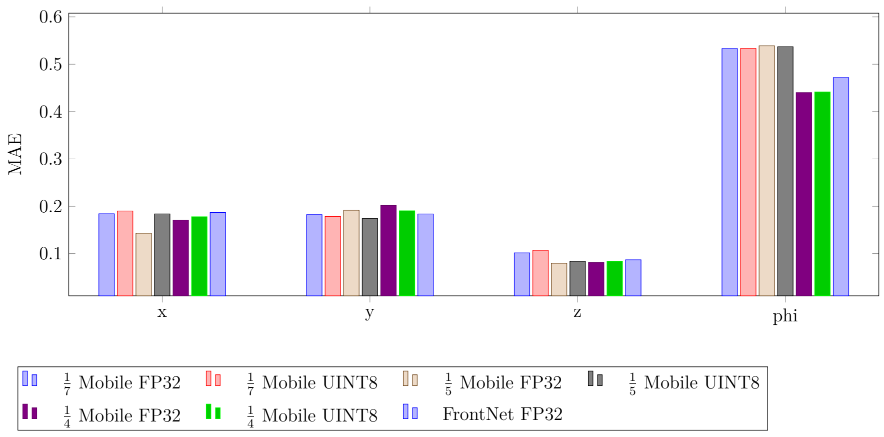


Figure 4.11: MAE of non quantized vs quantized ID Mobilenet v1 0.25 models

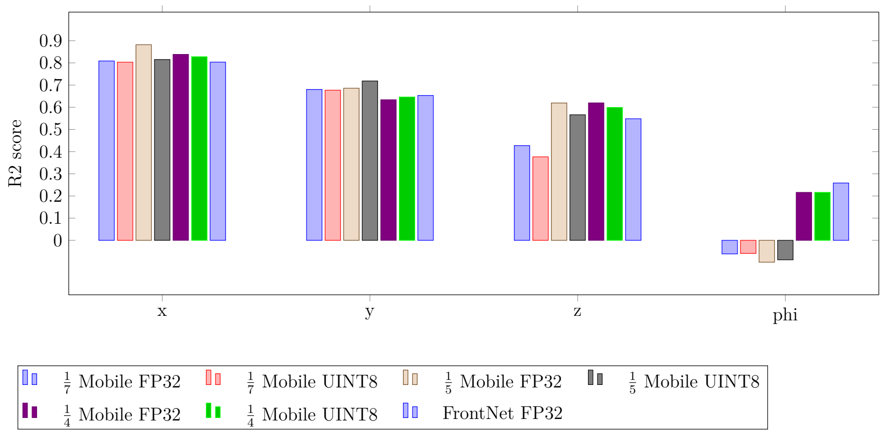


Figure 4.12: R2 score of non-quantized vs quantized ID Mobilenet v1 0.25 models

MobileNet v1 $\alpha = 1.0$

In the case of MobileNet v1 with $\alpha = 1.0$, the research with PIT has been done with 9 different values of strength (λ), in particular the values of λ employed and the respective results are listed in table 4.9 for x and y while, table 4.10 for z and ϕ. The values of λ used have been chosen to start from $\lambda = 0$ and iteratively increased in order to reach smaller networks. Starting from a total number of parameters equal to 213956 for the original MobileNet v1 with $\alpha = 0.25$, the most compact reached has 5824 parameters.
In the following graphs, the networks with $\lambda = 1 \cdot 10^{-8}$, $\lambda = 5 \cdot 10^{-8}$,$\lambda = 1 \cdot 10^{-10}$ are referred respectively as $\frac{1}{7}$ MobileNet, $\frac{1}{5}$ MobileNet, $\frac{1}{4}$ MobileNet in order to further stress the reduction in dimension with respect to the original model.

Table 4.9: Models with changing strength, x and y performance

Strength	**x**			**y**		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	0.0602	0.1818	0.8093	0.1466	0.2462	0.3686
$1 \cdot 10^{-11}$	0.0591	0.1817	0.8129	0.0887	0.1990	0.6182
$1 \cdot 10^{-10}$	0.0586	0.1786	0.8146	0.0598	0.1668	0.7425
$1 \cdot 10^{-9}$	0.0565	0.1802	0.8211	0.0705	0.1790	0.6965
$1 \cdot 10^{-8}$	0.0502	0.1692	0.8411	0.0852	0.2014	0.6330
$5 \cdot 10^{-8}$	0.0529	0.1754	0.8326	0.0786	0.1998	0.6617
$5 \cdot 10^{-7}$	**0.0517**	**0.1769**	**0.8364**	**0.0721**	**0.1940**	**0.6895**
$1 \cdot 10^{-6}$	0.0442	0.1632	0.8599	0.0894	0.2119	0.6151
$1 \cdot 10^{-5}$	0.0470	0.1660	0.8513	0.0955	0.1975	0.5887

Table 4.10: Models with changing strength, z and ϕ performance

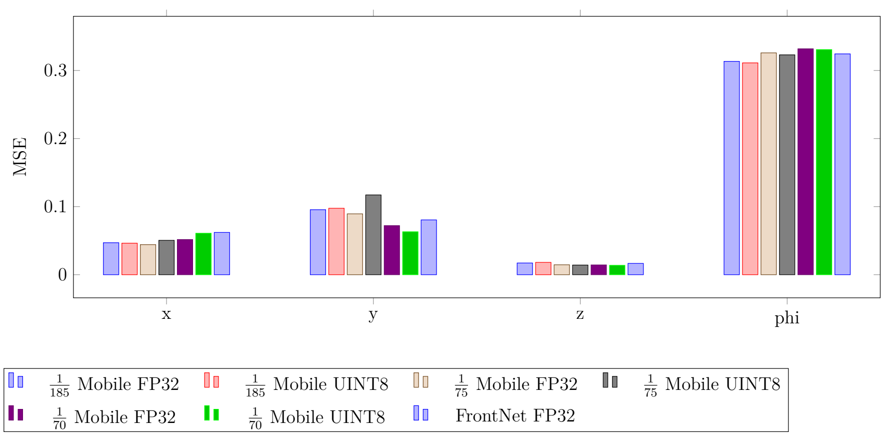
Strength	**z**			ϕ		
	MSE	**MAE**	**R2**	**MSE**	**MAE**	**R2**
0	0.0239	0.1092	0.3472	0.4472	0.5126	-0.0223
$1 \cdot 10^{-11}$	0.0171	0.0909	0.5335	0.3475	0.4506	0.2056
$1 \cdot 10^{-10}$	0.0151	0.0869	0.5885	0.3865	0.4717	0.1165
$1 \cdot 10^{-9}$	0.0142	0.0844	0.6132	0.3003	0.4190	0.3135
$1 \cdot 10^{-8}$	0.0159	0.0872	0.5664	0.2664	0.3944	0.3912
$5 \cdot 10^{-8}$	0.0141	0.0811	0.6154	0.2938	0.4205	0.3285
$5 \cdot 10^{-7}$	**0.0144**	**0.0819**	**0.6076**	**0.3319**	**0.4438**	**0.2414**
$1 \cdot 10^{-6}$	0.0146	0.0844	0.6016	0.3259	0.4333	0.2550
$1 \cdot 10^{-5}$	0.0172	0.0865	0.5308	0.3134	0.4291	0.2835

Quantization of MobileNet v1 1.0 models: Fake quantized

Table 4.11 and figures 4.13, 4.14 and 4.15 report the results for MobileNet v1 1.0. Also, in this case, the loss in performance is sufficiently low and all the networks maintain the results of the FP32 test set.

Table 4.11: Non quantized vs quantized FQ MobileNet v1 1.0 models

Name	$\frac{1}{185}$ **Mobile**	$\frac{1}{185}$ **Mobile 8**	$\frac{1}{75}$ **Mobile**	$\frac{1}{75}$ **Mobile 8**	$\frac{1}{70}$ **Mobile**	$\frac{1}{70}$ **Mobile 8**	**FrontNet**
Strength	**1E-5**	**1E-5**	**1E-6**	**1E-6**	**5E-7**	**5E-7**	**0**
FLOPS	$6.3 \cdot 10^6$	$6.3 \cdot 10^6$	$1.17 \cdot 10^7$	$1.17 \cdot 10^7$	$1.24 \cdot 10^7$	$1.24 \cdot 10^7$	$1.47 \cdot 10^7$
Par [MB]	0.07	0.07	0.16	0.16	0.18	0.18	1.14
# par	17603	17603	41566	41566	46256	46256	304356
MSE x	0.0470	0.0463	0.0442	0.0505	0.0517	0.0609	0.0621
MSE y	0.0955	0.0976	0.0894	0.1172	0.0721	0.0631	0.0805
MSE z	0.0172	0.0181	0.0146	0.0142	0.0144	0.0138	0.0166
MSE phi	0.3134	0.3112	0.3259	0.3230	0.3319	0.3307	0.3244
MAE x	0.1660	0.1664	0.1632	0.1729	0.1769	0.1956	0.187
MAE y	0.1975	0.1984	0.2119	0.2682	0.1940	0.1779	0.1837
MAE z	0.0865	0.0913	0.0844	0.0836	0.0819	0.0803	0.0927
MAE phi	0.4291	0.4225	0.4333	0.4312	0.4438	0.4399	0.4367
R2 x	0.8513	0.8535	0.8599	0.8401	0.8364	0.8070	0.8035
R2 y	0.5887	0.5799	0.6151	0.4953	0.6895	0.7283	0.6532
R2 z	0.5308	0.5066	0.6016	0.6114	0.6076	0.6239	0.5481
R2 phi	0.2835	0.2887	0.2550	0.2618	0.2414	0.2440	0.2585



H

Figure 4.13: MSE of non quantized vs quantized FQ Mobilenet v1 1.0 models

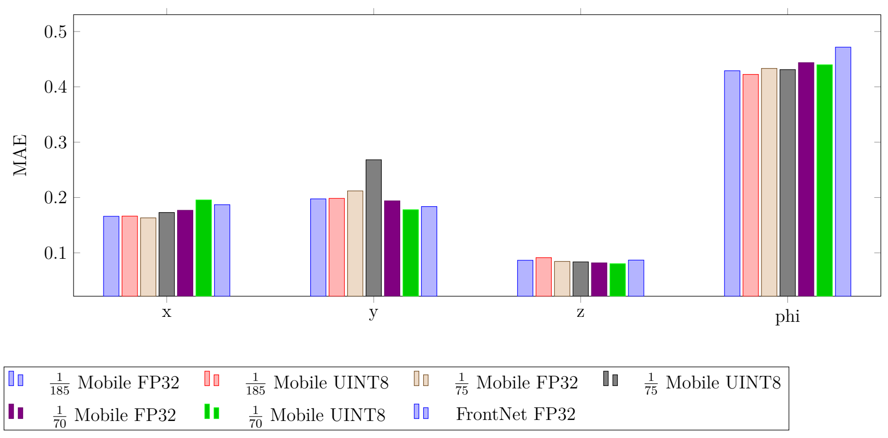


Figure 4.14: MAE of non quantized vs quantized FQ Mobilenet v1 1.0 models

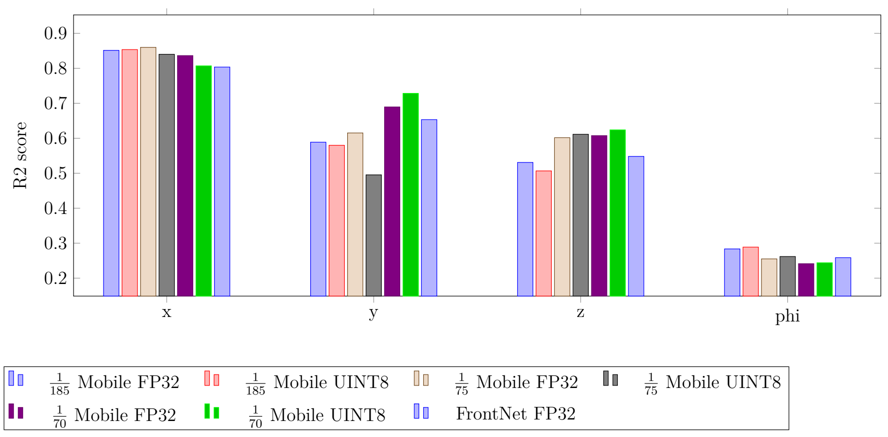


Figure 4.15: R2 score of non-quantized vs quantized FQ Mobilenet v1 1.0 models

Quantization of MobileNet v1 1.0 models: Integer deployable

Even in the case of MobileNet v1 1.0 in the step from fake quantized to integer deployable there is no loss of performance in any of the cases reported below. Details may be found in table 4.12 and graphs 4.16, 4.17 and 4.18.

Table 4.12: MobileNet v1 models, width multiplier = 1, Integer deployable

Name	$\frac{1}{185}$ **Mobile**	$\frac{1}{185}$ **Mobile 8**	$\frac{1}{75}$ **Mobile**	$\frac{1}{75}$ **Mobile 8**	$\frac{1}{70}$ **Mobile**	$\frac{1}{70}$ **Mobile 8**	**FrontNet**
Strength	**1E-5**	**1E-5**	**1E-6**	**1E-6**	**5E-7**	**5E-7**	**0**
FLOPS	$6.3 \cdot 10^6$	$6.3 \cdot 10^6$	$1.17 \cdot 10^7$	$1.17 \cdot 10^7$	$1.24 \cdot 10^7$	$1.24 \cdot 10^7$	$1.47 \cdot 10^7$
Par [MB]	0.07	0.07	0.16	0.16	0.18	0.18	1.14
# par	17603	17603	41566	41566	46256	46256	304356
MSE x	0.0470	0.0461	0.0442	0.0580	0.0517	0.0613	0.0621
MSE y	0.0955	0.0818	0.0894	0.0790	0.0721	0.0761	0.0805
MSE z	0.0172	0.0173	0.0146	0.0148	0.0144	0.0148	0.0166
MSE phi	0.3134	0.3037	0.3259	0.3344	0.3319	0.3153	0.3244
MAE x	0.1660	0.1650	0.1632	0.1886	0.1769	0.1935	0.187
MAE y	0.1975	0.1791	0.2119	0.1965	0.1940	0.1912	0.1837
MAE z	0.0865	0.0885	0.0844	0.0851	0.0819	0.0824	0.0927
MAE phi	0.4291	0.4214	0.4333	0.4407	0.4438	0.4301	0.4367
R2 x	0.8513	0.8540	0.8599	0.8164	0.8364	0.8060	0.8035
R2 y	0.5887	0.6476	0.6151	0.6600	0.6895	0.6723	0.6532
R2 z	0.5308	0.5286	0.6016	0.5974	0.6076	0.5968	0.5481
R2 phi	0.2835	0.3058	0.2550	0.2357	0.2414	0.2793	0.2585

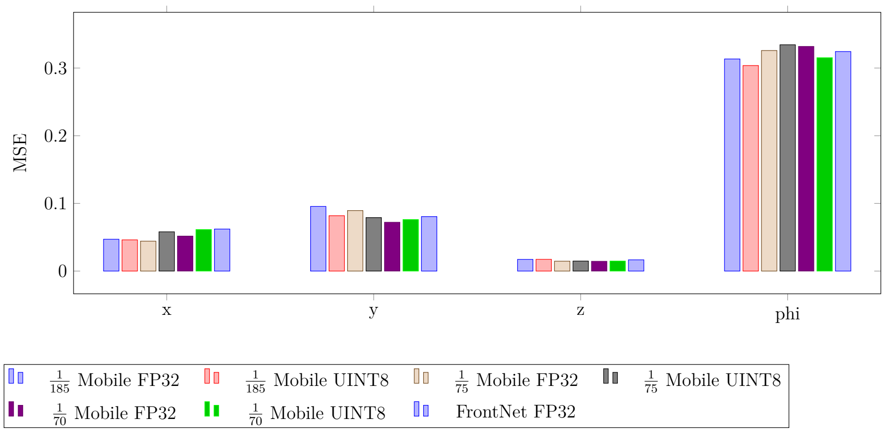


Figure 4.16: MSE of non quantized vs quantized ID Mobilenet v1 1.0 models

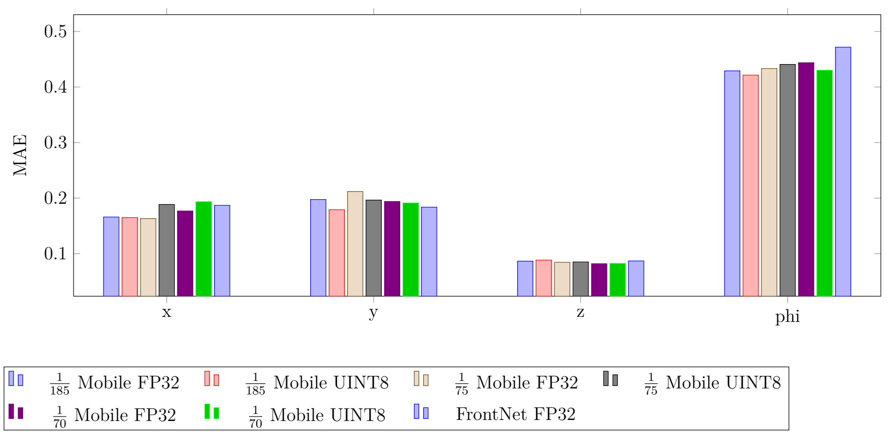


Figure 4.17: MAE of non-quantized vs quantized ID Mobilenet v1 1.0 models

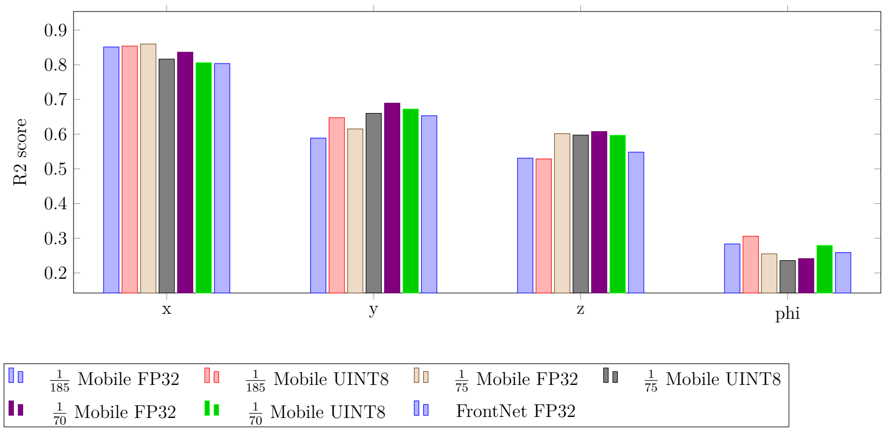


Figure 4.18: R2 score of non-quantized vs quantized ID Mobilenet v1 1.0 models

4.1.3 Criterion of selection

With the aim of optimizing the speed and reducing swaps from memory, without losing performance, a number of CNN have been chosen. Further attention to the selection of the networks was used in order to reduce the number of FLOPs and maintain elevated network execution performance. The models here selected are chosen between the networks explained in the space of solutions explored in the previous chapters. In particular the main considered are networks with a number of parameters lower than 300 thousand. This number of parameters is considered given the dimension of the L2 memory of the target GAP8 device, which is 512 KB, but should also contain the code of the program ($\sim 150k$).
All the performance reported in this section have been obtained on the test set.

The networks obtained from the searches done with PIT have been quantized in order to obtain avoid possible performance degradation after the choice of the networks to deploy. Figures 4.19, 4.21 and 4.20 report the average R2 performance (computed on x, y and z) for all the NNs and highlight in green the architectures reported in the tables and graphs below while, in blue are represented the seed Frontnet.

All the networks found by the NAS that belong to the Pareto front created analyzing inference cycles with respect to MAE were reported in figure 4.22 stressing the seed networks derivation with different colors. Considering all the architectures reported in figure 4.22 the deployed ones have been chosen picking the one with the highest performance in each seed network subset.
The network here reported as F_{seed} is Frontnet seed, F_{small} is the architecture with strength $\lambda = 2.5 \cdot 10^{-11}$ that is $\frac{1}{7}$ of the original Frontnet in size.
MobileNet networks that are referred as $M^{0.25}_{small}$ is $\frac{1}{4}$ of the seed MobileNet with width multiplier 0.25 and have been obtained with strength $\lambda = 10^{-10}$ while, $M^{1.0}_{small}$ is $\frac{1}{70}$ of the seed network MobileNet with width multiplier 1.0 that has a strength $\lambda = 5 \cdot 10^{-7}$. Although every network has been quantized, only the ones obtained through this selective approach have been reported in the next sections and have been tested infield.

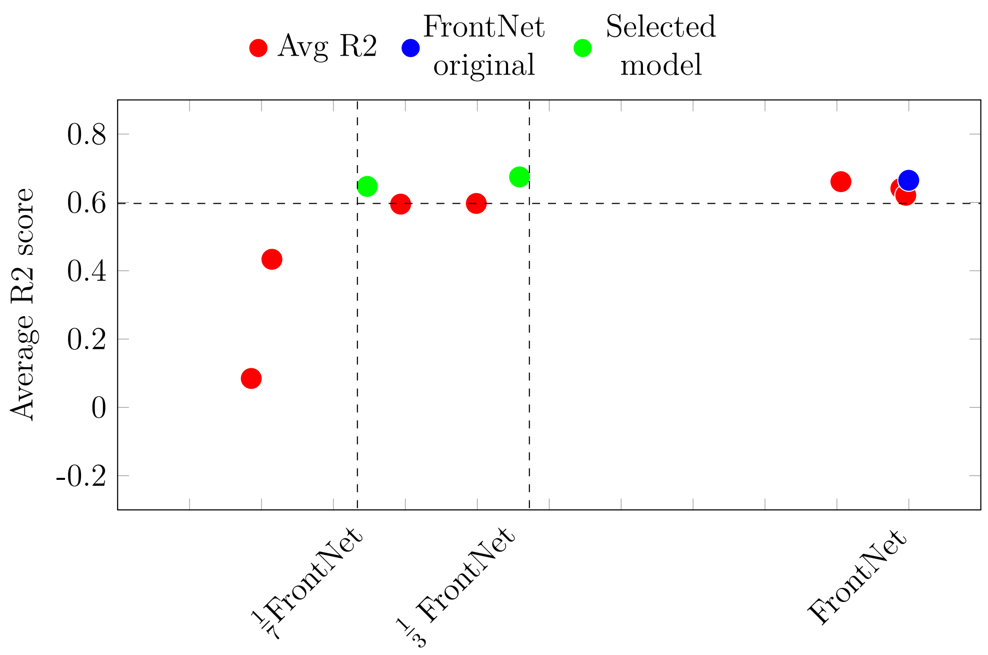


Figure 4.19: Average of FLOPs and size to average R2 x, y, z score FrontNet

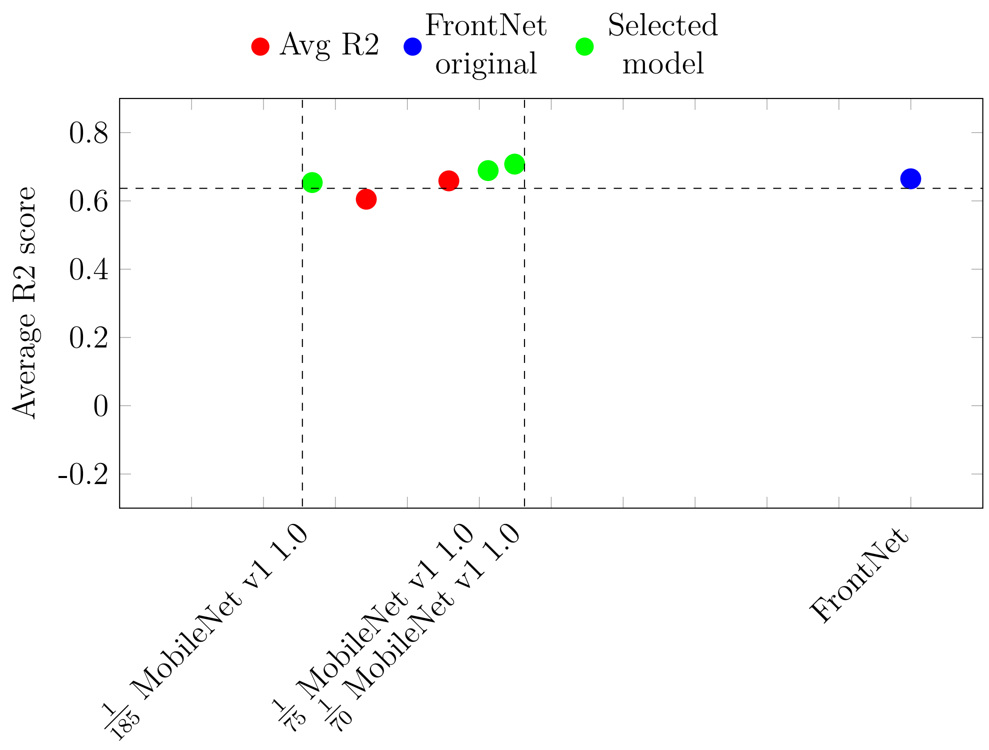


Figure 4.20: Average of FLOPs and size to average R2 x, y, z score MobileNet v1 1.0

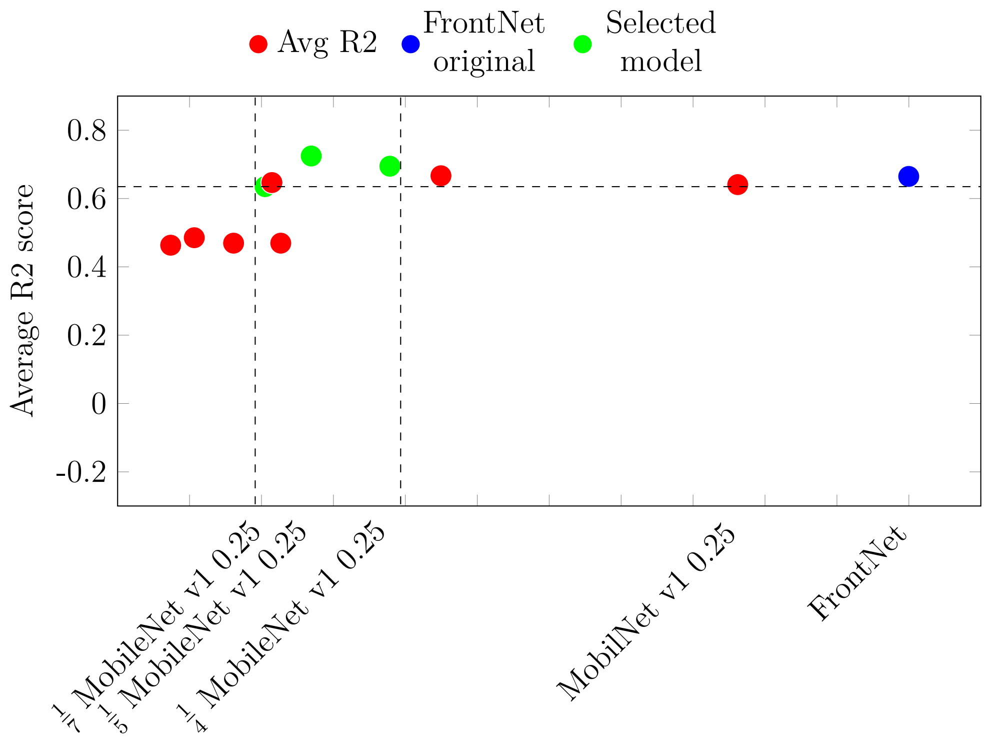


Figure 4.21: Average of FLOPs and size to average R2 x, y, z score MobileNet v1 0.25

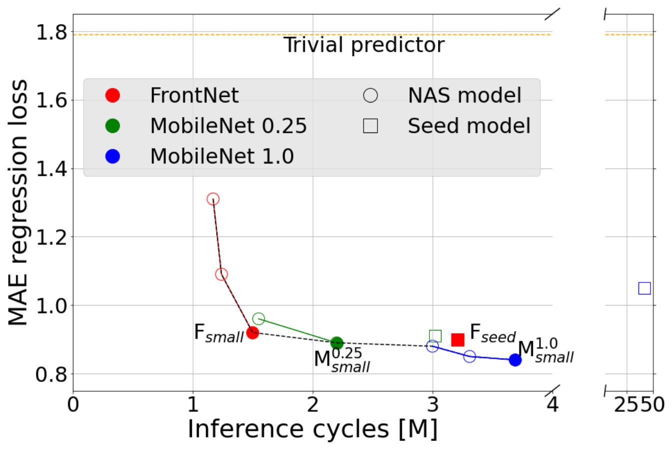


Figure 4.22: Pareto curves of the networks extracted from the NAS in the clock cycles vs. MAE space (lower is better).

.

In figure 4.23 are reported the quantized integer deployable and the floating point version of the networks to be tested infield. In all the cases reported, as reported above in the respective sections, there are no loss in performance.

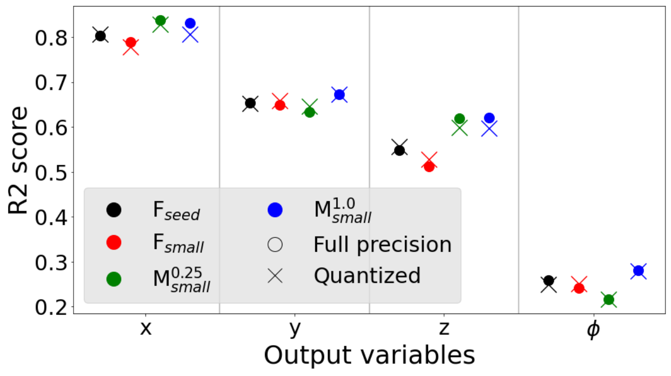


Figure 4.23: R2 of non-quantized vs quantized ID models (higher is better).

4.2 Results: power viewpoint and throughput

In this section are reported the results of the extensive tests performed on the SoC for the various networks.
All the networks reported below have been tested in the most power-hungry configuration for the GAP8 SoC, namely FC@250 MHz and CL@170 MHz, since in a robotic application the power requirement of the actual inference does not represent a crucial part of the whole, as reported in figure 4.24. By increasing the throughput the tracking performance improves consequently given that the error and the variance of the error of each inference remain constant. In fact, just 86.9 mW corresponding to 1.15 % of the total power is used by the GAP8 in the most performant version of networks employed (MobileNet$^{0.25}_{small}$). The rest of the power consumption is mainly due to the motors, accounting for more than 95% of the total. A relevant portion of the total energy cost, equivalent to 3.6%, may be considered for the crazyflie electronics. Finally, 4.3% of the power used by the AI deck (0.05% of the total) is absorbed by the camera.

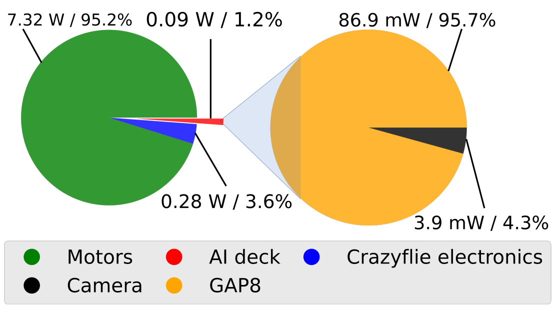


Figure 4.24: Power consumption breakdown of the Crazyflie 2.1 with inference onboard (Mobilenet$^{0.25}_{small}$)

.

Figure 4.25 reports different power consumption and consequently fps, for each network, depending on the frequencies used by the cluster and the fabric controller. Although the infield test has been performed only for the most power-hungry configuration, figure 4.25 and 4.26 report also the most energy efficient configuration (FC@25 MHz and CL @ 75 MHz) and the configuration with the highest power saving (FC@25 MHz and CL@25 MHz). Even though these two settings are not of interest for the drone's control application, they may be considered for standalone pose estimation tasks through IoT devices, in which a long-lasting time is of crucial importance.
The networks obtained thanks to the NAS need consistently less power (in the order of 5%) if compared to the seed network. It is worth mentioning that MobileNet-derived architectures need higher maximum power if compared to Frontnet cases. Even if the NAS performs a reduction of all the networks it does not change the peculiar energy characteristics of the networks that are almost immutated for a specific seed architecture. In fact, acting on the channels the technique modifies only the relative duration of each layer thus it results in a shortening or enlargement of the power consumption of the specific layer.

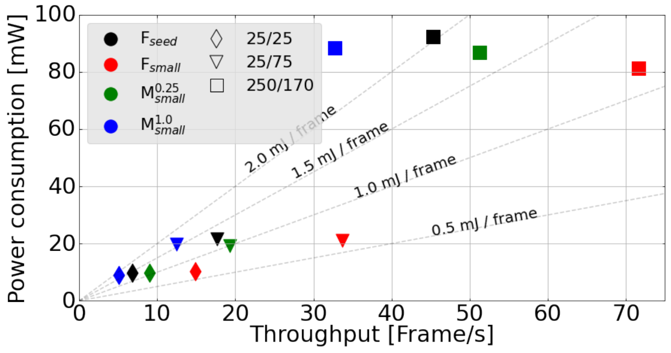


Figure 4.25: Throughput vs. power consumption for the four models at three different frequencies operating points (FC/CL).

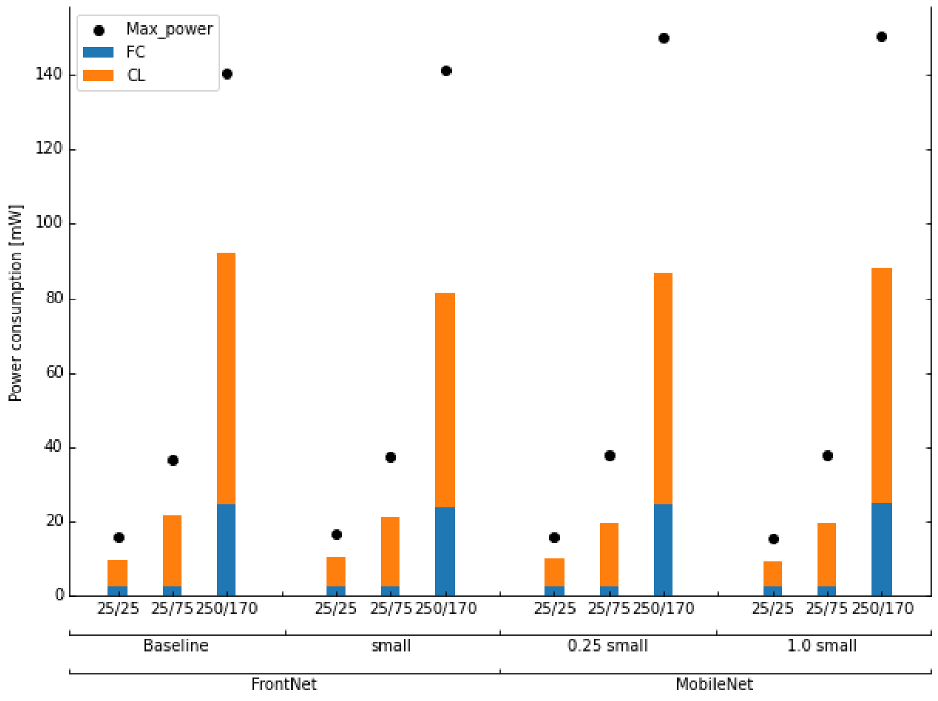
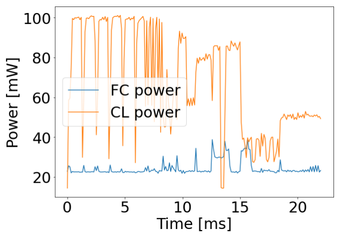


Figure 4.26: Power consumption of all the configurations.

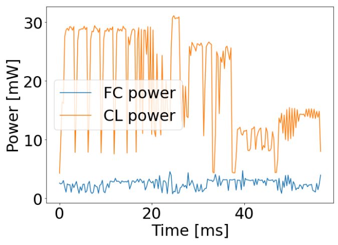
Table 4.13: Computation and memory footprint for inference on one frame (**F**: PULP-Frontnet, **M**: MobileNet).

Network	Params	MAC	Cycles	Memory	P [mW]	T [fps]
$\mathbf{F}_{seed}$	304 k	14.7 M	3.2 M	499 kB	92.2	45.3
$\mathbf{F}_{small}$	44 k	7.6 M	1.5 M	231 kB	81.3	71.6
$\mathbf{M}^{0.25}_{small}$	65 k	7.4 M	2.2 M	591 kB	86.9	51.2
$\mathbf{M}^{1.0}_{small}$	54 k	12.4 M	3.7 M	415 kB	88.3	32.7

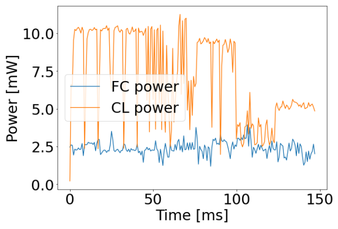
As reported in table 4.13 the NAS extended in this thesis was able not only to reduce the size of the networks (evaluated considering the # of parameters) but also to reduce MACs, cycles and the total memory employed.
The memory values reported in table 4.13 set an upper bound of the memory that should be used if all the parameters and outputs of the various layers would be maintained until the end of each inference.
MACs and cycles instead approach the computational complexity problem from two different points. On the one hand, the MACs metric provides a measure that is independent of the number of cores employed and from the SoC architecture. On the other side, cycles are evaluated Specifically on the GAP8 and are directly related to the time needed through the frequency at which the processor is used
In order to deeply understand the behaviors and differences of the networks power profiling (at 64 kbps) has been performed for all the configurations reported above by means of a GAPuino board located in ETH-Zurich laboratories.
All the power profiles are reported in figures 4.27a, 4.27b and 4.27c for Frontnet seed networks, figures 4.28a, 4.28b and 4.28c concerning the reduced version of Frontnet. Power profiles of MobileNet v1 networks are instead reported in figures 4.29a, 4.29b and 4.29c considering MobileNet$^{0.25}_{small}$ and in figures 4.30a, 4.30b and 4.30c for MobileNet$^{1.0}_{small}$. Considering the figures reported below, a crucial difference may be observed between Frontnet-based networks and MobileNet networks. All the configurations of the first seed network have relatively long phases in which the processor consumes a reduced amount of power but is active. On the other side, the MobileNet-based networks have consistently higher power consumption. This is phenomenon is due mainly to the higher number of layers in the MobileNet networks and to the different types of convolution employed in the two architectures.



(a) 250/170

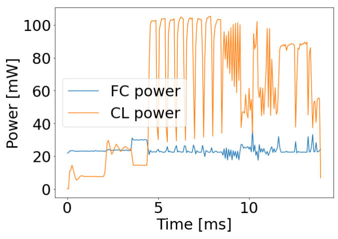


(b) 25/75

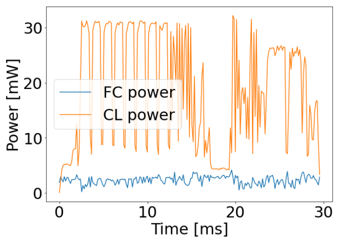


(c) 25/25

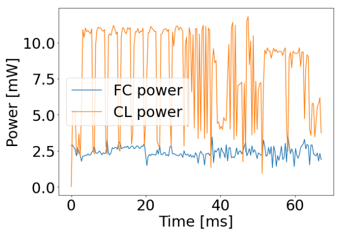
Figure 4.27: Frontnet seed power profiling



(a) 250/170

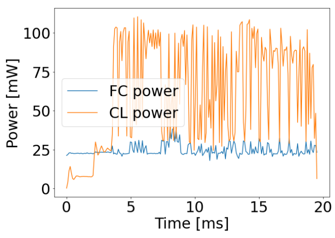


(b) 25/75

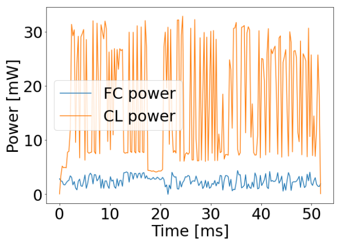


(c) 25/25

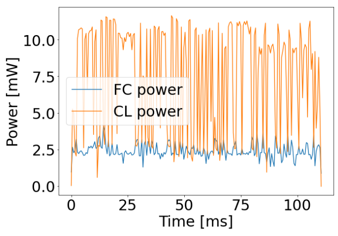
Figure 4.28: Frontnet$_{small}$ power profiling



(a) 250/170

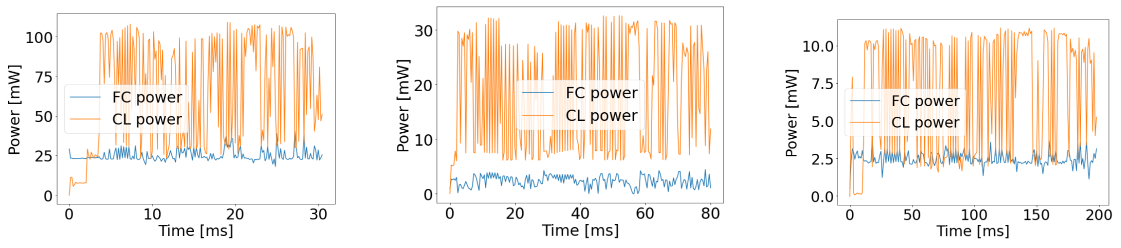


(b) 25/75



(c) 25/25

Figure 4.29: MobileNet$^{0.25}_{small}$ power profiling

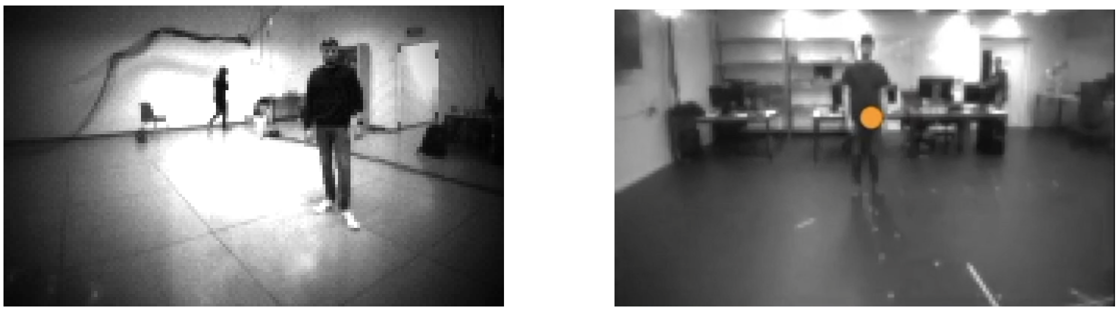


(a) 250/170 **(b)** 25/75 **(c)** 25/25

Figure 4.30: MobileNet$^{1.0}_{small}$ power profiling

4.3 Results: control accuracy and generalization standpoints

For all the networks selected the in-field tests described in section 3.3.4 have been performed providing accurate results on the inference and closed loop control accuracy. Since the laboratories used for the trials were never seen in the training phase these experiments should be considered valid also for generalization purposes. Furthermore, the new laboratory presents difficult backgrounds with monitors, windows, chairs, and tables that contribute to the validation of the generalization statements. In figure 4.31 are reported two images taken from the onboard camera of the drone of the two labs that depict the very different environment of testing if compared to training.



(a) Training lab **(b)** Testing lab

Figure 4.31: Onboard camera's images in two different environments

In the testing room, the approach developed in this work provides SoA results allowing the networks to complete the whole path in contrast to the Frontnet seed version that only completed 85% of it on a three runs average. Even more, derived architectures were able to reduce the control error and the inference error evaluated through the MAE between the distance obtained via the optitrack tracking system

(position error < 0.2 mm) and the distance estimation onboard.
In order to provide an upper bound on the inference accuracy evaluated in the tests, an experiment performed controlling the drone with the mocap's pose measure has been done. Figure 4.32 reports the path followed while, table 4.18, in the Mocap row, provides some details on the control accuracy obtained.

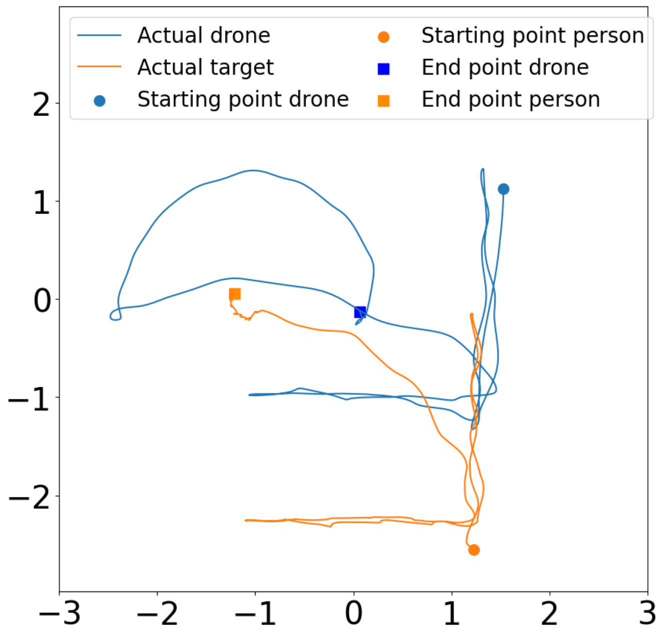


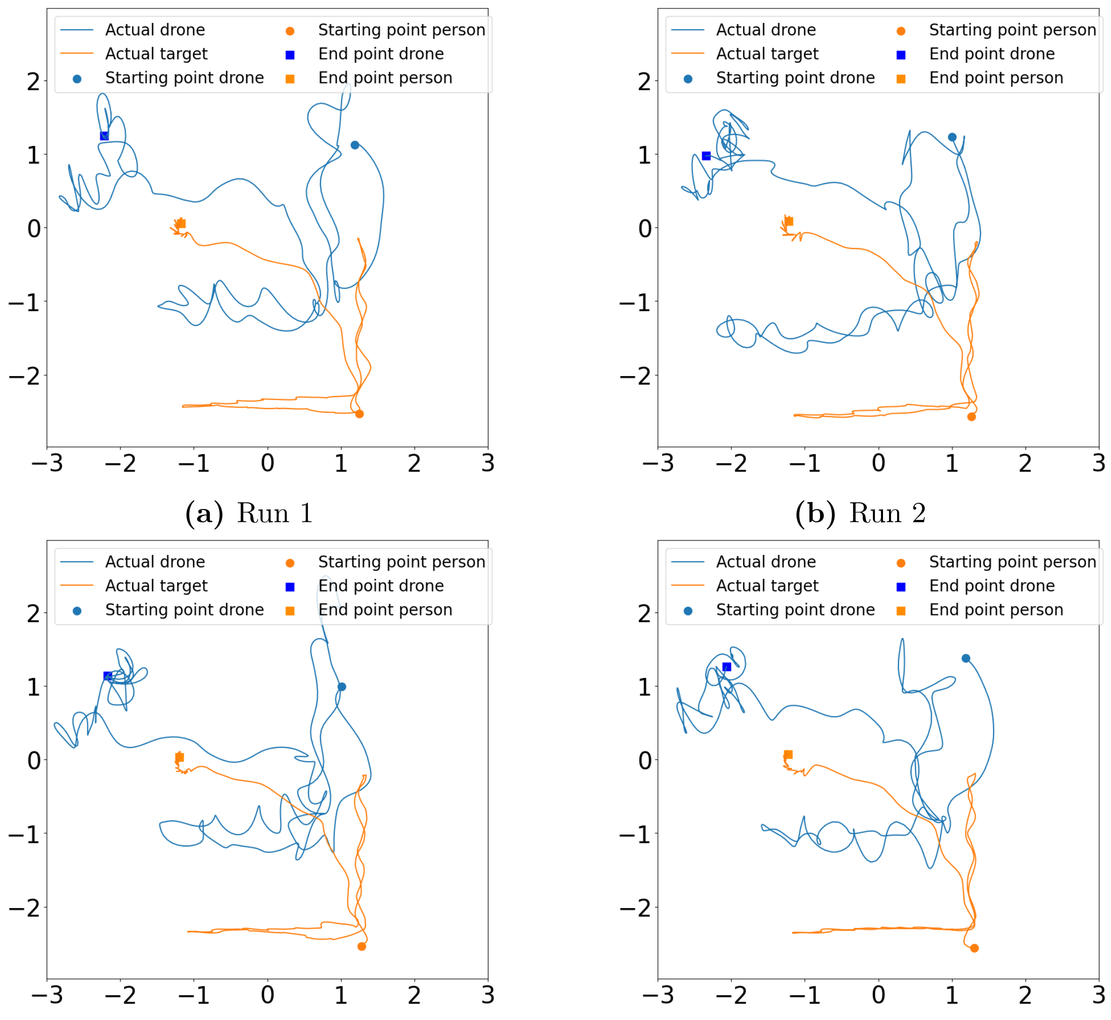
Figure 4.32: Mocap controlled drone path

Mean pose errors in table 4.18 are obtained as the differences between the drone's desired pose and the actual drone pose, as a consequence these metrics embed the control system error and, if the pose estimation is perfect, as in the case of the mocap control, the metrics express the errors of the controller alone.

4.3.1 Frontnet$_{seed}$

In the case of Frontnet$_{seed}$ only 140 s of total flight have been achieved compared to a total of 165 s of the whole path, equivalent to 85% of the whole way. In particular, the main weakness of Frontnet may be seen in the last part of the course, where the ϕ prediction is stressed with an in-place rotation of the target.
The mean MAEs over the 4 runs are 0.33, 0.12, and 0.77 respectively for x, y, and ϕ. Instead, for what concerns the controller errors, e_{xy}=0.72 m and e_{θ}=0.78 rad, representing an increase of more than 270% on the mocap test control errors. On the one hand, exploring more in detail the scatter plot related to the ϕ variable, an

almost random inference set may be seen with r=0.1196 as reported in figure 4.34d, this clarifies the malfunctions of the prediction system in the rotation phase (part 6 in 3.13). On the other side, the others variable used for control, x and y reported in figure 4.34a and 4.34b, show a relevant positive correlation with respective perfect predictors. Finally, concerning z, a clusterized output mirrors the limited data variability on the z-axis in the training set.



(a) Run 1

(b) Run 2

(c) Run 3

(d) Run streamer

Figure 4.33: Frontnet$_{seed}$ path

Considering these 4 experiments reported, Frontnet$_{seed}$ shows an incomplete tracking with consequently poor MAE error metric's performance that should be linked to the reduced number of layers and to the elevated number of parameters that do not allow sufficient generalization performance.

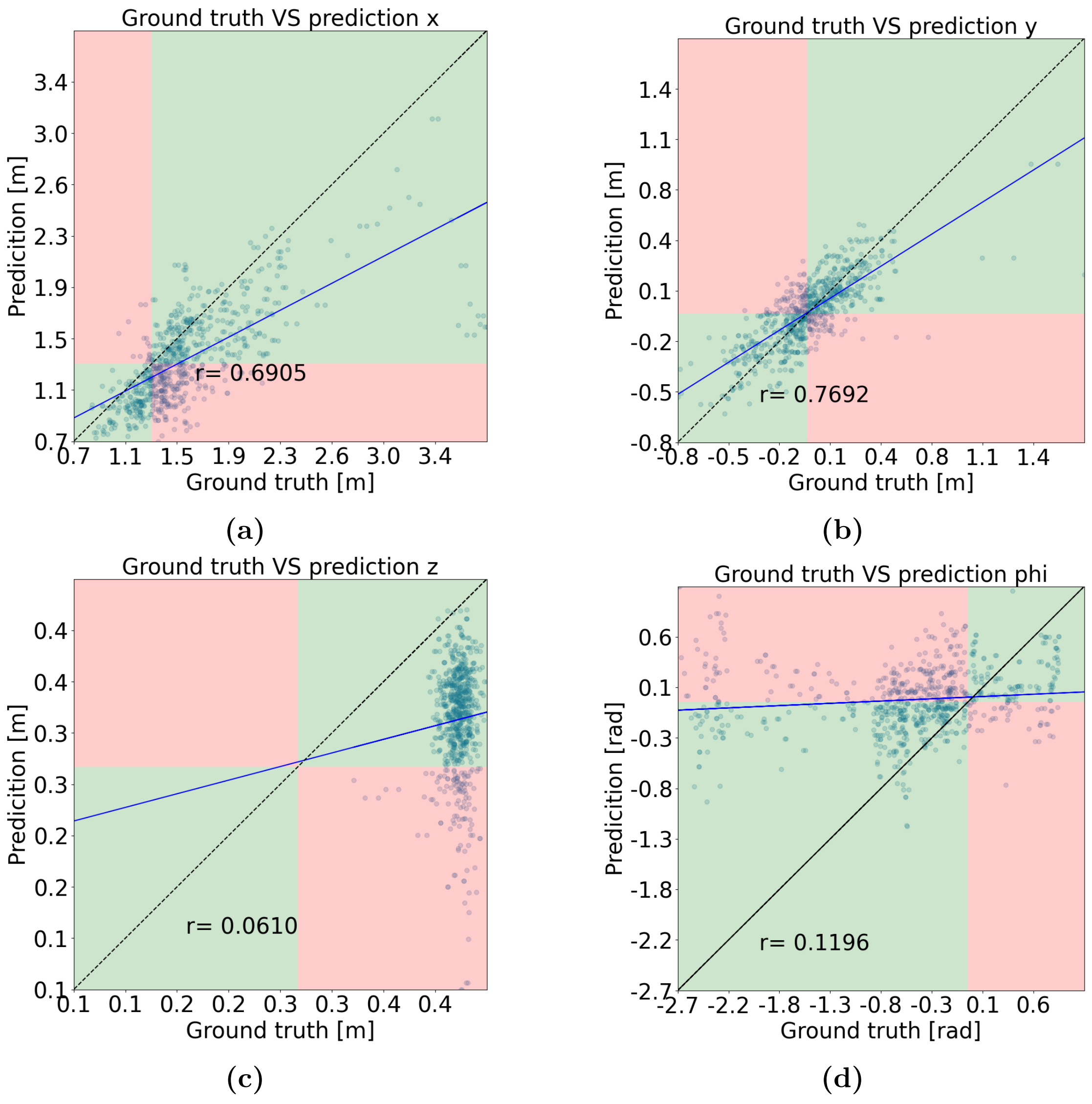
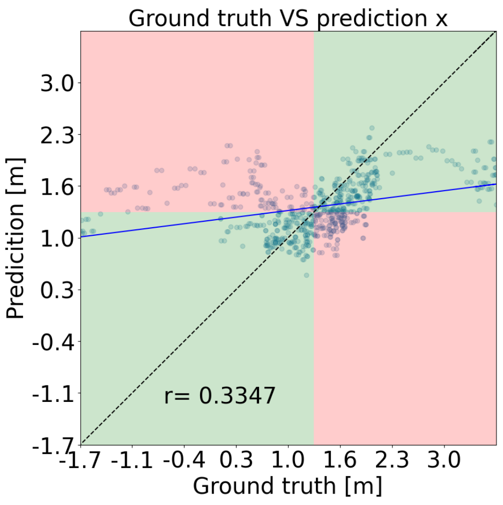


Figure 4.34: Frontnet seed scatterplot

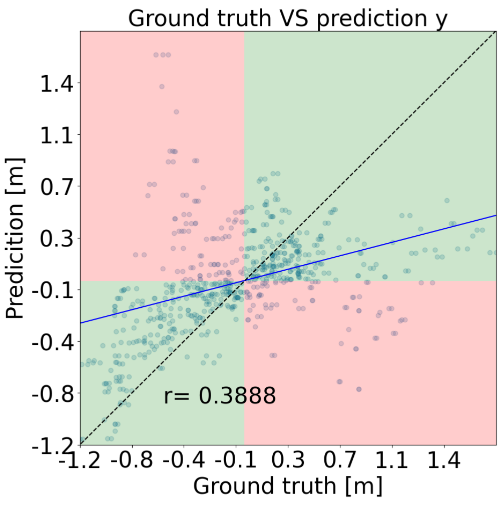
4.3.2 Frontnet$_{small}$

The reduced version of Frontnet does not provide satisfactory results in the onfield tests, in fact, it only reaches 35% of the complete path on a 4 runs average, although with the testing set it provides comparable R2 and MAE score to Frontnet$_{seed}$. Further insights are provided by figures 4.35a, 4.35b, 4.35c and 4.35d which respectively depict the scatterplot of variables x, y, z and ϕ.

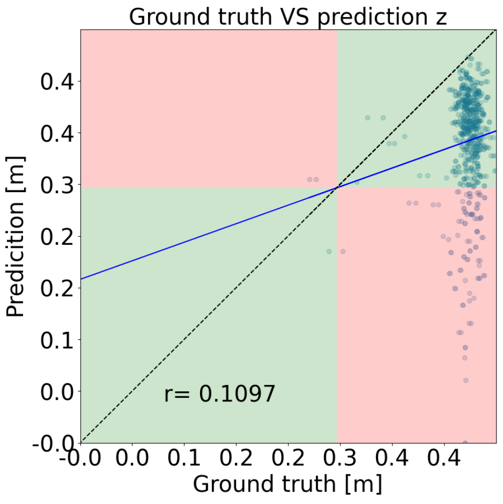
In the case of Frontnet$_{small}$ the problem in the tracking of the person seems due to the insufficient prediction performance of x and y. The variable ϕ, considered the weakness of the Frontnet model, is not stressed in these tests since the drone barely tracks the person until phase 5 of figure 3.13 and as so it has a relatively high regression performance (r=0.4068) since only the easier fraction of the angles' range is present in the tests.
The reduced number of layers in combination with the reduced number of parameters does not allow the network to perform correctly.



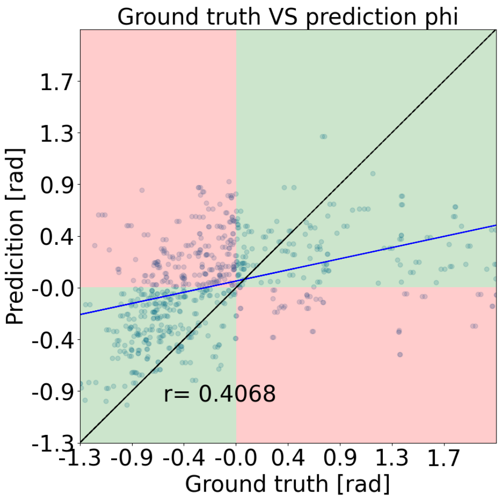
(a)



(b)



(c)

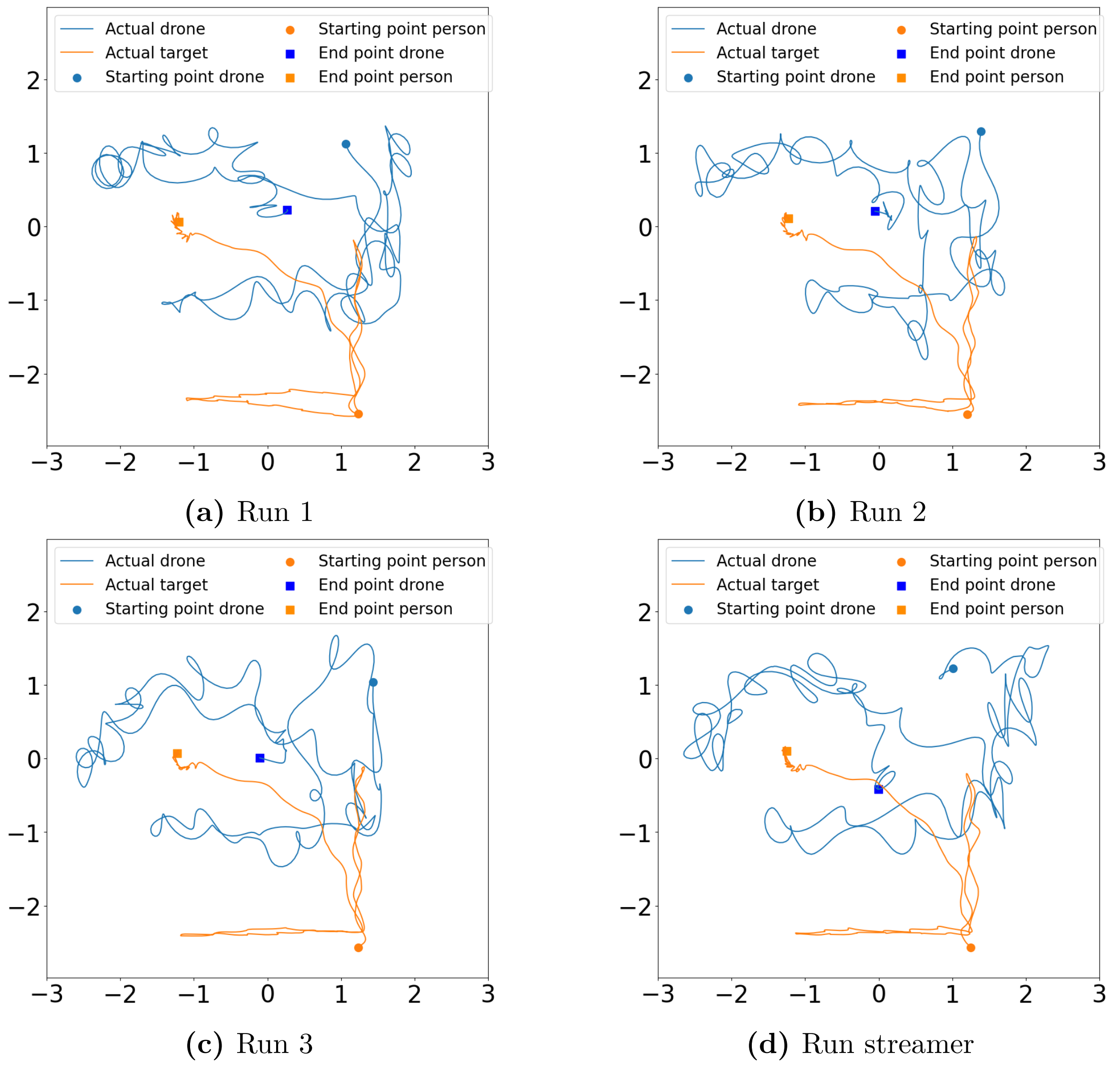


(d)

Figure 4.35: Frontnet$_{small}$ scatterplot

4.3.3 MobileNet$^{0.25}_{small}$

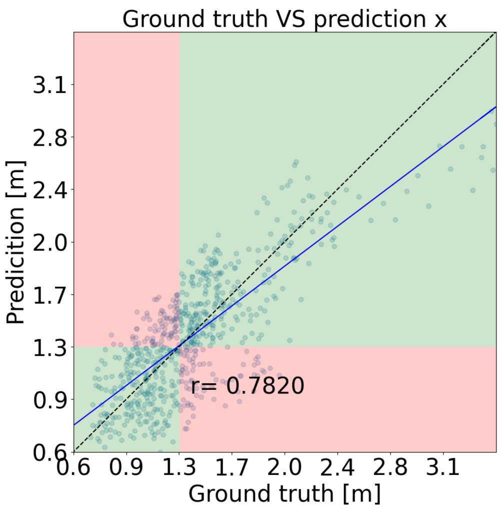
The network obtained from MobileNet with width-multiplier=0.25 is able to complete the whole path and, it reaches satisfactory errors for what concerns the control. In particular it achieves an e_{xy}=0.49 m and an e_{θ}=0.59 rad. Instead, considering the MAE performance this work achieved 0.25, 0.11 and 0.52 respectively on x, y and ϕ. Figures 4.36a, 4.36b, 4.36c and 4.36d show the paths followed by the drone and the target with the network under analysis.



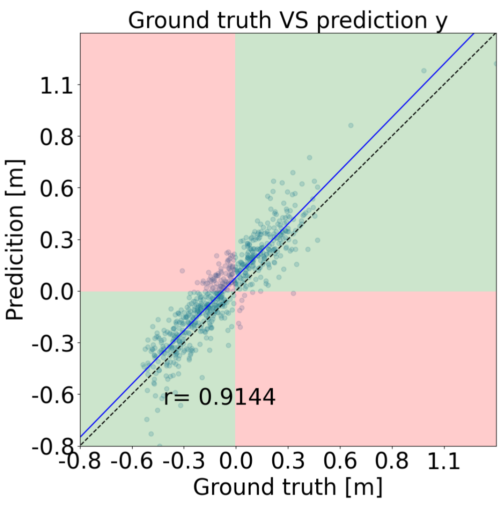
(a) Run 1 (b) Run 2 (c) Run 3 (d) Run streamer

Figure 4.36: MobileNet$^{0.25}_{small}$ path

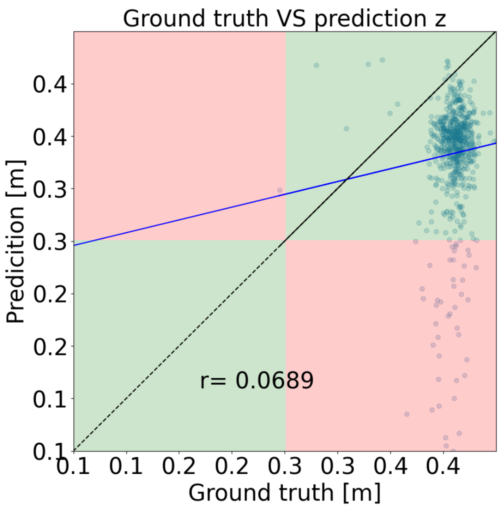
Figures 4.37a, 4.37b, 4.37c and 4.37d report the scatterplots of the predicted features. In this case, x and y have a high correlation with the perfect predictors while ϕ has a sufficiently high correlation to perform in the proper way. On the other side, the weak z performance confirms that the choice to fix z is proper. Further details may be observed in figures 4.38a and 4.38b where an almost perfect following happens.



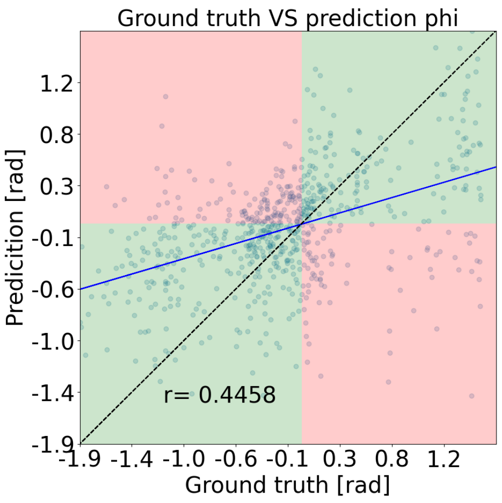
(a)



(b)



(c)

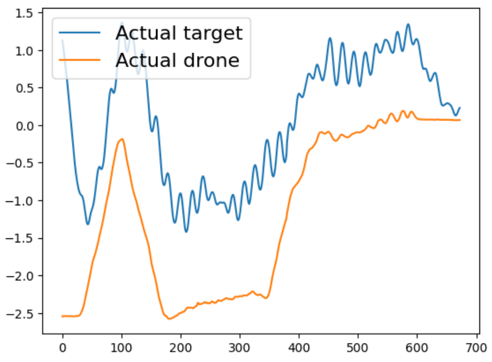


(d)

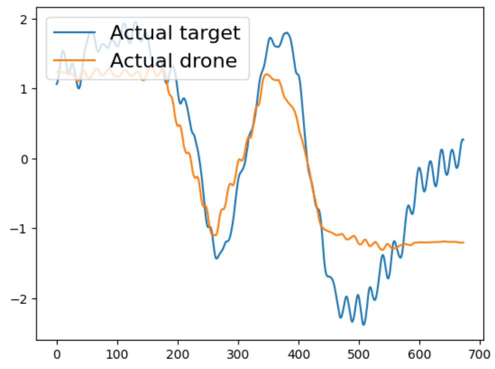
Figure 4.37: MobileNet$^{0.25}_{small}$ scatterplot

In the case of this architecture, the elevated number of layers gives the correct ability to learn and generalize correctly providing a reliable model that could be ported into different environments. Further strength is given to the particular

NN by the reduced number of parameters providing even more generalization capabilities.
The improvement obtained with this network is particularly remarkable since from the first Frontnet network the prediction accuracy of the angle ϕ was challenging and in fact presented reduced MAE performance in the original work.



(a) x

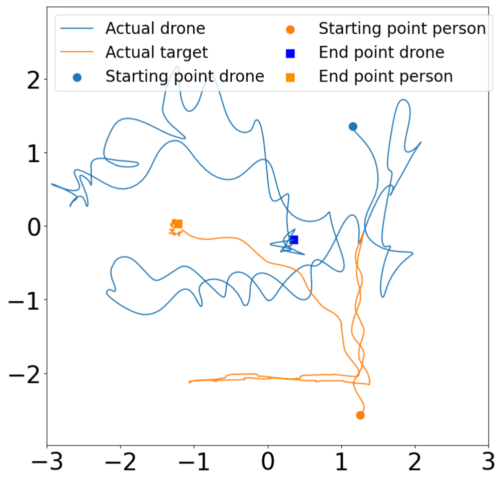


(b) y

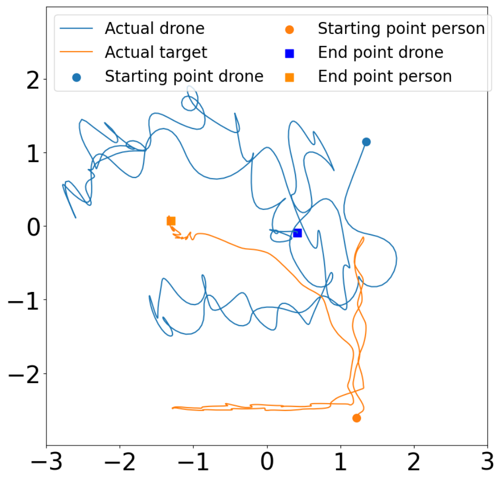
Figure 4.38: MobileNet$^{0.25}_{small}$ tracking performance in time (from frame 400 has been performed the in-place rotation of the target)

4.3.4 MobileNet$^{1.0}_{small}$

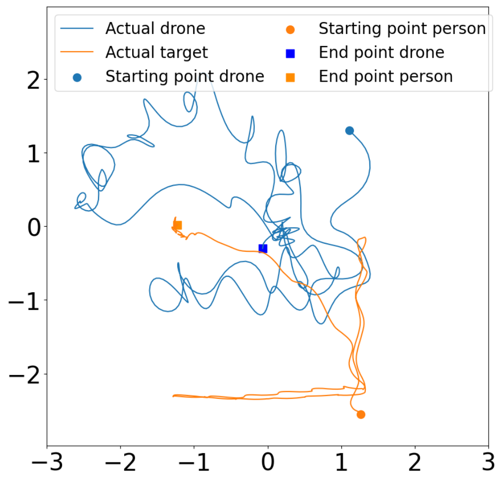
Even this MobileNet-derived network is able to complete the whole path and, it scores an MAE of 0.31, 0.13, and 0.52 respectively for x, y, and ϕ.
The control system errors are e$_{xy}$=0.58 m and e$_{\theta}$=0.59 rad. Figures 4.39a, 4.39b, 4.39c and 4.39d depict the paths obtained in the various runs.
Figures 4.40a, 4.40b, 4.40c and 4.40d report the scatterplots of the predicted features. Also, in this case, x and y have a high correlation with respect to the perfect predictors, ϕ has correlation=0.5156 that allows the tracking of the angle to perform in the proper manner. Although, z performance seems to present a positive correlation the vertical cluster at Ground truth=0.4m confirms that the choice to fix z is proper.
Also in this case, the ability to complete the path showed by the architecture allows the extension of the use cases to never seen environments and provides a wider range of applications. Further details may be observed in figures 4.41a and 4.41b where an almost perfect following happens although some variability on the x prediction may be observed with a relevant oscillation effect.
From frame 400, the inplace rotation has been performed as a consequence the tracking performace seen in figures 4.41a and 4.41b seems to perform poorly.



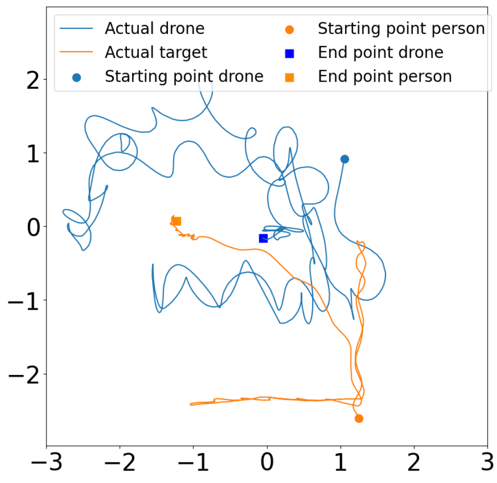
(a) Run 1



(b) Run 2

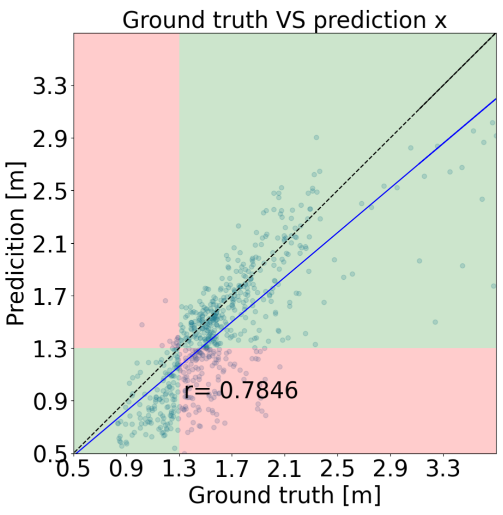


(c) Run 3

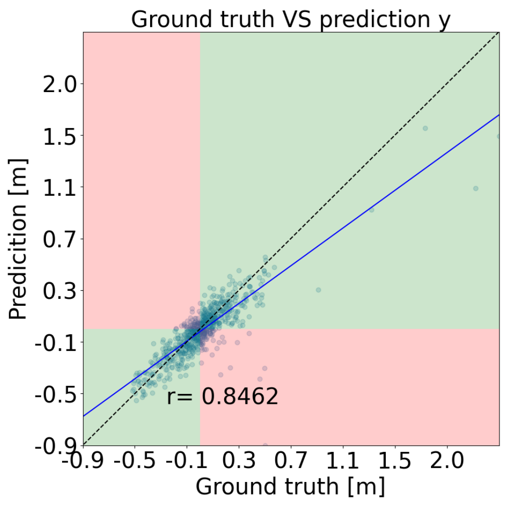


(d) Run streamer

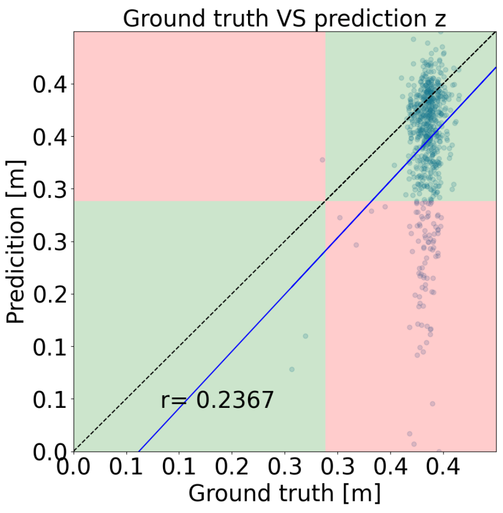
Figure 4.39: MobileNet$^{1.0}_{small}$ path



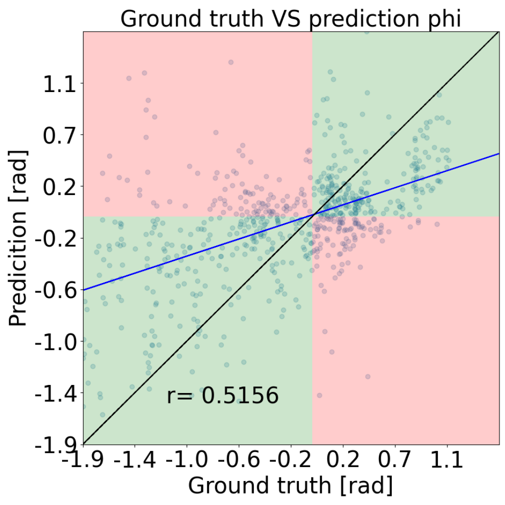
(a)



(b)

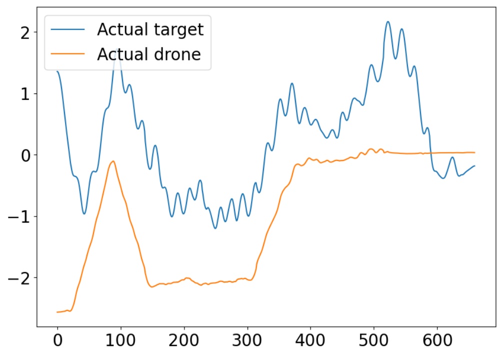


(c)

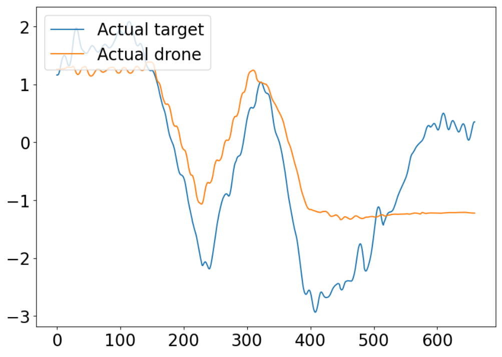


(d)

Figure 4.40: MobileNet$^{1.0}_{small}$ scatterplot



(a) x



(b) y

Figure 4.41: MobileNet$^{1.0}_{small}$ tracking performance in time (from frame 400 has been performed the in place rotation of the target)

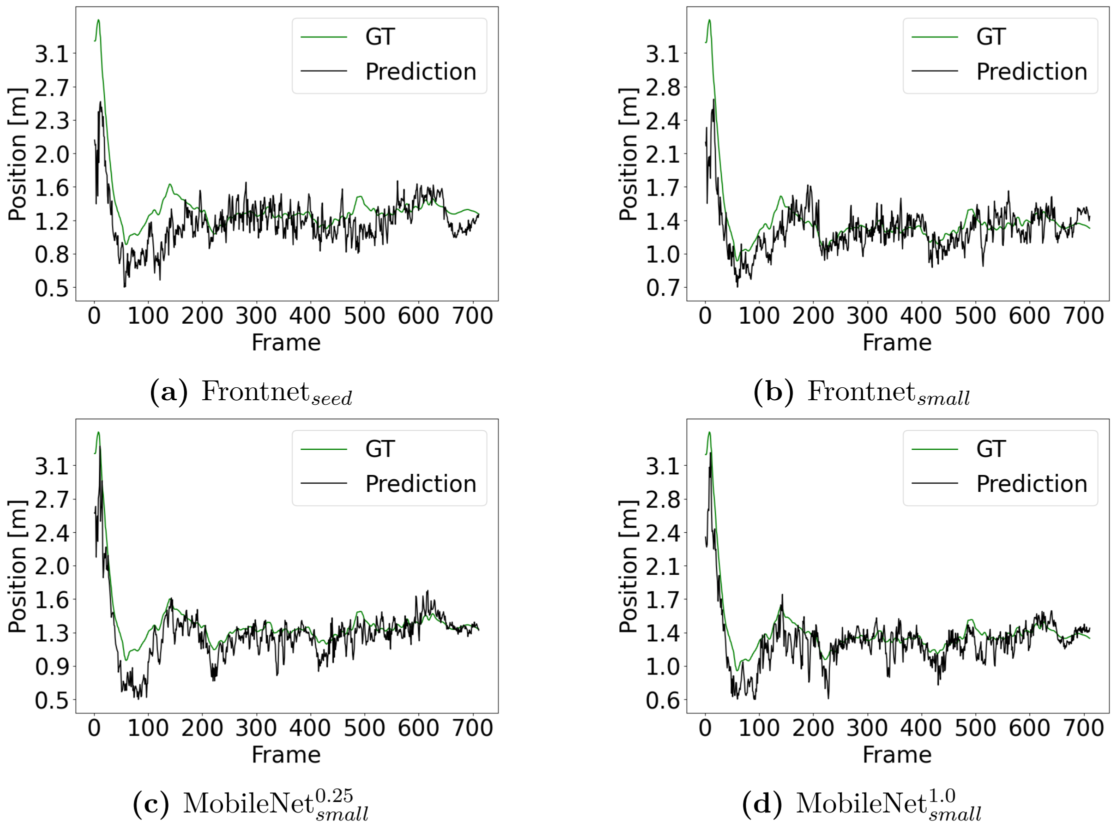
4.3.5 Cross tests validation with flying drone

Cross tests have been done in order to validate the performace and avoid a misleading attribution of the error to the specific network, understanding how the others NNs would have worked in a specific condition.
For these experiments only the runs with the streamer have been used since the presence of images is crucial to perform post processing inferences.

Mocap control tests

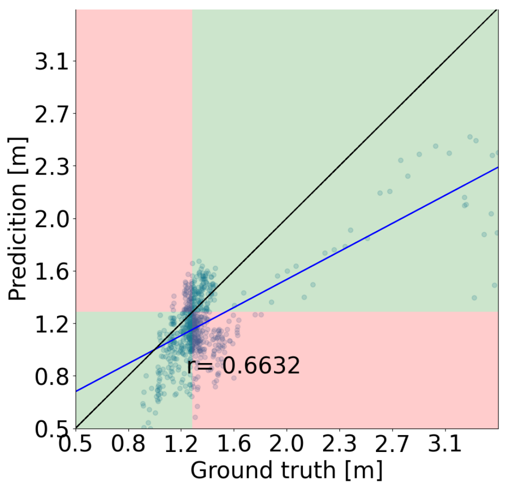
These tests have been realized controlling the drone through the motion capture system and, as such, reducing to almost 0 the prediction error of the drone relative pose. In fact, controlling the drone through the mocap is practically equivalent to a perfect prediction system. This cross test have been done in order to prove that the errors in the predictions' value are mostly related to the actual network rather then caused by the pitch oscillation for example. During the test with the mocap, the pitch oscillations are reduced to the minimum achieving the less variable system that can be obtained with this particular controller.
Figure 4.42, 4.44, 4.46 and 4.48 provide a representation of the x, y, z and ϕ behaviour of the various NNs in time while, figures 4.43, 4.45, 4.47 and 4.49 depict the performance through a scatter plot comparing Ground Truths (GTs) and predictions respectively for x, y, z and ϕ with changing architecture.
Tables 4.14, 4.15, 4.16 and 4.17 report MAE and MSE for every control variables.
In figure 4.42a is present an underestimation from frame 450 to frame 550 that may lead the drone to acquire more distance and as such perform less if comprard with figures 4.42b, 4.42c and 4.42d.
Analyzing instead the graphs concerning the y axis, 4.45c, namely the most accurate

model in field MobileNet$^{0.25}_{small}$ seems the best performing network, although every architecture performs sufficiently accurate prediction on this variable. The graphs representing the z variables, 4.46 and 4.47, in particular with 4.47a display significant and meaningful predictions for the z axis except for the Frontnet$_{small}$ version where the inferences do not perform properly. Finally, for what concerns the ϕ variable, the two versions of Frontnet present more than 50% of the sample in wrong areas, namely the two red ones, resulting in incorrect predictions most of the time and as such, they justify the absence of tracking in the experiments reported above in figure 4.33 and also for the version of Frontnet$_{small}$.

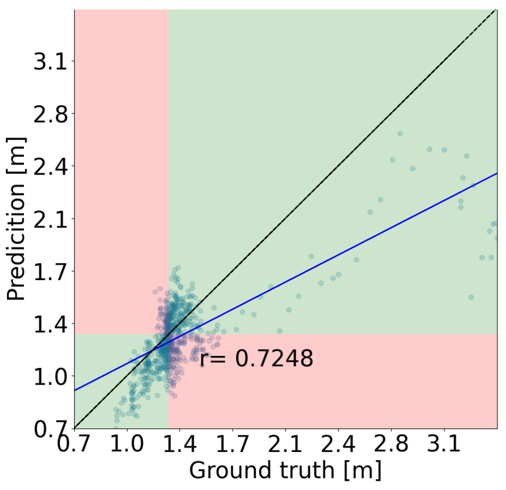


(a) Frontnet$_{seed}$ **(b)** Frontnet$_{small}$ **(c)** MobileNet$^{0.25}_{small}$ **(d)** MobileNet$^{1.0}_{small}$

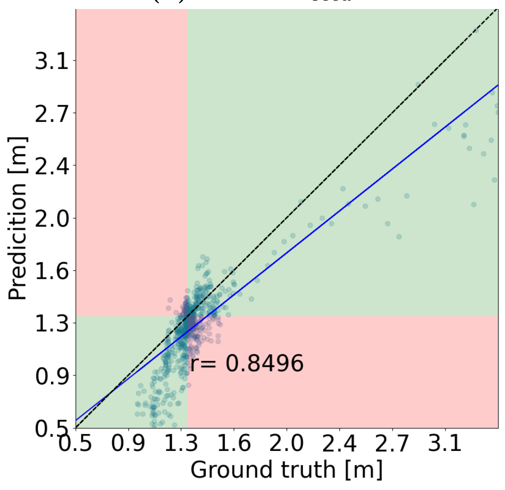
Figure 4.42: x predictions in time with changing network



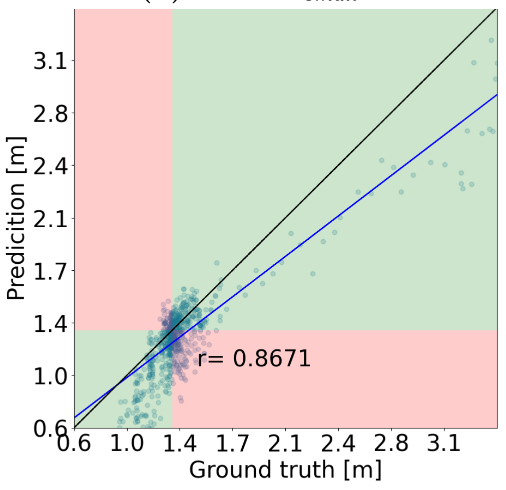
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$



(c) MobileNet$^{0.25}_{small}$

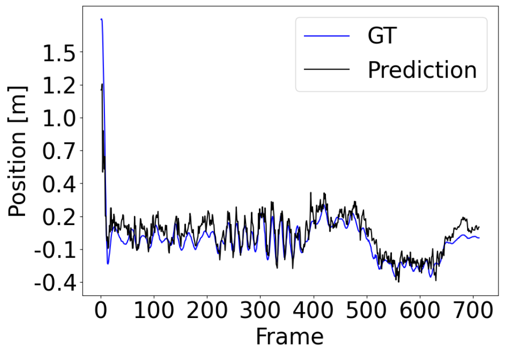


(d) MobileNet$^{1.0}_{small}$

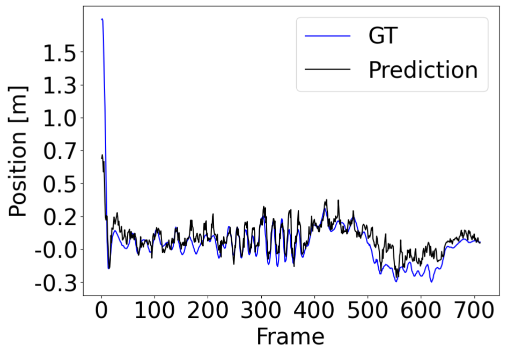
Figure 4.43: GT vs predictions x

Table 4.14: MAE and MSE variable x

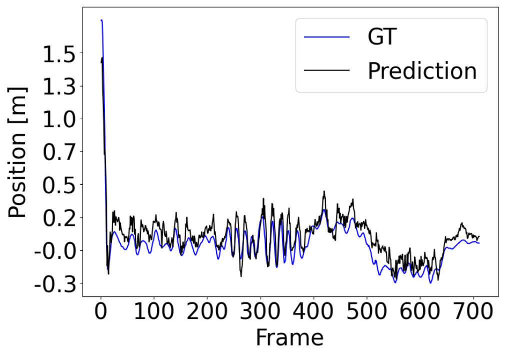
Network	MAE	MSE
$\mathbf{F}_{seed}$	0.21	0.09
$\mathbf{F}_{small}$	0.16	0.06
$\mathbf{M}^{0.25}_{small}$	0.15	0.05
$\mathbf{M}^{1.0}_{small}$	0.14	0.04



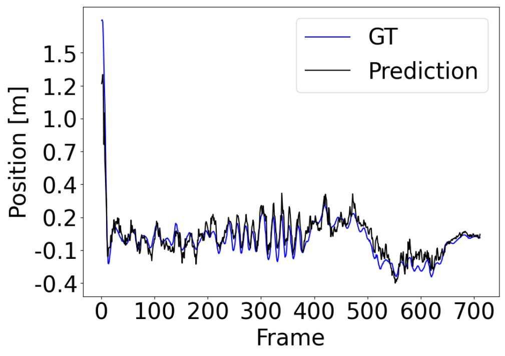
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$

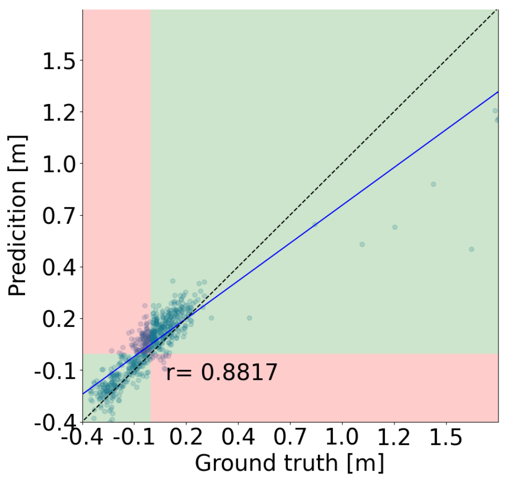


(c) MobileNet$^{0.25}_{small}$

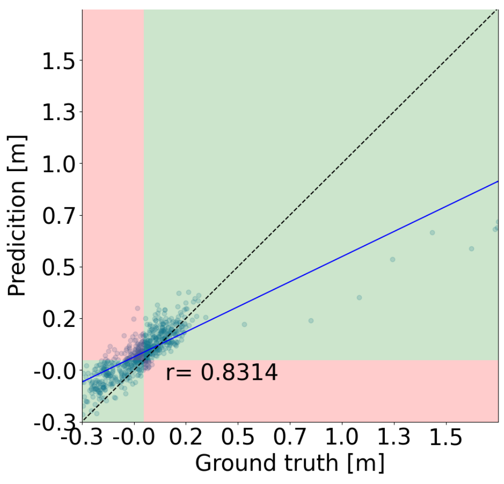


(d) MobileNet$^{1.0}_{small}$

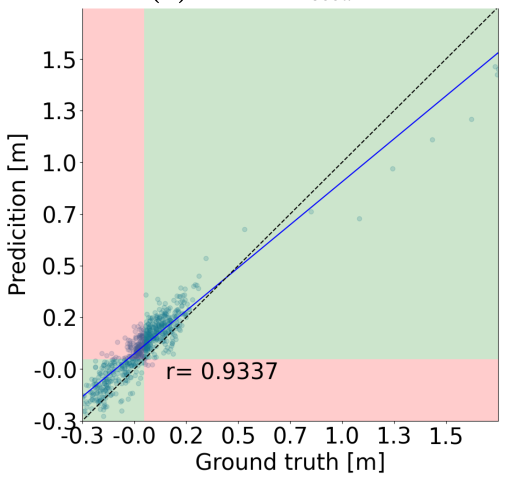
Figure 4.44: y predictions in time with changing network



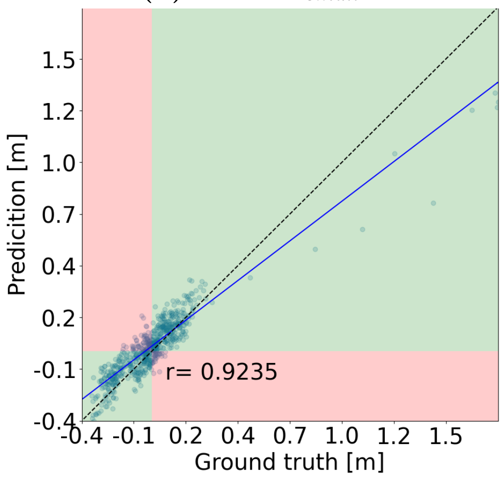
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$



(c) MobileNet$^{0.25}_{small}$

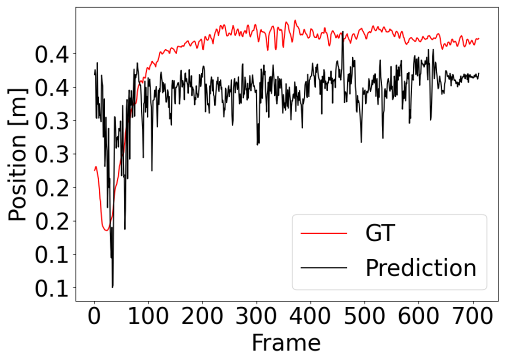


(d) MobileNet$^{1.0}_{small}$

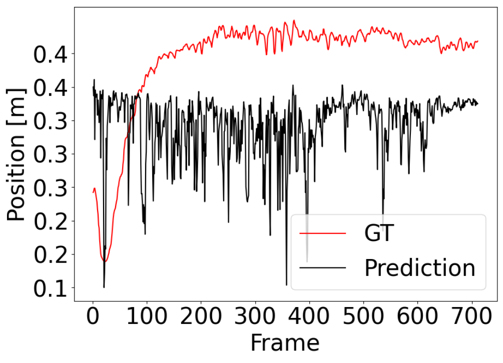
Figure 4.45: GT vs predictions y

Table 4.15: MAE and MSE variable y

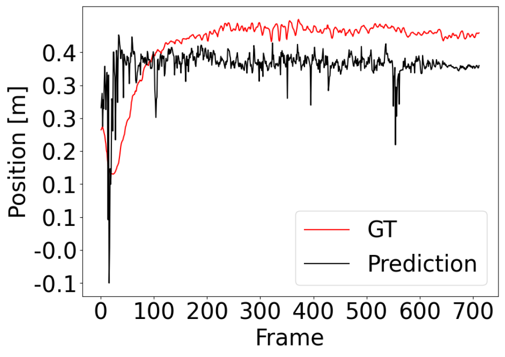
Network	**MAE**	**MSE**
$\mathbf{F}_{seed}$	0.08	0.01
$\mathbf{F}_{small}$	0.08	0.02
$\mathbf{M}^{0.25}_{small}$	0.08	0.01
$\mathbf{M}^{1.0}_{small}$	0.06	0.01



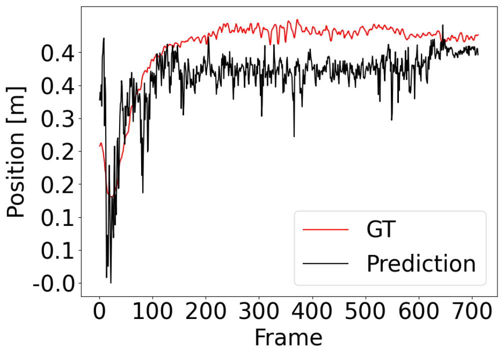
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$

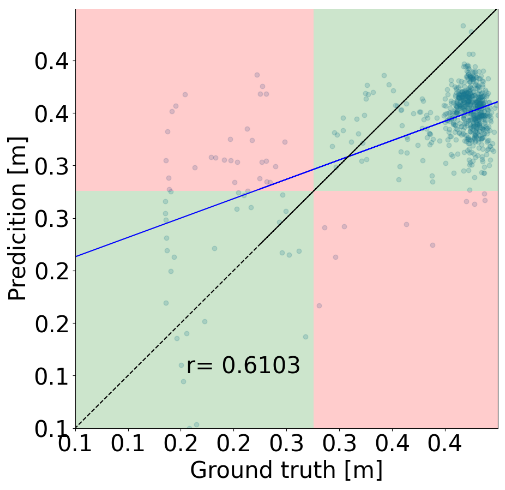


(c) MobileNet$^{0.25}_{small}$

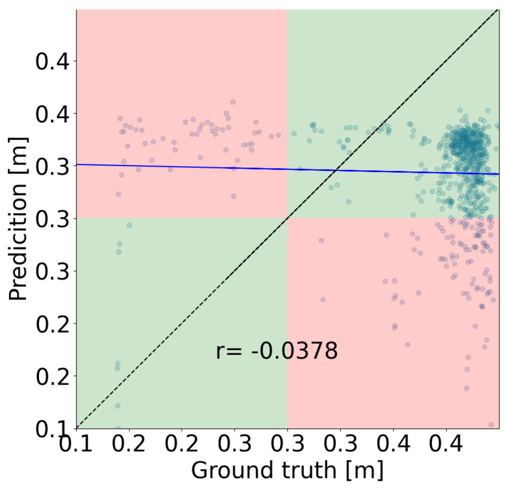


(d) MobileNet$^{1.0}_{small}$

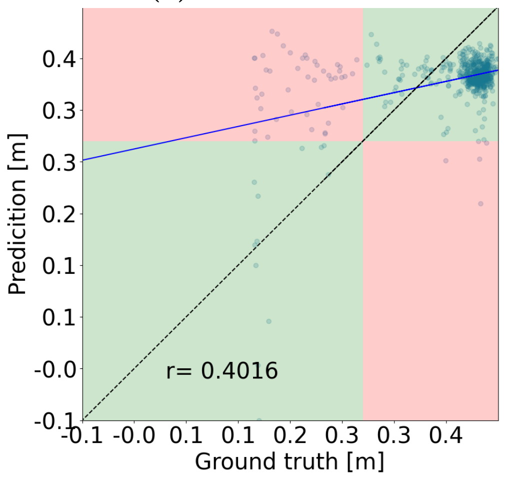
Figure 4.46: z predictions in time with changing network



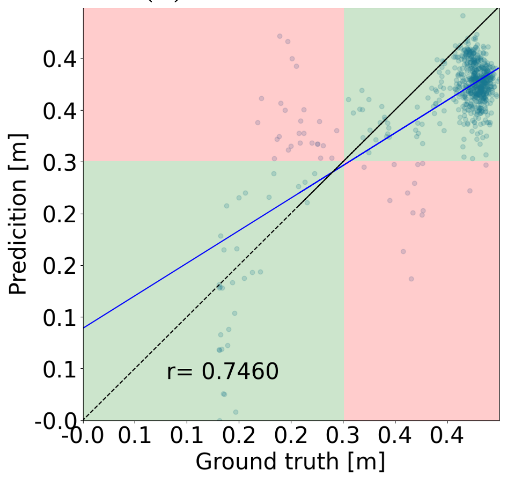
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$



(c) MobileNet$^{0.25}_{small}$

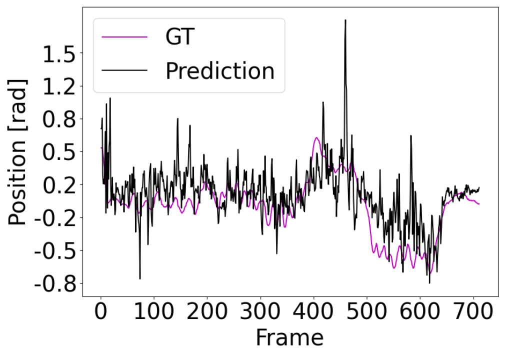


(d) MobileNet$^{1.0}_{small}$

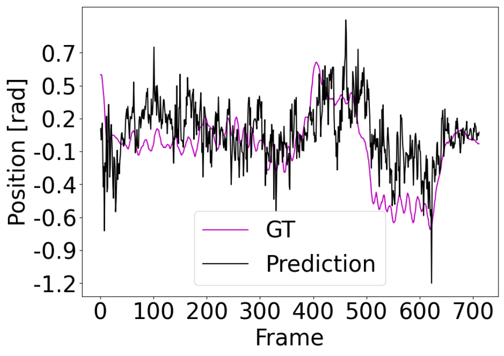
Figure 4.47: GT vs predictions z

Table 4.16: MAE and MSE variable z

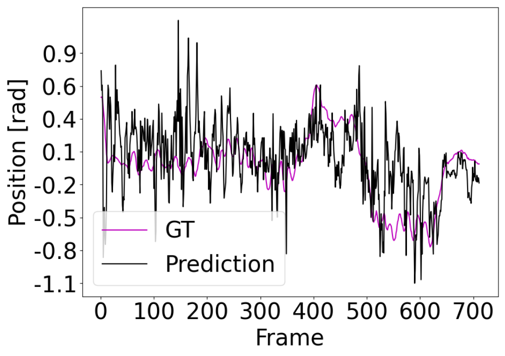
Network	MAE	MSE
$\mathbf{F}_{seed}$	0.08	0.01
$\mathbf{F}_{small}$	0.11	0.02
$\mathbf{M}^{0.25}_{small}$	0.07	0.01
$\mathbf{M}^{1.0}_{small}$	0.06	0.01



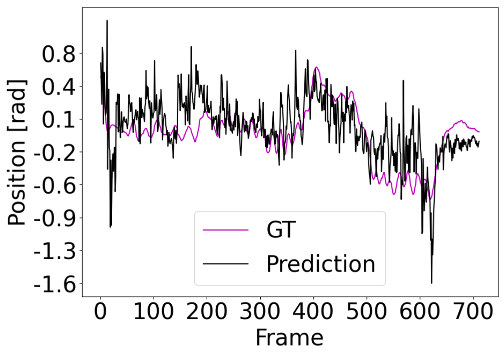
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$

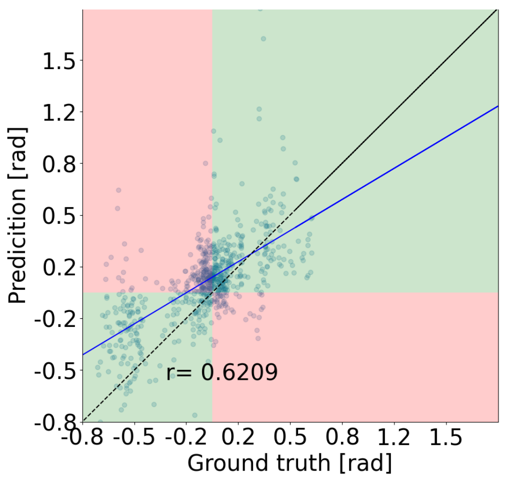


(c) MobileNet$^{0.25}_{small}$

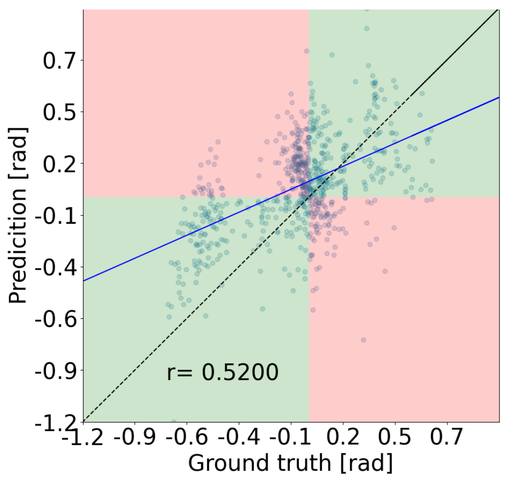


(d) MobileNet$^{1.0}_{small}$

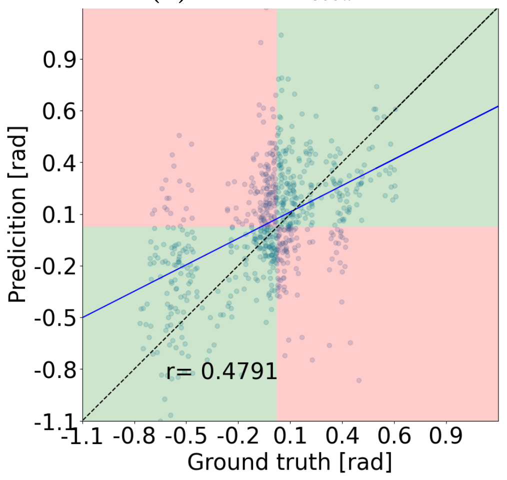
Figure 4.48: ϕ predictions in time with changing network



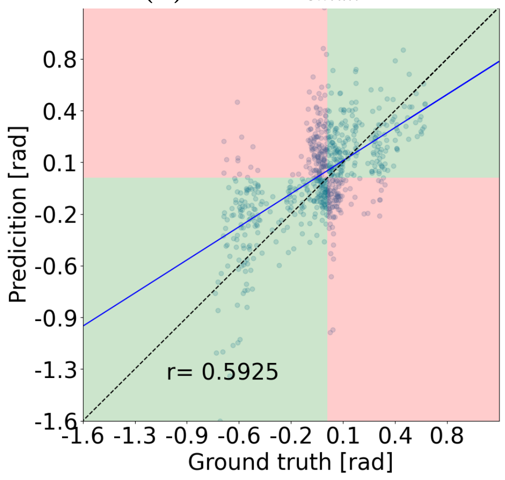
(a) Frontnet$_{seed}$



(b) Frontnet$_{small}$



(c) MobileNet$^{0.25}_{small}$



(d) MobileNet$^{1.0}_{small}$

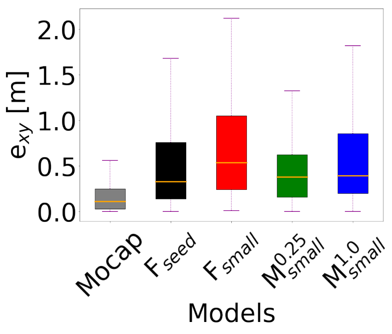
Figure 4.49: GT vs predictions ϕ

Table 4.17: MAE and MSE variable ϕ

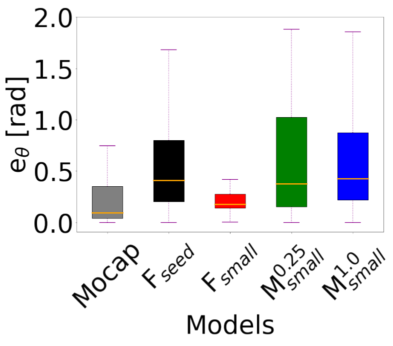
Network	**MAE**	**MSE**
$\mathbf{F}_{seed}$	0.20	0.08
$\mathbf{F}_{small}$	0.23	0.08
$\mathbf{M}^{0.25}_{small}$	0.24	0.09
$\mathbf{M}^{1.0}_{small}$	0.21	0.08

4.3.6 Infield tests summary

The NAS extension proposed in this thesis was able to design two derived networks from MobileNet v1 with width multipliers of 0.25 and 1.0 that track the person for 100% of the whole path in all 4 runs, improving the state of the art network, Frontnet, that follows the target for only 85% of the way. MobileNet$^{0.25}$ based networks achieved lower control errors and MAE, improving by 32% over the state of the art for what concern the former and the ϕ MAE and thus allowing to solve the main issue of Frontnet$_{seed}$. Considering that the test environment was never seen during the training phase, the NAS technique allows an improvement of the generalization capabilities of the networks with a reduction of the number of parameters, memory size and in the case of MobileNet$^{0.25}_{small}$ of cycles to perform the inference.
MobileNet$^{0.25}_{small}$ is the best performing model considering all the aspects as reported in table 4.18, in fact it scores the lowest MAE and Mean pose errors. Furthermore, this network provides the smaller variance on the absolute position error e$_{xy}$ as reported in figure 4.50a. In order to evaluate the contribution of this thesis and express it in a tangible way the difference between the Mean pose error of the mocap and the one from Frontnet$_{seed}$ and MobileNet$^{0.25}_{small}$ have been calculated. These errors can be expressed as the following values $e_{xy-inf-network} = e_{xy-network} - e_{xy-mocap}$ while for the angle error $e_{\theta-inf-network} = e_{\theta-network} - e_{\theta-mocap}$. As a result, $e_{xy-inf-Frontnet_{seed}} = 0.54$, $e_{\theta-inf-Frontnet_{seed}} = 0.57$, $e_{xy-inf-MobileNet^{0.25}} = 0.31$, $e_{\theta-inf-MobileNet^{0.25}} = 0.38$, $e_{xy-inf-MobileNet^{1.0}} = 0.40$ and $e_{\theta-inf-MobileNet^{1.0}} = 0.38$. Considering the inference errors reported above, the improvements obtained through this work can be up to 43% for the distance error and up to 33% for the angle error. These achievements have particular importance since the percentage measures do not consider the completion of the whole path and, they have been obtained in a never seen before environment.



(a)



(b)

Figure 4.50: In-field control errors distribution (lower is better). Boxplot whiskers mark the 5th and 95th percentile of data.

Table 4.18: In-field experiment results (* run that does not end the path)

Network	**Flight time [s]**	**Completed path [%]**	**MAE**			**Mean pose error**	
			x	y	ϕ	e_{xy} [m]	e_θ [rad]
Mocap	165	100	0.0	0.0	0.0	0.18	0.21
$\mathbf{F}_{seed}$	140	85	0.33*	0.12*	0.77*	0.72*	0.78*
$\mathbf{F}_{small}$	58	35	0.81*	0.53*	0.55*	0.65*	0.42*
$\mathbf{M}^{0.25}_{small}$	**165**	**100**	**0.25**	**0.11**	**0.52**	**0.49**	**0.59**
$\mathbf{M}^{1.0}_{small}$	**165**	**100**	0.31	0.13	0.52	0.58	**0.59**

Chapter 5

Conclusions and future works

The work verts on the extension of PIT with the integration of the 2D component allowing to work with 2D convolutions for image recognition tasks and for the sake of this thesis the processing of images to predict the pose of the human. The NAS technology, here designed, should be considered as an enabler for future ML architectures deployment on power/memory constrained on the edge devices. Furthermore, this thesis provides a development pipeline from the design of NN to the deployment onboard a Crazyflie 2.1 nano drone, proving the functionality of the NAS with strict on-field tests and achieving tighter, faster, and more akin to generalize networks that were able to perform the predictions and allow the tracking in a never seen before environment.
Future extensions may include the optimization on other parameters except for the channel number for the NAS that should lead to further optimized networks. Even more, the results obtained with this thesis might be tested on a different environment, setting, and tasks moving the focus from nano drones to smartphones in order to save battery.
Other possibilities to increase the capabilities and tracking performance, as well as the generalization may be brought to life with the integration of further sensors such as a Time of Flight (ToF) multizone. Even more possibilities for generalization in never seen environments may arise with continual learning techniques allowing the nano drone to learn and adapt to new subjects in new locations.
Further work may be done in the direction of data acquisition in order to include more variability on the z-axis thus, allowing the NN to perform correct predictions when a person lowers or with different target heights. The inclusion of more variability on the z-axis should lead the networks to learn and consequently predict the variables for complete control in all directions.

Appendix A

Metrics of evaluation

The use of metrics of comparison for this work is strictly related to the evaluation of models in order to provide a comparable solution and asses the performance of the networks employed. Several metrics have been used, in particular Mean Squared Error, Mean Average Error, R2 score and the regression coefficient, respectively analyzed in section A.1, A.2, A.3 and A.4.

A.1 MSE

The MSE, namely Mean Squared Error, is the sum of the squared distances between the predicted output (y_i) and the true value ($\hat{y}_i$) averaged through all the occurrences of the measure.

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n} \tag{A.1}$$

A.1: Mean Squared Error

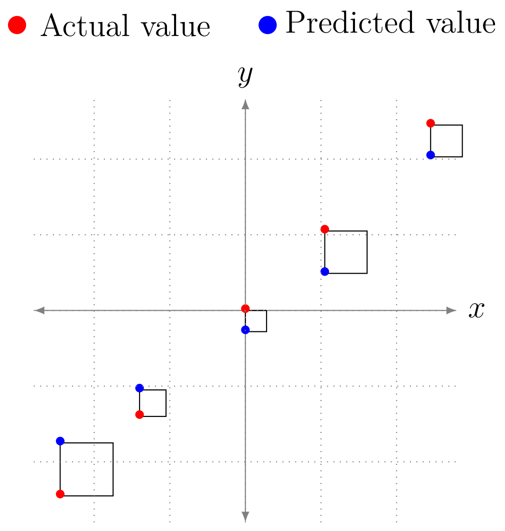


Figure A.1: Representation of the meaning of MSE through area of squares

Visualizing it trough geometry, it can be perceived as the average of the square represented in figure A.1.

A.2 MAE

Mean Average Error, also known with its acronym MAE, is the average of all the distances between predictions and labels. As reported in figure A.2 the MAE may be perceived as the average of all the segments.

$$MAE = \frac{1}{n}\sum(|y_i - \hat{y}_i|) \tag{A.2}$$

A.2: Mean Absolute Error

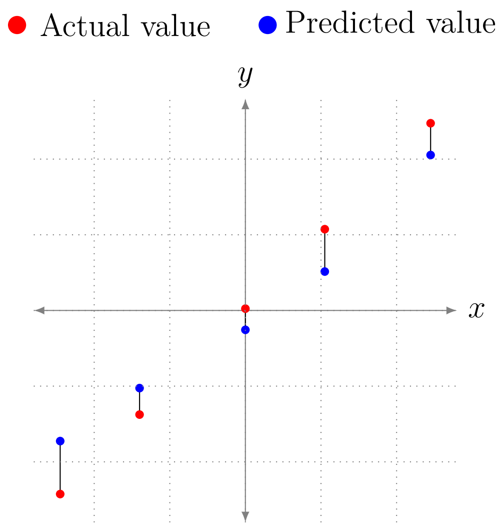


Figure A.2: Representation of the meaning of MAE through segments

A.3 R2

The most reliable measure of performance employed is the R2 score as reported in equation A.3 but, this metric may result in negative values if constant errors are present, even if they are of reduced magnitude.
For completeness the adjusted R2 score is reported in equation A.4, but for the sake of this thesis the only R2 used is the one of equation A.3.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{A.3}$$

where

$$SS_{res} = \sum_i (y_i - f(x_i))^2 \text{ is the sum of residual squares.}$$
$$SS_{tot} = \sum_i (y_i - \overline{y})^2 \text{ is the total sum of squares.}$$

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n-1)}{n-k-1} \tag{A.4}$$

where

k is the number of independent regressors

A.4 The Pearson Coefficient

The Pearson Coefficient determines how much two variables are related. The value of the correlation coefficient, above indicated with r, ranges between +1 and -1. Values near +1 mean high positive correlation, values near 0 represent the absence of correlation and, instead close to -1 mean strong negative correlation.
Table A.1 and figure A.3 report common cases and a classification of them depending on the value of the coefficient r.

$$r = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{A.5}$$

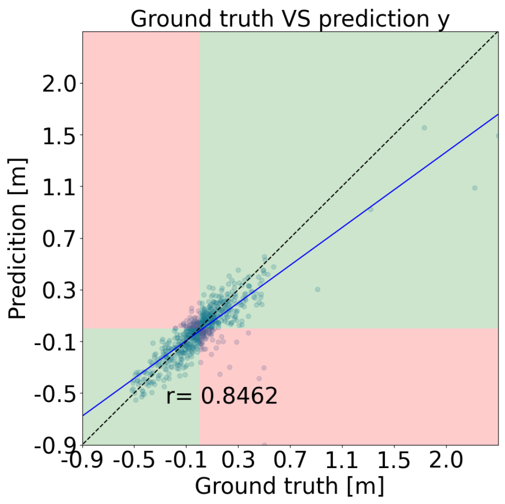
where

n is the number of elements

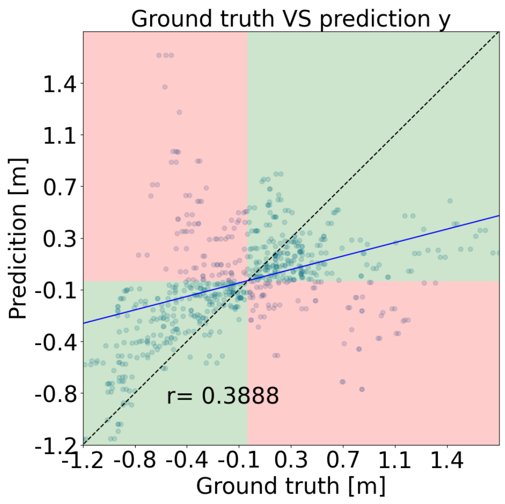
x and y are the variables on which the correlation has to be calculated

Table A.1: Correlation types

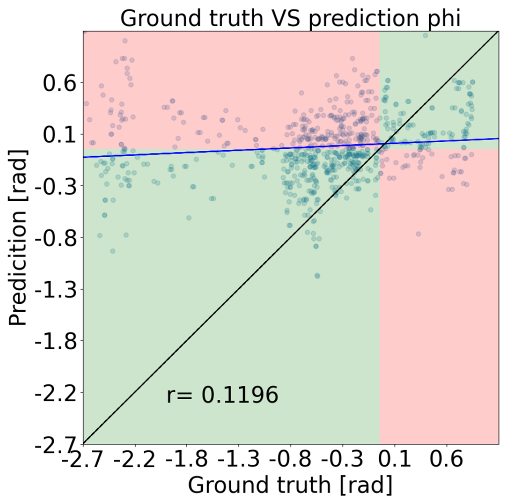
Pearson correlation coefficient	**Value**	**Direction**
$r > 0.5$	Strong	Positive
$0.3 < r < 0.5$	Moderate	Positive
$0 < r < 0.3$	Weak	Positive
0	No correlation	No correlation
$-0.3 < r < 0$	Weak	Negative
$-0.5 < r < -0.3$	Moderate	Negative
$r < -0.5$	Strong	Negative



(a) High positive correlation



(b) Moderate positive correlation



(c) Weak positive correlation

Figure A.3: Correlation cases helpful for this work.

Bibliography

[1] Matteo Risso, Alessio Burrello, Daniele Jahier Pagliari, Francesco Conti, Lorenzo Lamberti, Enrico Macii, Luca Benini, and Massimo Poncino. «Pruning In Time (PIT): A Lightweight Network Architecture Optimizer for Temporal Convolutional Networks». In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Dec. 2021. DOI: 10.1109/dac18074.2021.9586187. URL: https://doi.org/10.1109%2Fdac18074.2021.9586187 (cit. on p. ii).

[2] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. *Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs*. 2021. DOI: 10.48550/ARXIV.2103.10873. URL: https://arxiv.org/abs/2103.10873 (cit. on p. ii).

[3] Peter M. Williams. «Bayesian Regularization and Pruning Using a Laplace Prior». In: *Neural Computation* 7.1 (Jan. 1995), pp. 117–143. ISSN: 0899-7667. DOI: 10.1162/neco.1995.7.1.117. eprint: https://direct.mit.edu/neco/article-pdf/7/1/117/812982/neco.1995.7.1.117.pdf. URL: https://doi.org/10.1162/neco.1995.7.1.117 (cit. on p. 4).

[4] R. Tibshirani. «Regression Shrinkage and Selection via the Lasso». In: *Journal of the Royal Statistical Society: Series B* (1996) (cit. on p. 4).

[5] Yann LeCun, John Denker, and Sara Solla. «Optimal Brain Damage». In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf (cit. on p. 5).

[6] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. *Efficient Neural Architecture Search via Parameter Sharing*. 2018. DOI: 10.48550/ARXIV.1802.03268. URL: https://arxiv.org/abs/1802.03268 (cit. on p. 6).

[7] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. *SMASH: One-Shot Model Architecture Search through HyperNetworks.* 2017. DOI: 10.48550/ARXIV.1708.05344. URL: https://arxiv.org/abs/1708.05344 (cit. on p. 6).

[8] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. «MnasNet: Platform-Aware Neural Architecture Search for Mobile». In: (2018). DOI: 10.48550/ARXIV.1807.11626. URL: https://arxiv.org/abs/1807.11626 (cit. on p. 7).

[9] Han Cai, Ligeng Zhu, and Song Han. *ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware.* 2018. DOI: 10.48550/ARXIV.1812.00332. URL: https://arxiv.org/abs/1812.00332 (cit. on p. 8).

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search.* 2018. DOI: 10.48550/ARXIV.1806.09055. URL: https://arxiv.org/abs/1806.09055 (cit. on p. 9).

[11] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. «Understanding and Simplifying One-Shot Architecture Search». In: *ICML.* 2018 (cit. on p. 9).

[12] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. *MorphNet: Fast and Simple Resource-Constrained Structure Learning of Deep Networks.* 2017. DOI: 10.48550/ARXIV.1711.06798. URL: https://arxiv.org/abs/1711.06798 (cit. on p. 9).

[13] Alvin Wan et al. *FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions.* 2020. DOI: 10.48550/ARXIV.2004.05565. URL: https://arxiv.org/abs/2004.05565 (cit. on p. 11).

[14] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax.* 2016. DOI: 10.48550/ARXIV.1611.01144. URL: https://arxiv.org/abs/1611.01144 (cit. on p. 12).

[15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167 (cit. on p. 16).

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 23).

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. DOI: `10.48550/ARXIV.1704.04861`. URL: `https://arxiv.org/abs/1704.04861` (cit. on p. 25).