POLITECNICO DI TORINO


Master degree course in Data Science and Engineering

Master Degree Thesis

# Adaptive Deep Neural Networks for Human Pose Estimation on Autonomous Nano-drones

**Supervisors**
Prof. Daniele Jahier PAGLIARI
Dr. Daniele PALOSSI
Dr. Alessio BURRELLO
Matteo RISSO
Luca CRUPI

**Candidates**
Mustafa Omer Mohammed Elamin Elshaigi
Student number: 289815

Academic Year 2022-2023

# Abstract

Fully autonomous nano-scale unmanned aerial vehicles (UAVs) represent a specific category of UAVs characterized by dimensions below 10 centimeters and weighing only a few grams. They enable new AI-enhanced smart applications within the Internet of Things (IoT) domain. These UAVs leverage their exceptional speed and maneuverability to rapidly collect data from onboard sensors and devices in the surrounding environment. Due to their tiny form factor and compact size, they are particularly suitable for indoor applications that require safe navigation close to humans. These applications encompass surveillance, monitoring, ambient awareness, and interaction with smart environments. Recent research shows that deploying AI models in nano-drone systems with onboard computations offers several advantages, including reduced operational costs, minimized inference latency for real-time scenarios, and enhanced data security and privacy. However, these small devices face significant constraints in terms of memory and computational resources.

The concept of adaptive inference, which forms the foundation of dynamic networks, has emerged as one of the most attractive research topics over the past years. These networks dynamically adjust their structures and parameters during inference, providing several advantages over static models. These advantages include efficient computational resource allocation by selectively activating network components based on input, improved representation power through data-dependent network architecture or parameters, and addressing the trade-off between accuracy and efficiency when dealing with varying computational budgets.

The objective of this thesis is to apply adaptive inference techniques to construct networks for estimating and maintaining the relative 3D pose of a nano-UAV with respect to a moving person in the environment. The task requires mapping a low-resolution image to the relative pose of the subject, consisting of 3D coordinates (X, Y, Z), and a rotation angle ($\phi$) with respect to the gravity Z-axis. Given the requirement for real-time performance under

strict constraints on computational power and latency, we propose an input-dependent adaptive inference methodology based on the Big/Little model approach. This approach is based on constructing a network of two models, a big accurate model, and a more efficient but less accurate little model, and utilizing a decision function to control the execution of the two models, with the goal of reducing the execution of the big model. By utilizing pre-trained models optimized for the task, our methodology aims to minimize computational cost and inference latency with minimal changes in network regression performance. Our approach has yielded promising results, achieving a reduction of $\approx 0.015$ in Mean Absolute Error ($MAE_{sum}$), along with reductions of around 25% and 29% in inference cycles and NMACs, respectively, by executing the big model only 40% of the time. To further demonstrate the effectiveness of our method, we designed an output decision metric replicating the original Big/Little model approach and showcased our input decision function. We constructed three networks to test each methodology, fixing one big model and alternating the Little model.

Our experimental results show the superiority of our input decision function on MSE compared to randomly selecting the big or little model to be solely executed. It accurately distinguishes between hard and easy samples, assigning them to the big and little models, respectively, with minimal overhead in computing the decision metric. Furthermore, it addressed the limitations of the original Big/Little approach, which involved the default execution of the little model, leading to increased latency and computational cost.

The proposed method demonstrates great potential in terms of generality and flexibility, allowing for a controlled trade-off between network regression performance, inference latency, and computational cost by selecting suitable thresholds and models based on application needs and target devices. By building a network with a fast lightweight little model, we have successfully controlled the number of inference cycles to be below 2 million cycles, which is $\sim 2\times$ faster than executing the big model continuously (i.e. $\approx 3.7$ million cycles). The execution plan ranges from 10% to 50% utilization for the big model, allowing for efficient resource allocation. Importantly, we observed a minimal increase in $MAE_{sum}$ of 5% compared to the regression performance of the static big model.

**Keywords:** Adaptive inference, Dynamic Neural Networks, NAS, Artificial intelligence, autonomous unmanned aerial vehicle (UAV), convolutional neural networks, CNNs, nano-UAV, ultra-low-power, nano-drones, big/little.

# Acknowledgements

I would like to express my sincere gratitude to Professor Daniele PAGLIARI and Co-Professor Dr. Alessio BURRELLO for their invaluable guidance, time, and support throughout my thesis journey. I am deeply thankful to Matto RISSO for his exceptional assistance as well. I would also like to express my appreciation to our colleagues at IDSIA - Istituto Dalle Molle di studi sull'intelligenza artificiale, namely Dr. Daniele PALOSSI and Luca CRUPI, for providing me with the opportunity to collaborate and for their invaluable feedback and support. This experience has been incredibly enlightening, allowing me to gain profound knowledge in various technical and theoretical aspects.

While this should be a joyous occasion, my heart is filled with sorrow for my country and particularly for my family. Unfortunately, they are unable to join me during this significant moment as they have been enduring the hardships of war for the past three months. I sincerely hope for the swift cessation of this war and the safe return of my family. I dedicate this achievement to my family, especially to my brother who has been like a second father to me, and above all to my mother. She is an exceptional woman who defied societal norms in a country that fought against women's right to education. Despite having to pause her academic pursuits after my father's passing to raise us, she earned her master's degree in her fifties and her Ph.D. in her sixties. She has always been my beacon of light and source o inspiration. has taught numerous generations at the top university in my country, motivated solely by appreciation and respect from her students. May God protect her and my homeland.

Lastly, I would like to take a moment to congratulate myself. The journey has been tough, with many challenges to overcome, especially during the challenging times of the COVID pandemic and the ongoing war in my country. Graduating from one of the best universities in Italy, and globally, as an international student from a developing nation was no easy task. I feel proud of my achievement and that I have managed to conquer these obstacles.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

**Human-Drone-Interaction (HDI)** is considered a new emerging subfield of **Human-robot-interaction** with highly attractive research topics. Ongoing research focuses on evaluating and developing new control modalities, enhancing human-drone communication, evaluating interaction distance, and exploring new use cases. Nano-drones, also known as nano-scaled unmanned aerial vehicles (UAVs) [26], have emerged as a novel category of drones characterized by their compact form-factor, measuring just a few $cm^2$. These drones have demonstrated remarkable speed and maneuverability while operating on a low power budget. They possess the capability to swiftly gather data from their onboard sensors as well as a diverse range of deployed devices in the surrounding environment. The integration of advanced smart sensors with AI technology on these versatile nano-drones has proven to be highly suitable within the domain of the Internet of Things (IoT). Moreover, their tiny size makes them exceptionally well-suited for indoor applications that require safe navigation in close proximity to humans. Such applications include surveillance, monitoring, ambient awareness, and interaction with smart environments.

The deployment of AI models in Nano-Drones systems, with on-board computations, offers several advantages, such as reduced operational costs by avoiding data transmission to the cloud, which consumes significant power. On-board computations also enhance data security and data privacy, and most importantly, reduce inference latency, which is crucial for real-time scenarios. However, these advantages come with significant challenges that need to be addressed. These challenges primarily arise from the severe constraints on memory occupation (network size) and memory footprint (quantified as computational cost in terms of NMACs) due to the limited availability of

computational resources. Additionally, in real-time scenarios, the constraint on inference latency necessitates minimizing the time required for dynamic trajectory selection and detection of potential obstacles to prevent collisions.

The concept of adaptive inference has garnered substantial interest in the field of deep learning. It serves as the fundamental principle behind dynamic networks. Unlike static models, which rely on fixed computational graphs and parameters during inference, dynamic networks have the capability to modify their structures or parameters in response to input variations. This adaptability yields numerous benefits, including enhanced accuracy, improved computational efficiency, reduced inference latency, and increased adaptiveness. Consequently, dynamic neural networks have emerged as a promising research area within the field of deep learning.

Our work addresses a vision-based task that aims to predict the pose estimation of a user using low-resolution images, enabling effective **HDI**. The task, proposed by [26], is a multi-output regression problem that estimates and maintains the relative 3D pose of a nano-UAV with respect to a moving person in the environment. The objective is to map a low-resolution image to the relative pose of the subject, represented as a 3D point in space ($X$, $Y$, $Z$), and a rotation angle with respect to the gravity Z-axis ($\phi$).

The traditional approach to autonomous navigation, known as the localization-mapping-planning cycle, is not always suitable for nano-drones operating in crowded cities and dense indoor environments. In such areas, where drones fly at low altitudes and GPS signals are often obstructed or weakened, the traditional approach becomes inadequate, especially in real-time scenarios requiring active exploration of unknown environments, collision avoidance, and mapping. Moreover, this approach is computationally expensive for platforms with limited computational resources, such as commercial nano-drones. Although lighter algorithms based on convolutional neural networks (CNNs) have been proposed for basic reactive navigation of small drones without environment maps, their computational and power requirements still exceed the severely limited resources of nano-drones.

Previous work, such as the Proximity NN [23], addresses the same task of visual human pose estimation. The Proximity NN utilized the GPU of a remote desktop computer in conjunction with a Parrot Bebop 2 quadrotor drone flying near the user and streaming high-resolution (1920×1080 pixels) front-facing images to the remote computer. This allowed the model to estimate the subject's pose relative to the drone, determine the appropriate control input, and send it back to the drone, achieving its control task of staying in front of the user. Building upon this, the authors of PULP-Frontnet

[26] enhanced the work by running the models and computations entirely on board a commercial nano-drone (Crazyflie 2.1). They achieved equivalent drone behavior with significantly reduced computational cost and latency by utilizing low-resolution grayscale images (up to 160×96×1 pixels).

Another approach to the same task was taken by [6], who applied a novel NAS algorithm proposed by [30] to automatically discover multiple Pareto-optimal convolutional neural networks (CNNs) specifically designed for visual pose estimation. They enhanced the methodology and generated a range of models with varying characteristics, including regression performance, model size, number of multiply-accumulate (MAC) operations, and inference cycle. Two seed CNNs, PULP-Frontnet [26] and MobileNetv1 [14], were used in the process.

This thesis work extends the achievements made by PULP-Frontnet [26] and [6], aiming to address the challenges associated with deploying AI models on nano-drones by exploring adaptive inference and dynamic network approaches. Despite significant efforts dedicated to designing various types of dynamic networks, current research does not cover all types of machine learning tasks. To the best of our knowledge, all existing adaptive inference approaches have primarily been developed and tailored for supervised classification or natural language processing tasks. Most techniques have been tested on optimizing large CNN-based models or large RNN-based language models [11]. However, it is not straightforward to adopt the same techniques in regression tasks, where direct indicators of model confidence cannot be extracted using softmax probabilities, as in the case of supervised classification, or dynamic hidden state updates using accumulated halting scores.

This thesis objective is centered around creating a comprehensive methodology that extends the concept of adaptive inference to multi-regression visual pose estimation and can be applied to tiny models deployed on small-edge devices. The aim is to effectively reduce latency and computational cost while maintaining comparable regression performance. In our work, we conducted two experiments. The primary approach involved adopting ConvNets Adaptive Inference Graphs [41]. Although ConvNet-AIG was originally designed for a classification task, we believe its applicability extends to our scenario. It is a data-driven adaptive approach that does not rely on any information derived from the output vector. The ConvNets are constructed by building a high-level architecture similar to residual networks (ResNets [12]), with a gating mechanism embedded within each ConvBlock, enabling decisions on whether to execute the layer or not. Our main contribution, the

second experiment, focused on implementing the Big/Little DNN models approach [27]. This methodology involves building a network consisting of two models with distinct characteristics: a more efficient "little" model with low latency and computation cost but lower accuracy, and a "big" model that offers high accuracy but is computationally demanding with a longer inference cycle. A Decision Function is employed to determine when to execute the larger model. However, since this approach is not directly applicable to our case, we proposed two alternative methodologies: an input decision function based solely on the input inference samples, computed prior to invoking the models, and an output decision function based on the predictions of the Little model. We constructed several networks utilizing pre-trained models that were specifically optimized for the target task using Neural Architecture Search (NAS) [6]. We selected two models based on the MobileNetV1 architecture [14], namely $Msmall_{1.0}$ and $Msmall_{0.25}$, with width multipliers of 1.0 and 0.25, respectively. Additionally, we chose two models, $Fsmall_{22}$ and $Fsmall_{11}$, based on FrontNet [26].

The proposed approach involved maintaining a fixed distinct big model ($Msmall_{1.0}$) while varying the little model among three models with different characteristics: $Msmall_{0.25}$, $Fsmall_{22}$, and $Fsmall_{11}$. The evaluation of our proposed input decision function approach was conducted in two phases. Firstly, we assessed the performance by randomly assigning input frames to either the big or little model. We then performed a similar evaluation for the output decision function approach. Additionally, we compared the performance of our networks to state-of-the-art models, including FrontNet and $Msmall_{1.0}$, both of which were optimized for the specific task at hand. The evaluation metrics considered included the sum of mean absolute error for all regression variables ($MAE_{sum}$), the reduction in latency measured by inference cycles, and the computational cost quantified as the number of multiply and accumulate operations (NMACs).

The evaluation of the proposed input decision function approach was conducted in two phases. Firstly, we assessed the performance of randomly assigning input frames to either the big or little model. We then performed a similar evaluation for the output decision function approach. Additionally, we compared the performance of our networks to state-of-the-art models, including FrontNet and $Msmall_{1.0}$, both optimized for the task at hand. The evaluation metrics encompassed the sum of mean absolute error for all regression variables ($MAE_{sum}$), the reduction in latency measured by inference cycles, and the computational cost quantified as the number of multiply and accumulate operations (NMACs). Utilizing the input decision function

approach, our proposed method achieved ISO error rates comparable to the static big model $Msmall_{1.0}$. Moreover, we exceeded the regression performance of the big static model, resulting in a 0.015 reduction in $MAE_{sum}$. Additionally, we reduced inference cycles and NMACs by approximately 25% and 29%, respectively. Our approach offers flexibility and generality as we can control the execution of the big and little models according to the application's needs through a predefined parameter selection. This allows us to optimize for higher speed, higher accuracy, or lower power consumption by choosing an appropriate little model for deployment.

The remaining chapters of this thesis work are organized as follows: Chapter 2 provides a comprehensive introduction to the theoretical aspects related to our work. It explains the technical aspects of machine learning, deep learning, and learning tasks. Furthermore, it offers a brief overview of the target hardware, Nano Drone applications, resource limitations, and the concept of adaptive inference and dynamic neural networks. Chapter 3 is dedicated to discussing the related work to the thesis, specifically exploring previous approaches in automated **HDI**. It focuses on visual pose estimation and emphasizes the currently available adaptive inference approach. In Chapter 4, we describe the proposed methodologies in detail and provide insights into the reasoning behind their adoption. We also explain the experimental setup, the steps undertaken to conduct each experiment, and the evaluation process and metrics employed. Chapter 5 presents our findings, including a detailed comparison of the results accompanied by visualizations. Finally, in Chapter 6, we provide a conclusion summarizing the key findings of our work. Additionally, we highlight possible enhancements and suggest experiments that can be conducted in the future.

# Chapter 2

# Background

This thesis focuses on leveraging adaptive inference and dynamic neural network techniques to enhance the performance of deep neural networks in addressing the 3-D pose estimation task through onboard computation on nano-drones. This section provides a comprehensive summary of adaptive or dynamic neural networks, encompassing the framework, current research, applications, advantages, and challenges associated with their implementation in specific machine learning tasks. It also provides a brief explanation of machine learning (ML) and deep learning (DL), highlighting the types of learning and task categories. Additionally, the section discusses the characteristics, applications, advantages, and significant challenges related to the target device, nano-sized unmanned aerial vehicles (Nano-UAVs), particularly the limitations imposed by their limited computational resources. Furthermore, it explores the deployment approaches for ML and DL models on edge devices. Subsequently, it explains the utilization of neural architecture search (NAS) techniques to obtain optimized DL network architectures suitable for deployment on such small devices.

This thesis explores the application of adaptive inference methodologies in two previous studies related to nano-sized unmanned aerial vehicles (UAVs). The first study, conducted by [26], addresses the complex task of precise estimation and continuous tracking of the relative three-dimensional (3-D) pose of a UAV in relation to a human subject. The authors introduced PULP-Frontnet, a novel Convolutional Neural Network (CNN) that visually estimates the pose of a freely-moving human subject and successfully performs onboard computations, achieving remarkable regression performance. Building upon the work of [26], further advancements were made by [6], who leverage a novel neural architecture search (NAS) technique to automatically

identify several Pareto-optimal convolutional neural networks (CNNs) for the same task. Their approach surpasses the performance of PULP-Frontnet by significantly reducing in-field error and lowering latency and computational costs. This thesis focuses on achieving additional enhancements by optimizing inference latency and reducing Multiply and Accumulate (MAC) operations while maintaining comparable performance. These enhancements are crucial due to the hardware constraints imposed by limited memory usage, minimal latency, and energy consumption.

## 2.1 Machine learning and Deep learning

As stated by [34], Machine Learning (ML) encompasses the automated identification of meaningful patterns from historical data to enhance future performance. ML primarily focuses on the development of autonomous learning techniques, wherein algorithms can be modified or improved based on past experiences represented by data, without the need for external human intervention. The ultimate objective of this process is to create a computer program capable of successfully executing a given task by leveraging the knowledge acquired from input data. ML has found widespread applications across various domains, yielding cutting-edge outcomes in tasks that are too intricate for traditional algorithms or require adaptation to dynamic environmental changes. A range of ML algorithms exists, including Decision Trees, Random Forests, Support Vector Machines, regressions, K-Nearest Neighbors, and Neural Networks, among others. These algorithms generally demonstrate superior performance in scenarios with small datasets and well-defined features, or in situations where computational resources are limited. Nevertheless, [24] noted certain limitations of these models, emphasizing their performance's dependency on data quality and representational power. Even an advanced and complex machine learner is likely to suffer reduced performance with inadequate data representation, while a simpler machine learner can achieve high performance with well-crafted features. Consequently, feature engineering plays a vital role in addressing these challenges by constructing features and data representations from raw data. Feature engineering constitutes a significant portion of the effort involved in a machine learning task, as it is often domain-specific and requires a considerable amount of human input and time for data preparation, feature extraction, and feature engineering.

Deep learning, as illustrated in Figure 2.1, represents a subset of machine

learning techniques that leverage artificial neural networks to address intricate problems. Central to deep learning algorithms is the automation of extracting meaningful representations or abstractions from data. The hierarchical structure of deep learning algorithms is inspired by the human brain's neocortex, specifically the deep, layered learning process observed in primary sensorial areas. This emulation allows deep learning algorithms to autonomously extract features and abstractions from the input data. These algorithms demonstrate notable advantages when learning from extensive amounts of unlabeled data, often acquiring data representations in a layer-wise manner through a greedy approach.

While deep learning algorithms facilitate automated feature extraction, enabling researchers to obtain discriminative features with minimal domain knowledge and human effort, designing an optimal architecture for a specific task is nontrivial. Determining the ideal number and type of layers, selecting the appropriate optimizer, and engaging in hyperparameter tuning are among the challenges that arise in architectural design. The shift in focus from feature engineering, as seen in classical machine learning, to model architecture design challenge in deep learning [24].

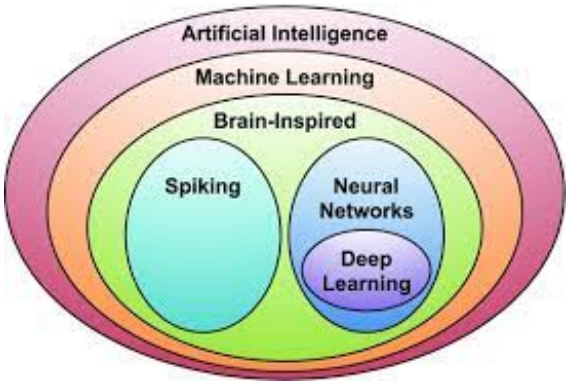



Figure 2.1: Deep learning in the context of artificial intelligence [3]

### 2.1.1 Types of learning

ML can be divided into different learning paradigms that aim to describe the relationship between the learner algorithm and the environment, there exist several ML approaches as indicated in [31]. We will discuss briefly the two major used approaches which are Supervised and unsupervised learning. Machine learning encompasses various learning paradigms, each seeking to describe the interplay between the learner algorithm and the environment.

Fig.2.2 encapsulates the different ML approaches available. In this discussion, we will provide an overview of the two primary approaches commonly employed: supervised and unsupervised learning.

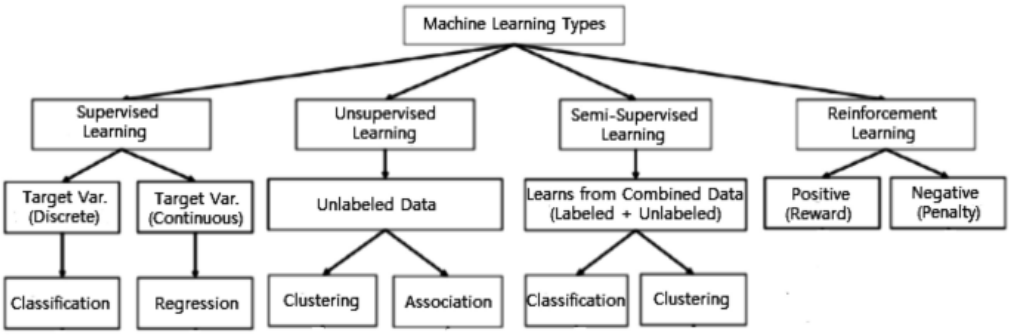



Figure 2.2: ML/DL fields segmented by task [31]

### Supervised learning

Supervised learning [34] is a method based on the idea of learning from examples represented by training data where the outputs are already known, The goal of the algorithm is to learn the relationships between the inputs and outputs and then make predictions on a separate, unseen dataset where the outputs are unknown called test data.

Formally the train data are composed of a set of $n$ ordered pairs $(x_1, y_1)$, $(x_2, y_2), \ldots, (x_n, y_n)$ where $x_i$ represents a set of measurements for a single example, and $y_i$ is the corresponding label. The test data are composed of $m$ examples $(x_{n+1}, x_{n+2}, \ldots, x_{n+m})$, and the goal of the algorithm is to predict their corresponding labels $(y_{n+1}, y_{n+2}, \ldots, y_{n+m})$.

More detailed information about Supervised learning, its applications, and algorithms can be found in [34].

### Unsupervised learning

Unsupervised learning [3] is an approach that enables the learning process to be implemented in the absence of labeled data, thereby eliminating the need for explicit labels. In this context, the agent acquires knowledge of significant features or internal representations necessary to uncover unknown structures or relationships within the input data. Techniques such as generative networks, dimensionality reduction, and clustering are commonly classified under unsupervised learning. Several members of the DL family, such

as restricted Boltzmann machines, autoencoders, and Generative Adversarial Networks (GANs), have exhibited notable performance in nonlinear dimensionality reduction and clustering tasks. Additionally, Recurrent Neural Networks (RNNs), including approaches like Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM), have found applications in unsupervised learning across various domains. It is important to note that unsupervised learning has limitations, such as the inability to provide precise data categorization and its computational complexity. For example, clustering algorithms like k-means or hierarchical clustering require iterative processes to group data points, which can be computationally intensive. However, the main reason is the lack of labeled data, which requires unsupervised learning algorithms to process a larger amount of data compared to supervised learning algorithms. This increased volume of data necessitates more computational resources and time. Among the popular unsupervised learning techniques, clustering stands out as one of the most widely employed methods.

### 2.1.2 Machine Learing Tasks

Machine learning enables us to address challenging tasks that are beyond the capabilities of fixed programs created by human programmers. As stated in [10], the essence of learning lies not in the learning process itself, but rather in acquiring the capability to accomplish a specific task. For example, if our objective is to enable a robot to walk, then walking becomes the task at hand.

Presently, machine learning finds application in a diverse array of tasks. Within the field of machine learning, the primary tasks encompass the following:

#### Classification Tasks:

In this specific task, the computer program is assigned the responsibility of determining the category to which a given input belongs from a set of $k$ categories. To address this task, the learning algorithm aims to generate a function denoted as $f : \mathbf{R}^n \rightarrow 1, ...., k$. In this context, when $y = f(x)$, the model assigns an input, represented by the vector $x$, to a specific category indicated by the corresponding numeric code $y$. Various variations of the classification task exist, including cases where the function $f$ outputs a probability distribution across multiple classes [10].

An illustrative example of a classification task is binary image classification, which is particularly relevant in the field of medical devices, such as the detection of diabetic retinopathy from retinal fundus images. The primary goal is to accurately classify retinal fundus images as either exhibiting signs of diabetic retinopathy or being considered normal [39].

**Regression Tasks:**

In this type of task[10]., the computer program is asked to predict a numerical value given some input. To solve this task, the learning algorithm is asked to output a function $f : \mathbf{R}^n \rightarrow \mathbf{R}$. When $y = f(x)$. This type of task is similar to classification, except that the format of the output is different. An example of a regression task is The task of PPG (Photoplethysmography) heart rate monitoring using regression models involves predicting the numerical value of a person's heart rate based on the PPG signal obtained from a sensor on a wrest-worn device [5].

### 2.1.3 Convolutional Neural Networks CNN

Convolutional Neural Networks (CNNs) represent a specialized type of neural network designed specifically for processing data that exhibits a known, grid-like topology. Figure 2.3, illustrates a typical architecture employed in CNN-based models. CNNs have been applied across various domains, encompassing time-series data, which can be regarded as a one-dimensional grid with regularly spaced samples, and image data, which can be perceived as a two-dimensional grid composed of pixels. The designation "convolutional neural network" signifies the utilization of the convolution operation as a fundamental operation within the network's architecture [10].

The utilization of **sparse interactions**, **parameter sharing**, and **equivariant representations** constitutes a significant enhancement strategy for machine learning systems. These key features provide CNNs with several advantages over conventional feed-forward neural networks (DNNs). The most popular and extensive use of parameter sharing occurs in CNNs applied to computer vision. Natural images exhibit statistical properties that are invariant to translation. For example, a photograph of a cat remains a cat photo even if it is shifted one pixel to the right. CNNs take advantage of this property by sharing parameters across multiple image locations. The same feature, represented by a hidden unit with the same weights, is computed over different locations in the input. This approach dramatically reduces

the number of unique model parameters and allows for larger network sizes without requiring a proportional increase in training data [10].

Notably, CNNs exhibit a closer resemblance to the human visual processing system, displaying high optimization for processing two-dimensional (2D) and three-dimensional (3D) images, as well as efficacy in learning and extracting abstractions of 2D features. The max pooling layer in CNNs effectively handles shape variations. Furthermore, CNNs employ sparse connections and tied weights, resulting in a significantly reduced parameter count compared to equivalently sized fully connected networks. Additionally, CNNs trained with gradient-based learning algorithms encounter fewer issues associated with diminishing gradients. Due to the direct minimization of an error criterion by the gradient-based algorithm, CNNs are capable of generating highly optimized weights [1].

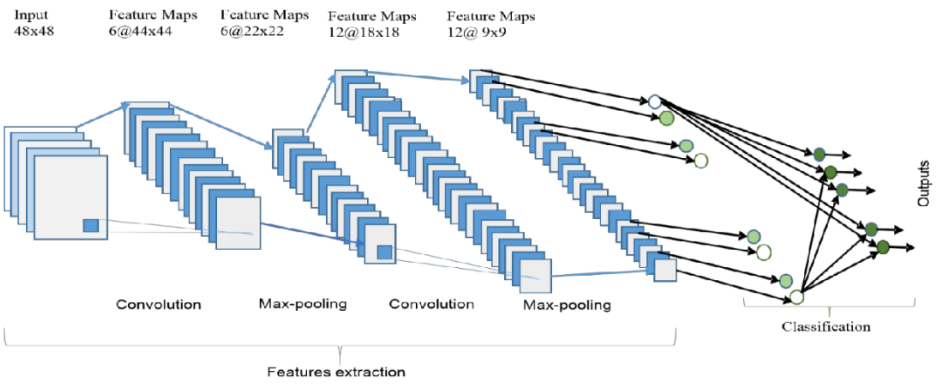



Figure 2.3: Typical architecture of the CNN-based model includes an input layer, multiple convolutions and max-pooling layers, a fully-connected layer, and a classification layer [1]

## 2.2 Unmanned aerial vehicles (UAVs)

The past decade has witnessed remarkable growth in the utilization of Unmanned Aerial Vehicles (UAVs) across diverse domains. The Federal Aviation Administration (FAA) report [28] predicts that the number of UAVs will exceed 3.2 million flying units by 2022. The integration of artificial intelligence (AI) algorithms into UAVs has emerged as a significant advancement in the realm of the Internet of Things (IoT). Numerous applications have been explored in various fields, encompassing search and rescue operations, human-drone interaction (HDI), precision agriculture [33], monitoring and

transportation tasks, provision of network and cellular coverage, enhancement of ground network connectivity, coordination with ground vehicles, and data collection from ground sensors [2]. However, the majority of these systems are predominantly reliant on standard- or microsized UAVs.

Nano-UAVs are an emerging class of aircraft distinguished by their compact size (subten centimeters), lightweight (a few tens of grams), and limited power requirements (sub-Watt). Figure 2.4 illustrates a size-based categorization for some of the most prevalent small drones. These UAVs possess the capability to perform onboard data analytics, enabling them to selectively identify pertinent information for transmission to the IoT backbone. Consequently, they have the potential to bridge the gap between costly, weather-dependent, and low-resolution satellite imagery and ground-based imagery, which is confined to human-level perspectives and the availability of accessible roads. Leveraging specialized aerial cameras and cloud-based data analytics, farmers can continually monitor the quality of crop growth. Moreover, transportation drones capable of secure takeoff and landing in close proximity to buildings unleash the full potential of e-commerce telecommunication infrastructure [9].

Nano-UAVs systems encompass various components[26], including sensors like cameras, which capture information-rich data but pose interpretation challenges. Consequently, the UAVs must tackle complex perception tasks onboard to enable autonomous operation. However, a notable limitation arises in the onboard computing capability of nano-aircraft, which has traditionally been confined to basic microcontroller units (MCUs). These MCUs provide a computational capacity of merely a few hundred million operations per second (MOp/s), proving inadequate to meet the real-time requirements of cutting-edge perception and navigation algorithms employed in state-of-the-art (SoA) systems. The primary challenges associated with the adoption of these Nano-Drones systems have been concisely summarized in [3] as follows:

- The minimum real-time frame rate is required to select a new trajectory on the fly or to detect a suspected obstacle in time to prevent a potential collision, which is a crucial point in our work for real-time pose estimation tasks.

- Maintaining the quality of results while utilizing an embedded ultralow-power low-resolution camera.

- The crucial need for a strategy aimed at reducing the memory footprint

and computational load to more easily fit within the available resources while exploiting the architectural parallelism at best to meet the real-time constraint.


Figure 2.4: UAVs categories by size

### 2.2.1 Bitcraze Crazyflie 2.1

To have a full picture of the restricted resource constraints we have to deal with, we will briefly describe some of the characteristics of the system under investigation within the scope of our research. Fig.2.5 shows a sample of the Nano-Drone, and described in the data sheet crazyflie 2.1 Data Sheet :

- ***Onboard microcontrollers*** *:*
  A. STM32F405 main application MCU (Cortex-M4, 168MHz, **192kb** SRAM, **1Mb** flash)
  B. nRF51822 radio and power management MCU (Cortex-M0, 32Mhz, 16kb SRAM, **128kb** flash)
  C. micro-USB connector
  D. On-board LiPo charger with 100mA, 500mA and 980mA modes available
  E. Full-speed USB device interface
  F. Partial USB OTG capability (USB OTG present but no 5V output) 8KB EEPROM
- ***Mechanical specifications*** *:*
  A. Takeoff weight: 27g
  B. Size (WxHxD): 92x92x29 mm (motor-to-motor and including motor mount feet)

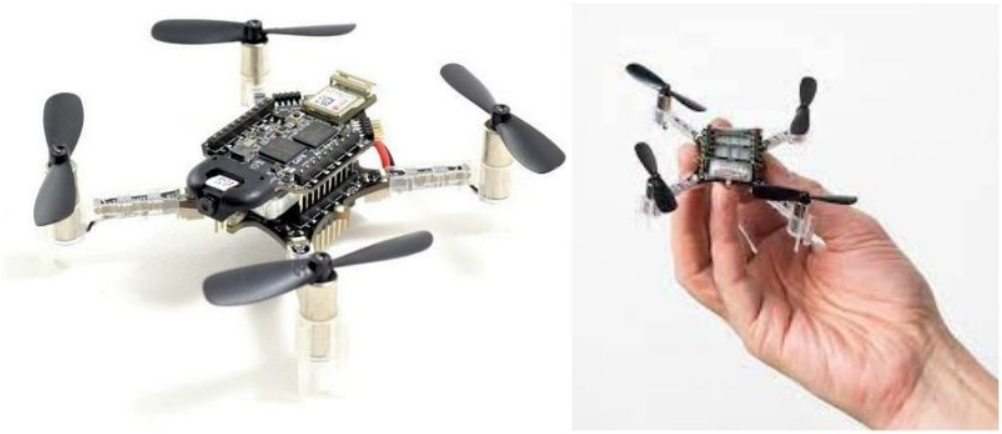


Figure 2.5: BitCraze Crazyflie 2.1 Drone Sample

### 2.2.2 PULP architecture and GAP8 System-on-Chip

The "PULP" (Parallel Ultra-Low Power) processing paradigm has emerged as a promising solution in the pursuit of enhanced performance and energy efficiency for low-power edge devices, attracting attention from both industry and academia. PULP computers combine a traditional Microcontroller Unit (MCU), designed for I/O-centric operations, with a programmable general-purpose accelerator dedicated to executing data-parallel computational kernels [8]. These computational kernels often involve fundamental linear algebra operations, which play a critical role in various artificial intelligence applications. By integrating these components within a single System-on-Chip (SoC), PULP computers provide a comprehensive solution to meet the demanding requirements of performance and energy efficiency in low-power edge devices.

The PULP paradigm, specifically the commercial implementation known as the GAP8 System-on-Chip (SoC) by GreenWaves Technologies, represents the brain of the nano-drone Bitcraze Crazyflie 2.1, as depicted in figure 2.6. The GAP8 SoC incorporates a total of nine identical RISC-V cores, with one core designated as the fabric controller (FC) that serves as the primary core within the Microcontroller Unit (MCU). The remaining eight cores are employed to construct a parallel general-purpose programmable accelerator, forming the cluster (CL).

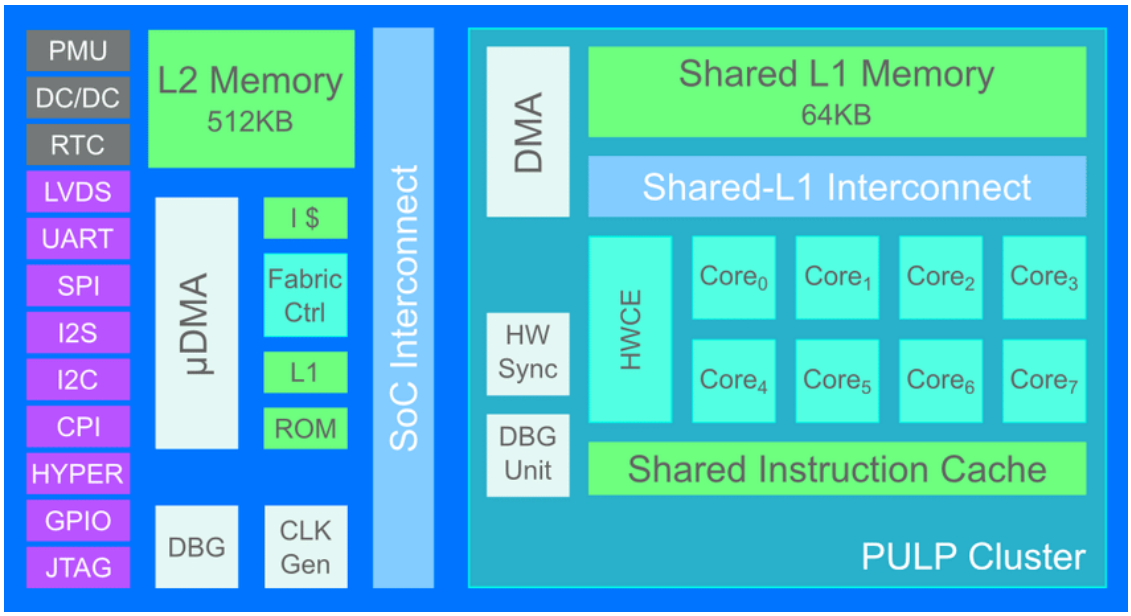



Figure 2.6: GAP8 System-on-Chip architecture [26]

The key characteristics of the novel design of GAP8 by GreenWaves Technologies, are outlined as follows :

A - **MCU class energy consumption**

- Highly efficient parallelization
- Sophisticated architecture (including instruction set architecture extensions)
- Explicit memory movement

B - **Agility**

- Fine-grained compute / energy scaling
- Ultra fast state transitions

C - **Programmability**

- Applicable to many real-world problems – not just CNNs
- Exploits fast evolution of state-of-the-art
- Single code model across architecture

## 2.3 Adaptive Neural Network

Adaptive or Dynamic Neural Networks are an emerging research topic in deep learning [11]. In various industrial fields, particularly in Robotics and IoT, the focus extends beyond accuracy to optimizing efficiency in computations, power management, inference latency, memory allocation, and representation power.

Static models have played a crucial role in diverse areas, including computer vision (CV), natural language processing (NLP), and the Internet of Things (IoT). In recent years, significant advancements have been made with successful deep models such as AlexNet [19], VGG [35], GoogleNet [36], ResNet [12], and Transformers [40]. These architectural innovations have facilitated the training of deeper and more accurate. Furthermore, research on neural architecture search (NAS) [4], [22] has accelerated the process of designing more powerful structures. However, most prevailing deep learning models perform inference in a static manner, where both the computational graph and network parameters are fixed after training. This static approach may limit their representation power, efficiency, and interpretability.

In contrast, dynamic networks have the ability to adapt their structures or parameters based on the input during inference, offering advantageous properties absent in static models. Some of these properties include:

- ***Efficiency*** Dynamic networks can allocate computations on demand, by selectively activating model components (e.g., layers, channels, or sub-networks) conditioned on the input.

- ***Representation*** Power Dynamic networks have significantly enlarged parameter space and improved representation power due to the data-dependent network architecture or parameters.

- ***Adaptivenes*** Dynamic models can trade-off between accuracy and efficiency for dealing with varying computational budgets on the fly.

- ***Compatibility*** Dynamic networks are compatible with the most recent advanced state-of-art optimization techniques such as architectural innovations in lightweight models, NAS approaches or acceleration methods developed for static models to boost their efficiency further, such as network pruning, weight quantization, knowledge distillation, and low-rank approximation.

- ***Generality*** Many dynamic models are general approaches that can be applied seamlessly to a wide range of applications, such as image classification, object detection, and semantic segmentation. Moreover, the techniques developed in CV tasks are proven to transfer well to language models in NLP tasks, and vice versa.

In general, the existing adaptive approaches can be categorized into 3 main categories as mentioned in [11] :

- ***sample-wise*** Aiming at processing different inputs in data-dependent manners, sample-wise dynamic networks are typically designed from two perspectives:

  A. adjusting model architectures to allocate appropriate computation based on each sample, and therefore reducing redundant computation for increased efficiency.

  B. adapting network parameters to every input sample with fixed computational graphs, with the goal of boosting the representation power with minimal increase in computational cost.

- ***spatial-wise*** spatially dynamic computation has great potential for reducing computational redundancy. In other words, making a correct prediction may only require processing a fraction of pixels or regions with an adaptive amount of computation. Moreover, based on the observations that low-resolution representations are sufficient to yield decent performance for most inputs.

- ***temporal-wise*** Adaptive computation can be performed along the temporal dimension of sequential data, such as texts (e.g., dynamic Recurrent Neural Networks (RNNs)) and videos (e.g., Temporal Convolutional Networks (TCNs)). Network efficiency can be improved by dynamically allocating fewer or no computations to unimportant temporal locations in the inputs.

### 2.3.1 Decision Making of Adaptive Networks

During the inference process, dynamic networks demonstrate the capacity to make data-dependent decisions that allow for the alteration of their architectures and parameters. Additionally, they can identify and select significant spatial or temporal regions within the input data. In their work, Han et al[11] provided a comprehensive summary of three frequently observed decision-making schemes in dynamic networks.

**Confidence-Based Criteria :**

Several dynamic networks ([15], [45]) have the capability to generate "easy" samples at early exits by satisfying specific confidence-based criteria. These methods generally involve estimating the confidence of intermediate predictions to be compared to a predefined threshold for decision-making purposes.

In **classification tasks**, the representation of confidence often relies on selecting the maximum element from the *SoftMax* output [15]. Alternatively, *Entropy-based* criteria have been explored, as demonstrated in the BranchyNet approach [37]. Another approach, known as the **Big/Little** models approach, incorporates the use of *score margin* [27]). The Big/Little DNN architecture, as illustrated in Fig2.7, consists of a more efficient but less accurate little DNN and a full-fledged big DNN. The main objective of this architecture is to minimize energy consumption by avoiding the execution of the big DNN whenever possible. The little DNN is initially executed for inference, and its result is directly used as the final inference result if it is deemed accurate. However, if the result from the little DNN is considered inaccurate, the big DNN is then executed to produce the final inference result.

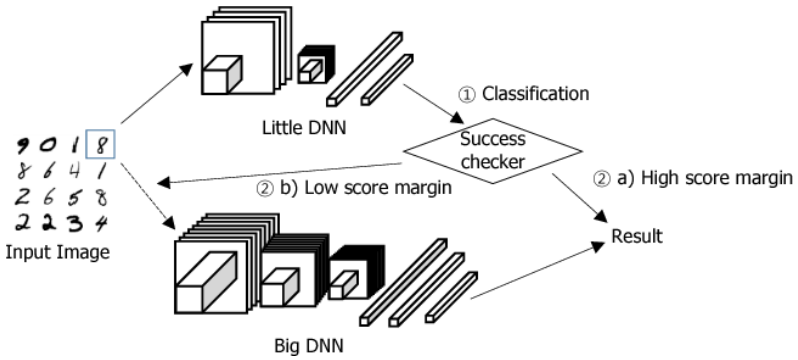



Figure 2.7: big/LITTLE DNN architecture [27]

Empirically, confidence-based criteria offer a straightforward implementation and typically do not necessitate specific training techniques. By manipulating the thresholds, a trade-off between accuracy and efficiency can be managed, with these thresholds often being tuned using a validation dataset. It is important to note, however, that the issue of overconfidence in deep models may impact the effectiveness of this decision paradigm. Specifically, when incorrectly classified samples receive high confidence scores at early exits, it can compromise the reliability of the decision-making process.

### 2.3.2 Policy Adaptive Networks

The network setup comprises a *SuperNet* and a *policyNet*, where the latter adapts the network topology based on different input samples. The *policyNet* processes each input sample and determines the activation of specific parts within the main network. This approach, known as Dynamic Routing, is commonly employed in a Mixture of Experts (ME) frameworks. ME is a popular and promising method for combining multiple neural network experts, where The experts are supervised by a gating network, as demonstrated in the works like in [21], [44]. The scheme has limitations including the lack of adaptability in policy networks, which are tailored for specific backbones and may not easily accommodate diverse architectures. Moreover, the substantial size of SuperNets (experts) makes them unsuitable for deployment on edge devices. Additionally, utilizing a neural network as a policy network can have negative implications on inference time and energy consumption due to computational constraints.

### 2.3.3 Adaptive Network with Gating Functions:

The gating function serves as a versatile and adaptable method for decision-making in dynamic networks. It can be easily integrated as a module within any backbone network at various locations. During inference, each module governs the local inference graph of a layer or block. By utilizing intermediate features, the gating functions efficiently generate binary-valued gate vectors to determine one of the following:

A- the activation of channels ([13], [7]).

B- the skipping of specific layers ([42], [41])

C- the selection of paths in a SuperNet ([20])

D- the allocation of computations to specific locations in the input ([18])

An example of an adaptive inference graph Network with Gating Functions is shown in Fig.2.8. In comparison to other decision policies, the gating functions exhibit remarkable versatility and applicability. However, their lack of differentiability necessitates specific training techniques, which will be discussed in subsequent sections.

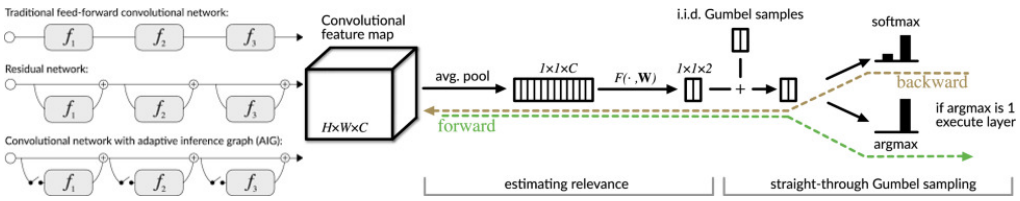



Figure 2.8: ConvNet-AIG taken from [41]
ConvNet-AIG (bottom left) follows a high-level structure similar to ResNets (center left ) by introducing identity skip-connections that bypass each layer. The key difference is that for each layer, a gate determines whether to execute or skip the layer. This enables individual inference graphs conditioned on the input

## 2.4 Neural Architecture Search (NAS) :

NAS tools play a pivotal role in the design phase of Deep Neural Networks (DNNs) by providing automated assistance in exploring an extensive architectural space encompassing various combinations of layers and hyperparameters. These tools are particularly significant on platforms with limited resources, aiming to minimize an objective function that encompasses both task-specific accuracy and non-functional cost metrics, including memory usage, latency, and energy consumption. An essential prerequisite in this domain involves addressing the strict memory limitations of edge devices, typically employing Microcontrollers (MCUs) with restricted Flash and RAM capacity. The primary focus revolves around achieving an optimal memory footprint for DL models, ensuring compliance with these constraints. Simultaneously, efforts are directed toward minimizing energy consumption by primarily reducing the overall number of operations (OPs) per prediction.

Early NAS algorithms primarily relied on evolutionary algorithms (EA) and reinforcement learning (RL) methodologies. These approaches provided the capability to navigate arbitrary search spaces and optimize any given cost function through an iterative process involving network sampling, training until convergence, performance evaluation, and subsequent utilization of the obtained information for guiding subsequent sampling. Nonetheless, this approach had notable limitations pertaining to computational resources and excessive time requirements, often amounting to 1000s of GPU hours. The design space itself often encompasses a staggering number exceeding $10^8$ potential architectures, thereby rendering manual design choices suboptimal and challenging to scale.

A progressive advancement in NAS is observed with the emergence of

differentiable neural architecture search (DNAS), which involves exploring a supergraph containing all potential architectures and selecting a singular path as the final neural network. DNAS exhibits the ability to explore extensive combinatorial spaces within the time required to train a single model. However, recent research by [46] highlighted additional limitations concerning ***memory costs*** that constrain the search space due to the necessity of storing feature maps in GPU memory during training. Furthermore, it was revealed that the cost associated with these memory requirements ***grows linearly*** with the number of options available per layer.

The recent study conducted by [30] has shed light on the limitations of previous differentiable neural architecture search (DNAS) approaches. It was revealed that existing DNAS methods primarily focus on optimizing either the model size or the number of operations (OPs) separately. However, from the perspective of a designer, the key concern lies in minimizing energy consumption (OPs) while adhering to a given memory constraint. To address this issue, the authors proposed a novel problem formulation that can be applied to any DNAS framework. This formulation enables the identification of a set of Pareto-optimal architectures within the accuracy versus OPs space while considering a fixed model size constraint. By introducing this novel approach, the authors have made significant progress in addressing the core challenges related to energy optimization in DNAS.

# Chapter 3

# Related work

Adaptive Neural Networks have become a very attractive field of research in the past few years, The vast increase in the amount of data created by the sensors is accelerating the emergence of the edge computing paradigm [29], we can summarize its advantages as follows :

- **Improve System Performance**: Edge computing in IoT can achieve ms level of data processing. resulting in an efficient reduction of the overall delay of the system.

- **Protect Data Security and Privacy**: Cloud platform service utilizes centralized data security protection solutions. However, serious consequences can occur if the data get leaked. on the contrary, Edge computing in IoT allows enterprises to deploy security solutions locally, reducing the risk of data leakage during transmission and the data volume stored in the cloud platform, so as to minimize security and privacy risks.

- **Reduce Operational Costs** Edge computing in IoT can reduce data uploading volume, thereby reducing data migration volume, bandwidth consumption, energy consumption, and latency. More specifically in computer vision tasks (CV), sending images to the cloud is an extremely costly process.

- **lower latency with more accurate etimations** : for real-time applications. Wireless WAN links (4G, LoRA, etc.) have significant Round-Trip Times (RTTs). Additionally, they can be unstable or unavailable in some places. In edge computing, since all computations are performed

on board, the latency for all phases of the inference pipeline can be measured accurately.

The last two points are the most crucial in our research. Our work aims to utilize the adaptive neural network framework to Optimize the onboard Computational cost and inference latency.

**Human-Drone-Interaction** (**HDI**) considered as a subfield of the **Human-robot interaction** and can be defined as "the study field focused on understanding, designing, and evaluating drone systems for use by or with human users" [38]. Despite the common characteristics between the two fields, the drone's unique characteristic to freely fly in a 3D space, and unprecedented shape makes human-drone interaction a research topic of its own. As **HDI** is a relatively new field, ongoing research focuses on 1-evaluating and developing new control modalities, 2- enhancing human-drone communication, 3-evaluating interaction distance, and 4- developing new use cases. Our work addresses a vision-based task that aims to predict a pose estimation of a user utilizing low-resolution images, and enabling effective HDI [26].

In this section, we aim to conduct a comprehensive comparison of the existing works in dynamic neural networks, Human Drones Interaction models deployed on Nano-UAVs, and Human Pose Estimation models.

## 3.1 Human Drones Interaction (HDI)

Fully autonomous nano-scale unmanned aerial vehicles (UAVs) have proven to have a highly suitable implementation of AI smart sensors within this domain of IoT [25]. Leveraging their exceptional speed and maneuverability, these UAVs possess the capacity to rapidly acquire data from their onboard sensors as well as from a diverse array of deployed devices within the surrounding environment. Their tiny form-factor makes them exceptionally well-suited for indoor applications wherein they must safely navigate in close proximity to humans. Such applications include surveillance, monitoring, ambient awareness, and interaction with smart environments, among others.

The traditional approach to autonomous navigation employed is commonly referred to as the **localization-mapping-planning cycle**. This approach encompasses a series of interconnected steps, involving the estimation of the robot's motion utilizing either offboard techniques, such as Global Positioning System (GPS), [32] mentioned the major limitation of this approach, as crowded cities and indoor dense environments are the typical operating areas for Nano-drones, they are required to fly at low altitudes, where GPS

signals are often shadowed, or indoors where GPS signals that are frequently obstructed or diminished, and to actively explore unknown environments while avoiding collisions and creating maps. The authors developed The Swarm of Micro Flying Robots(SFLY) a swarm of vision-controlled micro aerial vehicles (MAVs) capable of autonomous navigation, three-dimensional (3-D) mapping, and optimal surveillance coverage in GPS-denied environments. The system utilizes only a single onboard camera and an inertial measurement unit (IMU). However, all localization-mapping-planning-cycle-based approaches are very expensive for computationally constrained platforms such as commercial Nano-Drones, Moreover, it is inadequate in real-time scenarios applications.

Recent results have shown that much lighter algorithms, based on convolutional neural networks (CNNs), are sufficient for enabling basic reactive navigation of small drones, even without a map of the environment. However, their computational and power needs are unfortunately still above the allotted budget of current navigation engines of nano-drones, which are based on simple, low-power microcontroller units (MCUs). PULP-Frontnet [26] represents the state-of-the-art for the task of pose estimation of a user from low-resolution images. The authors based their work on Proximity NN [23] as the same vision-based task was addressed. The Proximity NN has been demonstrated with a remote commodity desktop computer's GPU and coupled with a Parrot Bebop 2 quadrotor flying near the user and streaming front-looking high-resolution images to the remote computer. This allows the model to estimate the subject's pose relative to the drone, determine the appropriate control input, and send it back to the drone, achieving its control task, i.e., staying in front of the user. on the other hand, PULP-Frontnet achieved equivalent quality of robot behavior by employing a novel streamlined DL model with up to $\approx 24\times$ and $\approx 33\times$ fewer operations and memory, respectively. Unlike Proxy NN, PULP-Frontnet exploits all the advantages mentioned in (3) by running the model runs entirely aboard a Crazyflie 2.1 nano-drone, Fig.(2.5), with no need for any external computer/infrastructure. PULP-Frontnet minimizes its prediction to a 3D point in space ($x$, $y$ , $z$) and a rotation angle w.r.t. the gravity z-axis($\phi$).

The vision task under consideration has been previously tackled by Cereda et al[6]. In their study, the authors employed a novel neural architecture search (NAS) technique to automatically discover multiple Pareto-optimal convolutional neural networks (CNNs) specifically designed for visual pose estimation. Initially, two seed CNNs, namely PULP-Frontnet and MobileNetv1

[14], were selected as seed models. Building upon the NAS algorithm proposed by Risso et al.[30], the authors enhanced the methodology and generated a range of models with varying characteristics, encompassing regression performance, model size, number of multiply-accumulate (MAC) operations, and inference cycle. Their most optimal model demonstrated a noteworthy improvement of 32% in reducing in-field control error compared to the approach presented by PULP-Frontnet[26]. Furthermore, this model achieved a real-time inference rate of ∼50 Hz with a power consumption of 90 mW. Despite the notable achievements of both studies, it is important to note that their models remain static once deployed, lacking the ability for further optimization. This limitation prompted numerous recent investigations demonstrating the presence of intrinsic redundant computations within convolutional networks, particularly in layers, channels, and blocks, when dealing with diverse sets of visually complex input images in the domain of computer vision [41], [27],[11]. Our contribution focuses on developing a general adaptive methodology to efficiently control the inference, latency, Number of computations, and consequently the energy consumption with fully onboard computations. Our approach has proven to achieve a reduction in both NMACs and Inference cycles with a negligible margin of increase or decrease in regression performance. we will conduct a comprehensive investigation and analysis to provide empirical evidence supporting these results in the chapter.5.

## 3.2 Adaptive Inference/ Dynamic Neural Networks

The concept of *adaptive inference*, which forms the foundation of dynamic networks, has been explored even prior to the widespread adoption of modern deep neural networks (DNNs). Classical approaches to achieve adaptive inference involve constructing a model ensemble using either a cascaded structure [47] or a parallel arrangement [17], and selectively activating specific models based on the input. Additionally, spiking neural networks (SNNs) [16] perform data-dependent inference by propagating pulse signals. However, the training strategy for SNNs significantly differs from that of popular convolutional neural networks (CNNs) and their application in vision tasks is relatively limited.

In the context of deep learning, the concept of dynamic inference in conjunction with modern deep architectures has emerged as an attractive research area, gaining significant attention over the past years. Despite the substantial efforts dedicated to designing various types of dynamic networks, there remains a dearth of systematic and comprehensive reviews on this subject matter [11]. To the best of our knowledge, all current adaptive inference approaches have been primarily developed and tailored for supervised classification tasks. However, our research objective centered around the creation of a comprehensive methodology that extends the concept of adaptive inference to the domain of Multi-regression visual pose estimation. In our work, we took inspiration from [27], the authors propose a novel concept called **Big/Little** DNN (**BL-DNN**), which aims to efficiently reduce the energy consumption required for DNN execution at a negligible loss of inference accuracy. The **BL-DNN** consists of two core components: a compact DNN with low energy consumption (*Little*), and a comprehensive full-fledged large DNN (*Big*). The primary objective of the BL-DNN is to minimize energy usage by strategically minimizing the execution of the large DNN whenever feasible. The concept behind this approach is straightforward: the *Little* model is executed initially, and its predictions are considered as the final inference result if the network exhibits sufficient confidence. However, if the confidence falls below a certain threshold, the *Big* model is invoked to provide a more reliable inference outcome. the authors demonstrated that their proposed methodology reduces the total energy consumption by obtaining the inference result only with the *Little*, energy-efficient DNN in most cases.

Our main contribution is adopting the Big/Little adaptive approach to the task under investigation (Visual pose estimation ) and overcoming the limitations in the existing approach. The big little methodology is centered around the utilization of a *Decision Function* that determines the necessity of executing the large model. This approach has gained significant popularity in the domain of classification tasks due to the ease of constructing a decision function based on Softmax probabilities, which inherently represents a probability distribution reflecting the model's confidence in its predictions. Constructing a decision function for a Multi-Output regression task, where the network generates a vector of independent real number predictions, presents a complex challenge. In our research, we propose and evaluate two distinct methodologies for addressing this task:

- ***Input decision function***: Based only on the input inference samples and it's computed initially before invoking the models.

- ***Output decision function***: Based on the predictions of the *Little* model on the input inference samples.

Our proposed approach demonstrated significant effectiveness by achieving remarkable outcomes across multiple dimensions. It not only optimized computational complexity and inference latency but also provided the flexibility to balance accuracy, computational cost, and latency by controlling the selection of the *Little* model and the threshold parameter within the decision function. A more detailed description of our approach, along with a thorough analysis, is presented in the subsequent chapters, namely 4 and 5.

# Chapter 4

# Methodology

This chapter is devoted to a comprehensive description of the methodologies employed to address the task at hand. It encompasses a brief introduction to the task, the utilized datasets, the experimental setup, the decision function, and the metrics associated as well as the three distinct approaches undertaken to tackle the task.

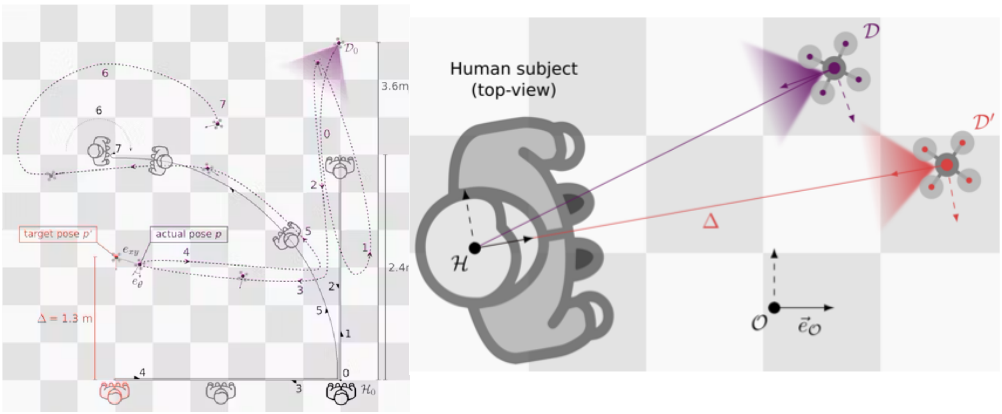



Figure 4.1: Top-view of the in-field experimental setup [26] proposed a three reference frame: $D$, $H$, and $O$. Top-down view of the human subject $H$ walking sideways to their right and the drone $D$ (violet) trying to stay in front at a distance $\Delta$ by moving toward target pose $D^{'}$ (red)

The task at hand was proposed by [26] as a multi-output regression problem that aims to estimate and maintain the nano-UAV's relative 3-D pose with respect to a person while they freely move in the environment. The goal is to map one low-resolution image to the relative pose of the subject in the form of a 3D point in space ($X$, $Y$, $Z$), and a rotation angle w.r.t. the

gravity Z-axis ($\Phi$). An illustration of the infield test and the expected drone behavior is provided in the following figure :

## 4.1 Data Sets

In this section, we will briefly introduce the data collection and augmentation applied. The dataset used to train, validate, and test the PULP-Frontnet models was collected in a 10 x 10 m room equipped with a motion capture system (mocap), composed of 12 Optitrack PM13 cameras. The dataset collection setup involved utilizing the onboard QVGA grayscale camera provided by Himax. In this setup, the quadrotor was equipped with a mocap target (i.e., reflective marker) and secured in a horizontal attitude (zero pitch and roll) on a wheeled cart with adjustable height. The captured images exhibit varied backgrounds, and lighting conditions, and sometimes contain cluttered objects such as furniture, lab equipment, and people other than the subject

### 4.1.1 Data Augmentation

The authors of PULP-FrontNet [26] employed various techniques to enhance the model's generalization. The images captured by the camera initially had dimensions of 160 x 160 pixels. For training samples, they randomly cropped them to a size of 160 x 96 pixels. Additionally, other augmentation techniques were applied, including contrast and brightness adjustments, gamma correction, the addition of a synthetic vignetting effect with random radius and strength, and smoothing using a Gaussian kernel.

To further increase the diversity of the dataset, the authors also incorporated horizontal flipping of the images. This flipping process involved adjusting the ground truth accordingly by inverting certain variables. By doing so, the distribution of these variables in the dataset became symmetrical. Figure 4.2 provides a visual representation of the employed data augmentation techniques.


Figure 4.2: Original dataset image (left) is cropped at a random height to simulate pitch variations; a random subset of photometric, optical, and geometric augmentations (top) is then applied. Bottom: ten random augmentations originating from the same source image [26]

### 4.1.2 Data Prepration

The final training dataset consisted of 26,290 samples, including the augmented images. This dataset was shuffled and divided, with 80% (21,032 samples) allocated for training and 20% (5,258 samples) for evaluation. In contrast, the test dataset comprised 4,028 consecutive frames without augmentation.

## 4.2 Adaptive Inference With Adaptive Inference Graphs

As a preliminary experiment, we drew inspiration from ConvNet-AIG [41]. The motivation behind this choice stems from the infeasibility of using a decision function based on model confidence due to the nature of our regression task, as described earlier. Instead, we aimed to adopt an approach that modifies the internal representation of the model. Although ConvNet-AIG was originally designed for a classification task, we believe it can be applicable to our scenario since it does not depend on any information derived from the output vector.

The fundamental concept underlying ConvNets was built around constructing a high-level architecture similar to residual networks (ResNets) [12]. To achieve this, a gating mechanism is embedded within each ConvBlock, allowing for the decision of whether to execute the layer or not. The internal representation of these gates, as applied in the ConvBlocks, is illustrated in Figure 2.8.

In our experimental setup, we employed a similar gating mechanism used in the FrontNet model architecture [26], with certain modifications made to the ConvBlocks. These modifications are depicted in Figure 4.3. In this architecture, each block receives a feature map from the preceding layer/block

and passes it along to all three key components of the ConvBlockAIG, as described below:

- ***classical convolutional pattern*** consist of convolution followed by batch norm and Relu activation layer, this pattern is repeated twice with a per-layer scaling effect.

- ***Gate layer*** The gate layer begins with an average pooling layer, followed by a fully connected (FC) layer with 16 output nodes. Batch normalization is then applied, followed by another FC layer with only 2 output neurons. The output of the gate layer is then passed through a Gumbel-Softmax layer, which helps prevent model collapse and grants gradient propagation during training. During inference, the Softmax layer is replaced by an Argmax operation to make a binary decision. For more detailed information, please refer to [41].

- ***Skip connection*** The skip connection layer serves the purpose of adjusting the size and/or depth of the feature map according to the output dimensions of the 4.2 component.

The output of the Gate is then multiplied by the output of the original convBlock 4.2, then added to the output of the skip connection, equation 4.1 describes the computational unit graph at inference time :

$$X_l = X_{l-1} + g_l(X_{l-1}) \cdot f_l(X_{l-1}) \tag{4.1}$$
$$\text{Where } g_l(X_{l-1}) \in \{0,1\}$$

Following the FrontNet architecture, we constructed two distinct models by fixing the initial layers, which include a $5 \times 5$ convolutional layer followed by a $2 \times 2$ max-pooling layer. These layers contribute to a $4\times$ reduction in the output feature map size due to a striding factor of two in both the horizontal and vertical directions. Subsequently, a series of ConvBlockAIG (4.3) are stacked in a repeated pattern block. Each block doubles the number of output channels and achieves a $4\times$ reduction in the output feature map size. The final part of the model incorporates a dropout stage and is followed by a fully connected layer that outputs the pose as a point in 3-D space (x, y, z) and a rotation angle relative to the gravity z-axis ($\Phi$).

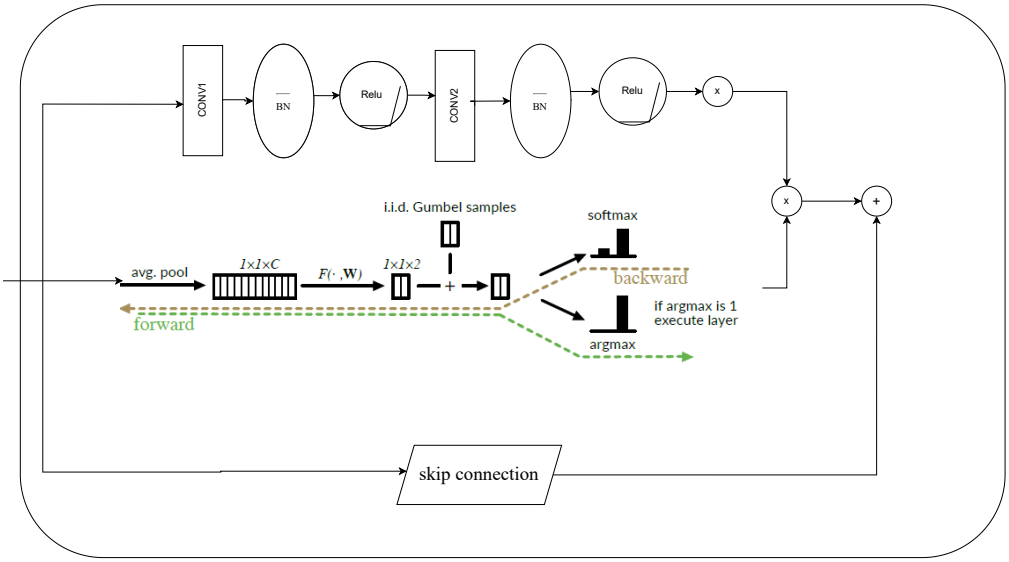



Figure 4.3: The internal representation of our proposed ConvBlockAIG with the gating mechanism applied on Convblocks of FrontNet [26]

## 4.3 Adaptive Inference with Big/Little DNN models

In the second phase of the experiment, we employed the Big/Little DNN approach introduced by [27]. Our approach was based on leveraging a diverse range of pre-trained models that were specifically optimized for the target task. These pre-trained models were developed by [6] using Neural Architecture Search (NAS), resulting in a collection of model architectures with varying characteristics in terms of inference latency (inference cycles), memory occupation (model size/number of parameters), and the number of multiply and accumulate operations (NMACs). By adopting this approach and utilizing these diverse models, we gain the flexibility to choose different combinations based on the specific application requirements. For instance, one may choose to execute a "little" model with the lowest number of inference cycles (minimal latency) but with higher NMACs (higher power consumption) and lower regression performance.

Our experimental setup consisted of several distinct phases: the evaluation of random models (employing stochastic model selection), the design of the decision function, the extraction of candidate metric thresholds, the execution of the testing loop, and the collection of results pertaining to NMACs (Number of Multiply and Accumulate Operations), the number of inference

cycles, and the Mean Absolute Error (MAE). In the subsequent sections, we will provide a comprehensive description of each of these phases.

Regarding the metrics employed within the decision function, we employed two distinct methodologies: input metrics-based and output metrics-based. The input metrics were computed on the input image prior to executing the models, while the output metrics were computed based on the predictions generated by the little model. In the following sections, we will provide a detailed explanation of the specific metrics utilized and their relevance in the decision-making process.

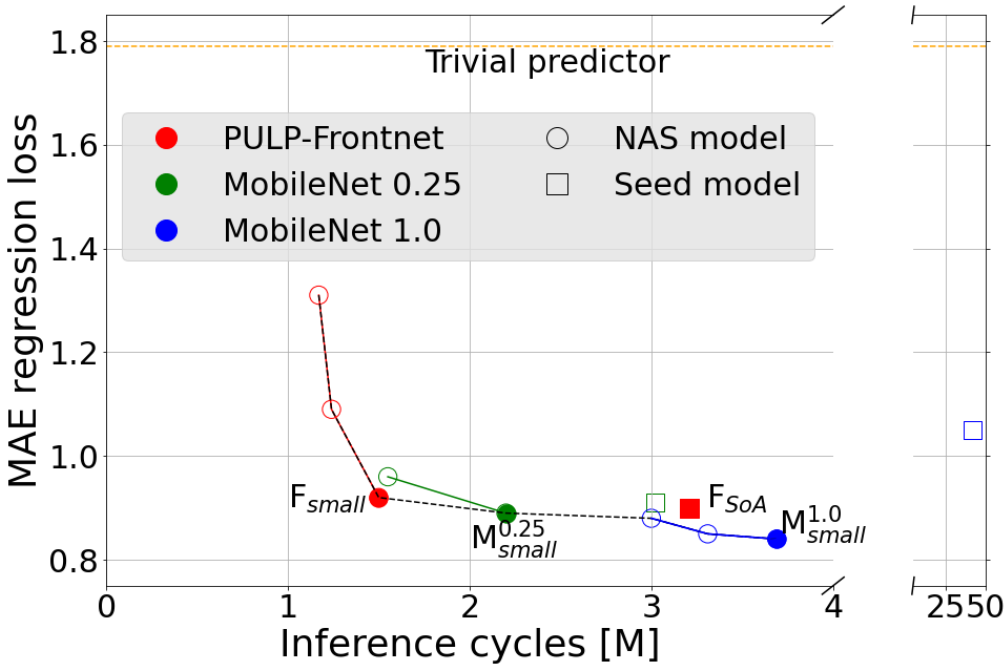



Figure 4.4: Pareto curves of the networks extracted from the NAS in the clock cycles vs. MAE space (lower is better) [6]

In the phase of selecting models for our study, we utilized four pre-trained models we will refer to as "*Static*" obtained from [6]. These models were characterized based on their performance and resource requirements, as depicted in Figure 4.4. Among them, we identified the following models: $Msmall_{1.0}$, positioned at the bottom right point of the blue curve, representing the most accurate model but with the highest computational cost and latency. $Msmall_{0.25}$, located at the center green point, strikes a balance between accuracy and significantly lower computational cost and latency. Furthermore, we choose two FrontNet-based models from the red curve, namely $Fsmall_{22}$

and $Fsmall_{11}$. These models were selected due to their lightweight nature and fast execution. Specifically, $Fsmall_{11}$, positioned at the top left point on the red curve, corresponds to the fastest model but with lower accuracy compared to the others. We fixed the $Msmall_{1.0}$ as the $Big$ model, while the remaining three models were employed as the little models. By alternating between these three little models, we were able to generate three distinct combinations.

In order to evaluate the effectiveness of our methodologies, we employed the regression performance metric utilized in the study by [6], namely the Mean Absolute Error (MAE). This metric effectively captures the disparity between the predicted values and the corresponding ground truth values as the summation of MAE for all regression variables, including x, y, z, and Φ. Our evaluation encompassed two distinct phases: the comparison between adaptive methods and random model selection, as well as the comparison between adaptive and static approaches. Additionally, we conducted a comprehensive analysis of the performance exhibited by various input and output metrics, allowing us to gain insights into their respective capabilities. Throughout these evaluations, we examined the trade-offs between MAE and Inference Cycles, as well as MAE and NMACs, providing valuable insights into the relationship between regression accuracy and computational efficiency.

## 4.4 Random model selection

As a baseline for our experiments, we decided to perform a stochastic process for assigning the test samples to Little/Big models, the intuition is that our methods with decision functions should outperform selecting random model selection approach in terms of regression performance, assuming that it will effectively assign the "$Hard$" samples to the big model and "$Easy$" samples to the Little model, yielding in more comparable results with the static models. To this purpose, we constructed our experimental setup as follows :

1. **Test sample selection**: In real applications where the samples are passed sequentially, one sample at a time, a binary decision is typically made for each sample using a Bernoulli random variable, where 0 represents the selection of the Little model and 1 represents the selection of the Big model, In our experiments, we aimed to optimize the process and take advantage of parallelization as indicated in 4.5 lines 10,

we choose to perform the index selection apriori by permutating (shuffling) the test data, then in lines 12-13 we select the models based on a parameter (ratio = p )that determined the proportion of samples to be assigned to the Big model and the Little model.

2. **External loop**: In 4.5 line-2 We conducted a comparison by varying the $P_{Big}$ parameters that control the proportion of test samples assigned to the Big model, ranging from 0 to 1 with a step size of 0.1. This resulted in eleven data points, including two extreme cases and nine intermediate points. We compute the expected NMACs for each point as indicated in 4.5 line-5. The assignment of the test samples to models follows the rule:

$$P_{Big} = \begin{cases} 0, & \text{Invoke only the Little model} \\ 0 < P_{BIG} < 1, & P_{BIG} \times \text{len(test set)} \rightarrow \text{Big , 1- } P_{BIG} \times \text{ len(test set)} \rightarrow \text{Little} \\ 1, & \text{Invoke only the Big model} \end{cases}$$

3. **Internal loop**: 4.5 line-8: Given the stochastic nature of our experiment, we performed ten internal runs with different random seeds. In line-25 the results for Mean Absolute Error (MAE) were averaged across these runs to obtain a more reliable estimate.

The following pseudo-code can summarize the undertaken steps for the experiment :

```
1  // # external loop
2  for p in [0.0, 0.1,0.2 .... , 1.0]  : // identify the Prob_big for the external loop (11 candidates)
3
4      //  Compute the expected overall Nmacs for the current Prob_big [e.g. p= 0.1 => invok big 10%  and little 90% of the time]
5      expected_NMAC = (NMAC_big * p) + (NMAC_little * (1-p))
6
7      // # internal loop
8      for seed in [s1.....s10]  : // identify the seeds for the internal loop (10 seeds)
9          // # Select the indexes randomly and split according to the probability big
10         ix_Big ,ix_Little = split(permutation (size) , ratio = p)
11         // select the test samples by the indexes
12         Big_test_set = test_set[ix_Big]
13         little_test_set = test_set[ix_little]
14         //   Results accumulators
15         y_preds = []
16         gt_labels = []
17         // Invok the models and update the predictions and ground truth labels lists
18         y_preds , gt_labels += Inference (model = Big , data_set = Big_test_set)
19
20         y_preds , gt_labels += Inference (modellittle , data_set = little_test_set)
21
22         // compute MAE for the current run
23         MAE_run(i) = MAE(y_preds , gt_labels)
24     /// Compute MAE for the current probability by averaging the MAE of all runs
25     MAE_P = AVG(MAE_run(1) , MAE_run(2) ..... , MAE_run(10))
```

Figure 4.5: Random-Model-Selection-Pseudo Code

## 4.5 Decision Function on Input Metrics

The primary objective of the Big/Little approach [27] is to develop a method that effectively differentiates between 'Hard' Images, assigned for the Big model, and 'Easy' images, suited for the Little model. A key challenge lies in formulating a decision function capable of making this distinction. The decision functions proposed in the literature rely on the softmax probabilities generated by the Little model as an indicator of model confidence. However, in our specific task, the direct utilization of model predictions for constructing a decision function is not feasible since the model outputs independent real-valued numbers instead of softmax probabilities. Additionally, the traditional approach always involves invoking the Little model, thereby introducing computational overhead, increased inference latency, and power consumption.

To overcome these limitations, we propose an alternative decision function based entirely on information extracted from the input image, prior to invoking the models. As visually illustrated in Figure 4.6, this approach eliminates the additional computational cost associated with utilizing the Little model. Our input decision function leverages image similarity metrics, exploiting the sequential nature of the data in our task. We assume that consecutive frames captured within a short time window, such as an average frame rate higher than 30 frames per second in real-time applications, share significant similarities. By considering this temporal coherence, our methodology introduces minimal overhead, involving the storage of the previous image and the computation of the similarity metric, in comparison to the original approach presented by [27].

### 4.5.1 Mean Squared Error (MSE)

The mean squared error (MSE) stands out as a metric that has garnered substantial attention within the field. Its simple formulation and clear interpretation made it one of the most widely used metrics in many fields including image processing. The metric is defined as in equation.4.2.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{4.2}$$

where n is the number of pixels in an n-dimensional image vector, and xi and yi denote the gray levels of the $i_{th}$ pixels of the original and coded image vectors x and y, respectively. Mathematically, the MSE represents the

average squared distance between two vectors, in this case, x and y. MSE score is a non-negative value on the interval [0,∞). A lower MSE score indicates a smaller error and, consequently, a higher level of visual quality. Conversely, a higher MSE score corresponds to a more substantial error and lower visual quality.

The MSE suits our task perfectly since our objective is to minimize the computational cost and the inference latency. The computational cost of the metric is almost negligible. In the context of our task, where the input images have dimensions of 160 × 96 × 1 (H, W, C). The overhead of MSE ≈ 40 kMAC can be computed using the following formula:

Number of MAC operations MSE = n × (1 m + 1 a) + (n-1) a + 1 d
where :

$n \equiv$ Number of pixels in the image $(H \times W \times C) = 160 \times 96 \times 1 = 15{,}360$ pixels
$a \equiv$ additions
$m \equiv$ multiplications
$d \equiv$ divisions

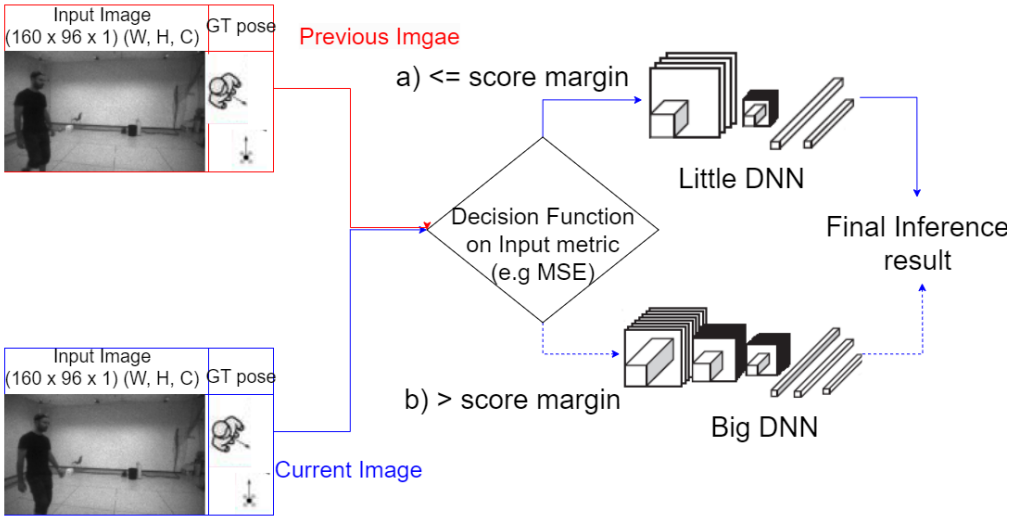



Figure 4.6: Adaptive Inference with Big/Little: decision function on MSE as input Metrics

### 4.5.2 structural similarity index (SSIM)

The Structural Similarity Index (SSIM) is a widely used metric for assessing the similarity between images. It measures the perceived quality of an image

by comparing its structural information with a reference image. SSIM takes into account three components: luminance, contrast, and structure. The index ranges from -1 to 1, with 1 indicating a perfect match between the images. The SSIM was introduced by Wang et al. [43] to overcome the limitations of existing metrics in capturing the human perception of image quality. They proposed SSIM as an alternative metric that incorporates the structural information of the images.

The SSIM formula involves comparing local image patches and computing their similarities. It consists of combining three primary components: *luminance*, *contrast*, and *structure*. Given two images denoted as x and y, with corresponding pixel values x(i, j) and y(i, j) at each pixel location (i, j), the SSIM is calculated as follows:

1. ***The luminance component***: estimated as the mean pixel intensity Computed as :

$$\mu_x = \frac{1}{N}\sum_{i,j} x(i,j) \qquad\qquad \mu_y = \frac{1}{N}\sum_{i,j} y(i,j)$$

   the luminance is a function of $x$ and $y$ utilizing ($\mu_x$,$\mu_y$ ) as follows:

$$l = \frac{2 \cdot \mu_x \cdot \mu_y + C1}{\mu_x^2 + \mu_y^2 + C1}$$

   Where the constant C1 is included to avoid instability when ($\mu_x^2$,$\mu_y^2$ )is very close to zero. Specifically, the author suggested the following constant :

$$C1 = (K1 \cdot L)^2$$

   Where $L$ is the dynamic range of pixel values (typically $2^b - 1$, where $b$ is the number of bits per pixel).

2. ***The constants component***: the authors used the standard deviation (the square root of variance) as an estimate of the signal contrast. An unbiased estimate in discrete form is given by:

$$\sigma_x = \left(\frac{1}{n-1}\sum_{i,j}[x(i,j) - \mu_x]^2\right)^{1/2} \quad , \sigma_y \; = \left(\frac{1}{n-1}\sum_{i,j}[y(i,j) - \mu_y]^2\right)^{1/2}$$

The contrast comparison c(x, y) is then a comparison function of ($\sigma_x$,$\sigma_y$) as :

$$c = \frac{2 \cdot \sigma_x \cdot \sigma_y + C2}{\sigma_x^2 + \sigma_y^2 + C2}$$

Where C2 is a constant to avoid instability (zero division) computed in the same way as c1 (i.e. C2 = $(K1 \cdot L)^2$). This function is consistent with the contrast masking characteristic observed in the Human Visual System (HVS) since this measure tends to be less sensitive to the case of high base contrast $\sigma_x$ than low base contrast with the same amount of contrast change (i.e. $\Delta\sigma = \sigma_x$-$\sigma_y$).

3. ***The Structure component*** Structure comparison is conducted after luminance subtraction and variance (contrast) normalization. The correlation (inner product) between the normalized images ($\frac{(x-\mu_x)}{\sigma_x}$, $\frac{(y-\mu_y)}{\sigma_y}$) is a simple and effective measure to quantify the structural similarity.

$$s = \frac{\sigma_{xy} + C3}{\sigma_x \cdot \sigma_y + C3}$$

$C3$ is another small constant to avoid division by zero.

4. ***The SSIM Index*** The Structural Similarity Index can be defined as a function that combines the three mentioned functions :

$$\text{SSIM} = f\left(l(x,y), c(x,y), s(x,y)\right)$$

To simplify the expression the authors choose C3 = C2/2, hence the final formula to compute the SSIM is as follows:

$$\text{SSIM(x,y)} = \frac{\left(2 \cdot \mu_x \cdot \mu_y + C_1\right)\left(2 \cdot \sigma_{xy} + C_2\right)}{\left(\mu_x + \mu_y + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C2\right)}$$

The selection of specific values for constants (K1, K2, C1, C2, C3) in the SSIM formula may vary depending on the implementation and the specific application requirements. These values are typically determined empirically to achieve the desired performance. The SSIM $\in$ [-1,1] can be interpolated as follows:

$$SSIM = \begin{cases} -1, & \text{perfect anti-correlation} \\ 0, & \text{No similarity} \\ 1, & \text{perfect similarity} \end{cases}$$

In the context of our study, we encounter the challenge of real-time application, and since this metric involves conducting three computational steps and manually configuring multiple parameters, we decided to avoid it for now as it is expected to introduce computational overhead and latency. However, we plan to optimize the implementation in our future work.

### 4.5.3 Normalized Root Mean Squared Error (NRMSE)

*RMSE* is the square root of the average of squared errors. The effect of each error on *RMSE* is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on *RMSE*. Consequently, *RMSE* is sensitive to outliers. The normalized Root Mean Squared Error (*NRMSE*) is often expressed as a percentage, where lower values indicate less residual variance. There are no consistent means of normalization in the literature. In our study, we applied a function provided by skimage.metrics, with three options for normalization ['euclidean', 'min-max', 'mean']. In the case of Euclidean normalization, we averaged by the Euclidean norm of the reference image. Given $X$,$Y$ where $X$ is the reference image :

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

$$NRMSE = RMSE \times \sqrt{N}/\|x\|$$

Where N ≡ Number of pixels in the reference image

## 4.6 Decision Function on Output Metrics

In order to further demonstrate the effectiveness of our proposed methodology, we implemented a decision function inspired by the original Big/Little approach introduced by [27]. As visually illustrated in Fig.4.7, this decision function leverages the predictions of a smaller model, referred to as the "Little" model, to assess model confidence. We designed a metric based on the distance between consecutive frames, considering the sequential nature of our data. We assumed that consecutive frames should exhibit only small shifts in the coordinates of the drone and, consequently, small deviations in the consecutive model predictions.

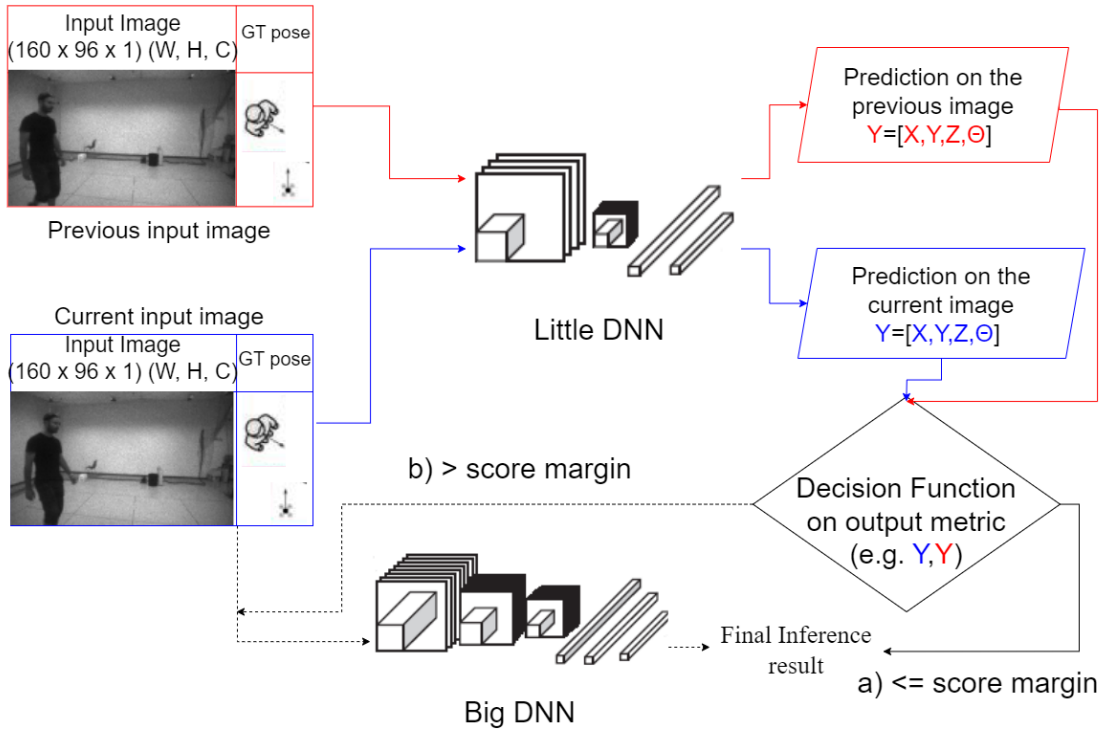



Figure 4.7: Adaptive Inference with Big/Little: decision function on Output Metrics

In our study, we devised four straightforward metrics based on the Absolute Error (AE) between the regression-dependent variables: ($x$,$y$,$\Phi$). Additionally, we considered the sum of the absolute errors as a metric. we decided to exclude the variable $z$ from the analysis due to its significantly lower standard deviation compared to the other variables (i.e., $\approx 0.2$). The execution of our approach can be summarized by the following steps:

1. In the upper left part of figure 4.7, as a default setting, $'Little'$ model is invoked on the input frame producing an output vector of predictions, denoted as $Ypred_{(t-1)}$, store the predictions in a buffer to be compared with subsequent predictions.

2. The current frame is passed to the $'Little'$ model to obtain the prediction vector $Ypred_{(t-1)}$

3. To establish more general thresholds for the metric, we normalized the

prediction vectors using the Min-Max normalization technique, constraining the values to the interval ∈ [0,1]. The normalization was performed using the formula:

$$\boxed{X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}}$$

4. On the right part of figure 4.7, the 2 normalized vectors are fed to the decision function which computes the metric score for the selected variable as follows:

$$\text{Score} = \mid \hat{Y}_T \text{ - } \hat{Y}_{T-1} \mid$$

where :

$\hat{Y} \equiv$ reprsents one of the regression variables ∈ [x,y,Φ,sum] normalized
$SUM_{AE} \equiv$ the sum of the normalized Absolute error on all regression variables

5. Compare the score to a pre-defined Score margin if :

   a. path (A) Score ≤ Score Mergin → *Little* model prediction = Final prediction.

   b. path (B) Score > Score Mergin → pass the same current frame to Big model → Invoke the *Big* model → prediction = Final prediction.

While this approach, which relies on model predictions, is expected to yield more reliable results, our experimental results indicate a decrease in performance compared to the Input-Metrics decision function, particularly in terms of mean absolute error (MAE). Moreover, the constant execution of the "Little" model introduces additional overhead, negatively impacting inference latency and computational costs. It is noteworthy that the metric itself incurs a negligible computational cost (i.e., X × W ≈ 15 kMACs). A comprehensive numerical analysis of the performance of all metrics will be conducted in the subsequent chapter 5, providing a more detailed evaluation of their effectiveness.

# Chapter 5

# Experimental Results

This chapter is dedicated to presenting the experimental results of the proposed methodology applied to the tasks discussed in the previous chapters. We will conduct a comprehensive analysis to evaluate the effectiveness of our approach, utilizing the Big/Little approach outlined in Chapter 4. This analysis encompasses both quantitative and visual assessments to compare the performance of input and output decision metrics. We will employ various evaluation metrics, including the Mean Absolute Error (**MAE**) as a measure of regression performance, the number of Multiply-Accumulate (**MAC**) operations as a metric for computational cost, and inference cycles as an indicator of inference latency. Furthermore, this analysis will provide insights into the behavior of the models and demonstrate how alternating between Little models with different characteristics can yield diverse results. These findings can be leveraged to optimize the approach according to the specific requirements of the target application.

To evaluate the effectiveness of our approach, we conducted a baseline experiment employing random model selection, referred to as "Stochastic." This experiment serves as a benchmark for evaluation purposes. We hypothesized that the random model selection behavior would demonstrate a linear decline in Regression Error (MAE) as we move from the lower bound (representing the Little model with the lowest regression performance) to increase computational cost and inference latency. Conversely, the upper bound (representing the Big model) would exhibit the highest achievable regression performance but at the expense of maximum computational cost and inference latency. Between these extremes, a trade-off exists that can be customized to specific requirements. However, our experimental results indicate that randomly assigning test samples for inference to either the Big or Little model does not

yield optimal outcomes, as it fails to fully exploit the distinctive characteristics of each model. In the subsequent sections, we will elaborate on how our approach aims to effectively leverage these characteristics to enhance overall performance.

The second experiment involved the development of a decision function based on input metrics, which was executed beforehand to determine the appropriate model for the current inference task. As previously discussed in Chapter 4, this decision function was designed using an image similarity metric, specifically the Mean Squared Error (MSE). This metric demonstrated superior performance in achieving a favorable balance between accuracy, latency, and computational cost with minimal overhead in terms of computation. In the subsequent sections, we present a comprehensive analysis of the effectiveness of this decision function and the overall system that was constructed around it.

The final experiment in our study involves the implementation of a decision function utilizing output metrics. As explained in Chapter 3, designing a decision function that directly utilizes information from the model output for our specific task is not a straightforward process. In Chapter 4, we proposed four simple output metrics to showcase the significant enhancements our approach brings to the original Big/Little methodology proposed by [27]. These metrics serve as a benchmark, comparing our approach to a similar methodology that relies on output decision functions. We aim to provide further evidence of the advancements achieved by our contribution. By utilizing these output metrics, we demonstrate the notable improvements our approach offers in comparison to the original methodology, shedding light on the capabilities and benefits of our proposed methodology.

The evaluation process includes multiple phases, each involving specific configurations. Throughout all phases, we maintained the Big model as a fixed reference ($Msmall_{1.0}$), while alternating between three different Little models ($Msmall_{0.25}$, $Fsmall_{22}$, $Fsmall_{11}$). By using these distinct combinations of Little models, we can explore and compare their unique characteristics. The subsequent sections will present detailed numerical evaluations to analyze the differences observed among these models.

Finally, it is important to highlight that our primary approach, which is based on ConvNets Adaptive Inference graphs [41], as discussed in Chapters 4 and 3, did not yield the anticipated results. While this work is still in progress, we have decided not to share the specific outcomes at this time. We are working on further optimizing this approach to enhance its effectiveness, and we look forward to providing updates on our findings in future work.

## 5.1 Experimental setup

The experimental setup for our study involved pre-trained models that were optimized using Neural Architecture Search (NAS) by [6]. Therefore, our experiments did not involve any additional training of the models. Instead, we conducted re-testing on the test data to verify the consistency and reliability of the results obtained. The experiments were implemented using Python 3.8 and various libraries, including Sklearn, Pandas, NumPy, torch, Torchvision, Tqdm, Argparse, and Matplotlib, for data visualization. All experiments were performed using the PyTorch framework and the PyTorch TorchVision library. For image similarity, we utilized the Skimage library, which offers a diverse range of image similarity metrics along with well-documented and well-implemented functionalities.

While some of our experiments were carried out on a single GPU, specifically the nVidia Tesla V100 16GB, the majority of the experiments were conducted on a local machine equipped with a standard CPU, specifically the Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz - 2.59GHz. It is important to note that we have not yet performed an actual deployment of the proposed methodology, as this is planned for future work. However, we have relied on the actual infield test results conducted by our colleagues and the findings presented by them in [27].

## 5.2 Results On Static models

Before conducting our experiments we needed to compute the upper extrema (Big model accuracy on whole test data) and the lower extrema ( Little model accuracy on whole test data ) by running the four candidate models [$Msmall_{1.0}$,$Msmall_{0.25}$, $Fsmall_{22}$, $Fsmall_{11}$] and computing the $MAE_{sum}$r as follows:

$$\mathrm{MAE_{SUM}} = \mathrm{MAE}_X + \mathrm{MAE}_Y + \mathrm{MAE}_Z + \mathrm{MAE}_\Phi$$

In our thesis work, we included the number of MAC operations calculated using the ptflops library, specifically by utilizing the "*get_model_complexity_info*" class. It is important to note that the estimated number of MAC operations serves as a preliminary reference. To accurately assess the actual computational cost on the target hardware or platform. It is worth mentioning that using alternative libraries, such as "torchinfo summary" may result in

different estimates of MAC operations. This variation can occur because certain libraries exclude certain layers (e.g., batch normalization, pooling layers) from the computations.

Regarding the inference cycles, as the actual deployment of our models has not been conducted, we relied on the actual deployment results on the specific hardware reported by our colleagues in [6],[26]. The regression test performance, along with the corresponding inference cycles and NMACs, for the four candidate models are presented in the following table5.1:

| Network | MAE | | | | | #param [M] | #NMAC [K] | #cycles[M] |
|---|---|---|---|---|---|---|---|---|
| | X | Y | Z | Φ | SUM | | | |
| MobileNet $Msmall_{1.0}$ | 0.157 | 0.193 | 0.07 | 0.425 | 0.845 | 42 | 10.68 | 3.7 |
| MobileNet $Msmall_{0.25}$ | 0.158 | 0.179 | 0.081 | 0.453 | 0.871 | 18 | 5.58 | 2.2 |
| Frontnet $Fsmall_{11}$ | 0.257 | 0.137 | 0.212 | 0.519 | 1.125 | 14.7 | 5 | 1.2 |
| Frontnet $Fsmall_{22}$ | 0.2 | 0.189 | 0.103 | 0.449 | 0.941 | 44.5 | 7.6 | 1.5 |
| Frontnet (SoA) | 0.19 | 0.18 | 0.09 | 0.44 | 0.900 | 304 | 14.7 | 3.2 |

Table 5.1: TEST SET EXPERIMENT RESULTS

## 5.3 Input Metric Vs Random Model Selection

### 5.3.1 Input Metrics Benchmarking phase

As an initial step, to select the optimal input metrics for our task we conducted an experiment for benchmarking the estimated performance of several image similarity metrics, namely Mean Squared Error (MSE), Normalized Mean Squared Error (NMSE_euclidean), and structural_similarity (SSIM) as previously described in details in chapter 4.5. We built our benchmarking strategy around an assumption that the deviation in coordinates of 2 consecutive frames, which reflects the temporal dependency, can be represented by the absolute difference between the 2 ground truth labels (i.e Diff = $|Y_{(T)}$ - $Y_{(T-1)}|$ ), and we took the sum of the differences for all regression variables (i.e $Diff_{sum}$ =$Diff_x$ +$Diff_y$+$Diff_z$+$Diff_\Phi$). To measure the strength of the relation between these two quantities (i.e. MSE, $Diff_{sum}$) we utilized the Pearson correlation coefficient, we found the $MSE$ to be the metric with the highest Pearson correlation coefficient value. for this reason and the simplicity of implementation and interpolation as discussed previously in chapter 4.5.1, we decided to implement only the MSE for now and

the other will be implemented in future work.

### 5.3.2 MSE Vs Random Model Selection

In our initial experiment, we compare two methods for selecting the appropriate model for inference: random selection and our proposed decision function utilizing an input metric based on ($MSE$). These methods are executed prior to invoking the models for inference, hence the assessment will be only in terms of regression performance, as measured by the ($MAE_{SUM}$).

The objective of this experiment is to demonstrate that the input decision function can effectively exploit the temporal dependencies of consecutive input frames. By utilizing this information, we aim to assign the easier frames to the little model and reserve the big model for more challenging frames, thereby reducing computational load. In the case of random model selection, the execution of the big model is controlled by a pre-defined parameter $P_{Big}$, where $P_{Big} = 0.1$ implies that the big model will be invoked 10% of the time, while the little model will be invoked 90% of the time ($P_{Little} = 1 - P_{Big} = 0.9$). In the subsequent sections of this thesis, we will utilize the term **"Execution Plan"** to denote the process of selecting the desired proportion of execution time for the big and little models. For example, a balanced execution plan corresponds to a 50/50% execution ratio, indicating an equal allocation of 50% of the application run time for both the big and little models.

To determine the appropriate thresholds for each $P_{Big}$ value in our predefined set [0.1, 0.2, ..., 1], we conducted a search on the test set. We computed the $MSE$ on consecutive frames and identified the thresholds that corresponded to the desired $P_{Big}$ values. This process was performed for all three combinations of the models mentioned earlier. For each combination, the lower extrema (Little model) and upper extrema (Big model) were obtained from deploying all models on the target hardware as described in5.1. To calculate the intermediate values, we utilized a straightforward formula to compute the inference cycles, nonetheless, the same formula is used to interpolate the number of MAC operations :

$$\text{InferenceCycles(p)} = InferenceCycles_{Big} \times \text{p} + InferenceCycles_{Little} \times \text{(1-p)}$$

where :

$$\begin{aligned}
\text{InferenceCycles}(p) &: \text{Number of Inference Cycles of the whole system} \\
InferenceCycles_{\text{Big}} &: \text{Actual Number of Inference Cycles of the Big model} \\
InferenceCycles_{\text{Little}} &: \text{Actual Number of Inference Cycles of the Little model} \\
p &: \text{the portion of execution time where the big model is invoked} \\
1-p &: \text{the portion of execution time where the Little model is invoked}
\end{aligned}$$

After implementing our proposed approach using all three combinations of models, we present the visualized results in Figures [5.3, 5.2, 5.1]. The figures illustrate the ($MAE_{sum}$) numerical results obtained by our method compared to the random model selection approach at various intermediate points. Our proposed metric consistently outperforms the random model selection approach, indicating that our method effectively assigns the test samples to the optimal model. In theory, we would expect the big model to yield lower errors across all samples. However, our experimental findings contradict this expectation, as can be indicated in the figures, by alternating between the models we achieved lower overall MAE with much lower computational cost and lower latency. we plan to conduct a comprehensive analysis by examining the MAE for each regression variable individually across all model combinations. This analysis aims to provide deeper insights into the observed behavior.

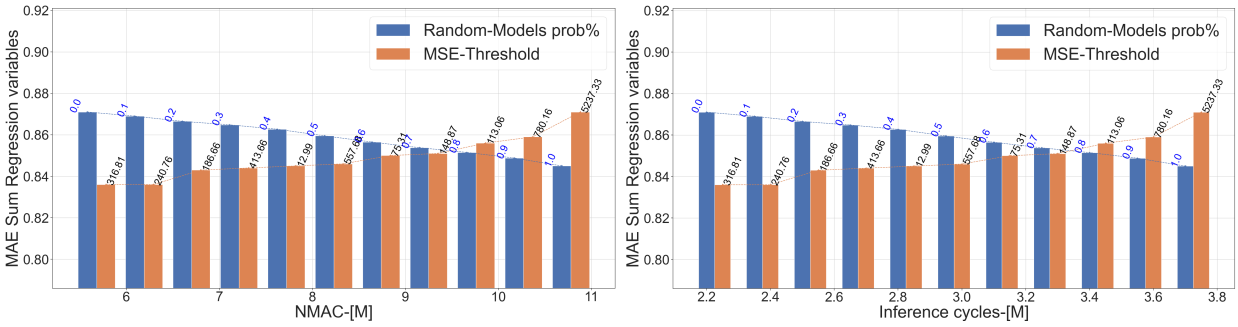



Figure 5.1: Input Metric MSE | $MSmall_{1.0}$ vs $Msmall_{0.25}$

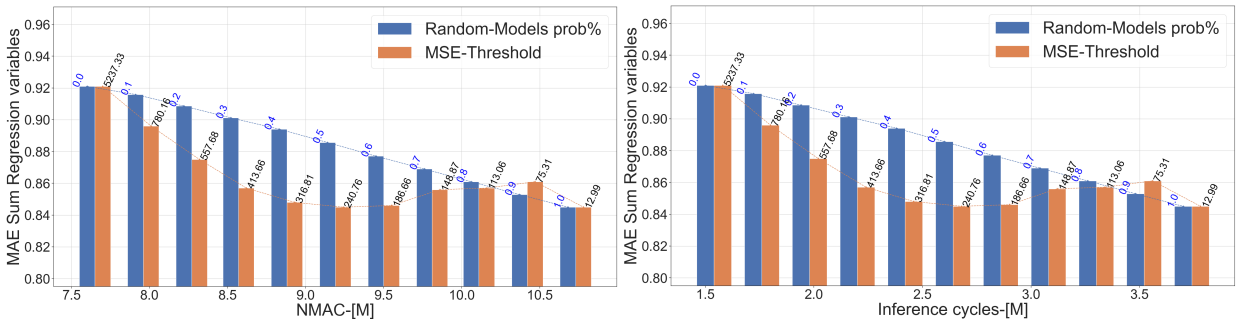



Figure 5.2: Input Metric MSE | $MSmall_{1.0}$ vs $Fsmall_{22}$

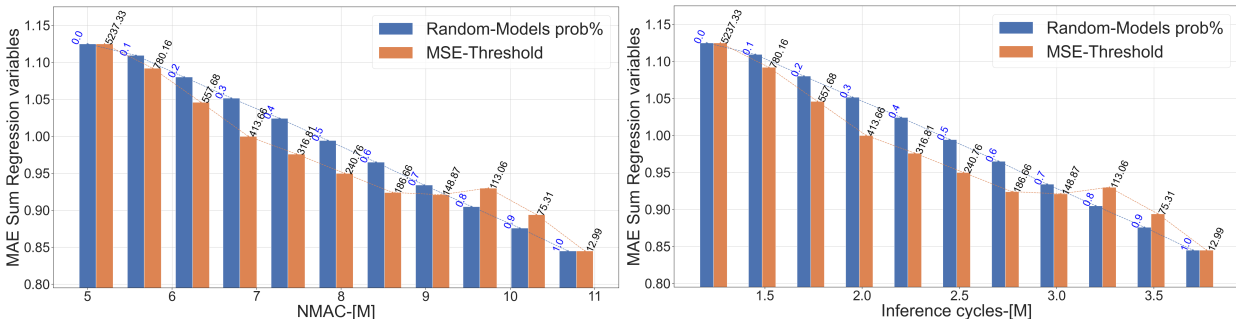



Figure 5.3: Input Metric MSE | $MSmall_{1.0}$ vs $Fsmall_{11}$

### 5.3.3 Results for MSE | $MSmall_{1.0}$ vs $Msmall_{0.25}$

In this experiment, we employed two models, denoted as $Msmall1.0$ and $Msmall0.25$. These models are based on the MobileNetV1 architecture with width multipliers of 1.0 and 0.25, respectively. They were optimized for the specific task at hand using NAS techniques, as proposed by [6]. Figure 5.1 illustrates the performance of our method compared to random model selection and an upper bound scenario where only the Big model is executed. Notably, our method outperformed both random model selection and the upper bound in terms of $MAE_{sum}$ at intermediate points. For example, at $P_{Big} = 0.4$, our method achieved a lower error of 0.836 compared to 0.863 for random model selection and even surpassed the upper bound with an error of 0.845. To gain deeper insights into the behavior of the models, we extended our investigation to analyze the individual regression variables, as shown in Figure 5.4. We observed that as the execution of the Big model increased, the accuracy of predicting $\Phi$ (phi) improved, while on the contrary, the error in predicting $Y$ increased. This observation is intuitive, as the Little model exhibited better performance in predicting $Y$. However, it is important to

note that the final $MAE_{sum}$ is highly influenced by $MAE_{\Phi}$. Analyzing the intermediate points, we found that balancing the execution of the two models, with $P_{Big} \in [0.4,0.5,0.6]$, yielded the optimal trade-off between the increase in $MAE_Y$ and the reduction in $MAE_{\Phi}$, with an approximate 0.3% reduction in NMACs compared to executing only the Big model.

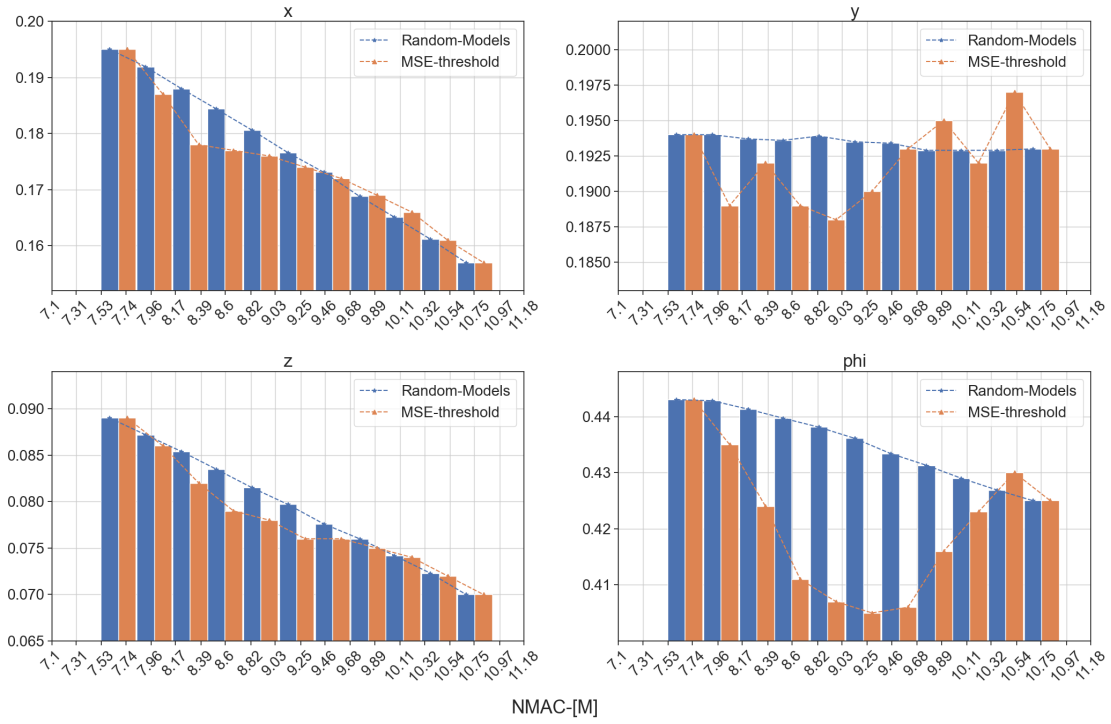



Figure 5.4: Network performance in terms of MAE[x-y-z-Φ]

### 5.3.4 Results for MSE | $MSmall_{1.0}$ vs $Fsmall_{22}$

The second model combination we employed utilized a model based on FrontNet [26]. The FrontNet model demonstrated notable advantages in terms of computational cost and latency reduction compared to the model based on the MobileNet architecture but in the cost of a decrease in accuracy. This can be due to its lightweight and straightforward model architecture. The results presented in Figure 5.2 show that by executing the FrontNet model 50% of the time, we achieved the same $MAE_{sum}$ while experiencing a remarkable reduction of over 30% in terms of inference cycles (latency). Figure 5.5 provides further insights into the performance of the FrontNet model combination. It is evident that this comparable performance is primarily driven

by the significant reduction in error for the Φ variable, which has the highest contribution to the $MAE_{sum}$. Additionally, it is worth noting that the error in the $Z$ variable shows minimal variability when comparing the decision function based on MSE or random model selection. This observation aligns with the intuition mentioned in previous sections, as the drone is typically positioned in front of the user and its $Z$ coordinate does not frequently change.

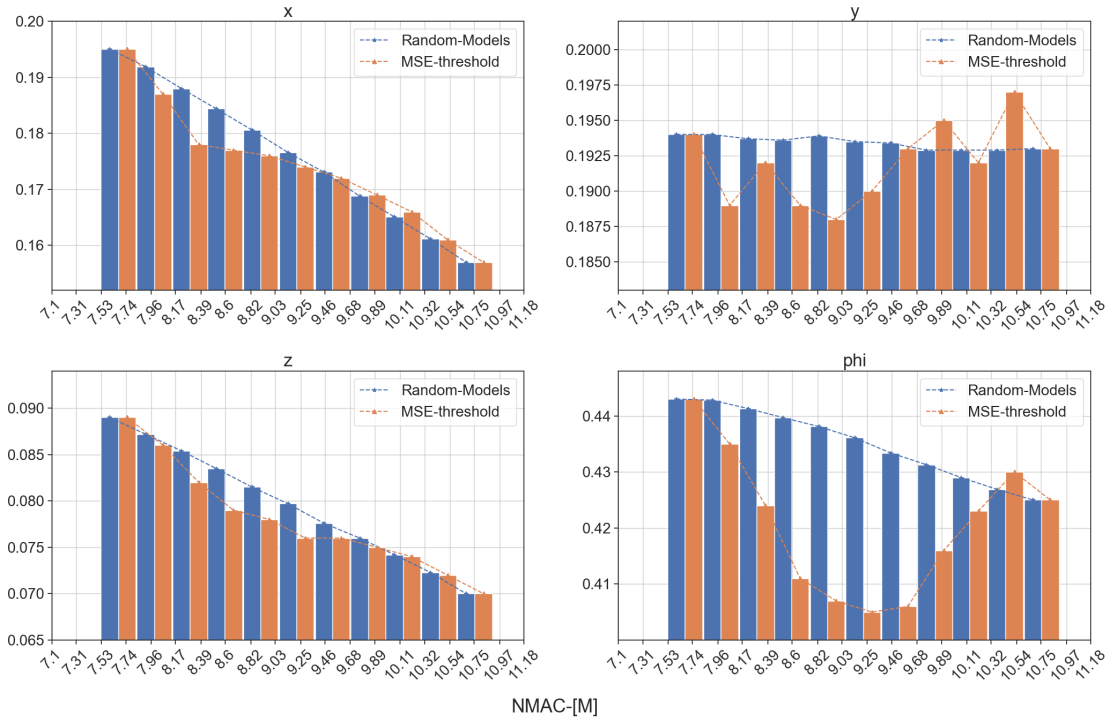



Figure 5.5: Network performance in terms of MAE[x-y-z-Φ]

### 5.3.5 Results for MSE | $MSmall_{1.0}$ vs $Fsmall_{11}$

The final model combination we utilized involved a model based on FrontNet [26], referred to as $Fsmall11$. This model exhibits a highly compressed architecture, resulting in exceptional speed with only 1.2 million inference cycles, approximately 32% of the latency of $MSmall1.0$. However, this efficiency comes at the cost of increased error, as the $MAE_{sum}$ is ≈ 32% higher compared to $MSmall1.0$.

Nonetheless, our empirical results demonstrate that by executing the $Fsmall11$ model with a balanced (50/50%) execution plan, we can achieve a significant

reduction of ≈ 35% in latency. However, this reduction in latency is accompanied by an increase of ≈ 12% in error compared to only executing the Big model.

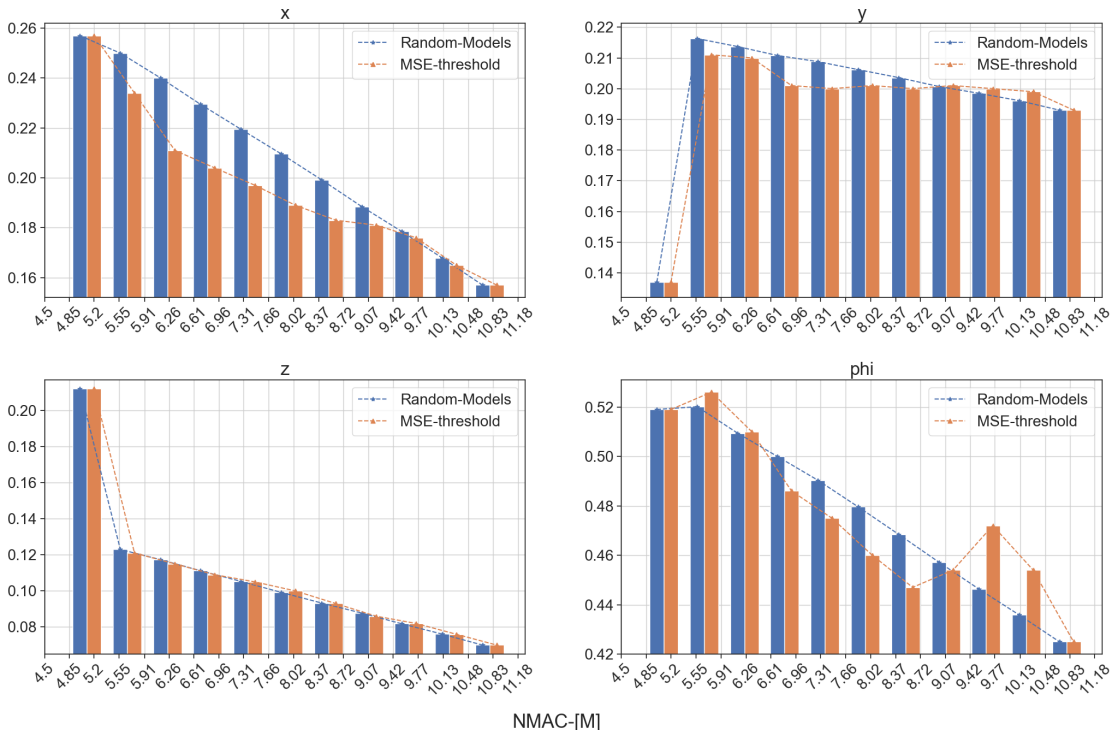



Figure 5.6: Network performance in terms of MAE[x-y-z-Φ]

In general, the utilization of the decision function based on the input metric ($MSE$) has yielded highly promising results in optimizing computational cost, latency, and, in some cases, even accuracy, offering a high degree of flexibility in selecting the factor to optimize. The adoption of FrontNet-based models has led to a significant reduction in terms of inference latency, enhancing the system's overall execution speed. On the other hand, the combination with MobileNet-based models has resulted in higher regression performance and lower errors, while still maintaining acceptable reductions in latency and computational cost.

Our experiments have demonstrated the generalizability and customization potential of our proposed methodology by showcasing the selection of an appropriate little model for the specific application at hand. Moreover, we have shown that systematic assignment of frames yields better system performance compared to random assignment, and in certain scenarios, it can even outperform the performance achieved by executing the big model

exclusively.

## 5.4 Output Metrics Vs Random Models' Selection

To further explore the effectiveness of the proposed methodology, we adopted a similar methodology a similar approach to the original Big/Little method presented by [27] was adopted. The decision functions were designed using four distinct metrics. Our aim was to keep the design as simple as possible; therefore, we employed the absolute difference between the little model's predictions on the previous frame (referred to as $Y_{pred}$ at time $T$) and the current frame (referred to as $Y_{pred}$ at time $T-1$). The model's output, denoted as $Y_{pred}$, consists of a vector [X, Y, Z, Φ]. Additional details regarding these metrics are provided in the previous section 4.6. To evaluate the effectiveness of these metrics, we followed a similar evaluation process as in comparing the proposed approach to random model selection. The difference lies in the execution time of both approaches. In the case of random model selection, both approaches are executed after the little model, and a binary decision is made to either keep the predictions as the final inference results or invoke the big model. On the other hand, in our proposed approach, the decision is made by comparing the output of the decision function to a predefined threshold.

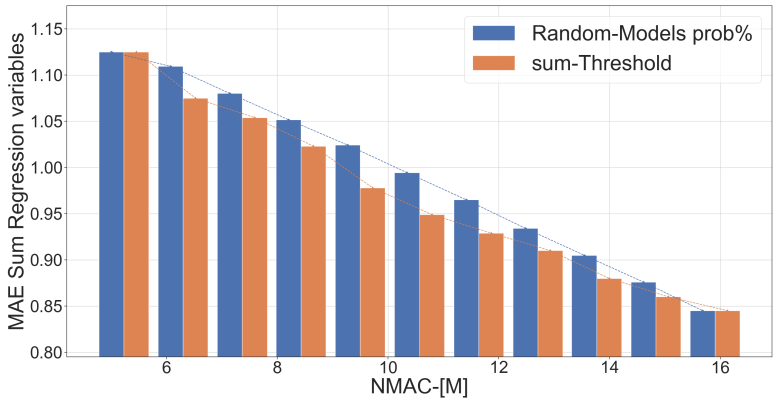
### 5.4.1 Decision on Sum errors

The first output decision function is constructed by calculating the sum of the absolute differences in errors as follows:

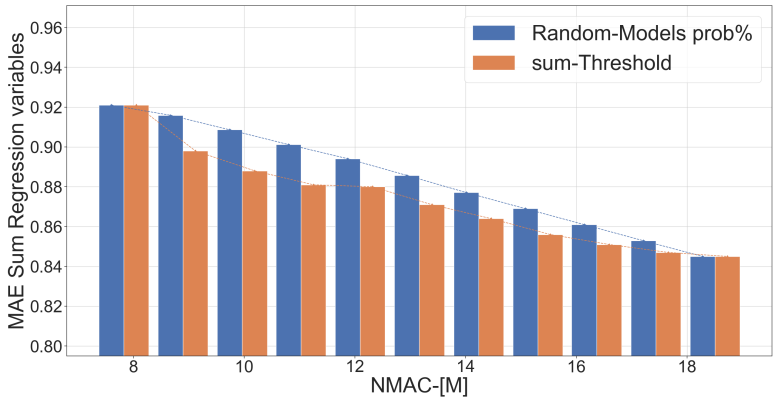
$$Score = |\hat{Y}_T^x - \hat{Y}_{T-1}^x| + |\hat{Y}_T^y - \hat{Y}_{T-1}^y| + |\hat{Y}_T^z - \hat{Y}_{T-1}^z| + |\hat{Y}_T^\Phi - \hat{Y}_{T-1}^\Phi| \quad (5.1)$$

where :

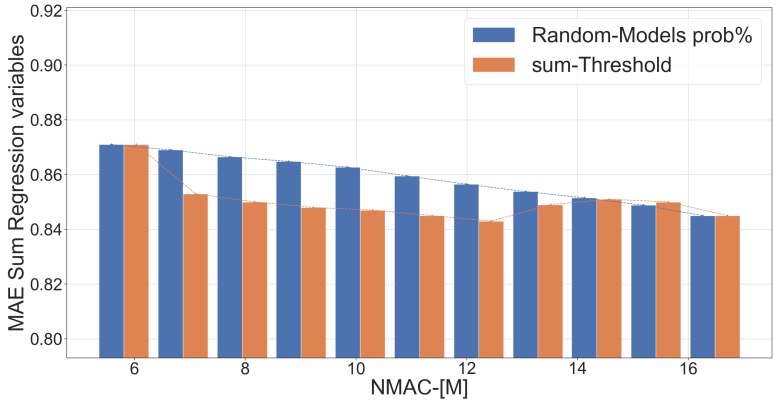
$\hat{Y}_T^x \equiv$ the normalized current prediction of the model for the variable x
$\hat{Y}_T^y \equiv$ the normalized current prediction of the model for the variable y
$\hat{Y}_T^z \equiv$ the normalized current prediction of the model for the variable z
$\hat{Y}_T^\Phi \equiv$ the normalized current prediction of the model for the variable Φ
$Score \equiv$ the sum of the normalized Absolute Error on [x, y, z,Φ]




(a) $MSmall1.0$ and $Fsmall11$




(b) $MSmall1.0$ and $Fsmall22$



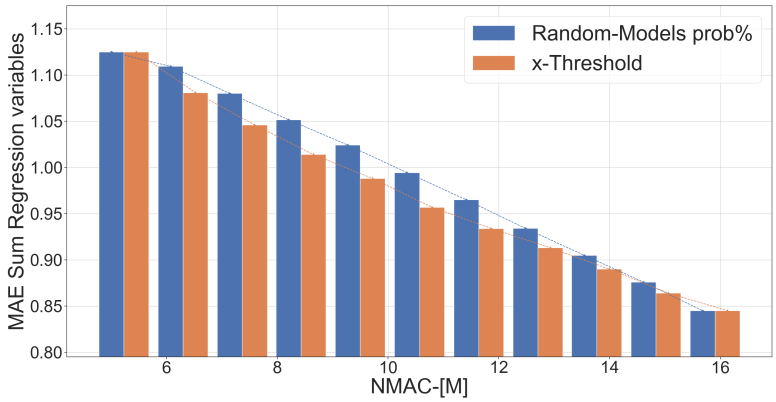

(c) $MSmall1.0$ and $Msmall0.25$

Figure 5.7: Decison on sum - MAE-SUM vS NMACs Results

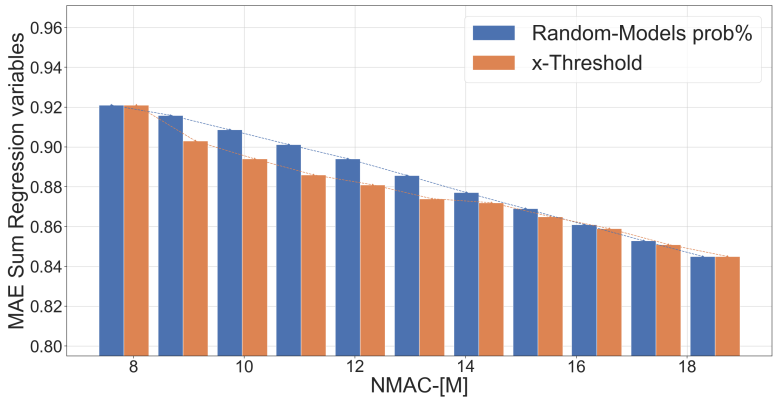
Figures 5.7a, 5.7b, and 5.7c depict the evaluation of the decision function based on sum errors 5.1, demonstrating superior performance compared to random model selection across different combinations of models. However, the observed improvement is not substantial when compared to using only the big model or the small model individually. Among the model combinations, the $MSmall1.0$ and $Fsmall011$ combination show the most significant improvements, characterized by a steeper error curve. This is primarily due to the notable variation in performance and computational cost between these two models. Nevertheless, achieving a satisfactory trade-off between performance, computational cost, and latency becomes challenging when considering the default execution of the little model. For instance, adopting a balanced execution plan with equal weights (50/50%) leads to a modest reduction of approximately 17.5% in latency and 9.8% in NMACs. However, this approach also results in an increase of approximately 12% in error compared to exclusively executing the static big model ($MSmall1.0$). It is worth noting that by utilizing the input decision function described in Section 5.3.5 under the same execution plan, it becomes possible to achieve equivalent regression performance with a reduction in latency by $\times 2$ and a computational cost reduction of around 35% and 29%, respectively.

### 5.4.2 Decision on X errors

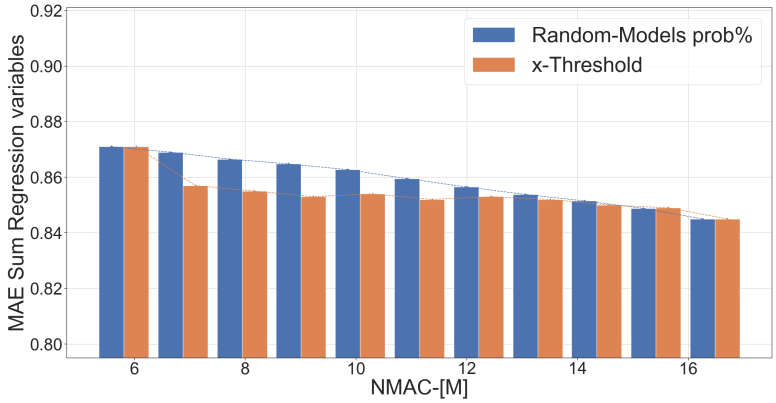
The second output decision function focuses on the error specifically on variable $x$, comparing the previous and current predictions of the little model. The outcomes of this decision function are depicted in Figures 5.8a, 5.8b, and 5.8c. However, the results are similar to those obtained from the decision function based on sum errors, showing no significant improvements. Similar observations were made for the decision functions based on variables $y$ and $\Phi$. Therefore, we will not present detailed results in this section. In the subsequent section, we will analyze the outcomes from a different perspective.




(a) $MSmall1.0$ and $Fsmall11$




(b) $MSmall1.0$ and $Fsmall22$




(c) $MSmall1.0$ and $Msmall0.25$

Figure 5.8: Decison on x - MAE-SUM vs NMACs Results

### 5.4.3 Analysis of Output Decision Metrics in Relation to Network Performance on Individual Regression Variables

In this section, we conducted an analysis to investigate the relationship between the variables used in the decision metric and the network's predictions for these variables. We calculated $MAE_{sum}$ for each regression variable and visualized the results in Figures 5.10a, 5.10b, and 5.9. The focus of this analysis was on the curves corresponding to the variable $x$, which showed significant improvements in the network's performance when the decision metric was based on $x$ compared to alternative decision metrics. A similar trend was observed for the variable $y$, indicating that the little model performed better in predicting this variable.

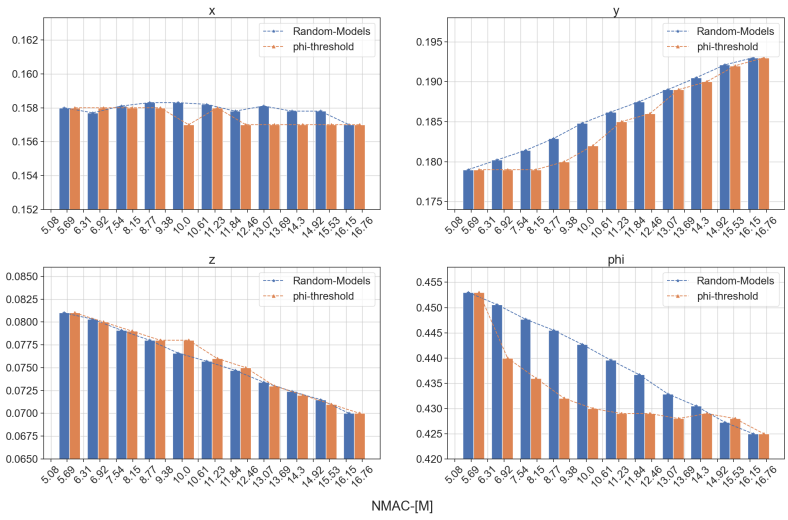
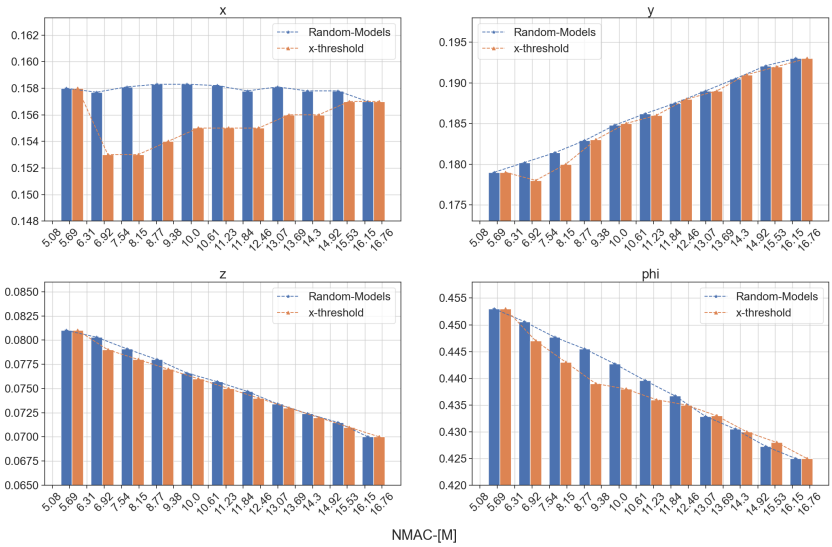
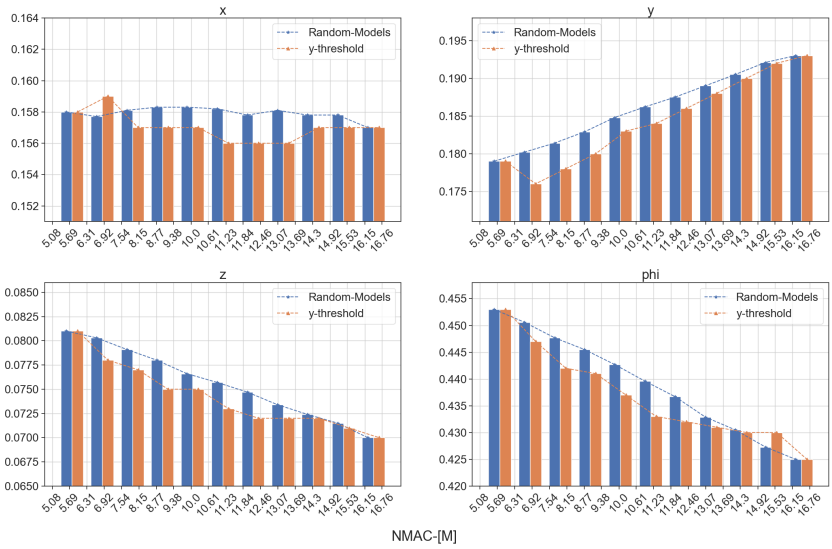



Figure 5.9: Decision on Φ

Furthermore, the examination of the decision metric curves for the variable Φ and the sum of errors ($Sum$) provided valuable insights. We observed a proportional relationship between the decision metric and the variable Φ, with Φ contributing the most to the MAE for the $Sum$ decision metric due to its larger margin of error. In contrast, the variable $z$ was found to be the easiest to predict, as it reflects the target height, which remains relatively constant throughout the task.




(a) Decision on $x$




(b) Decision on $y$

Figure 5.10: Analysis of Output Decision Metrics in Relation to Network Performance on [x, y] Regression Variables

## 5.5 Input Metrics Vs Output Metrics Vs Static models

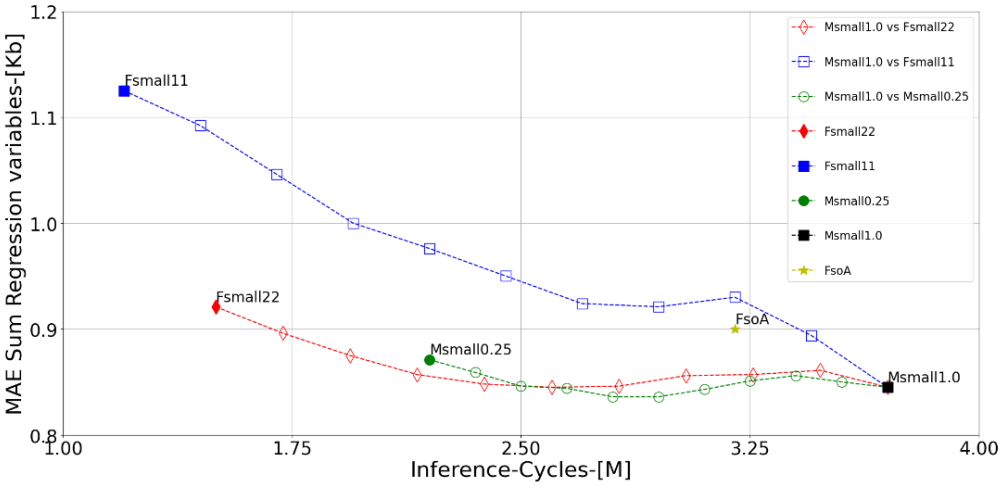
In this section, we conduct a comprehensive evaluation to compare the performance of the proposed Big/Little approach, utilizing both input and output decision functions, with the alternative approach of deploying a single static model. Our analysis focuses on three key aspects: computational cost, inference latency, and the trade-off in regression performance. This assessment will contribute to the understanding of the advantages and limitations of each approach, enabling informed decision-making regarding the selection of the most suitable model deployment strategy.

### 5.5.1 Input Metrics Vs Static Models

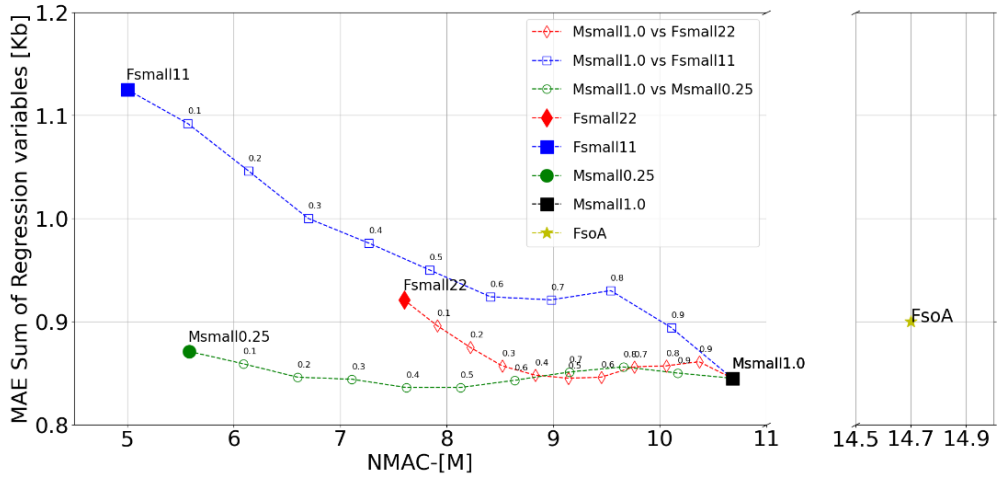
To illustrate the improvements achieved by the approach, we compare the obtained results with the static model execution paradigm. Figure 5.11b demonstrates the flexibility provided by the proposed method in selecting a deployment strategy that better aligns with the computational limitations of the target device. It can be observed that by deploying a network consisting of the $MSmall1.0$ and $Msmall0.25$ models, with execution plans ranging from 10% to 60% for the big model, we achieved better performance in terms of regression performance ($MAE_{sum}$) compared to all the static models and other adaptive networks with different little models. This improvement was accompanied by a significant reduction in computational cost compared to the static models.

Another option for faster inference is illustrated in Figure 5.11a, where the network denoted by $MSmall1.0$ and $Fsmall22$ exhibits promising results. By implementing an execution plan ranging from 10% to 40% for the big model, we achieve comparable regression performance while keeping the inference below 2 million cycles. This represents a substantial reduction in latency, $\approx 2\times$ faster than the execution of the $MSmall1.0$ static model, and $\approx 40\%$ faster than the state-of-the-art FrontNet model for the given task.

Lastly, we examine the configuration of $MSmall1.0$ vs $Fsmall11$, which demonstrates the greatest reduction in computational cost and latency. However, it also exhibits a significant decrease in network performance. This trade-off may be acceptable for applications that prioritize strict real-time inference and target devices with limited resources, where accuracy is not the primary optimization factor.




(a) Inference cycles Vs $MAE_{Sum}$



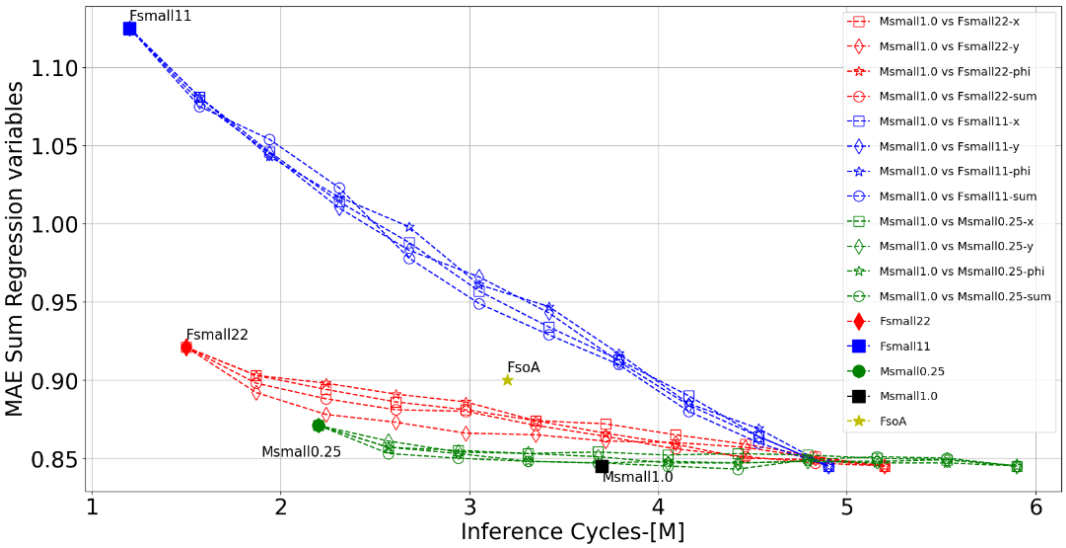

(b) NMACs Vs $MAE_{Sum}$

Figure 5.11: Analysis for the performance of Decision on MSE Vs Static Models

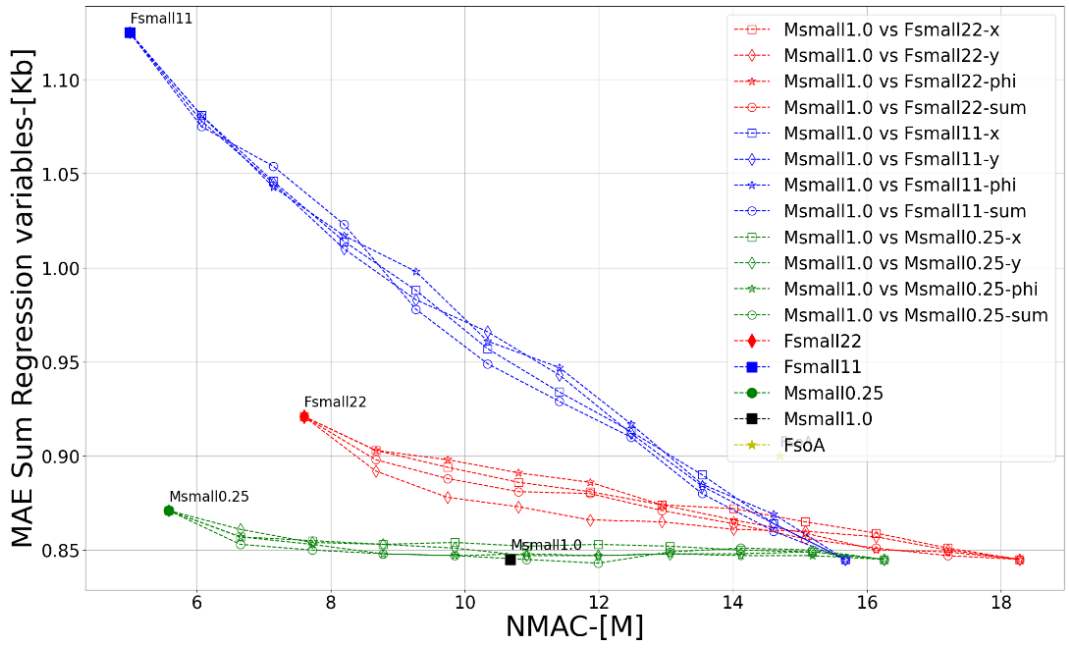
### 5.5.2 Output Metrics Vs Static Models

In this section, we evaluate the proposed methodology utilizing output decision functions, as depicted in Figures 5.12b and 5.12a. It is observed that while this approach provides the ability to control latency and computational cost, it comes at a significant compromise in terms of performance. Finding an optimal trade-off is challenging due to the default execution of the little model, which adds computational overhead to the network.

Additionally, the behavior of the different decision functions on variables such as $x$, $y$, $\Phi$, and $sum$ varies across different networks. For instance, for the $MSmall1.0$ and $Fsmall11$ network, the decision function based on $y$ error yields the best trade-off between regression performance, NMACs, and inference latency. Conversely, the $MSmall1.0$ and $Msmall0.25$ network achieves optimal performance through the decision function on sum errors. However, as discussed in the previous section (5.4.3), there exists a relationship between the variable used to construct the decision metric and the network's performance on that specific variable. While the network's predictions may improve for that particular variable, the cost of increased errors for other variables becomes evident, as the network becomes biased towards improving predictions for a single variable. In summary, the main limitations of this approach include the complexity of constructing the decision function and the overhead associated with the default execution of the little model. Despite efforts to optimize the execution plan, none of the plans outperformed the regression performance of the static big model. These findings highlight the challenges of developing an output decision function that effectively addresses a multi-regression problem, where improvements in predictions for one variable must be balanced with the overall performance across all variables.




(a) Inference cycles Vs $MAE_{Sum}$

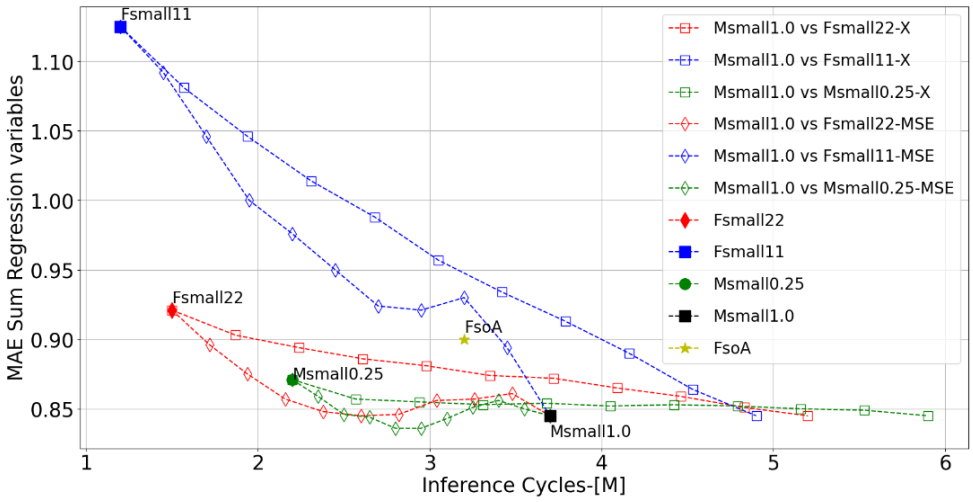



(b) NMACs Vs $MAE_{Sum}$

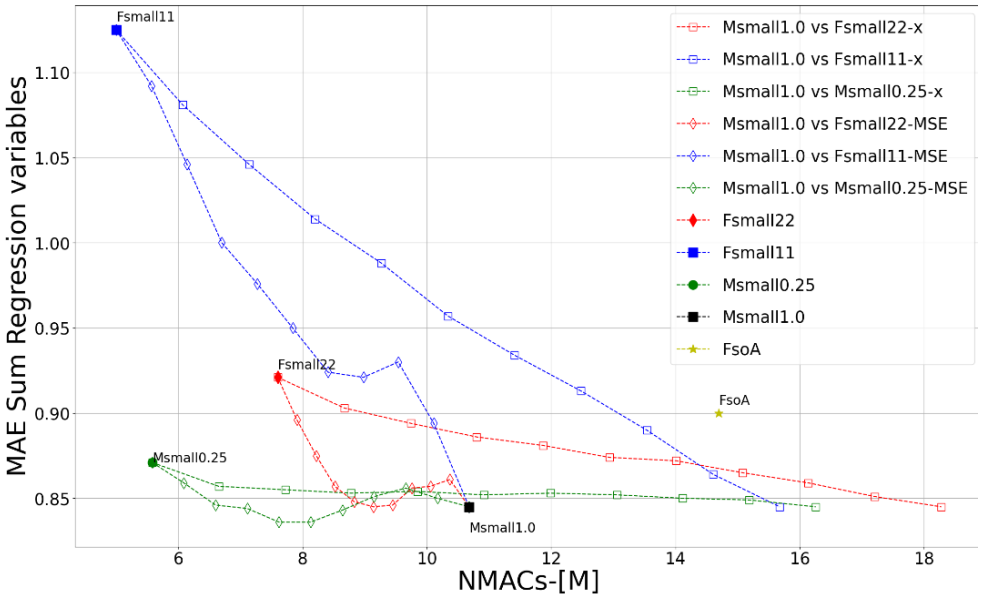
Figure 5.12: Analysis of output decision functions Vs Static Models

### 5.5.3 Input Metrics Vs output Metrics

In this section, we compare the performance of the decision function on input metrics versus output metrics in figures [5.13,5.14,5.15,5.16].

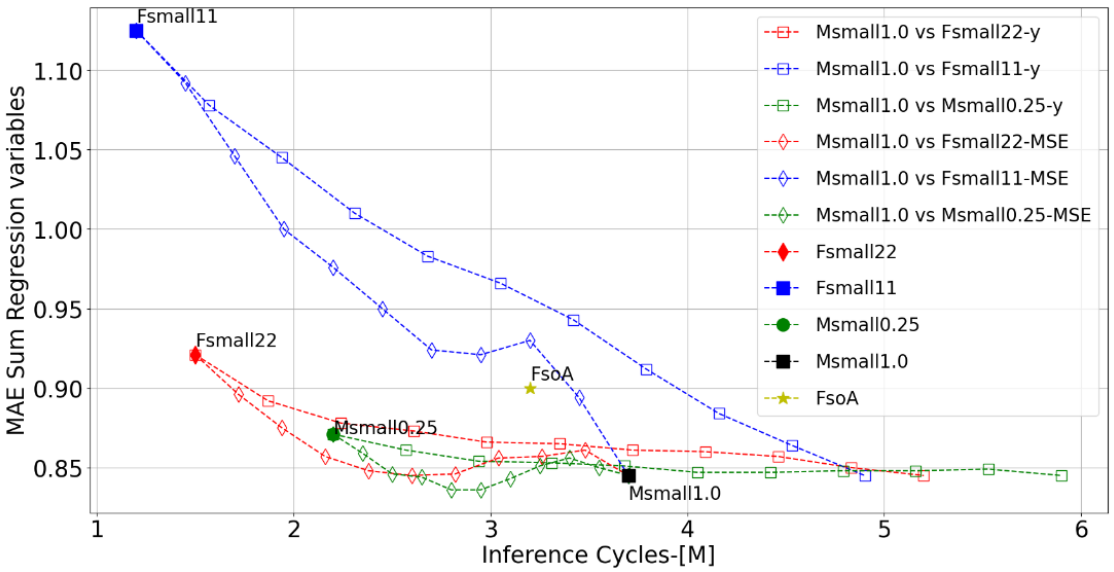



(a) Inference cycles Vs $MAE_{Sum}$

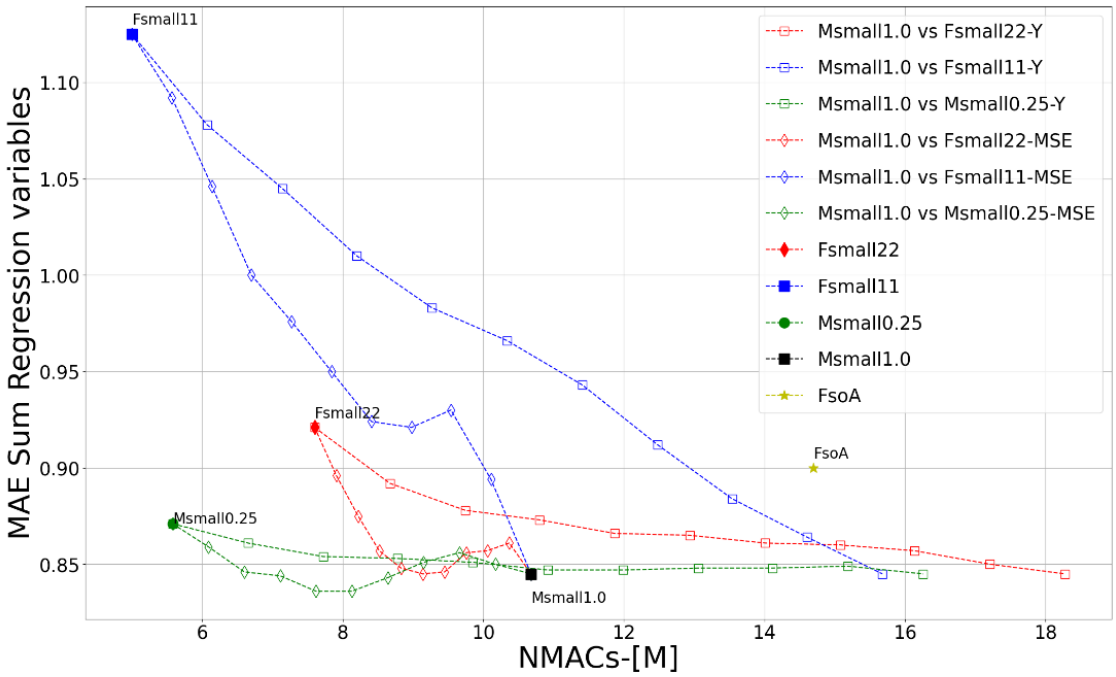



(b) NMACs Vs $MAE_{Sum}$

Figure 5.13: Analysis of input decision function (MSE) Vs output decision function on $X$

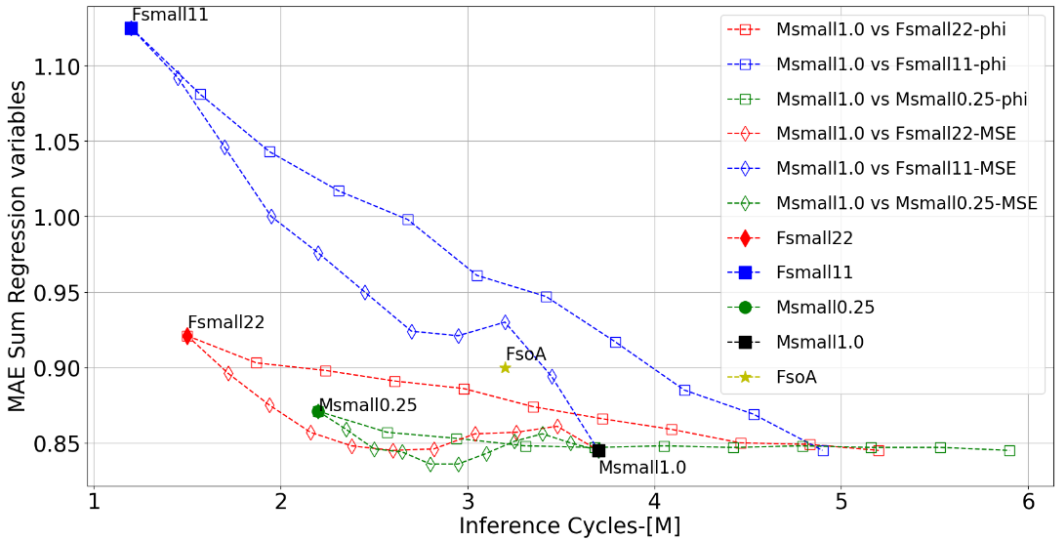



(a) Inference cycles Vs $MAE_{Sum}$

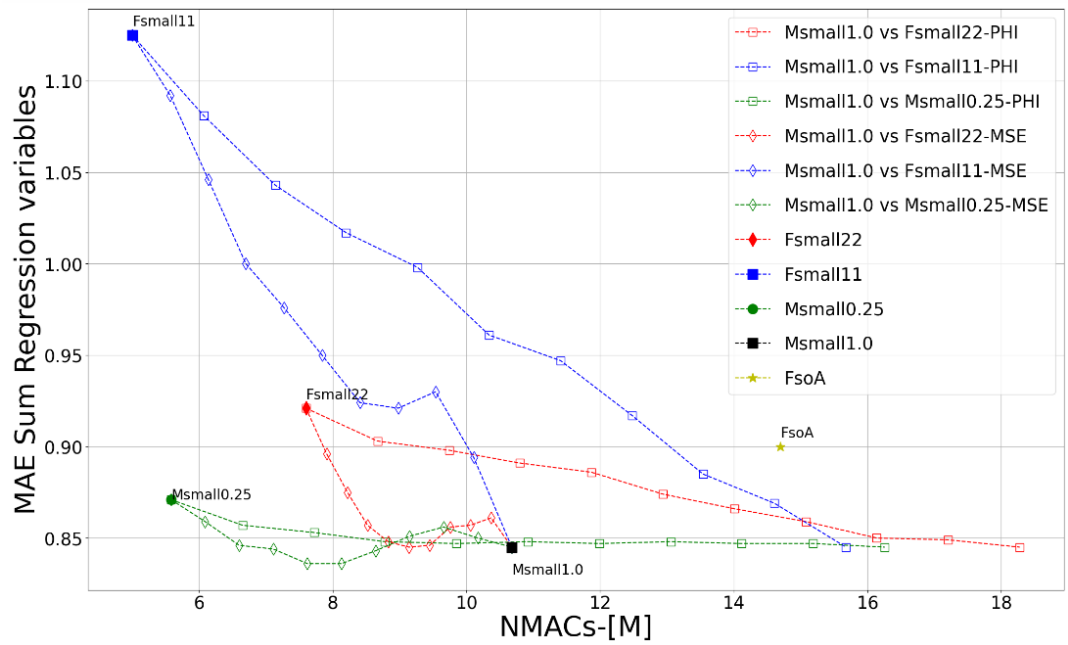



(b) NMACs Vs $MAE_{Sum}$

Figure 5.14: Analysis of input decision function (MSE) Vs output decision function on $Y$

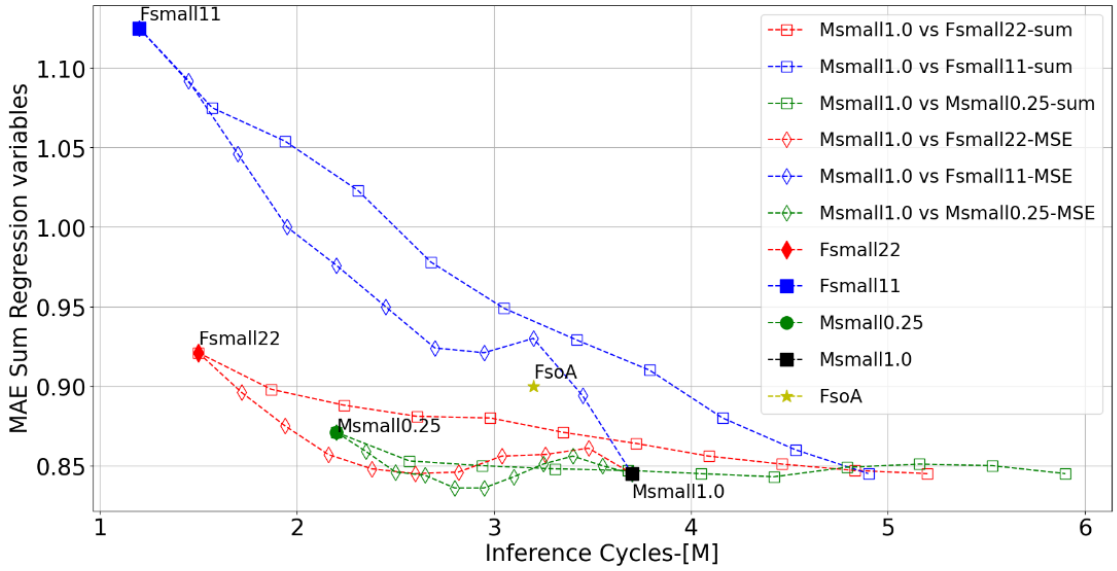



(a) Inference cycles Vs $MAE_{Sum}$

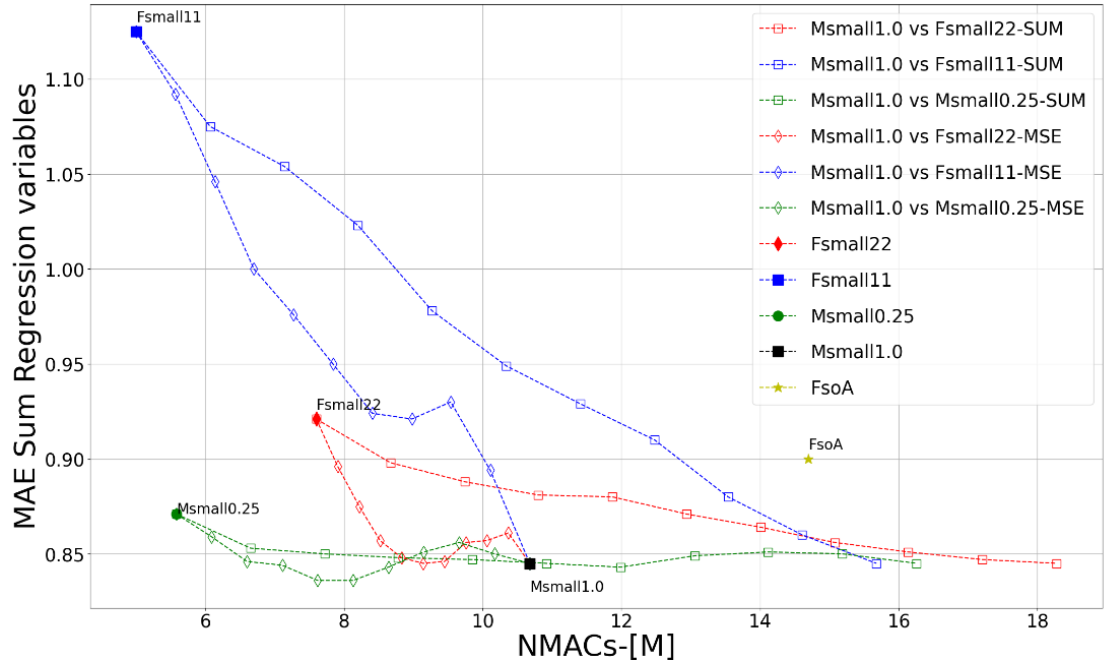



(b) NMACs Vs $MAE_{Sum}$

Figure 5.15: Analysis of input decision function (MSE) Vs output decision function on $\Phi$




(a) Inference cycles Vs $MAE_{Sum}$




(b) NMACs Vs $MAE_{Sum}$

Figure 5.16: Analysis of input decision function (MSE) Vs output decision function on *Sum* errors

The visual results indicate that, in the worst-case scenario with the input decision function, the big model is constantly executed. This execution is accompanied by the previously established negligible overhead associated with computing the metric itself. In contrast, the worst-case scenario for the output decision function involves executing the big model while considering the default execution of the little model. As a result, the overall computational cost and latency of the network experience a significant increase compared to the execution of the static big model alone.

Notably, the figures (5.13, 5.14, 5.15, 5.16) demonstrate that despite the simplicity of the Mean Squared Error (MSE) metric, it effectively distinguishes between hard and easy frames. This distinction allows for an optimal assignment of hard frames to the big model and easy frames to the little model. Consequently, this assignment strategy yields a substantial boost in network regression performance, surpassing even the performance of the big static models, while requiring less latency and computational resources. In contrast, the output decision function fails to achieve a satisfactory trade-off, even when considering the additional computations associated with this approach.

## 5.6 Overall Assessment of adaptive methods

In this final section, we present the evaluation results of adaptive Big/Little networks with input decision metrics and output decision functions, as well as the comparison with static models. As discussed in previous sections, the Big/Little approach with input decision functions demonstrated superior performance compared to other approaches. We provide a quantitative analysis of the best-performing adaptive networks and static models in table 5.2.

By employing the input decision function approach with Mean Squared Error (MSE) as the decision metric, two networks with different model combinations outperformed the most accurate static model ($MSmall1.0$). However, our main objective extends beyond improving network regression performance; we also prioritize the reduction of inference latency and computational costs. It is important to highlight that these results were achieved by employing different execution plans for the big model, showcasing the flexibility of our proposed methodology.

Compared to $MSmall1.0$, the network combination of $MSmall1.0$ and

$MSmall0.25$ achieved the largest reduction in $MAE_{sum}$, decreasing the number of Multiply and Accumulate (MAC) operations from 10.68 million to 7.62 million MAC operations, while achieving a $MAE_{sum}$ reduction to 0.836. Similarly, the network combination of $MSmall1.0$ and $Fsmall22$ achieved nearly the same $MAE_{sum}$ as $MSmall1.0$, but with a significant reduction in inference cycles from 3.7 million to 2.6 million cycles.

| Network Name | Network type | Adaptive decision Fuction | $prob_{Big}$ | $MAE_{SUM}$ | MAC Operations [M] | Inference Cycles [M] |
|---|---|---|---|---|---|---|
| MSmall0.25 | static | n.a. | n.a. | 0.871 | 5.58 | 2.2 |
| Fsmall22 | static | n.a. | n.a. | 0.921 | 5.58 | 1.5 |
| Fsmall11 | static | n.a. | n.a. | 1.125 | 5 | 1.2 |
| MSmall1.0 | static | n.a. | n.a. | 0.845 | 10.68 | 3.7 |
| FrontNet (SoA) | static | n.a. | n.a. | 0.9 | 14.7 | 3.2 |
| MSmall1.0 \| Fsmall22 | adaptive | Input Metric / MSE | 0.5 | 0.8450 | 9.14 | 2.6 |
| MSmall1.0 \|MSmall0.25 | adaptive | Input Metric / MSE | 0.4 | 0.8360 | 7.62 | 2.8 |

Table 5.2: Best performing Networks benchmarking results

# Chapter 6

# Conclusions and Future Work

In this thesis, we proposed a novel approach for input-dependent adaptive inference in drone pose estimation. Our task involves mapping low-resolution images to the relative pose of the subject, comprising 3D coordinates ($X$, $Y$, $Z$) and a rotation angle ($\Phi$) w.r.t the gravity Z-axis. Real-time performance is essential, considering strict constraints on computational power and latency.

We explore two approaches: Adaptive inference graphs utilizing convolutional neural networks and the Big/Little models approach. For the Adaptive inference graphs, we construct four networks with varying numbers of gates, using two seed networks (FrontNet and MobileNetV1). We conduct several experiments employing different training techniques and hyperparameters. However, the preliminary results do not yield satisfactory outcomes. Nevertheless, we consider this work as ongoing research with the potential for future improvements.

The primary contribution of our work lies in adopting the Big/Little model approach and addressing its limitations. We propose two methodologies: one utilizing an input decision function based on image similarity metrics and exploiting temporal dependencies between consecutive frames, and the other utilizing an output decision function on the predictions of the little model for consecutive frames.

We evaluate the proposed input decision function in two phases. Firstly, we evaluate its performance with random model selection, where input frames are randomly assigned to either the big or little model. We also conduct a similar evaluation for the output decision functions. Finally, we compare the

results of the input decision function with those of the output decision functions. We further compare the performance of the networks to state-of-the-art models, including FrontNet and $Msmall_{1.0}$, which are already optimized for the task at hand. The evaluation is based on the sum of mean absolute error for all regression variables ($MAE_{sum}$), as well as the reduction in latency represented by inference cycles and the computational cost measured in terms of the number of Multiply and Accumulate (NMACs).

To demonstrate the efficiency of the proposed methodology, we construct three different big/little networks, fixing one distinct big model and alternating between three little models with different characteristics. By utilizing the decision function on MSE and employing the network $Msmall_{1.0}$ and $Msmall_{0.25}$ with an execution plan ranging from 30% to 60% for the big model, we achieve comparable error rates to the static model $Msmall_{1.0}$ while reducing both latency and computational cost. With a 40% execution plan, we even exceed the regression performance of the static $Msmall_{1.0}$, resulting in a 0.015 reduction in $MAE_{sum}$ and reducing inference cycles and NMACs by approximately 25% and 29%, respectively.

The introduced input decision function effectively allocates resources by accurately distinguishing between 'hard' and 'easy' samples, assigning them to the big and little models, respectively, with a negligible computation overhead. Moreover, it addresses the limitations of the original big/little approach (output decision function), which suffers from increased latency and computational cost due to the default execution of the little model. Additionally, the proposed method offers flexibility in selecting models based on specific application requirements and target devices and enables a controllable trade-off between network performance, and inference latency. Additionally, we analyzed the effects of the decision variable used in the output decision function on the network performance of that variable. Our study highlights the challenges of designing a simple decision function that captures information about all regression variables without incurring additional computational overhead.

For future work, we plan to conduct in-field tests with different execution plans and explore dynamic threshold selection. Additionally, we aim to benchmark the results of MSE against other image similarity metrics, such as Structural Similarity Index (SSIM) and Normalized Root Mean Squared Error (NRMSE). Furthermore, we intend to continue enhancing the approach with ConvNets adaptive inference graphs.

# Bibliography


[1] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S. and Asari, V. K. [2018], 'The history began from alexnet: A comprehensive survey on deep learning approaches', *arXiv preprint arXiv:1803.01164* .

[2] Alzahrani, B., Oubbati, O. S., Barnawi, A., Atiquzzaman, M. and Alghazzawi, D. [2020], 'Uav assistance paradigm: State-of-the-art in applications and challenges', *Journal of Network and Computer Applications* **166**, 102706.

[3] Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M. and Farhan, L. [2021], 'Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions', *Journal of big Data* **8**, 1–74.

[4] Bello, I., Zoph, B., Vasudevan, V. and Le, Q. V. [2017], Neural optimizer search with reinforcement learning, *in* D. Precup and Y. W. Teh, eds, 'Proceedings of the 34th International Conference on Machine Learning', Vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 459–468.
**URL:** *https://proceedings.mlr.press/v70/bello17a.html*

[5] Burrello, A., Pagliari, D. J., Risso, M., Benatti, S., Macii, E., Benini, L. and Poncino, M. [2021], 'Q-ppg: Energy-efficient ppg-based heart rate monitoring on wearable devices', *IEEE Transactions on Biomedical Circuits and Systems* **15**(6), 1196–1209.

[6] Cereda, E., Crupi, L., Risso, M., Burrello, A., Benini, L., Giusti, A., Jahier Pagliari, D. and Palossi, D. [2023], 'Deep neural network architecture search for accurate visual pose estimation aboard nano-uavs', *arXiv e-prints* pp. arXiv–2303.

[7] Chen, J., Zhu, Z., Li, C. and Zhao, Y. [2019], Self-adaptive network pruning, *in* 'Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part I 26', Springer, pp. 175–186.

[8] Conti, F., Schilling, R., Schiavone, P. D., Pullini, A., Rossi, D., Gürkaynak, F. K., Muehlberghuber, M., Gautschi, M., Loi, I., Haugou, G. et al. [2017], 'An iot endpoint system-on-chip for secure and energy-efficient near-sensor analytics', *IEEE Transactions on Circuits and Systems I: Regular Papers* **64**(9), 2481–2494.

[9] Floreano, D. and Wood, R. J. [2015], 'Science, technology and the future of small autonomous drones', *nature* **521**(7553), 460–466.

[10] Goodfellow, I. J., Bengio, Y. and Courville, A. [2016], *Deep Learning*, MIT Press, Cambridge, MA, USA. `http://www.deeplearningbook.org`.

[11] Han, Y., Huang, G., Song, S., Yang, L., Wang, H. and Wang, Y. [2021], 'Dynamic neural networks: A survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **44**(11), 7436–7456.

[12] He, K., Zhang, X., Ren, S. and Sun, J. [2016], Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

[13] Herrmann, C., Bowen, R. S. and Zabih, R. [2020], Channel selection using gumbel softmax, *in* 'Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII', Springer, pp. 241–257.

[14] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H. [2017], 'Mobilenets: Efficient convolutional neural networks for mobile vision applications', *arXiv preprint arXiv:1704.04861* .

[15] Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L. and Weinberger, K. Q. [2017], 'Multi-scale dense networks for resource efficient image classification', *arXiv preprint arXiv:1703.09844* .

[16] Izhikevich, E. M. [2003], 'Simple model of spiking neurons', *IEEE Transactions on neural networks* **14**(6), 1569–1572.

[17] Jacobs, R. A., Jordan, M. I., Nowlan, S. J. and Hinton, G. E. [1991], 'Adaptive mixtures of local experts', *Neural computation* **3**(1), 79–87.

[18] Kong, S. and Fowlkes, C. [2019], Pixel-wise attentional gating for scene parsing, *in* '2019 IEEE Winter Conference on Applications of Computer Vision (WACV)', IEEE, pp. 1024–1033.

[19] Krizhevsky, A., Sutskever, I. and Hinton, G. E. [2017], 'Imagenet classification with deep convolutional neural networks', *Communications of the ACM* **60**(6), 84–90.

[20] Li, C., Wang, G., Wang, B., Liang, X., Li, Z. and Chang, X. [2021], Dynamic slimmable network, *in* 'Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition', pp. 8607–8617.

[21] Li, Y., Song, L., Chen, Y., Li, Z., Zhang, X., Wang, X. and Sun, J. [2020], Learning dynamic routing for semantic segmentation, *in* 'Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition', pp. 8553–8562.

[22] Liu, H., Simonyan, K. and Yang, Y. [2018], 'Darts: Differentiable architecture search', *arXiv preprint arXiv:1806.09055* .

[23] Mantegazza, D., Guzzi, J., Gambardella, L. M. and Giusti, A. [2019], Vision-based control of a quadrotor in user proximity: Mediated vs end-to-end learning approaches, *in* '2019 International Conference on Robotics and Automation (ICRA)', IEEE, pp. 6489–6495.

[24] Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R. and Muharemagic, E. [2015], 'Deep learning applications and challenges in big data analytics', *Journal of big data* **2**(1), 1–21.

[25] Palossi, D., Loquercio, A., Conti, F., Flamand, E., Scaramuzza, D. and Benini, L. [2019], 'A 64-mw dnn-based visual navigation engine for autonomous nano-drones', *IEEE Internet of Things Journal* **6**(5), 8357–8371.

[26] Palossi, D., Zimmerman, N., Burrello, A., Conti, F., Müller, H., Gambardella, L. M., Benini, L., Giusti, A. and Guzzi, J. [2021], 'Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs', *IEEE Internet of Things Journal* **9**(3), 1913–1929.

[27] Park, E., Kim, D., Kim, S., Kim, Y.-D., Kim, G., Yoon, S. and Yoo, S. [2015], Big/little deep neural network for ultra low power inference, *in* '2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)', IEEE, pp. 124–132.

[28] Price, H. [2018], 'Federal aviation administration (faa) forecast fiscal years 2017-2038', *Accessed: Feb* **25**, 2019.

[29] Qiu, T., Chi, J., Zhou, X., Ning, Z., Atiquzzaman, M. and Wu, D. O. [2020], 'Edge computing in industrial internet of things: Architecture, advances and challenges', *IEEE Communications Surveys & Tutorials* **22**(4), 2462–2488.

[30] Risso, M., Burrello, A., Conti, F., Lamberti, L., Chen, Y., Benini, L., Macii, E., Poncino, M. and Pagliari, D. J. [2022], 'Lightweight neural architecture search for temporal convolutional networks at the edge', *IEEE Transactions on Computers* .

[31] Sarker, I. H. [2021], 'Machine learning: Algorithms, real-world applications and research directions', *SN computer science* **2**(3), 160.

[32] Scaramuzza, D., Achtelik, M. C., Doitsidis, L., Friedrich, F., Kosmatopoulos, E., Martinelli, A., Achtelik, M. W., Chli, M., Chatzichristofis, S., Kneip, L., Gurdan, D., Heng, L., Lee, G. H., Lynen, S., Pollefeys, M., Renzaglia, A., Siegwart, R., Stumpf, J. C., Tanskanen, P., Troiani, C., Weiss, S. and Meier, L. [2014], 'Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments', *IEEE Robotics & Automation Magazine* **21**(3), 26–40.

[33] Shakhatreh, H., Sawalmeh, A. H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N. S., Khreishah, A. and Guizani, M. [2019], 'Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges', *IEEE Access* **7**, 48572–48634.

[34] Shalev-Shwartz, S. and Ben-David, S. [2014], *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge, UK.

[35] Simonyan, K. and Zisserman, A. [2014], 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556* .

[36] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. [2015], Going deeper with convolutions, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1–9.

[37] Teerapittayanon, S., McDanel, B. and Kung, H.-T. [2016], Branchynet: Fast inference via early exiting from deep neural networks, *in* '2016 23rd International Conference on Pattern Recognition (ICPR)', IEEE, pp. 2464–2469.

[38] Tezza, D. and Andujar, M. [2019], 'The state-of-the-art of human–drone interaction: A survey', *IEEE Access* **7**, 167438–167454.

[39] Ting, D. S., Peng, L., Varadarajan, A. V., Keane, P. A., Burlina, P. M., Chiang, M. F., Schmetterer, L., Pasquale, L. R., Bressler, N. M., Webster, D. R. et al. [2019], 'Deep learning in ophthalmology: the technical and clinical considerations', *Progress in retinal and eye research* **72**, 100759.

[40] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. [2017], 'Attention is all you need', *Advances in neural information processing systems* **30**.

[41] Veit, A. and Belongie, S. [2018], Convolutional networks with adaptive inference graphs, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 3–18.

[42] Wang, X., Yu, F., Dou, Z.-Y., Darrell, T. and Gonzalez, J. E. [2018], Skipnet: Learning dynamic routing in convolutional networks, *in* 'Proceedings of the European Conference on Computer Vision (ECCV)', pp. 409–424.

[43] Wang, Z., Bovik, A. C., Sheikh, H. R. and Simoncelli, E. P. [2004], 'Image quality assessment: from error visibility to structural similarity', *IEEE transactions on image processing* **13**(4), 600–612.

[44] Wu, Z., Nagarajan, T., Kumar, A., Rennie, S., Davis, L. S., Grauman, K. and Feris, R. [2018], Blockdrop: Dynamic inference paths in residual networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 8817–8826.

[45] Yang, L., Han, Y., Chen, X., Song, S., Dai, J. and Huang, G. [2020], Resolution adaptive networks for efficient inference, *in* 'Proceedings of the IEEE/CVF conference on computer vision and pattern recognition', pp. 2369–2378.

[46] Yuan, J., Liu, M., Tian, F. and Liu, S. [1912], 'Visual analysis of neural architecture spaces for summarizing design principles', *IEEE Transactions on Visualization and Computer Graphics* .

[47] Zhang, K., Zhang, Z., Li, Z. and Qiao, Y. [2016], 'Joint face detection and alignment using multitask cascaded convolutional networks', *IEEE signal processing letters* **23**(10), 1499–1503.