UNIVERSIDADE D
COIMBRA

Pedro António Ferreira Diamantino Coelho e Silva

Edge Computing to Enable Onboard Computation of Deep Learning Algorithms in Service Robots

Dissertation supervised by Prof. Rui Paulo Pinto da Rocha and submitted to the Department of Electrical and Computer Engineering of the Faculty of Sciences and Technology of the University of Coimbra, in partial fulfillment of the requirements for the Master Degree in Electrical and Computer Engineering, specialization in Computers

January, 2022

· U C ·

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Edge Computing para Potenciação de Aprendizagem Profunda a Bordo de Robôs de Serviço

Pedro António Ferreira Diamantino Coelho e Silva

Coimbra, Janeiro 2022

Abstract

This dissertation aims to research the possibility of implementing and executing Artificial Intelligence (AI) methods on service robots previously developed at the Instituto de Sistemas e Robótica - Universidade de Coimbra (ISR-UC). These systems have strict energy requirements in order to sustain services for long periods of time before they need recharged batteries. As such, the implementation of AI methods in general purpose computer architectures implies diverse constraints and might not be viable.

A cloud computing approach, that is, the relocation of local data and processing to large, remote data centers with more powerful computing capabilities, still implies heavy power consumption due to the use of networking resources as well as a higher latency. As such, we choose an edge computing paradigm, i.e. the processing of data as close as possible to the place in which data is acquired and results are needed.

To implement this edge computing approach, a system was implemented composed by the System on Module Jetson AGX Xavier, designed and manufactured by NVIDIA, a stereo camera, a laser rangefinder and a mobile robot. This system aims to verify public health measures vital to the COVID-19 pandemic using AI-based models, namely correct facial mask usage and social distancing, while maintaining low power consumption required by mobile platforms.

Resumo

Esta dissertação visa investigar a implementação e execução local de métodos de Inteligência Artificial (IA) em robôs de serviço do Instituto de Sistemas e Robótica - Universidade de Coimbra (ISR-UC). Estes sistemas têm requisitos de autonomia energética por forma a se manterem operáveis durante longos períodos de tempo, antes de necessitarem de recarregar a bateria. Como tal, o uso de inteligência artificial na arquitectura implementada de Central Processing Unit (CPU) + Graphics Processing Unit (GPU) implica diversos constrangimentos técnicos, incluindo energéticos, e pode ser mesmo inviável.

Uma abordagem usando o paradigma de *cloud computing*, i.e. o envio de dados recolhidos localmente para servidores remotos com mais recursos computacionais, implica custos pesados de consumo de energia devido ao uso de recursos de rede bem como maior latência. Como tal, investigaremos uma abordagem de *edge computing*, i.e. o processamento de dados o mais perto possível do local em que o resultado será utilizado.

Para a implementação da abordagem de *edge computing* foi construído um sistema composto pelo *System on Module* Jetson AGX Xavier, produzido pela NVIDIA, uma câmara estereo, um *laser rangefinder* e robô móvel. Este sistema tem como objectivo a verificação de normas de saúde públicas vitais durante a situação pandémica de COVID-19 recorrendo ao uso de modelos baseados em IA, nomeadamente o uso correcto de máscaras faciais e distânciamento social, proporcionando um consumo de potência sustentável numa plataforma robótica móvel.

Contents

List of Acronyms

AI Artificial Intelligence

AMCL Adaptive Monte Carlo Localization

ANN Artificial Neural Network

AP Average Precision

AUC Area Under ROC Curve

COCO Common Objects in COntext

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma-Separated Value

DL Deep Learning

DLA Deep Learning Accelerator

DNN Deep Neural Network

FN False Negative

FP False Positive

FOV Field of View

FPS Frames Per Second

GPU Graphics Processing Unit

IA Inteligência Artificial

IOU Intersection Over Union

ISR-UC	Instituto de Sistemas e Robótica - Universidade de Coimbra
LAN	Local Area Network
mAP	Mean Average Precision
ML	Machine Learning
NVDLA	NVIDIA Deep Learning Accelerator
R-CNN	Regions with CNN features
ROC	Receiver Operating Characteristic
ROS	Robot Operating System
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
SOM	System on Module
TN	True Negative
TP	True Positive
USB	Universal Serial Bus
VNC	Virtual Network Computing

List of Figures

1 Introduction

1.1 Context and Motivation

Advancements in the field of Artificial Intelligence (AI) over the past decade have enabled a vast variety of services and capabilities, such as object detection [8], face detection [9] and recognition [10], sound recognition [11], among others. These are interesting services to be implemented on and applied by service robots, defined by the International Organization for Standardization as a robot “that performs useful tasks for humans or equipment excluding industrial automation applications” [12].

Machine Learning (ML) is a subset of the field of AI and can be defined as the study of computer algorithms that automatically improve their performance, measured by some metric, through experience [13]. Deep Learning (DL) is then a subset of the field of ML and is based on the use of Artificial Neural Networks (ANNs), originally developed to mimic biological intelligence via the concepts of neurons, layers and how these are connected [14]. By connecting multiple layers of neurons, a Deep Neural Network (DNN) is created.

Graphics Processing Units (GPUs) are electronic circuits with a structure highly specialized towards parallel computing, which enable computations where calculations can be done in parallel and without logical dependencies. GPUs have seen a rapid development and are capable of accelerating parallelizable algorithms for ML applications [15] with more and more efficiency. However, their form factor and power consumption turns them inadequate for inclusion on board mobile service robots and in other embedded systems with power and size constraints.

Consequently, to implement AI methods on service robots, it is typical to resort to a *cloud computing* paradigm. Data is gathered in the environment in which it is produced then transmitted via a Local Area Network (LAN) or the Internet to a remote computational service, commonly called the cloud, with greater computational resources such as those provided by Amazon Web Services, Microsoft Azure and Google Cloud, among others. The transmitted data is then processed and results are transmitted back to the deployed system. However, this approach presents several shortcomings: higher latency, unsustainable to real-time based applications; the network's quality of service can be insufficient to the required reliability; a lack of data encryption can compromise the privacy and security of sensitive data [16]; an outage from the service provider will jeopardize the application's quality or functionalities [17]; among others.

In contrast, *edge computing* is a distributed computing paradigm designed to process and store data closer to the physical location where it is needed [18]. This paradigm decentralizes the processing of data, reducing data latency and required bandwidth. However, heavy processing of data could prove to be unfeasible as deployed systems could not have appropriate resources for the task.

1.2 Problem statement

Since the start of the COVID-19 pandemic, DL applications have been developed to evaluate public health measures such as social distancing [19]. However, the use of a GPU to accelerate the execution of these networks presents problems for mobile robots: excessive power consumption, inappropriate dimensions and weight. This dissertation aims to exploit an edge computing paradigm in service robots, and in embedded systems in general, in order to overcome these technical limitations encountered by GPU-based general purpose computer architectures.

To implement an edge computing based solution, the NVIDIA System on Module (SOM) Jetson AGX Xavier [20] provides powerful computational capabilities and was designed for edge-based AI applications. With this device, the computational power required for AI on service robots is achieved at a low energy expenditure with results obtained in real time.

Besides the NVIDIA SOM Jetson AGX Xavier, the following equipment was used in this dissertation: a Pioneer P3-DX mobile robot to be used as a service robot and a ZED 2 stereo camera. This hardware suite was integrated in a service robot controlled by the Jetson AGX Xavier as a case study for the use of the edge computing paradigm in an application requiring DL algorithms on board service robots, as well as the study of the implementation of DL algorithms on different computer architectures.

1.3 Objectives and contributions

The overall goal of this dissertation is to engineer a service robot based on the edge computing paradigm, capable of executing DL algorithms and other heavy computations required by AI methods, as well as localization, mapping and navigation algorithms commonly used on autonomous mobile robots, to be used and replicated in future research projects at the Instituto de Sistemas e Robótica - Universidade de Coimbra (ISR-UC). The use of a non-traditional computer architecture for robotic embedded systems aims to overcome current limitations posed by mobile robots employing general purpose computers (e.g., laptops) and to find a more practical trade-off between computational power and energy consumption, increasing available computational resources on service robots, while avoiding prohibitive power consumption from the robot's batteries.

This overall goal involves the following objectives:

- To carry out a comparative evaluation in terms of available computational resources and energy consumption of a general purpose architecture versus a SOM-based architecture in object detection using images;
- To study deep learning algorithms, their computational requirements and development frameworks;
- To develop a prototype system based on the integration of a SOM device and a stereo camera on a Pioneer P3-DX robot platform, employing the Robot Operating System (ROS) middleware to build a service robot capable of performing autonomous navigation and advanced perception tasks based on deep learning techniques;
- To perform a proof of concept by using the SOM-based service robot prototype to ensure COVID-19 prevention measures, namely wearing a mask and social distancing, within an open laboratory space common to several researchers.

1.4 Outline of the dissertation

This dissertation is composed by six chapters, each detailing different parts of the work. Chapter 1, introduces this project, the social and academic context it is inserted in, as well as the motivations behind it. Chapter 2 provides a brief description of fundamental concepts and a review of the state of the art in the area of DL applied to computer vision. Chapter 3 elaborates on the computational resources of the Jetson AGX Xavier as well as a comparison of various performance metrics against other devices in computational loads related to DL based functionalities, in order to contextualize the Xavier and its potential role for service robotics. Chapter 4 describes the system proposed in order to evaluate the assertion of this dissertation – the implementation of deep learning based services on board service robots – containing a review of the hardware and software elements as well as a review of relevant state-of-the-art solutions. The proposed system prototype is presented and discussed in Chapter 5 as well as a description of the AI model developed. Chapter 6 provides a summary of the work and a conclusion of this project, while providing a base for future projects that could be implemented at the ISR-UC in an academic context.

2 Background and Related Work

This chapter provides a general overview of fundamental concepts in the field of DL and of the robotics applications to be implemented over the course of this dissertation work.

2.1 Cloud computing versus edge computing

Dr. Karim Arabi, an Engineer and Vice-President at Qualcomm, defined edge computing as *"all computing outside the cloud happening at the edge of the network and more specifically in applications where real-time processing of data is required"*. Dr. Karim Arabi further defines cloud computing as operating on big data while edge computing operates on instant data, that is, real-time data generated by sensors or users [21].

Cloud computing is a computational model designed to enable ubiquitous, convenient and on-demand network access to a shared pool of diverse computational resources that can be quickly distributed with minimum effort [22]. A cloud computing model is defined by the following characteristics:

- On-demand self-service. A consumer can unilaterally and autonomously request and obtain computational resources such as storage, processing, memory among others, without human interaction with the service provider;
- Broad network access. Computational resources can be accessed from anywhere by a diversity of platforms, such as laptops, mobile phones or workstations;
- Resource pooling. Computational resources are pooled in such a way as to serve multiple consumers, being dynamically allocated according to demand;

- Rapid elasticity. Resources can be elastically provided and released, even automatically, in order to rapidly scale to consumer demand;
- Measured service. Resources are typically controlled and optimized by the cloud system on a pay-per-use basis. Usage can be monitored and controlled to both the consumer and the provider of the service.

Despite its advantages, this computational paradigm has also some shortcomings and issues, namely security vulnerabilities, higher latency, data privacy, a recurring monetary model, constant and frequent changes to service level agreements among others [23]. Consequently, despite offering the capability to use DL in mobile devices and robots, a cloud computing approach is sometimes ill suited for these systems due to the unreliability and latency of network connections in robotics applications.

A competing approach is the edge computing paradigm, aiming to compute data as closely as possible to the location in which it is generated using resources located at the edge of the Internet, such as mobile phones or laptops, in order to reduce the volume and distance that data must travel [18]. This approach is not heavily reliant on network connections but computational resources at the edge are not guaranteed to be capable to comply with the requirements demanded by mobile robotics applications. To enable this approach, there has been an investment on the development of embedded computing boards, or System on Modules (SOMs), capable of providing heavy computational resources such as GPU acceleration in field applications.

2.2 NVIDIA Jetson family

The NVIDIA Jetson devices are a set of SOM devices designed for edge computing. The devices feature a software stack designed to immediately enable the development and deployment of various computational tasks, such as image processing with OpenCV, DL runtime inference acceleration with TensorRT, CUDA enabled GPU acceleration and the Vision Programming Interface library for computer vision or image processing algorithms on CPU or accelerated by GPU or Programmable Vision Accelerator. These devices are developed to enable AI applications based on the edge computing paradigm and their use in robotic systems has begun to be adopted with promising results [24].

The NVIDIA Jetson family includes:

- Jetson Nano, the entry-level model;
- Jetson TX2;
- Jetson Xavier NX;
- Jetson AGX Xavier, the higher end model.

Among these devices, the Jetson AGX Xavier is the most powerful one and the most interesting case study for DL-based applications on board service robots.

2.2.1 NVIDIA Jetson AGX Xavier

The nature of the NVIDIA Jetson devices aligns with the goals and objectives of this dissertation, to enable DL applications on mobile robotics employing the edge computing paradigm. The NVIDIA Jetson AGX Xavier features a 512 core NVIDIA Volta GPU architecture and an 8 core ARM v8.2 64-bit Central Processing Unit (CPU). Other noticeable features are the modules dedicated to the acceleration of DL and Computer Vision applications, two NVIDIA Deep Learning Accelerator (NVDLA) and two Vision Accelerators. However, Bluetooth and WiFi are not natively implemented, complicating the interface with external devices and systems, and the 32GB eMMC storage is a limitating factor due to the memory that neural network weight files commonly require (YOLOv3 requires 248MB to store its weights for example). Furthermore, two USB 3.0 ports is typically insufficient for robotic applications as connections to mobile platforms, stereo cameras, laser range finders among other common hardware interfaced via USB 3.0.

In sum, the NVIDIA Jetson AGX Xavier has a powerful computer architecture capable of executing DL tasks on the edge [25] with low power consumption. However, the limited storage memory and number of USB 3.0 ports are limitations and must be taken in consideration during system design.

Figure 2.1: Jetson AGX Xavier Developer Kit.

2.3 Deep Learning

AI is a hard to define concept and it can be difficult to classify something as AI or not. Shane Legg and Marcus Hutter, leading machine learning researchers at Google's DeepMind Technologies, offer the following definition:

> "Intelligence measures an agent's ability to achieve goals in a wide range of environments." [26]

Machine Learning is defined by Tom Mitchell, researcher at the Carnegie Mellon University's Machine Learning Department, as follows:

> "Machine Learning is the study of computer algorithms that improve automatically through experience." [13]

DL is then defined as a field of Machine Learning, referring to the multi-layer networks applied. These models are based on Artificial Neural Networks (ANNs), initially developed by Warren McCulloch and Walter Pitts in 1943 [14], in an attempt to mimic artificial intelligence through the concepts of neurons and connections.

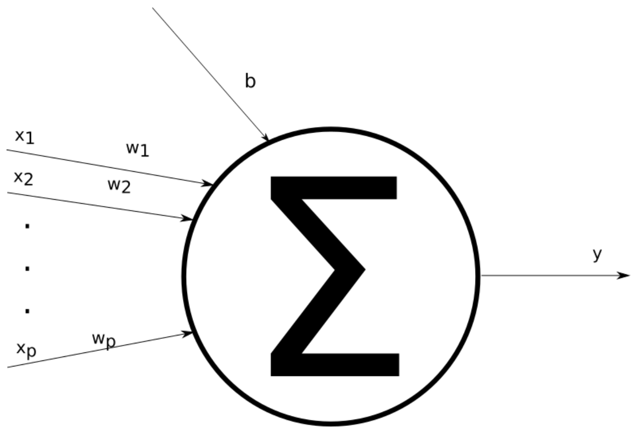


Figure 2.2: Linear Neuron.

2.3.1 Neurons, Layers and Networks

An artificial neuron is the basic element of ANNs. It is a mathematical function modelled after biological neurons. A linear neuron is composed by a set of n inputs $(x_1, ..., x_n)$, a set of n weights associated to these inputs and a scalar value b, a bias. The output of a neural network is a weighted sum y defined as:

$$y = \sum_{i=0}^{n} (x_i w_i) + b \tag{2.1}$$

Artificial neurons can be arranged into groups, defined as layers. The use of multiple layers, where the output of one layer is the input of the layer following it, provides the basis for Deep Neural Networks. Data is processed by being inserted in the input layer, and the results of that layer are then fed forward to the proceeding layers. Results can then be collected at the output layer. A visual representation of this can be seen in Figure 2.3. Each output neuron represents a class and the value computed on a neuron represents a confidence score. For example, in the case of image classification, the confidence score of each output neuron represents how confident the model is that the image contains an object of the class represented by that neuron.

The end goal for a neural network is to develop a model capable of predicting a certain quality based on the data used to train the network. The quality to predict is the label and the input variables are the features. Confidence scores are typically normalized in a range of $[0, 1]$. By setting a value, a confidence threshold, it is possible to interpret the results of the output network, for instance, a network detects an object when the confidence score is greater than 0.5.

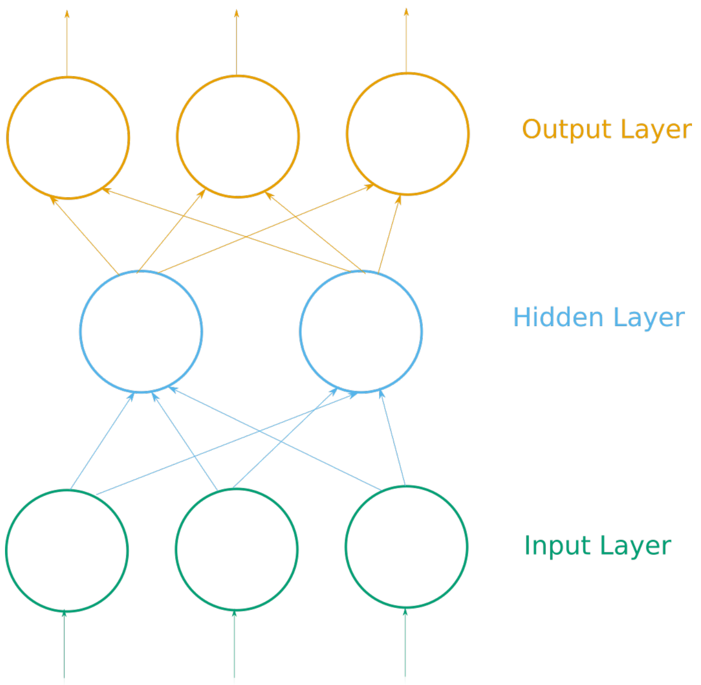


Figure 2.3: Three layered Neural Network.

Training a model is the gradual adjustment of every neuron's set of weights and bias in order to infer qualities correctly. To measure the quality of a model's predictions, its loss function is studied. There are many variations of loss functions, such as L_1 and L_2 losses:

- Mean Absolute Error, $L_1 = \frac{1}{N} \sum_{(x,y)\in D} (y - y')$,
- Mean Square Error, $L_2 = \frac{1}{N} \sum_{(x,y)\in D} (y - y')^2$,

wherein x is the dataset's feature, y is the known label, y' is the predicted value of the model and D is the training dataset.

However, a combination of linear neurons assumes a linear model, and it is common the need to predict a non-linear quality. In this case, a linear model cannot minimize the loss functions and has unsatisfying results. In order to imbue the model with non-linearity, each neuron's weighted sum output is put through an activation function, σ. Thus, the final output of a linear neuron, a, is the weighted sum, z, through the activation function:

$$a = \sigma(z) = \sigma(\sum_{i=0}^{n}(x_i w_i) + b). \tag{2.2}$$

The mathematical properties of this activation function, σ, such as monotonicity or continuous differentiability, will impact the performance of the model. However, according to the *No Free Lunch Theorem* [27], it is impossible to, *a priori*, compare the performance of any two activation functions.

2.3.2 Data Augmentation

A model's performance is dependent on variation in data used for its training. The more varied the training dataset is, the more robust the model is and is capable of generalizing, providing better predictions to a wider range of data that was not part of the training dataset.

Examples of data augmentation are cropping, scaling, flipping, color distortion among others. The data augmentation strategy must be chosen carefully. For example, in a dataset of cars it is possible to augment it by introducing data (images) with distorted colors or flipped by the vertical axis, as cars generally exist with a variety of colors and can be oriented in many directions, as shown in Figure 2.4. However, to introduce data augmentation via color distortion to a training dataset designed to detect flowers could lead to decreased performance, as shown in Figure 2.5, as most flowers tend to be a part of a strictly defined set of colors, no blue roses exist for example. The use of translation, rotation and flipping for data augmentation must also be judiciously used, as shown in Figure 2.6. While cars can be oriented in many directions, stop signs exist in a strict orientation.

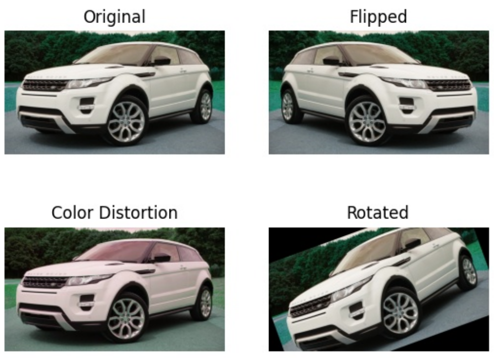


Figure 2.4: Data augmentation. The car image is resilient to flipping, slight rotation and slight color distortion.

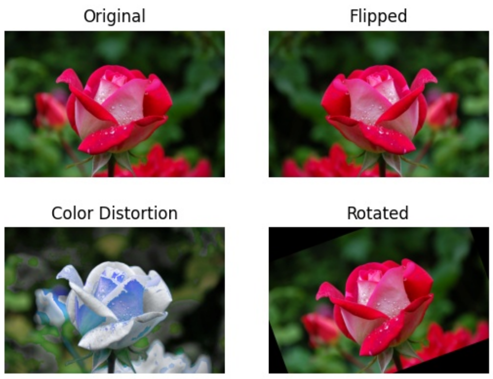


Figure 2.5: Data augmentation. The image of a rose, despite being resilient to affine transformations, is not resilient to color distortions.

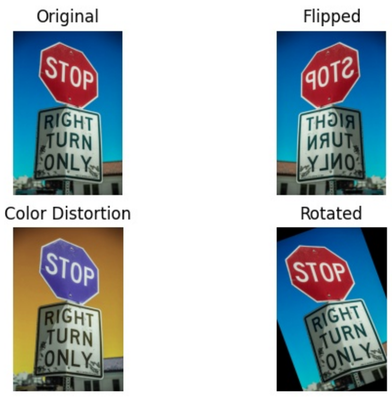


Figure 2.6: Data augmentation. The image of a stop sign is resilient to slight rotations, but isn't resilient to flipping. No potS signs exist and as such cannot be generalized.

2.3.3 Quality of predictions

The quality of a model is commonly defined according to its Mean Average Precision (mAP). A model's predictions can be True Positive (TP), True Negative (TN), False Positive (FP) or False Negative (FN). The ratios of these predictions will define the model's quality according to its Accuracy and Precision. A model's accuracy is defined as

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total number of predictions}}. \tag{2.3}$$

A model with accuracy 1 has no false positives nor false negatives. Its precision, describing the proportion of correct predictions from all predictions, is defined as

$$\text{Precision} = \frac{\text{True Positives}}{\text{TP} + \text{FP}}. \tag{2.4}$$

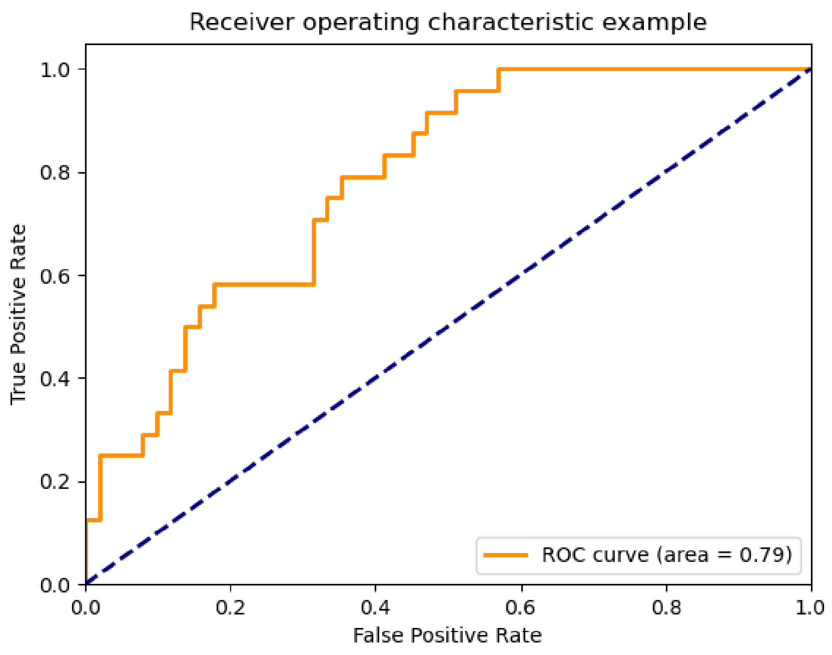


Figure 2.7: Receiver Operating Characteristic Curve example [1].

So a model that produces no false positives has a precision of 1. Another relevant metric is a model's recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.5}$$

A model producing no false negatives has a recall of 1.

An ideal model has a value of 1 for each of these metrics. However, actions to increase a model's precision tend to decrease its recall and vice-versa. If the confidence threshold increases in order for the model to produce less false positives, the number of false negatives tends to increase, increasing precision but decreasing recall. If the confidence score is decreased in order for the model to produce less false negatives, the number of false positives tends to increase, increasing recall but decreasing precision.

A Receiver Operating Characteristic (ROC) curve plots a model's performance for various classification thresholds as seen on Figure 2.7. It plots two paramenters: True Positive Rate and False Negative Rate. These values can be defined as follows:

$$\text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{2.6}$$

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{2.7}$$

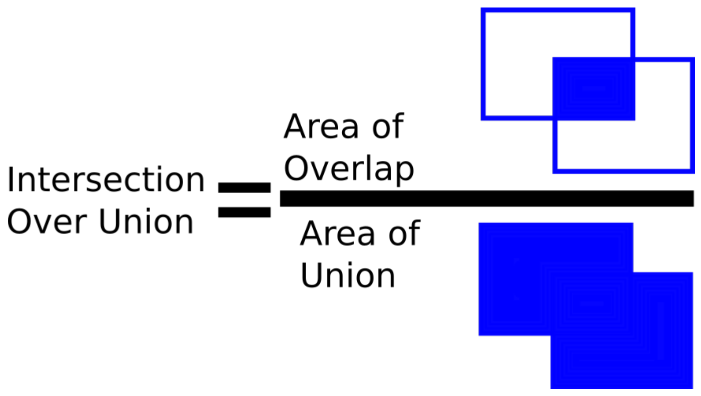


Figure 2.8: Intersection Over Union.

To qualify different ROCs performances, it is common to use the Area Under ROC Curve (AUC) metric. This metric represents the probability, between $[0, 1]$, that a random positive example will score higher than a random negative example, producing more true positives and less false positives. A model with AUC of 0 has strictly incorrect predictions and a model with AUC of 1 has strictly correct predictions.

A common metric to qualify a model's performance for object detection tasks is the Intersection Over Union (IOU). To compute this metrics, two things are required: the ground truth bounding box of the object and the model's bounding box prediction of the object. If the two bounding boxes overlap, the model has detected the object perfectly according to the ground truth. If the boxes have no overlap, the model has failed to detect the object. This metric is common for the evaluation of predictions by YOLO models, to be discussed in subsection 2.4.3.

Average Precision (AP) summarizes a ROC curve as the weighted mean of precisions achieved at each threshold, or IOU, where each precision is weighted by the increase in recall from the previous threshold,

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n, \tag{2.8}$$

where P_n and R_n are the precision and recall at the *nth* threshold. Calculating the AP at various values for IOU for each class of the model, it is possible to compute the mAP. This metric is commonly applied for challenges and competitions for deep learning applied to computer vision, such as COCO [28] or PASCAL [29].

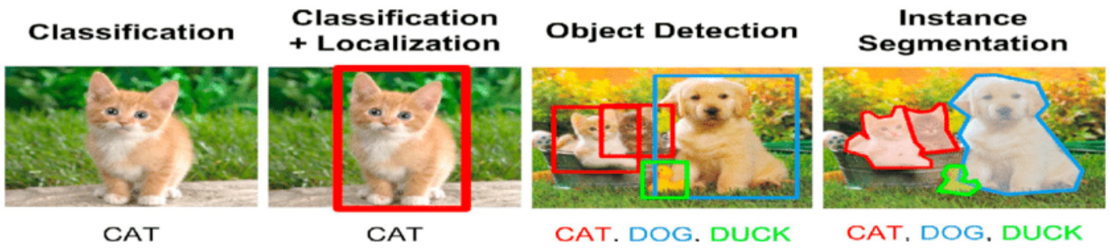


Figure 2.9: Classification, Detection and Segmentation [2].

2.4 Deep Neural Networks Applied to Computer Vision

Computer Vision is a branch of computer science dedicated to the study of how computers can obtain an understanding of digital imagery with sub-fields such as object detection, image restoration and 3D pose estimation with applications in medicine [30], industry [31] and military [32]. DNNs, specifically Convolutional Neural Networks (CNNs), have been applied to computer vision with main applications in image classification, detection and segmentation.

Image classification is the task of assigning a label to an input image from a fixed set of classes. No information on the position of the object in the image is output. Image detection is the labelling of one or more objects in an image as well as providing coordinates and an area of where the object is in the input image. Image segmentation is the partitioning of an image into multiple sets of pixels, or segments, where each set represents an object or feature in the image. The computational complexity of each task increases with the amount of information extracted. That is, image segmentation is more complex than image detection which is more complex than image classification.

2.4.1 AlexNet

AlexNet, the winner of the 2012 ImageNet contest, features 60 million parameters and 650,000 neurons. The size of this network would make conventional training of the network prohibitive but Kriezhevsky et al. circumvented this issue with the use of GPUs to accelerate training [3]. The success of AlexNet provided a fundamental basis for the use of deeper and bigger CNNs trained on large datasets and for their use on larger scale and higher resolution images. AlexNet is influential in computer vision, and as such must be included in any revision of the state of the art in computer vision.

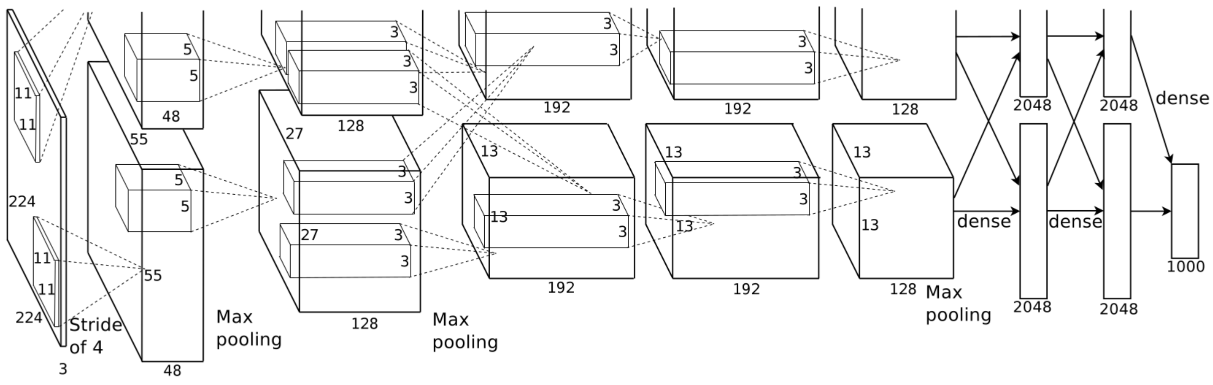


Figure 2.10: AlexNet architecture [3].

The network is composed by eight layers, five convolutional layers and three fully connected layers, with the output of the last layer feeding into a 1000-way softmax layer, producing a distribution of confidence values for each possible output label. The first convolutional layer filters the input image, with dimensions $224 \times 224 \times 3$ with 96 kernels of size $11 \times 11 \times 3$. The output of this first layer is fed into the input of the second convolutional layer and is filtered by 256 kernels of dimensions $5 \times 5 \times 48$. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$ and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$.

2.4.2 R-CNN

Regions with CNN features (R-CNN) employs three modules for object detection. The first generates a set of class-independent region proposals, where the likelihood of an object exists is high. The second module is a large CNN designed to extract a fixed length feature vector from each candidate region. The third module is a set of class specific linear Support Vector Machine, associating each candidate region with an object class. A 4096-dimensional feature vector is extracted from each candidate region employing a Caffe implementation of the CNN employed on AlexNet for feature extraction, requiring image data in the candidate region to be resized to a dimension of 227×227 pixels.

Employing a large CNN requires a vast amount of training data that is usually insufficient. A common solution for this problem is to apply unsupervised learning, where the classes of the training dataset are unknown. Girshick et al. employ a supervised pretraining on a large auxiliary dataset and complement it with domain-specific fine-tuning on a small dataset, concluding that this approach improves mAP performance by 8 percentage points, scoring

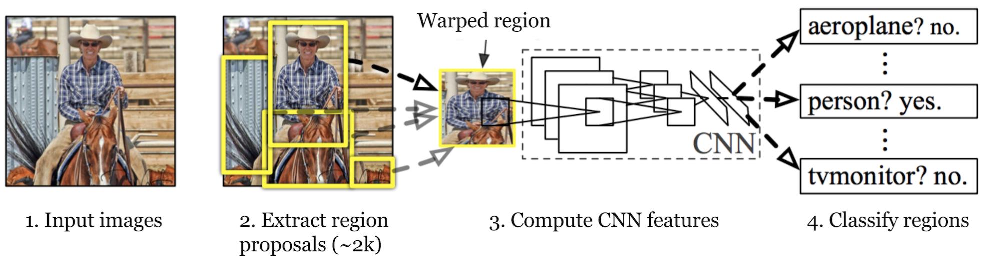


Figure 2.11: R-CNN architecture [4].

a mAP of 54% on the VOC 2010 dataset

2.4.3 YOLO

YOLO is a a DNN for object detection. It is based on one convolutional network that accepts as input the entire image, predicting multiple bounding boxes and class probabilities for detected objects in one operation to the image hence You Only Look Once, particularly in contrast with R-CNN models. This approach simplifies and reduces the network's size and has lead to faster executions times compared to other contemporaneous networks such as R-CNN [5]. This approach of inputting the whole image rather than applying a sliding window over it means that the model applies global contextual information rather than information limited by a sliding window. However, YOLO based models are known to struggle with the detection of smaller scale objects.

Object detection in a YOLO model begins with the segmentation of the input image into an $S \times S$ grid, where each grid cell is responsible for the detection of an object whose center is located in it. Each cell predicts B bounding boxes and confidence scores for these boxes, reflecting the system's confidence on the presence of an object in the cell. Confidence is formally defined as $Pr(Object) * IOU^{truth}_{pred}$, where $Pr(Object)$ is the probability of an object in the box and IOU^{truth}_{pred} how accurate the bounding box predicts the object. A bounding box is composed of 5 predictions: x, y, w, h and confidence. (x, y) represents the center of the bounding box relatively to the grid cell, (w, h) represent then *width* and the *height* of the bounding box. Each cell also predicts C conditional class probabilities, the probability of a class if an object is assumed to exist, that is $P(Class_i|Object)$. By computing

$$Pr(Class_i|Object) \times Pr(Object) \times IOU^{truth}_{pred} = Pr(Class_i) * IOU^{truth}_{pred} \tag{2.9}$$

class-specific confidence scores for each bounding box are obtained. The output of the network is a function of the grid dimensions S, the number of bounding boxes B and the

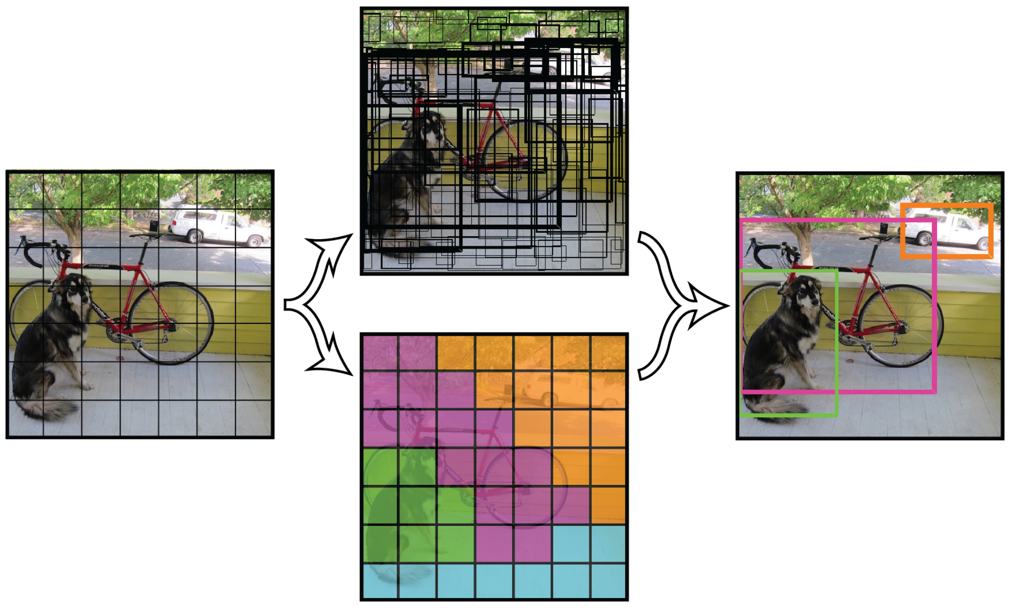


Figure 2.12: YOLOv1 Model. Segmentation of image into $S \times S$ grid cells, bounding boxes and confidence for each cell, class probability for each cell and final detections [5].

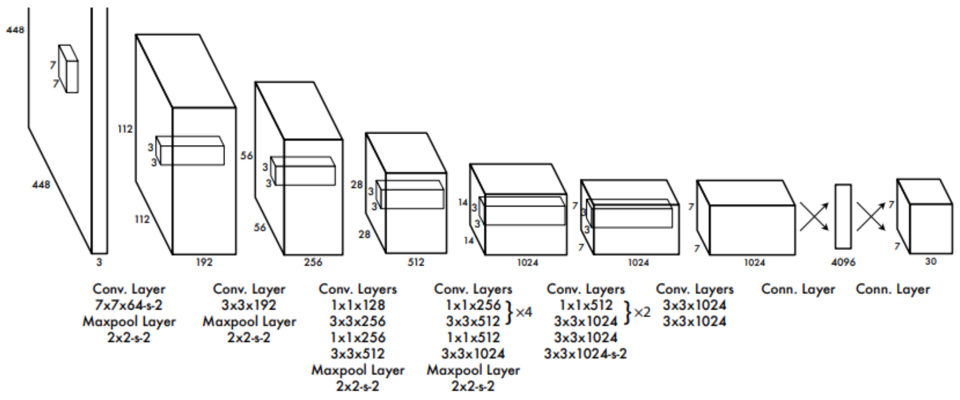


Figure 2.13: YOLOv1 Architecture [5].

number of labelled classes C, concluding to a $S \times S \times (B * 5 + C)$ sized tensor. This process is displayed in Figure 2.12.

The network's architecture Darknet, inspired by GoogLeNet's model, is composed by 24 convolutional layers followed by 2 fully connected layers, with the use of 1×1 reduction layers before 3×3 convolutions, taking advantage of former theoretical framework [33]. For the case of the evaluation of the PASCAL VOC dataset [29] composed by 20 classes, a 7×7 grid and 2 bounding boxes per grid cell, that is $S = 7, B = 2$, the output is a $7 \times 7 \times (2 * 5 + 20) = 7 \times 7 \times 30$ tensor.

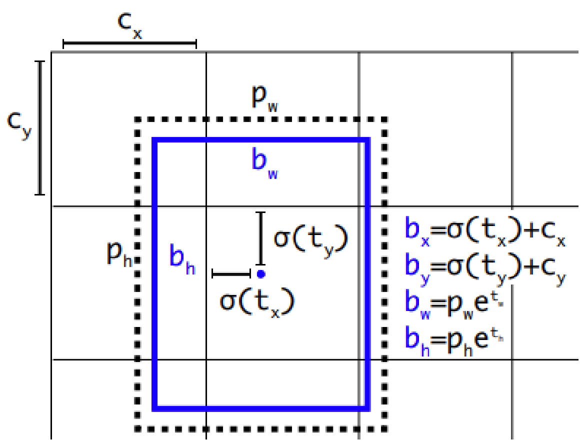


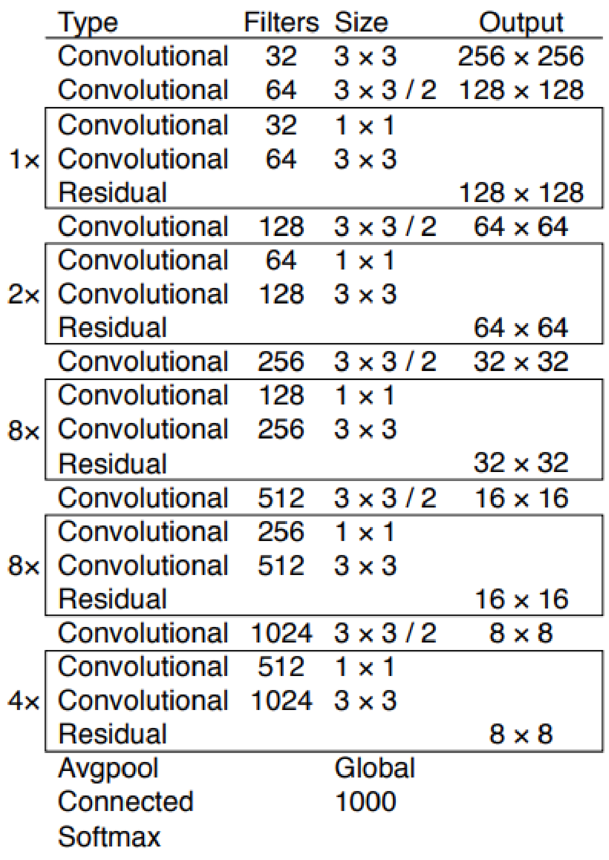
Figure 2.14: YOLOv2 bounding boxes predictions [6].

YOLOv2 [6] improves upon this model. It introduces the mechanism of anchor boxes for detection. Rather than implementing object detection as the result of fully connected layers with bounding boxes representing detection as relative coordinates to grid cells, (x, y, w, h), the result is now the deformation from a set of base bounding boxes, or anchor boxes, required to form a bounding box encompassing the detected object. However, the dimensions of these anchor boxes have an impact on the quality of the model's predictions, so these values must be carefully chosen or be trained alongside the model.

Another improvement is the introduction of convolutional layers downsampling the input image by a factor of 32, meaning that if an image with dimensions of 416×416 is inputted to the model, the resulting grid cell with dimensions $S \times S$ will have $S = 13$, or 13×13. Given that for object detection it is common for an object to be in a middle, an odd value for S will lead to one grid cell in the center of the image responsible for the detection of the centered object, as opposed to a group of four grid cells responsible for detection near the center.

YOLOv2 predicts 5 bounding boxes per grid cell. Each prediction is composed of 5 coordinates: $(t_x, t_y, t_w, t_h, t_o)$. If the cell is offset from the top left corner of the image by (c_x, c_y) and the anchor box has width and height (p_w, p_h), then the bounding box prediction corresponds to:

$$b_x = \sigma(t_x) + c$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.15: Darknet-53 [7].

To improve detection at different scales, at every 10 training batches, image dimensions are randomly chosen from a set of multiples of 32, the downsampling factor of the network, with minimum dimensions 320×320 and maximum dimensions 608×608. These different dimensions allow the model to train from features with different scales, placating the issue with small scale detections displayed in the YOLOv1 model.

Another change was the Darknet network model. It now implements 19 convolutional layers and 5 maxpooling layers creating the Darknet-19 network for feature extraction.

YOLOv3 [7] is the following iteration of YOLO-styled models. This model predicts detections at 3 different scales, extracting features from each scale. The base feature extractor is composed by 53 convolutional layers and the implementation of residual layers, forming the Darknet-53 network, displayed in figure 2.15. These changes improved the detection of small scale objects with a decline in the detection of medium and large size objects, comparatively to previous YOLO models.

YOLOv4 [34], developed by Bochkovskiy et al. while previous versions of the YOLO model were developed by Redmond et al., aims to generalize fast object detectors in production systems, verifying the influence of Bag-of-Freebies and Bag-of-Specials methods, as well as optimizations for parallel computations introducing modified state of the art methods for object detection to more general single GPU training systems.

Bochkovskiy et al. define “bag of freebies” as “*methods that only change the training strategy or only increase the training cost*”, offering data augmentation as an example. “*Plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection*” are defined as “bag of specials”, modules such as attention mechanisms [35] or enlarging the receptive field for each pixel [36]. The implementation of these methods resulted in an increase of AP scores by 2.7% on the MS COCO dataset, for an increase of 0.5% in computation. The result of this work is a state of the art detector available for conventional GPUs.

YOLOv5 [37] has been announced by Ultralytics. However, no clear relationship exists between this entity and the developers of the preceding YOLO versions, either Redmond et al. or Bochkovskiy et al., nor do any peer reviewed articles regarding the development of YOLOv5. Consequently, this version of YOLO was not taken into consideration for this dissertation.

2.5 SOMs in Edge Computing Applications

SOMs applied to DL-based applications using an edge computing paradigm typically have the following characteristics:

- A GPU-based application to be used on the edge;
- hardware (i.e., CPU, GPU) limitations due to concerns about power consumption and form factor;
- requirements to provide results in real-time.

An application developed to detect drowsiness in drivers has real-time requirements, as a driver could fall asleep at any moment, endangering themselves and others, and has been developed by Reddy et al. [38], using an AlexNet-like neural network for feature extractions due to a faster inference when compared to GoogleNet [39] and ResNet. Their application was deployed for use on an Jetson TK1 board, a SOM with 192 CUDA cores on a NVIDIA Kepler GPU architecture with a base clock speed of 870 MHz and a ARM Cortex-A15 CPU, due to its low power consumption and lower financial investment. The study demonstrated that detection of facial features, indicating a person's drowsiness, can be executed on the edge with low power consumption and fulfilling real-time requirements with a SOM.

Gu et al. [40] have developed a robotics application for the collection of tennis balls. This application uses a DL-based service to detect tennis balls in the environment as well as another DL-based method to minimize travel costs on path planning, abstracting the problem as a Travelling Salesman Problem and utilizing Pointer Network models to solve it. For deployment, a Jetson TX1 SOM board was used, featuring a NVIDIA Maxwell GPU architecture with 256 CUDA cores and a 4 core ARM Cortex-A57 CPU complex. Detection of tennis balls was done with a YOLOv1 model, using a USB camera to capture images of the environment and a Kinect depth camera to produce point cloud data of the environment in order to compute the coordinates of each tennis ball. It has been concluded that a YOLO model is capable of detecting objects in a mobile robotics environment as well as that other DL-based applications, such as Pointer Networks, can be executed on the edge under low power conditions with real-time results.

Sa et al. [41] have implemented a Jetson TX2, 256 CUDA core on a NVIDIA Pascal GPU architecture and a 4 core ARM Cortex-A57 CPU to execute semantic segmentation of multi-spectral images in order to evaluate the conditions of weeds in agricultural environments and to be carried in a micro aerial vehicle. A VGG16 [42] was modified for pixel wise segmentation of a multi-spectral image, classifying each image pixel as only crop, only weed and both crop and weed. A cloud computing approach was not implemented due to network latency and unreliable network connection. As such, the edge computing approach was implemented with the Jetson TX2, capable of providing results in real-time with a power consumption of 14 W during maximum work conditions. The Jetson TX2's metrics were compared with the results of a NVIDIA Titan X GPU: the Titan X performed 3.6 times faster but consumed 250 W, around 17.8 times more power than the Jetson TX2. The conclusion is a field deployed SOM with DL-based capabilities with computations executed locally, demonstrating low power consumption and real-time results.

These recent works have obtained positive results from the use of various SOMs on the edge with a low power consumption as well as the integration of the system on mobile platforms such as unmanned aerial vehicles or ground platforms. This provides a basis of comparison as well as precedent for the implementation of DL-based services of mobile systems on the edge under low power consumption conditions to be explored over this dissertation.

2.6 Summary

An edge computing approach is concordant with mobile service robots applications and the Jetson AGX Xavier was designed with this paradigm for local computations of AI applications. A general overview of the state of the art on DL applied to computer vision was discussed and it can be concluded that TinyYOLOv4 and TinyYOLOv3 models involve lighter computations than other suitable models such as R-CNNs while providing appropriate mAP metrics.

The implementation of other SOMs, such as the Jetson TX1, TX2 and TK1, on various applications on the edge to enable DL-based capabilities provides insights into this dissertation, such as results for facial feature extraction and deployment of YOLO-style models on mobile platforms, as well as a basis of comparison for the system to be implemented over this dissertation.

A study of the Jetson AGX Xavier's performance on object detection in previously gathered data and real-time data, and performance evaluation in CPU and GPU intensive computational tasks, are the logical next topics of this dissertation, which are addressed in the following chapter.

3 Benchmark Study

This chapter compares the computational resources and performance of a consumer-grade laptop, an ASUS U36S with traditional CPU and GPU architecture, and the Jetson AGX Xavier. By performing a variety of tasks designed to stress different capabilities in each system, it becomes possible to contextualize the Jetson AGX Xavier with the resources and systems previously employed at the ISR-UC for mobile robotic systems.

3.1 Conventional system using a laptop

The ASUS U36S is a discontinued laptop manufactured in 2011 that has been used to drive mobile robots in research projects developed at the ISR-UC. It features as CPU an Intel i5 2450M [43] with the following specifications:

- 32 nm lithography,
- Two CPU Cores with four total threads
- Maximum 3.10 GHz and base 2.50 GHz frequency,
- Average Power Consumption of 35 W under high-intensity workloads.

The system also features a NVIDIA GeForce 610M [44] GPU:

- 48 CUDA Cores,
- 1024 MB of DDR3 Memory,
- NVIDIA Fermi architecture, supporting CUDA up to CUDA 9.0.

The Fermi GPU architecture has since been unsupported by NVIDIA and as such is incompatible with CUDA 10.x versions. DL-based computations are typically GPU accelerated and Darknet requires at least CUDA 10.2 [45], rendering the ASUS U36S incompatible with the DL-based applications intended to be developed over this dissertation. Consequently, for the purposes of contextualizing the Jetson AGX Xavier's GPU performance for DL-based tasks, the NVIDIA GeForce 1070 will serve as basis for comparison. This GPU features [46]:

- 1920 CUDA Cores,
- 8 GB of DDR5 Memory,
- NVIDIA Pascal architecture, supporting CUDA Software Development Kit (SDK) versions from 8.0 to, as of time of writing, 11.5,
- Average power consumption of 150 W.

3.2 System using NVIDIA Jetson AGX Xavier

The NVIDIA Jetson AGX Xavier integrates:

- A CPU complex composed of four clusters of two NVIDIA Carmel ARMv8.2 CPUs, with each core having individual L1 cache memory, L2 cache memory shared per cluster and L3 cache memory shared by the complex,
- 512-CUDA core GPU with Volta architecture with 64 Tensor Cores – GPU cores designed to provide superior deep learning performance over regular CUDA cores [47],
- Two Deep Learning Accelerator (DLA) engines, a specialized architecture for DL inference operations [48],
- 32 GB 256-bit LPDDR memory,
- 32 GB eMMC 5.1 storage memory,
- Power budget supporting power consumption from 10 W up to 30 W.

The Volta GPU architecture supports CUDA 10.2 and cuDNN 8.0.2, a GPU-accelerated library of primitives for deep neural networks [49], two conditions required for Darknet. The more modern GPU architecture provides the Jetson AGX Xavier with computational resources for DL applications and the multi-core CPU complex is suitable for the multi-process nature of mobile robotics applications, consequently, it is to be expected that the Jetson AGX Xavier will outperform the ASUS U36S on CPU and GPU applications namely integer or floating-point operations and DL-based applications.

3.3 Testbed description

The open-source benchmark tool UnixBench [50] provides an indicator of the performance of an Unix system in tasks such as integer and floating point operation, process spawning, inter-process communication and data transfer. This benchmark's suit test scores compare the tested system with the baseline system "George", a SPARCstation 20-61 with 128 MB RAM, a SPARC Storage Array, and Solaris 2.3, whose ratings were set at 10.0, ever since 1995. As such, a test score of 100.0 indicates that there is a 10x performance enhancement compared to "George". The test suite is composed of the following tests:

- Dhrystone: Designed to measure CPU performance, mainly based on string handling operations,
- Whetstone: Designed to measure CPU performance, based on integer and floating-point C operations,
- Execl Throughput: Designed to measure the speed of CPU process context swaps,
- File Copy: Designed to measure the rate at which data can be transferred between files,
- Pipe Throughput: Designed to measure performance of inter-process communication of 512 bytes via a pipe,
- Process Creation: Designed to measure the cost of spawning a child process and terminating it,
- Shell Scripts: Quantifies the number of times a process can start a set of shell scripts designed to manipulate a data file,
- System Call Overhead: Designed to measure the computational cost of operating system calls.

To complement the UnixBench tests, both systems were tasked and evaluated on object detection using a YOLOv3 model pre-trained on the Common Objects in COntext (COCO) dataset. A YOLOv3 model was used over a YOLOv4 model due to the lower GPU and CUDA requirements, facilitating the execution of a DL-based application on both systems.

The UnixBench test suite was executed three times with the following Linux bash command:

```
$ for i in {1..3}; do mkdir -p ~/unixbench-testing/$i; ./run.sh --output
  ~/unixbench-testing/$i; done
```

Results were stored and analyzed on a Comma-Separated Value (CSV) document.

As robotic algorithms' tend to be composed of long sequences of mathematical operations, particular attention is assigned to CPU and GPU performance.

3.4 Comparative results and discussion

The testbed was executed on both systems and the results aggregated and analyzed. Observing the results in Tables 3.1 and 3.2, indicating the single-core and multi-core test runs visualized in Figures 3.1 and 3.2, it can be concluded the Xavier has a superior performance in the more relevant CPU tests, namely the Dhrystone and Whetsone tests, with 1.42x and 1.52x improvements, respectively, and an inferior performance on tests based on operating system calls and data transfer operations.

To test the performance of GPU-accelerated DL based applications for computer vision, various pre-trained YOLO models, acquired on Alexey Bochkovskiy's GitHub repository [45], were executed on both systems to perform object detection on the COCO 2017 dataset [28].

Results on the ASUS U36S were unobtainable as the system implements a NVIDIA GeForce 610M GPU, unsupported by CUDA 10.2, making it incompatible with GPU-accelerated YOLO models. To circumvent this issue, CPU inference was attempted but this approach led to the laptop's operating system shutting down mid-execution of the test. Similar results were noted on real-time object detection, demonstrating the inability of the ASUS U36S to execute DL based services.

Table 3.1: UnixBench results, one parallel process.

	ASUS U36S	Jetson AGX Xavier
Dhrystone 2 using register variables	2655.8	3783.9
Double-Precision Whetsone	414.8	630.9
Execl Throughput	638.8	286.3
File Copy 1024 bufsize 2000 maxblocks	1524.0	1346.0
File Copy 256 bufsize 500 maxblocks	1088.9	971.3
File Copy 4096 bufsize 8000 maxblocks	2835.8	2201.3
Pipe Throughput	866.1	627.3
Process Creation	871.0	123.9
Shell Scripts (1 concurrent)	2011.7	781.9
Shell Scripts (8 concurrent)	3317.4	1592.0
System Call Overhead	605.2	354.0

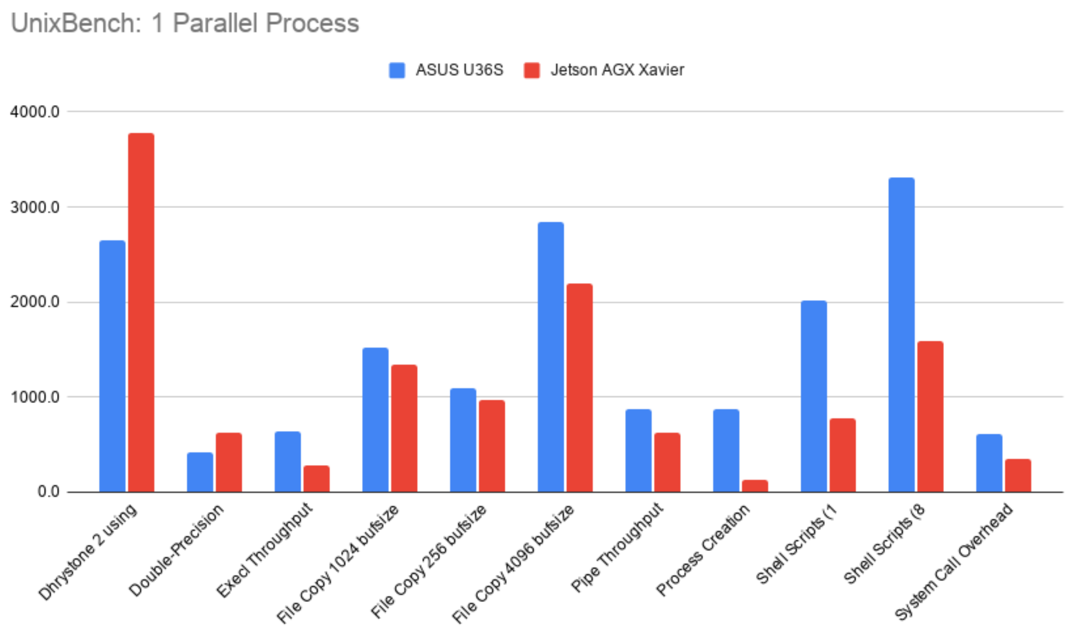


Figure 3.1: Unix Bench results, one parallel process.

Table 3.2: UnixBench results, four parallel processes.

	ASUS U36S	Jetson AGX Xavier
Dhrystone 2 using register variables	5029.6	11506.1
Double-Precision Whetsone	1408.4	2425.8
Execl Throughput	1996.7	804.6
File Copy 1024 bufsize 2000 maxblocks	2112.9	1223.7
File Copy 256 bufsize 500 maxblocks	1446.2	810.2
File Copy 4096 bufsize 8000 maxblocks	3997.4	2467.1
Pipe Throughput	2176.7	2417.7
Process Creation	1907.3	530.0
Shell Scripts (1 concurrent)	3935.6	1817.2
Shell Scripts (8 concurrent)	3682.5	1676.2
System Call Overhead	1798.4	852.9

Table 3.3: Images Per Second processed by various Darknet style neural networks and power consumed. Results from GeForce GTX 1070 GPU.

Network Name	Frames Per Second	Average GPU Power Consumption (W)
TinyYOLOv3	131	100
YOLOv3	41	170
TinyYOLOv4	125	100
YOLOv4	19.2	180

Given the nature of the results obtained on object-detection on the ASUS U36S, the test was conducted on a different desktop machine with a AMD Ryzen 5 3600 6 Core CPU and a NVIDIA GeForce GTX 1070 GPU.

Figure 3.3 is a set of histograms describing the occupancy of the GPU, the percentage of time over the past sample period during which at least one kernel was executing on the device, with a sampling period of one second. Table 3.3 displays the time elapsed for object detection on the COCO 2017 dataset.

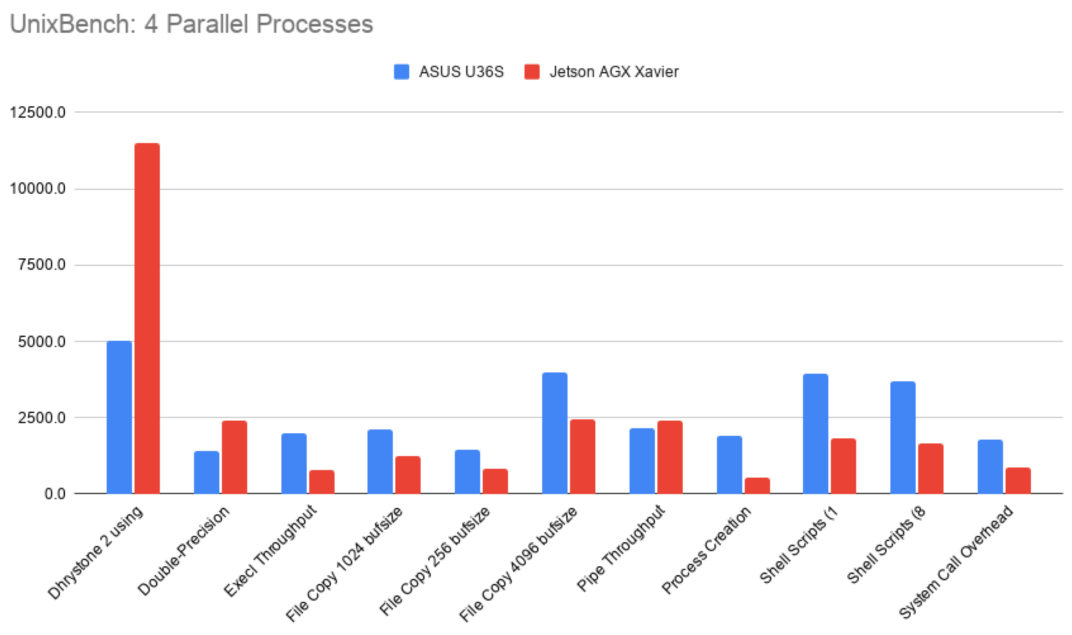
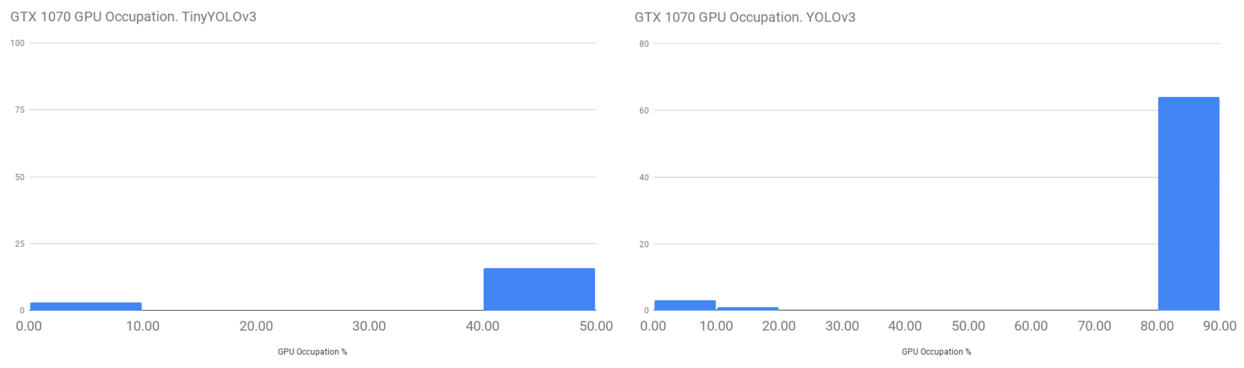
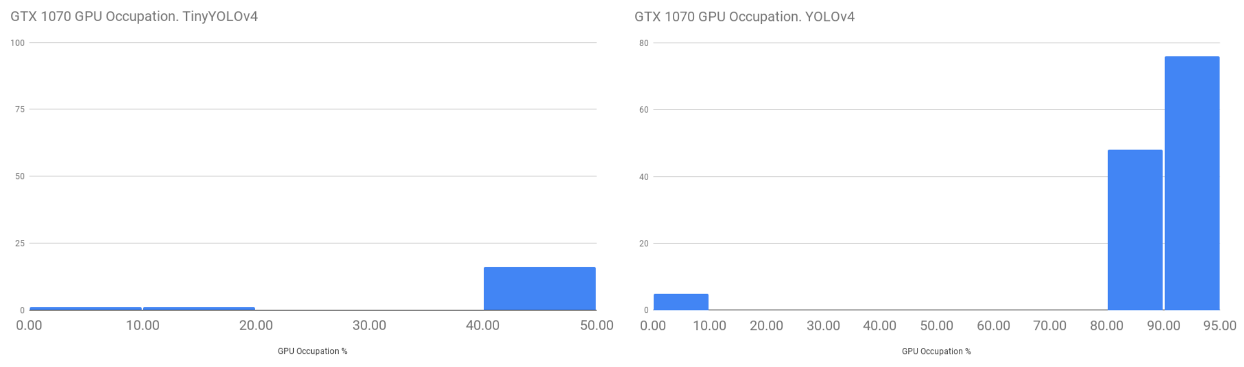


Figure 3.2: Unix Bench results, four parallel processes.



(a) TinyYOLOv3 and YOLOv3.



(b) TinyYOLOv4 and YOLOv4.

Figure 3.3: GPU metrics of the GeForce 1070 during object detection on the MSCOCO 2017 dataset.

Table 3.4: Default power budgets [51].

Power Budget (W)	Active CPU Cores	Max CPU Frequency (MHz)	Max GPU Frequency (MHz)
10	2	1200	520
15	4	1200	670
30	8	1200	900

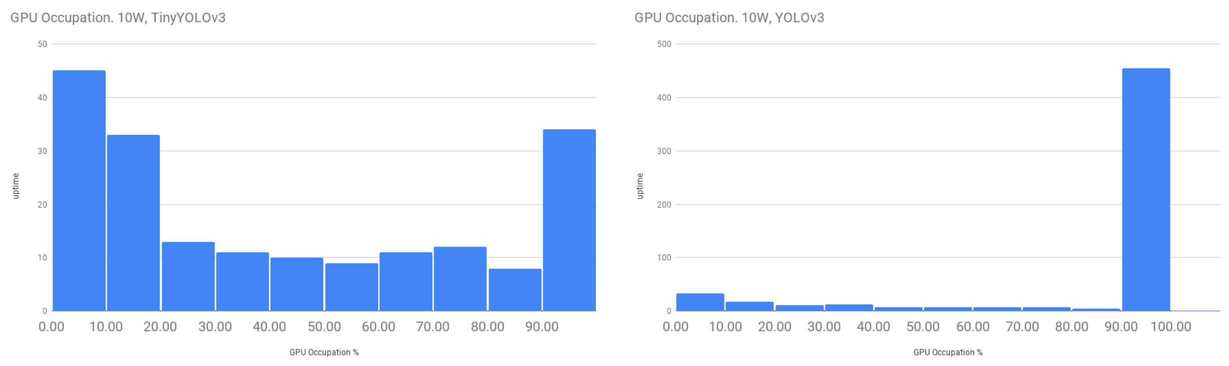
From this data, it is concluded that the GeForce 1070 GTX has the computational resources to process TinyYOLOv3 and TinyYOLOv4 models with strict real-time restrictions. YOLO models are also possible to be executed in real-time but present a higher load on the system, with GPU occupation values majorly in the 90-100% range. Power consumption metrics, as seen on Figure 3.3, display prohibitive values for every YOLO-based model, concluding that a traditional general purpose computer with GPU is not feasible for use in mobile service robots.

To test real-time applications, object detection was implemented on a video stream provided by a USB web camera. On every network tested, on the computer based on the GeForce 1070 GPU was used to process an average of 5 Frames Per Second (FPS). It can then be concluded that the limiting factor is not GPU capability, but data transfer over USB and miscellaneous overhead costs.

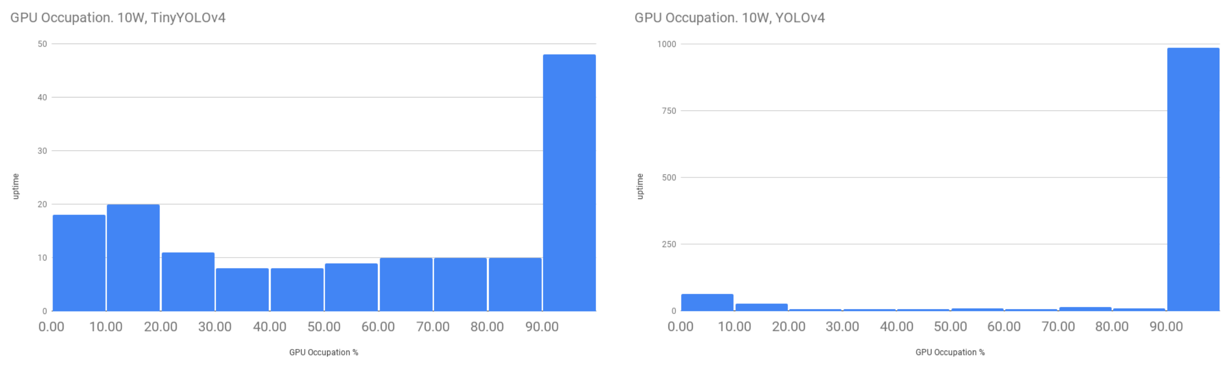
The same models were executed on the AGX Xavier on varied CPU and GPU clock frequencies designed for a consumed power budget of 10 W, 15 W and 30 W. Table 3.4 describes the CPU and GPU frequency and active CPU cores.

From the histograms represented in Figures 3.4, 3.5 and 3.6, demonstrating GPU occupancy, it is concluded that YOLOv3 and YOLOv4 nearly saturate the GPU, occupying consistently mostly 90% of the GPU, regardless of power mode. Object detection with TinyYOLOv3 and TinyYOLOv4 models do not saturate the GPU and display a more evenly distributed GPU occupation. However, TinyYOLOv4 tends to mostly occupy the GPU for over 90% between samples.

From Tables 3.5, 3.6 and 3.7, which show the average frame rate (in FPS) the power consumption for object detection for the AGX Xavier in various power modes, it can be seen



(a) TinyYOLOv3 and YOLOv3, 10W.

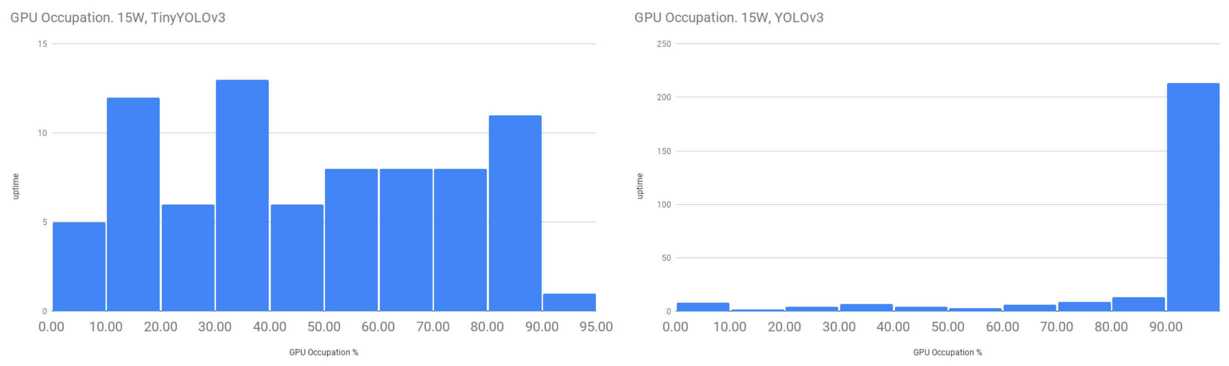


(b) TinyYOLOv4 and YOLOv4, 10W.

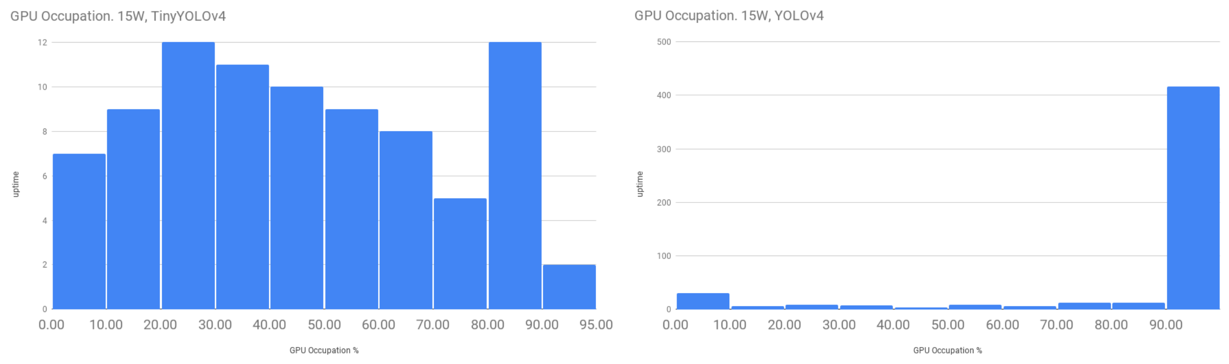
Figure 3.4: AGX Xavier's GPU statistics during object detection on the MSCOCO 2017 dataset, 10W power mode.

Table 3.5: Images Per Second processed by various Darknet style neural networks and power consumed. AGX Xavier, 10W power mode.

Network Name	Frames Per Second	Average GPU Power Consumption (W)
TinyYOLOv3	12.9	1.2
YOLOv3	4.2	2.3
TinyYOLOv4	15.4	1.3
YOLOv4	2.1	2.3



(a) TinyYOLOv3 and YOLOv3, 15W.

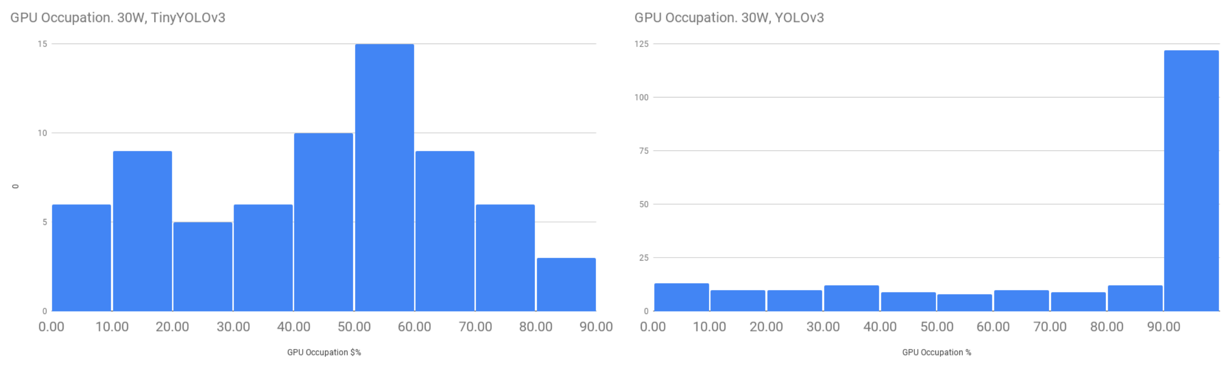


(b) TinyYOLOv4 and YOLOv4, 15W.

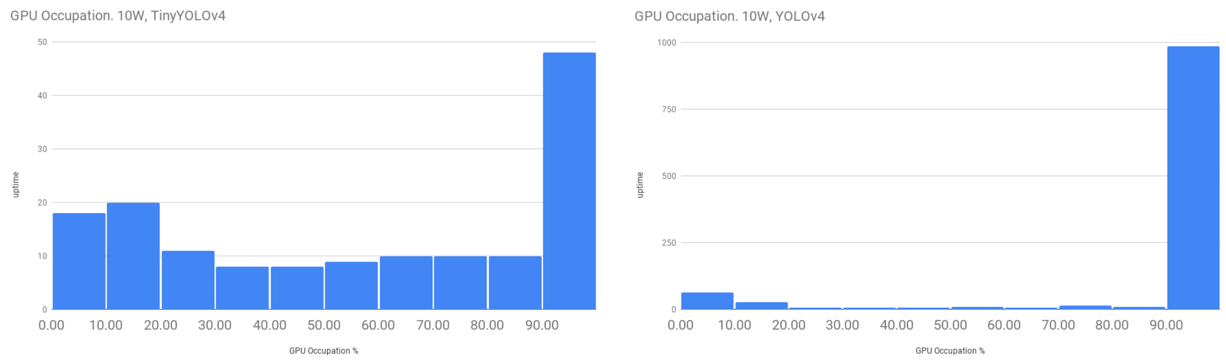
Figure 3.5: AGX Xavier's GPU statistics during object detection on the MSCOCO 2017 dataset, 15W power mode.

Table 3.6: Images Per Second processed by various Darknet style neural networks and power consumed. AGX Xavier, 15W power mode.

Network Name	Frames Per Second	Average GPU Power Consumption (W)
TinyYOLOv3	29.5	2.3
YOLOv3	9.3	4.3
TinyYOLOv4	27.0	2.2
YOLOv4	4.7	4.8



(a) TinyYOLOv3 and YOLOv3, 30W.



(b) TinyYOLOv4 and YOLOv4, 30W.

Figure 3.6: AGX Xavier's GPU statistics during object detection on the MSCOCO 2017 dataset, 30W power mode.

Table 3.7: Images Per Second processed by various Darknet style neural networks and power consumed. AGX Xavier, 30W power mode.

Network Name	Frames Per Second	Average GPU Power Consumption (W)
TinyYOLOv3	33.3	2.8
YOLOv3	11.2	6.8
TinyYOLOv4	35.7	2.9
YOLOv4	6.0	7.9

that TinyYOLO models have decreased Average Power Consumption and greater FPS when compared to their YOLO counterparts.

From these results it is concluded that real-time object detection can be executed with power consumption up to 10 W in the NVIDIA Jetson AGX Xavier. TinyYOLOv4 scored the FPS and the lowest power consumption leading to the conclusion that a TinyYOLOv4 model should be implemented in the system for DL applied to Computer Vision purposes.

3.5 Summary

The Jetson AGX Xavier's CPU complex has an approximate 2x improvement on integer and floating-point operations compared to traditional consumer-grade computer architectures. The Jetson AGX Xavier's GPU is capable of executing various YOLO and TinyYOLO models without saturation on 15 W and 30 W power modes and with GPU saturation on a 10 W operation mode. After comparing performance metrics such as FPS on execution of varied YOLO models, it is concluded that due to the highest FPS and low GPU power consumption, TinyYOLOv4 is the most suitable model to base a DL application on mobile systems.

After the contextualization provided in this chapter of the Jetson AGX Xavier with traditional consumer-grade computer architectures used at the ISR-UC, it is possible to design a mobile robotics system implementing a computational workload composed of typical robotics algorithms for mapping, localization and navigation as well as a concurrent GPU and DL-based workload. This system is studied and explored in the next chapter.

4 Proposed System

This chapter intends to describe the various elements, hardware and software, composing the robotic system to be implemented for this dissertation and a contextualization of these elements in the relevant state of the art.

4.1 Overall System Architecture

The system comprises the SOM Jetson AGX Xavier, the Pioneer P3-DX mobile robot and, the ZED 2 stereo camera. If the point cloud data produced by the ZED 2 camera proves to be unsuitable for navigation, the Hokuyo URG-04LX laser range finder can be implemented into the system, generating laser scan data for the robot navigation algorithms. The Pioneer P3-DX is the mobile platform responsible for the locomotion of the system. Data of the environment is provided by the ZED 2 stereo camera: a point cloud describing the distance of various objects to the system and RGB image captures of the surrounding environment. The Jetson AGX Xavier is to process the point cloud produced by the ZED 2 camera using various robotics algorithms, such as Simultaneous Localization and Mapping (SLAM) [52] and Adaptive Monte Carlo Localization (AMCL) [53] for simultaneous mapping and localization, and for robot navigation with respect to an *a priori* map, as well as to perform object detection on the captured images via a TinyYOLOv4 model trained for the detection of faces with or without facial masks. The Jetson AGX Xavier is also responsible for the communication with the Pioneer P3-DX platform's microcontroller enabling locomotion using the AriaCoda library [54], based on the previous ARIA library maintained by OmronAdept Technologies, Inc.

These system software components are integrated by using the ROS middleware, with various ROS nodes responsible for the communication between the multiple processes computing

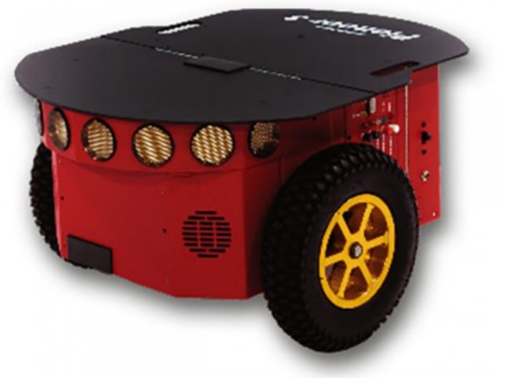


Figure 4.1: Pioneer P3-DX mobile robot.

Figure 4.2: ZED 2 Stereo Camera.

data and system components.

4.1.1 Pioneer P3-DX

The Pioneer P3-DX is a now discontinued small lightweight two-wheel two-motor differential drive robot, designed for research and industrial use, formerly manufactured by Adept Technologies, now Omron Adept after Omron acquisition of Adept Technologies in 2015 [55]. It is a 9 kg, capable of reaching speeds of up to 1.6 m/s, with an 8 to 10 hours of operational time supported by up to three 12 V lead-acid batteries, each with a capacity of 9 Ah. This model has been used for research projects conducted by multiple students and researchers at ISR-UC, e.g. [56] [57] [58] [59], and will be used in this dissertation as a platform example for the implementation of AI methods based on DL on service robots.

4.1.2 ZED 2 Stereo Camera

The ZED 2 is a stereo camera providing high definition three dimensional video and neural depth perception of the environment [60], powered over Universal Serial Bus (USB) 3.0. It is capable of capturing depth up to 20 meters with a frame rate as high as 100 FPS and a

Field of View (FOV) of up to 110º(H) x 70º (V). Its SDK implements visual SLAM in order to track its position and orientation in space and has native support for the NVIDIA Jetson device family, including the Jetson AGX Xavier.

The ZED 2's deep learning support applied to robotics makes it a powerful device for the purposes not only of the system studied in this dissertation, but as well as for future robotic applications developed at the ISR-UC.

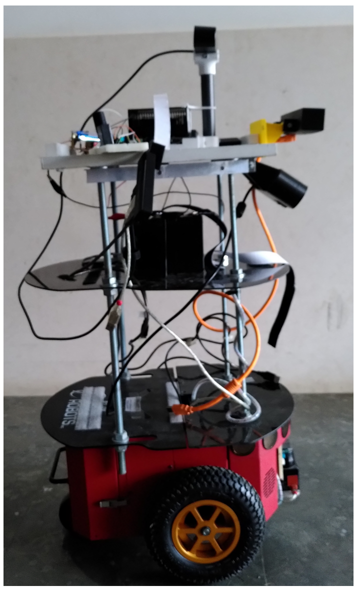
4.2 System Integration

The Jetson AGX Xavier was affixed with velcro straps to a platform one meter above the Pioneer P3-DX as seen on Figure 4.3a. A stand for the ZED 2 has been 3-D printed, using schematics available on the Thingiverse repository [61], and affixed to the Jetson AGX Xavier's carrier board.

The Pioneer P3-DX microcontroller, responsible for the actuators and sensors in the platform, and the ZED 2 camera are connected to the Jetson AGX Xavier via USB 3.0. In order to enable teleoperation with a Wii game console controller via Bluetooth and local network communication an Intel 8265 M.2 Wireless Network card was integrated. The system can then be interacted in real time by a computer connected to the same local network as the Jetson AGX Xavier by remotely connecting to a Virtual Network Computing (VNC) server set up on the Xavier.

A filesystem derived from Ubuntu 18.04, part of JetPack 4.4 [62], and ROS Melodic middleware suite were installed in the Jetson AGX Xavier in order to provide a software basis for the multiple algorithms implementing the system's functions. The ROS system is depicted in Figure 4.4. The ROS nodes can be described as follows:

- The *zed2* node is responsible for publishing data produced by the ZED 2, such as visual RGB data of the surrounding environment and constructed point cloud over the *rgb/image_rect_color* and *point_cloud/cloud_registered* topics, respectively;
- the *ROSARIA*, *move_base*, *amcl*, *estop* and *map_server* nodes are responsible for issuing motion directives to the Pioneer P3-DX's microcontroller and for the mapping, localization and navigation of the system in the environment;


(a) Side view of the system.

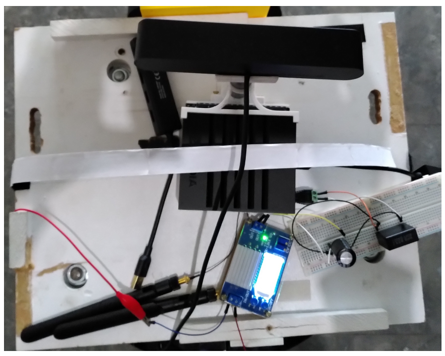

(b) Top down view of the system.

Figure 4.3: Service robot prototype.

- the *wiimote* node is responsible for Bluetooth communication with a Wii game console remote to be used for teleoperation of the mobile platform;
- the *urg_node* is responsible for aggregating the data collected by the Hokuyo laser rangefinder;
- the *RGB_Depth_DL* node that subscribes to topics published by the *zed2* node and executes the YOLOv4 model that detects faces with and without facial masks, as well as the computation of distances between detections, verifying social distancing measures.

Powering the Jetson AGX Xavier from the robotic platform's batteries proved to be laborious due to tight electrical requirements for proper powering of the SOM device:

- An input voltage between 9 and 19 V;
- the input voltage cannot suffer fluctuations greater than 0.5 V/s.

Power efficiency and voltage fluctuation are both inversely proportional with the input voltage, meaning that energy efficiency is higher with lower voltages and that more pronounced deviations in the input signal occur with lower voltage supply. Therefore, there is a trade-off: to choose a suitable voltage in order to produce a stable power signal ($\frac{dV}{dt} \leq 0.5V/s$)

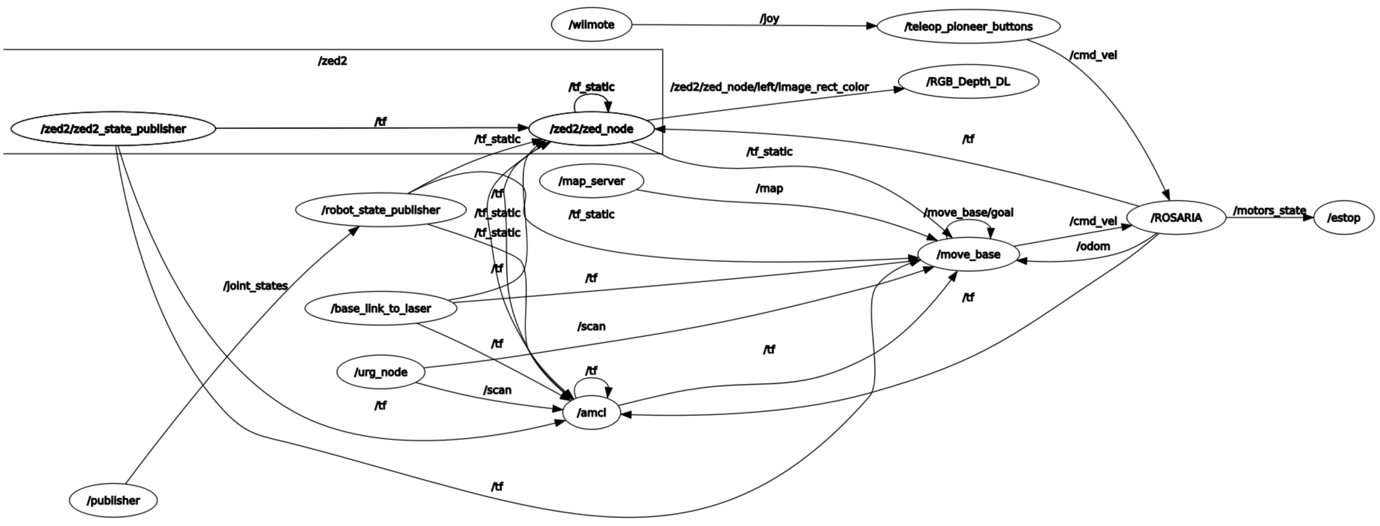


Figure 4.4: ROS workspace composed by various nodes responsible for sensors, actuators and data processing.

while maximizing efficiency. To note, during system boot-up and during the instantiation of the ROS system multiple processes start up in a limited time frame, causing significant current spikes and voltage drops.

The first explored approach was to power the Jetson AGX Xavier with the 12 V lead acid batteries that power the Pioneer P3-DX mobile platform. This approach could not provide a stable voltage signal in laboratory tests, despite measures to enhance the electrical circuit with the implementation of a DC-DC converter, an electrolytic 4.7 mF capacitor and a 10 μF plastic dielectric capacitor. Despite improving the battery's output voltage by reducing voltage deviations, the Jetson AGX Xavier's electrical requirements could not be satisfied with this approach as the embedded computer could not boot successfully due to sudden voltage drops as seen on Figure 4.5.

Therefore, a dedicated power source for the Xavier was implemented. Two 12 V lead-acid batteries in series, producing a 24 V voltage source, modulated down to 18.5 V via the DC-DC converter and both capacitors connected in parallel with the Xavier's input power port. This electrical montage provided a suitable power signal for operation of the Jetson AGX Xavier with a maximal GPU frequency of 670 MHz, corresponding to the 15 W power budget as per Table 3.4. Its schematic is included in Annex A.

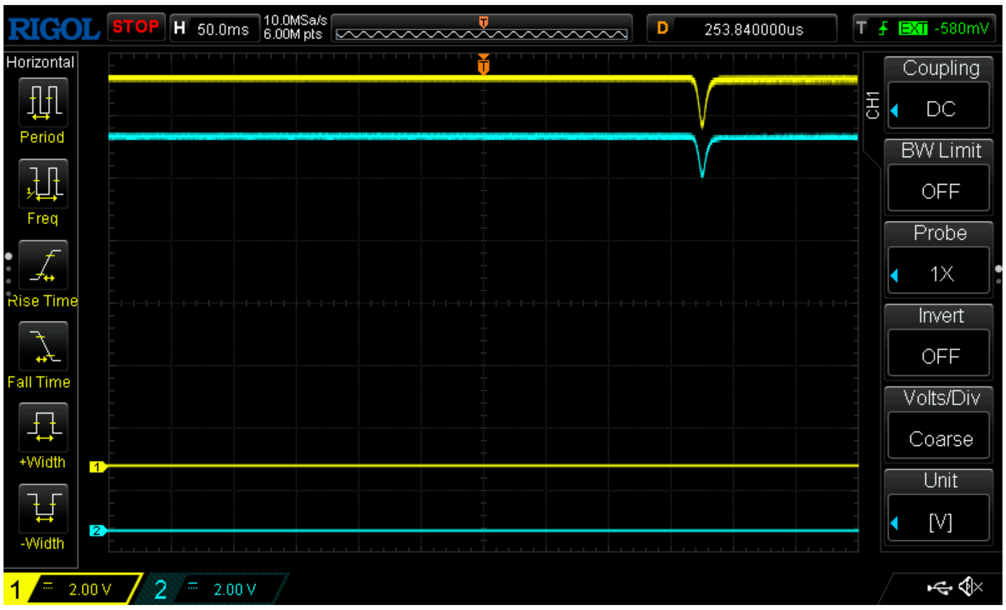


Figure 4.5: Voltage drops during laboratory tests. 2 Volts per division. The output voltage of the DC-DC converter is in yellow and the capacitor voltage, parallel to the Jetson AGX Xavier power port, is in cyan.

4.3 Preliminary Functional Tests

All hardware was successfully integrated into a cohesive system. The Pioneer P3-DX microcontroller, the ZED 2 camera and the Hokuyo laser range finder are connected to the Jetson AGX Xavier via a USB 3.0 hub. The RosAria [54] library was built from source and installed, allowing the interface of the Pioneer P3-DX platform from the Jetson AGX Xavier. The Intel 8265 M.2 Wireless Network card operates correctly, providing Bluetooth communication and remote desktop streaming via a VNC server. The RVIZ viewer, a 3D visualization tool integrated in ROS, also executes correctly on the system, allowing the visualization of the environment map, sensor input data and to set target poses in the mapped environment, providing target navigation poses.

Environment data for the navigation stack was aggregated by the point cloud generated by the ZED 2 stereo camera. This 3D data was collapsed into 2D data, on a plane parallel to the floor, in order to be processed by the mapping stack. However, this approach did not lead to acceptable navigation in the environment: navigation around corners and through tight corridors would occasionally lead to collisions, risking serious damage to the system. Consequently, the Hokuyo laser range finder is used to provide reliable 2D data of the surrounding environment. This approach lead to more robust navigation.

Darknet [45] was built from source with GPU, CUDNN [49] and OpenCV support. To facilitate the implementation of Darknet-based YOLO-style models in C++ applications, the open-source third-party library DarkHelp [63] was installed and successfully tested.

Remote desktop streaming with a 15 W power budget, with 4 active CPU cores, was also successfully implemented but had issues: difficult user interface and data visualization due to low frame rate remote desktop streaming. To provide a stable and responsive user interface with the system, the power budget was increased to accommodate the multi-process system by powering a total of 8 CPU cores, increasing the power budget of the system to around 19 W.

The robot navigation with respect to an *a priori* map of the environment was successful. However, the Jetson AGX Xavier would occasionally get out of range of the local area network access point and consequently disconnected from the network, denying remote desktop streaming until the system reconnected to the network via a different access point.

4.4 Summary

This chapter provided insight into the various hardware and software components composing the overall system, as well as describing the various difficulties in integration and how they were overcome.

As SOM computational architectures become more complex, their electrical requirements become stricter. Consequently, DL-based applications, as well as other complex GPU-based applications, depend not only on the computational resources available on the edge but also on electrical resources for system power.

The software stack was successfully implemented and tested on the system. Some applications, such as the AriaCoda library on which ROSARIA is based on, did not provide a suitable .deb file for installation on Unix-based ARM64 systems, meaning that they must be built from source or that suitable alternatives must be chosen instead.

A basic ROS-based navigation system for a service robot was successfully implemented and executed, demonstrating the possibility of concurrent execution of tasks such as mapping, inter-process communication and DL-based object detection in real-time.

The ZED 2 camera provides suitable RGB image data of the surrounding environment allowing object detection. However, 2D data provided by the Hokuyo laser range finder proved to be more suitable for mapping and navigation when compared to 2D data generated from the collapse of the 3D point cloud data captured by the ZED 2 camera as it strongly reduces the possibility of collision of the robot with obstacles, namely at corners.

Operation of the Pioneer P3-DX was also successful, leading to correct implementation of all hardware components into the system.

In the following chapter, the DL-based system for the enforcement of COVID-19 prevention measures is presented, providing insight into the workflow of a Darknet based model and a discussion on the results of the trained TinyYOLOv4 model.

5 Prototype to automatically check COVID-19 prevention measures

This chapter aims to describe the behavior and requirements of the system for checking the fulfillment of COVID-19 prevention measures: proper facial mask usage and social distancing – two meters in accordance with the Portuguese national health authority.

Power consumption is to be minimized while avoiding CPU and GPU saturation. It is required for the system to process at least five frames per second as given the nature of COVID-19 prevention measures. A high rate of processed frames is not required as failure to maintain social distancing must be prolonged in order for infection to possibly occur. The system is non-critical as failure to oblige task scheduling does not lead to damage to the environment, to people or to the system itself.

5.1 System Description and Requirements

The evaluation of facial masks is to be done with a custom trained TinyYOLOv4, to be executed and GPU-accelerated on the Jetson AGX Xavier. Social distance is to be evaluated by resorting to the point cloud generated by the ZED 2 stereo camera. By detecting correctly or incorrectly worn facial masks via DL, it is possible to determine the (x, y, z) world coordinates, centered on the left ZED 2 camera lens, corresponding to the (u, v) image coordinates of the detected faces. By computing the Euclidean distance of every pair of detected faces, it is possible to determine whether or not social distancing measures are being practiced. The navigation of this system in the environment is to be done via the Pioneer P3-DX mobile platform by a user inputting target poses, position and orientation, on the *a priori* mapped floor 0 of the environment used for the tests conducted.

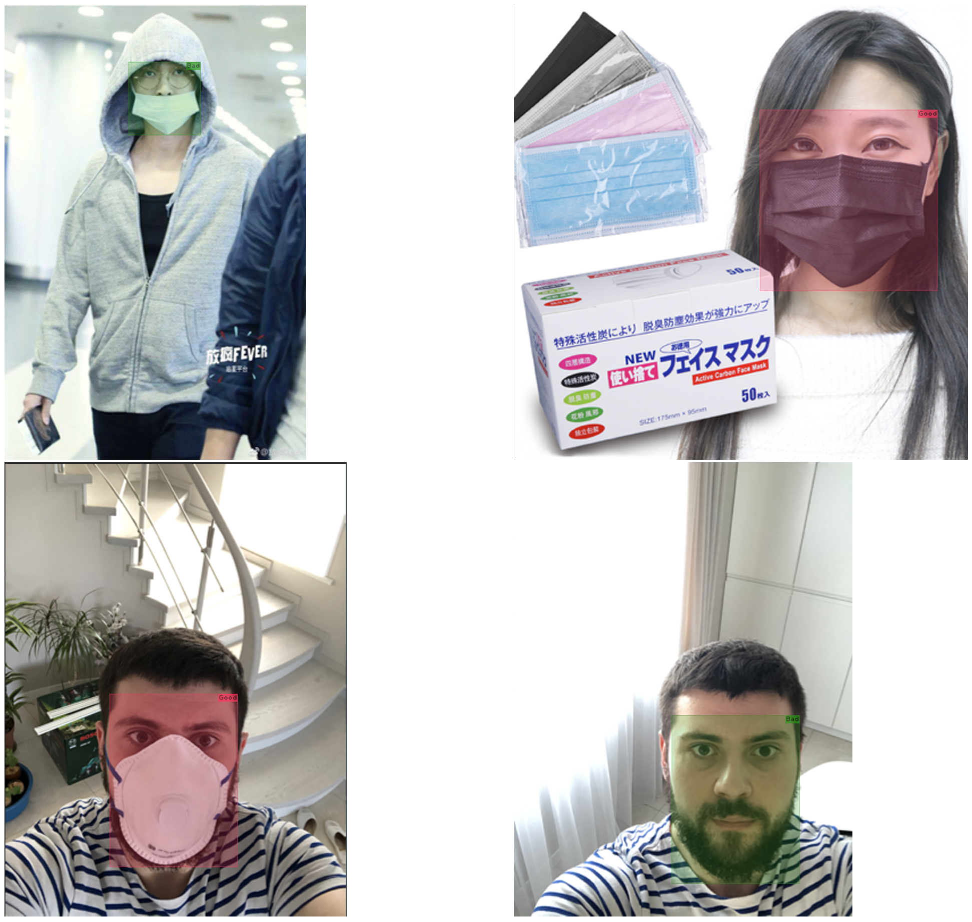


Figure 5.1: Sample of the training dataset.

5.2 Neural network tailored for the target application

A TinyYOLOv4 model was custom trained for this system with assistance of the open-source DarkMark [64] application. The training data set was obtained on the online platform Kaggle [65] and is composed of 853 images of faces with correctly worn facial masks, faces with incorrectly worn facial masks and maskless faces. The set of faces with incorrectly worn facial masks and of maskless faces was merged into a single class in order to simplify the model.

The model was trained for 9000 batches, with 64 images per batch and subdivisions of 16 images. That is, the dataset was arranged into 9000 groups of 64 randomly selected images, and 16 images were loaded onto the GPU at a time. After training a full batch of images, the model's weights are updated. Data was augmented by enabling color manipulation, symmetries on a vertical axis and arrangement of images in mosaics, producing a larger, more diverse training dataset and consequently a more generalized trained model. Training occurred on a GeForce 1070 GTX GPU. Annex C.3 provides more detailed information

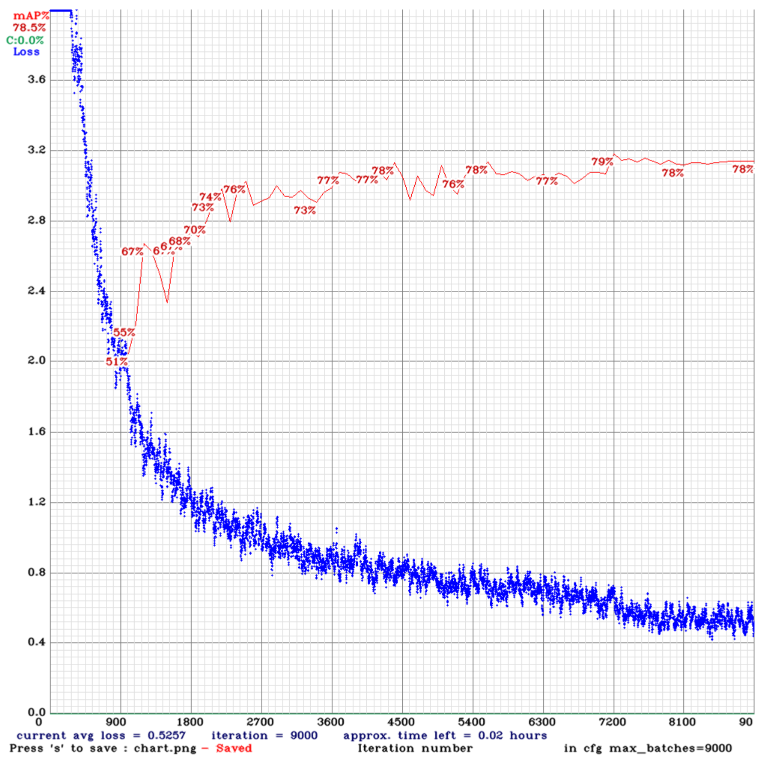


Figure 5.2: Training loss, mAP score of custom trained TinyYOLOv4 model.

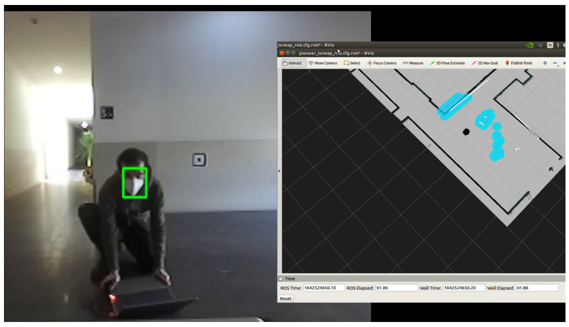
about the procedure that was used to train a customized YOLO model.

The model's best mAP@50% score during training was 79% and the weights corresponding to this performance were used on the final system. During training, loss consistently decreased until asymptotically reaching a value of 0.5 indicating that the model did not underfit, that is, the model is complex enough to be able to extract features of input data and produce a valid relationship between input data and class output.

5.3 Experimental tests

The TinyYOLOv4 model was tested with images, provided by a USB camera, of different faces, with and without facial masks under different sets of circumstances such as lighting, distance and facial mask color.

The floor 0 of the ISR-UC was first patrolled manually using Bluetooth-based teleoperation of the system. Afterwards, a set of poses was defined via the RVIZ tool in order to initiate autonomous navigation. The tests were repeated with the additional computational workload of the TinyYOLOv4 model and the verification of social distancing. The experimental tests can be seen in Figures 5.3a and 5.3b. All 8 available CPU cores of the Jetson AGX Xavier


(a) System autonomously navigating around the environment.

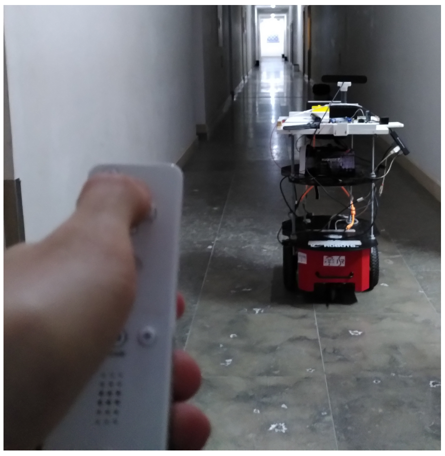

(b) Manually navigating the system around the environment.

Figure 5.3: Experimental tests – navigation of an *a priori* known environment manually and autonomously.

were powered and used with 1200 MHz maximal frequency. The GPU's maximal frequency was set to 670 MHz. Object detection operated at an average of 10 frames per second. The system is expected to navigate through the *a priori* known environment, both manually and autonomously, while processing visual and point cloud data generated by the ZED 2 stereo camera, or alternatively data produced by the Hokuyo laser rangefinder, concurrently with the trained TinyYOLOv4 model GPU-based workload under 20 W of power consumption.

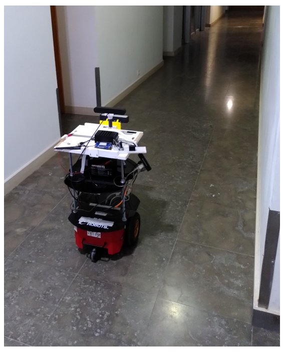
5.4 Results and discussion

CPU workload, related to the robot navigation and inter-process communication in ROS, was evenly distributed along each core by the Unix task scheduler with each core operating at an average occupancy of around 70%, as seen on Figure 5.5. Workload distribution was mostly even, i.e. each core had approximately the same occupancy rate.

The Jetson AGX Xavier's GPU did not saturate but did display an erratic behaviour over time, as seen on figure 5.6. Despite this, object detection maintained a rate of processed frames per second above 5, satisfying system requirements.

Navigation of the environment using the point cloud data generated by the ZED 2 had issues: the system would occasionally encounter difficulties navigating, namely through corners, and there was an elevated risk of collision. For this reason, the Hokuyo laser rangefinder was implemented into the system to provide environmental data as initially postulated.

Navigation with the Hokuyo laser rangefinder was possible and the mobile platform was capable of traversing all of the known environment without issues, as seen on Figures 5.4b and 5.4a. Route planning and obstacle detection were processed in such a way as to provide real-time navigation.


(a) System autonomously navigating through a corridor.

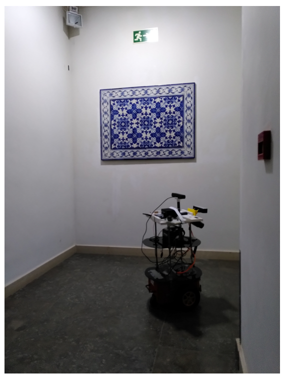

(b) System autonomously navigating around a corner.

Figure 5.4: Experimental tests – navigation of an *a priori* known environment manually and autonomously.

The system consumed under 20 W of power while executing a DL-based, GPU-accelerated service in parallel with a CPU-based system composed of robotics algorithms and data acquisition of USB-connected peripherals, providing a basis for more complex DL-based applications.

The custom trained TinyYOLOv4 model detects faces during practical applications and the system can successfully compute the distance between detected objects, as seen on figure 5.7. However, environmental lighting as well as the shape and color of facial masks are factors that influence the confidence scores and classification of the model. A YOLOv3 or YOLOv4

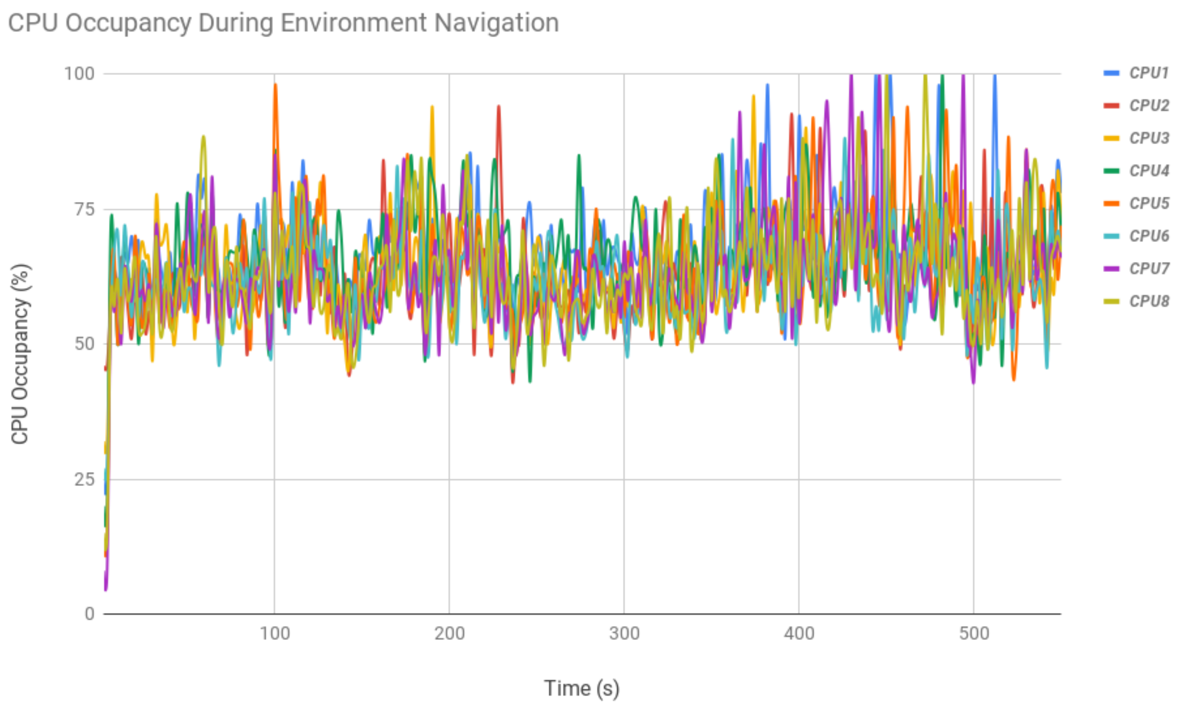


Figure 5.5: Workload distribution over all 8 CPU cores.

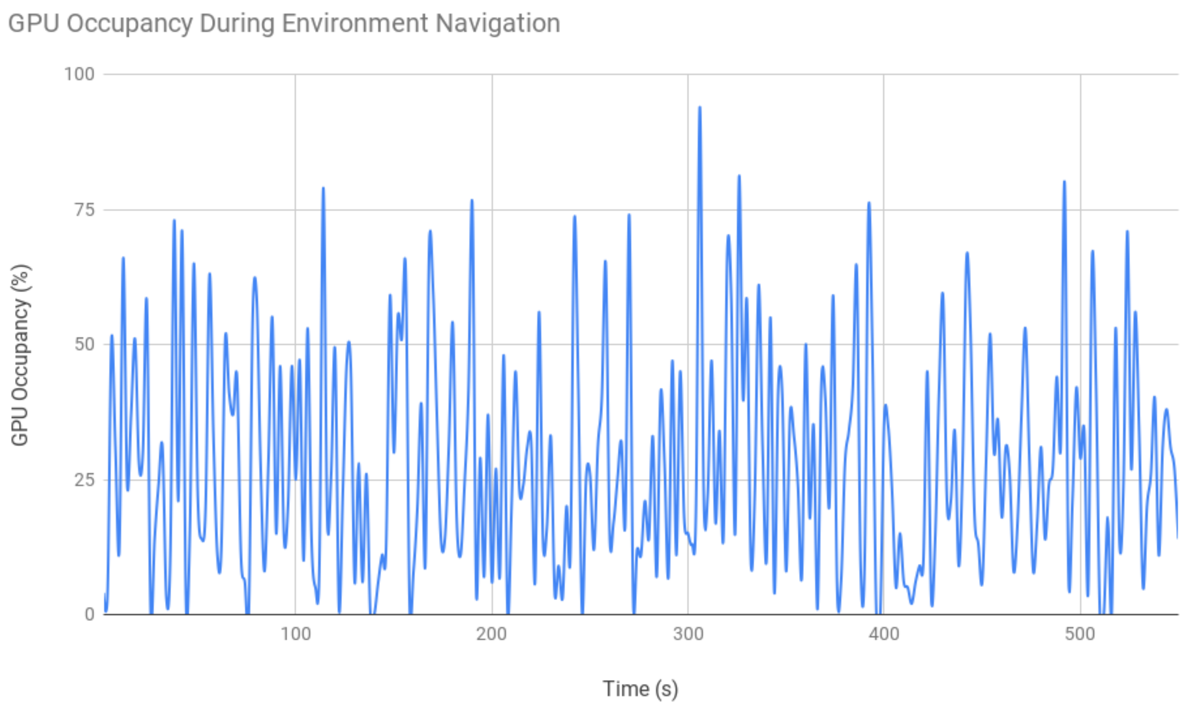


Figure 5.6: GPU occupancy during navigation.

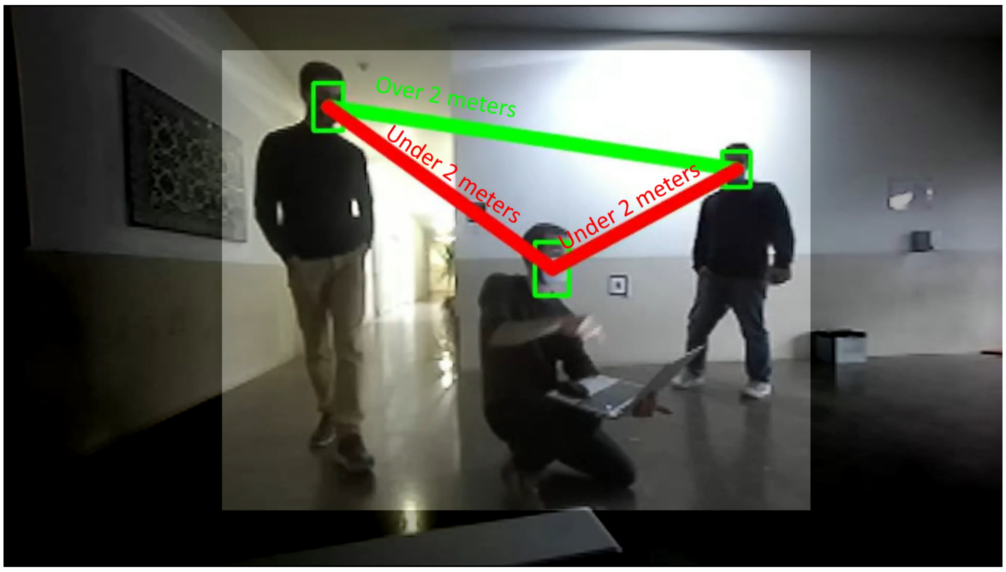


Figure 5.7: Practical application of facial mask detection and social distance evaluation. All three people are wearing masks, as indicated by the green squares. A green line indicates a valid social distance whereas a red line indicates an invalid social distance between individuals.

model would provide resiliency to these factors by extracting more complex features of input data at the cost of longer execution times, higher GPU workload, bigger model weights and higher power consumption.

5.5 Summary

This chapter described the execution and results of the prototype system. The trained TinyYOLOv4 model successfully detects correctly masked faces and maskless faces as intended but can be disturbed by environmental variables, such as lighting, and due to masks different to those present in the training dataset, such as masks with graphical designs.

The point cloud generated by the ZED 2 stereo camera could not provide good navigation on the *a priori* known environment. To solve this issue, the Hokuyo laser range finder was implemented to provide environmental data necessary for the navigational ROS stack.

The final chapter intends to provide a conclusion of this dissertation and to provide a direction for future work to be elaborated over this system.

6 Conclusion

In this dissertation, the possibility of implementing DL-based applications on mobile service robots employing an edge computing approach was explored and a prototype was successfully developed. As AI applications become more common place in the modern world, it becomes important to explore possibilities and approaches to implement DL-based applications in mobile robots on the edge.

The developed prototype system is composed of a Jetson AGX Xavier, a SOM designed and manufactured by NVIDIA, a ZED 2 stereo camera, a Hokuyo URG-04LX laser range finder and a Pioneer P3-DX mobile platform. A CPU-based robotics system is executed concurrently with a GPU-accelerated DL-based application in order to implement system functionality, i.e. environment navigation and COVID-19 prevention measures, while fulfilling system requirements without relocating any computational workload to external computers on the cloud.

This dissertation was based on six stages: an exploration of the Jetson AGX Xavier; contextualizing the Jetson AGX Xavier's capabilities with other machines used for similar systems; a study of DL and AI; a comparison of various DL models applied to computer vision and deciding which model is most suitable to this prototype system; an exploration of electrical setups in order to satisfy electrical requirements for mobile powering of the Jetson AGX Xavier; and field deployment of the system by manually patrolling an *a priori* known environment.

The system locally executed a robotics-based application concurrently with GPU-accelerated DL-based services without prohibitive power consumption, providing a hardware and software basis for more complex applications on the edge to be developed.

6.1 Future work

Over this dissertation, various possibilities were discussed but could not be explored without risking digressing the main objective of this dissertation. These include exploring the heterogeneous architecture of the Jetson AGX Xavier, namely the NVDLAs [48] modules, accelerating the TinyYOLOv4 model via TensorRT [66], deploying and using models based on a ONNX [67] framework in order to minimize the effects of the fragmentation of the DL community divided into multiple frameworks and libraries – such as TensorFlow, Keras or PyTorch.

It is possible to integrate the ZED 2 stereo camera more intimately into the system, such as utilizing the produced depth map to detect noticeable differences between the *a priori* point cloud data and the locally generated point cloud data. By detecting the presence of an unknown object via the depth map, it is possible to change the DL service from object detection to image classification, a typically less intensive problem.

In order to facilitate further development of this system and deployment, this system was integrated into a Docker container and is available at a Docker repository [68].

7 Bibliography

[1] "Receiver Operating Characteristic (ROC)." `https://scikit-learn/stable/auto_examples/model_selection/plot_roc.html`. Accessed on 2022-01-18.

[2] A. Khan, "Machine Learning in Computer Vision," *Procedia Computer Science*, vol. 167, 04 2020.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, pp. 1106–1114, 2012.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv e-prints*, p. arXiv:1506.02640, June 2015.

[6] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[7] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, 2018.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[9] S. Sudhakar Farfade, M. Saberian, and L.-J. Li, "Multi-view Face Detection Using Deep Convolutional Neural Networks," *arXiv e-prints*, p. arXiv:1502.02766, Feb. 2015.

[10] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[11] J. Li, W. Dai, F. Metze, S. Qu, and S. Das, "A comparison of Deep Learning methods for environmental sound detection," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 126–130, 2017.

[12] IFR, "International Federation of Robotics." `https://ifr.org/service-robots`. Accessed on 2021-05-31.

[13] T. Mitchell, *Machine Learning.* McGraw-Hill, 1997.

[14] G. Palm, "Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity," in *Brain Theory* (G. Palm and A. Aertsen, eds.), (Berlin, Heidelberg), pp. 229–230, Springer Berlin Heidelberg, 1986.

[15] D. Steinkraus, I. Buck, and P. Simard, "Using GPUs for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pp. 1115–1120 Vol. 2, 2005.

[16] M. Karra, "Using cloud solutions for translation: Yes or no? – IAPTI." `https://www.iapti.org/iaptiarticle/using-cloud-solutions-for-translation-yes-or-no-2/`. Accessed on 2021-05-28.

[17] L. Seltzer, "Your infrastructure's in the cloud and the Internet goes down. Now what? | ZDNet." `https://www.zdnet.com/article/the-internet-is-down-what-next/`. Accessed on 2021-05-28.

[18] E. Hamilton, "What is Edge Computing: The Network Edge Explained." `https://www.cloudwards.net/what-is-edge-computing/`, Dec. 2018. Accessed on 2021-01-13.

[19] S. Gupta, R. Kapil, G. Kanahasabai, S. S. Joshi, and A. S. Joshi, "SD-Measure: A Social Distancing Detector," in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 306–311, IEEE, 2020.

[20] "Jetson AGX Developer Kit." `https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit`. Accessed on 2021-1-1.

[21] MIT, *Trends, Opportunities and Challenges Driving Architecture and Design of Next Generation Mobile Computing and IoT Devices | Microsystems Technology Laboratories*, 2015.

[22] P. Mell, T. Grance, *et al.*, "The NIST definition of Cloud Computing," *Natl. Inst. Stand. Technol., 53 (6) (2009),*, 2011.

[23] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 27–33, 2010.

[24] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ROS-compatible stereo visual SLAM methods on a nVidia Jetson TX2," *Measurement*, vol. 140, pp. 161–170, 2019.

[25] H. Liu, R. A. Rivera Soto, F. Xiao, and Y. J. Lee, "YolactEdge: Real-time Instance Segmentation on the Edge," *arXiv e-prints*, p. arXiv:2012.12259, Dec. 2020.

[26] B. Goertzel and P. Wang, *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms : Proceedings of the AGI Workshop 2006*. Frontiers in artificial intelligence and applications, IOS Press, 2007.

[27] D. H. Wolpert, "The Lack of A Priori Distinctions Between Learning Algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[28] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 740–755, Springer International Publishing, 2014.

[29] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, June 2010.

[30] T. Pun, G. Gerig, and O. Ratib, "Image analysis and computer vision in medicine," *Computerized Medical Imaging and Graphics*, vol. 18, no. 2, pp. 85–96, 1994. Multimedia Techniques in the Medical Environment.

[31] J. Villalba-Diez, D. Schmidt, R. Gevers, J. Ordieres-Meré, M. Buchwitz, and W. Wellbrock, "Deep Learning for Industrial Computer Vision Quality Control in the Printing Industry 4.0," *Sensors*, vol. 19, no. 18, 2019.

[32] Z. Yang, W. Yu, P. Liang, H. Guo, L. Xia, F. Zhang, Y. Ma, and J. Ma, “Deep transfer learning for military object recognition under small training set condition,” *Neural Computing and Applications*, vol. 31, pp. 6469–6478, Oct 2019.

[33] S. Arora, A. Bhaskara, R. Ge, and T. Ma, “Provable Bounds for Learning Some Deep Representations,” in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 584–592, PMLR, 22–24 Jun 2014.

[34] A. Bochkovskiy, C.-Y. Wang, and H. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *ArXiv*, vol. abs/2004.10934, 2020.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[36] Y. Gu, Z. Zhong, S. Wu, and Y. Xu, “Enlarging Effective Receptive Field of Convolutional Neural Networks for Better Semantic Segmentation,” in *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 388–393, 2017.

[37] “ultralytics/yolov5.” `https://github.com/ultralytics/yolov5`. Accessed on 2022-01-17.

[38] B. Reddy, Y.-H. Kim, S. Yun, C. Seo, and J. Jang, “Real-Time Driver Drowsiness Detection for Embedded System Using Model Compression of Deep Neural Networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper With Convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[40] S. Gu, X. Chen, W. Zeng, and X. Wang, “A Deep Learning Tennis Ball Collection Robot and the Implementation on NVIDIA Jetson TX1 Board,” in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 170–175, 2018.

[41] I. Sa, Z. Chen, M. Popović, R. Khanna, F. Liebisch, J. Nieto, and R. Siegwart, “weednet:

Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 588–595, 2018.

[42] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2015.

[43] "Intel® Core™ i5-2450M Processor (3M Cache, up to 3.10 GHz) - Product Specifications." `https://www.intel.com/content/www/us/en/products/sku/53452/intel-core-i52450m-processor-3m-cache-up-to-3-10-ghz/specifications.html`. Accessed on 2021-10-18.

[44] "GeForce 610M | Specifications | GeForce." `https://www.nvidia.com/en-gb/geforce/gaming-laptops/geforce-610m/specifications/`. Accessed on 2021-10-18.

[45] Alexey, "AlexeyAB/darknet." `https://github.com/AlexeyAB/darknet`. Accessed on 2021-05-28.

[46] "NVIDIA GeForce 10 Series Graphics Cards." `https://www.nvidia.com/en-eu/geforce/10-series/`. Accessed on 2021-12-17.

[47] "NVIDIA Volta AI Architecture." `https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/`. Accessed on 2021-11-04.

[48] "NVDLA Primer — NVDLA Documentation." `http://nvdla.org/primer.html`. Accessed on 2021-11-04.

[49] "NVIDIA cuDNN | NVIDIA Developer." `https://developer.nvidia.com/cudnn`. Accessed on 2021-11-04.

[50] J. Tread, "cloudharmony/unixbench." `https://github.com/cloudharmony/unixbench`, Feb. 2021.

[51] "NVIDIA Jetson Linux Developer Guide : Clock frequency and power management | NVIDIA docs." `https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-325/index.html`. Accessed on 2021-05-31.

[52] I. Cvišić, I. Marković, and I. Petrović, "Recalibrating the KITTI Dataset Camera Setup for Improved Odometry Accuracy," in *European Conference on Mobile Robots (ECMR)*, 2021.

[53] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in *Proc. of the National Conference on Artificial Intelligence*, 1999.

[54] R. Hedges, "AriaCoda." `https://github.com/reedhedges/AriaCoda`, July 2021. Accessed on 2021-09-09.

[55] "OMRON to acquire u.s. based adept technology | OMRON global." `https://www.omron.com/global/en/media/press/2015/09/c0916.html`. Accessed on 2021-06-01.

[56] D. S. B. Amorim, "Seguimento de um humano por um robô companheiro | Estudo Geral." `http://hdl.handle.net/10316/41250`.

[57] D. Portugal and R. P. Rocha, "Performance estimation and dimensioning of team size for multirobot patrol," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 30–38, 2017.

[58] R. P. Rocha, D. Portugal, M. Couceiro, F. Araújo, P. Menezes, and J. Lobo, "The chopin project: Cooperation between human and robotic teams in catastrophic incidents," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–4, 2013.

[59] J. M. Santos, M. S. Couceiro, D. Portugal, and R. P. Rocha, "A Sensor Fusion Layer to Cope with Reduced Visibility in SLAM," *Journal of Intelligent & Robotic Systems*, vol. 80, pp. 401–422, Dec. 2015.

[60] "ZED 2 - AI stereo camera | stereolabs." `https://www.stereolabs.com/zed-2/`. Accessed on 2021-06-01.

[61] "ZED stand by rbonghi - Thingiverse." `https://www.thingiverse.com/thing:3208903`. Accessed on 2021-09-09.

[62] "JetPack SDK 4.4 archive | NVIDIA Developer." `https://developer.nvidia.com/jetpack-sdk-44-archive`. Accessed on 2021-08-18.

[63] "Darkhelp." `https://www.ccoderun.ca/darkhelp/api/index.html`. Accessed on 2021-08-18.

[64] S. Charette, "What is DarkMark?." `https://github.com/stephanecharette/DarkMark`, Sept. 2021. Accessed on 2021-10-03.

[65] “Mask Dataset.” https://makeml.app/datasets/mask. Accessed on 2021-10-8.

[66] “NVIDIA TensorRT.” https://developer.nvidia.com/tensorrt, Apr. 2016. Accessed on 2021-11-23.

[67] “Open Neural Network Exchange.” https://github.com/onnx. Accessed on 2021-11-23.

[68] “Docker Hub.” https://hub.docker.com/repository/docker/pedrr/isr_xavier. Accessed on 2021-11-23.

[69] “CUDA GPUs.” https://developer.nvidia.com/cuda-gpus, June 2012. Accessed on 2021-12-20.

Appendix A

Electrical Requirements

The Jetson AGX Xavier must be powered with a voltage level between 9 and 19 V and without voltage fluctuations over 0.5 V/s. The Pioneer P3-DX mobile platform is powered by a set of three 12 V lead-acid batteries and is responsible to power the platform's micro-controller and to power the various sensors and actuators, such as motors. This is not a constant load on the battery pack as not only the motors are not guaranteed to be powered at a constant rate over a long period time but motors are also an inductive load, meaning that the powering voltage will suffer fluctuations over time. Consequently, despite the 12 V provided by the mobile platform being of sufficient value, the characteristics of the batteries and the load they power make it impossible for them to be used to power the Jetson AGX Xavier as well.

To solve this issue, two 12 V lead-acid batteries were acquired. By connecting the batteries in parallel, a dedicated 24 V signal is obtained and then modulated down to 19 V via a DC-DC converter. However, these batteries are not regulated and as such a sudden change, such as the sudden start of multiple processes responsible for the booting up of the Jetson AGX Xavier, in the output load will cause fluctuations on the voltage source. An electrolytic 4.7 mF and a 10 μF polyester dielectric capacitor were connected in parallel with the Jetson AGX Xavier's 2.5mm x 5.5mm barrel connector in order to reduce high frequency fluctuations in the voltage source due to sudden load changes. The diagram is represented in Figure A.1 and its circuit diagram in Figure A.2.

As lead-acid batteries have considerable weight, a different approach should be considered in the future. It is suggested to investigate the usage of Lithium Polymer, or LiPo, batteries with 5 cells as it is possible to obtain around 18 V of voltage but with a significantly lower

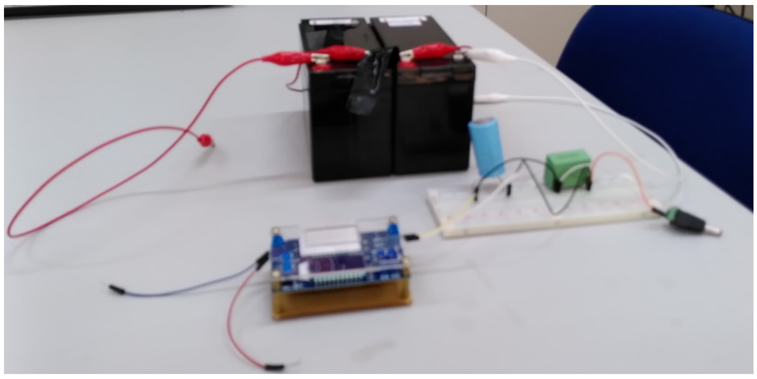


Figure A.1: Electrical montage for portable power source for the Jetson AGX Xavier.

mass and form factor.

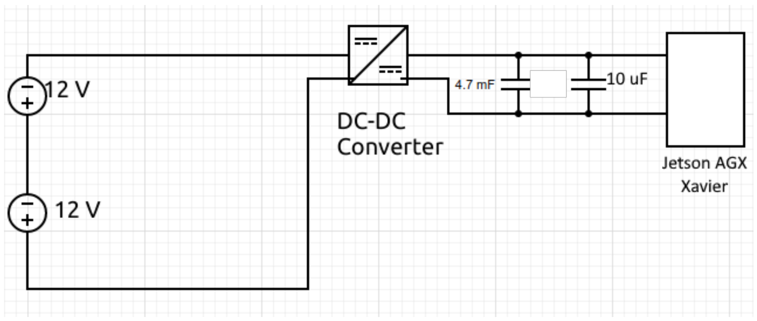


Figure A.2: Electrical diagram for circuit powering the Jetson AGX Xavier.

Appendix B

System Dockerfile

A Docker container is an operating system level virtualization that allows the deployment of software in a standardized environment. Over the course of this dissertation a Docker container was constructed in order to encapsulate the operation of the Pioneer P3-DX platform by a system with an ARM64 architecture. However, it must be noted that the AriaCoda library is an open-source project still in development. Consequently for each future release of this software, the Dockerfile from which this Docker container is constructed should be revisited and edited accordingly.

It would also be interesting to encapsulate the ZED 2 SDK and the developed TinyYOLOv4 model. However, this would require the installation of OpenCV 4.3 (or subsequent releases) in the Docker environment. As OpenCV is a heavy investment in storage memory, around 4 GB, to have it installed in two environments (in the Docker container and in the base Jetson AGX Xavier system) is not considered to be worth the investment given that storage memory is a limiting factor of the Jetson AGX Xavier as it has 32 GB of storage memory.

To use this Docker container, you should first pull it from Docker Hub. Open a terminal and input:

```
$ Docker pull pedrr/isr_xavier:1.1
```

After which, you can start the container with the following bash command:

```
$ Docker run -it --rm --net=host --priviliged --runtime nvidia -e
  DISPLAY=$DISPLAY --device=/dev/ttyUSB0 -v /tmp/.X11-unix/:/tmp/.X11-
  unix pedrr/isr_xavier:1.1
```

Here is a brief explanation of the command:

- The -it flag starts the container in interactive mode,
- the –rm flag removes the container after it is stopped,
- –net=host allows the container to share the host's network namespace,
- the –priviliged flag concedes root permissions to the container,
- the –runtime nvidia option allows the use of NVIDIA GPUs,
- –device=/dev/ttyUSB0 connects the Pioneer P3-DX's microcontroller to the container via USB

Then you must install the AriaCoda library:

```
cd src/AriaCoda && make install -j8
```

As of time of writing, the "make install" rule fails to complete successfully, it is possible that future releases of the AriaCoda library will solve the issue and the installation can be fully integrated in the Dockerfile. In its current state, despite the error in the installation process, it is possible to operate the Pioneer P3-DX mobile platform. You can now build the ROS environment:

```
cd /workspace && catkin_make
```

After constructing the ROS environment, you can start the RosAria and teleoperation stack:

```
roslaunch mrl_pioneer pioneer_wii.launch
```

The roslaunch process in the Docker container can be connected to by other ROS nodes outside the container. As such, in a different terminal, you can launch the ZED 2 and the DL-based application's ROS nodes:

```
roslaunch zed_wrapper zed2.launch
```

```
rosrun cv_node cv_node
```

Appendix C

How to develop a custom YOLO model

This appendix aims to provide a tutorial to the installation of the Darknet framework, the DarkHelp helper library and the DarkMark application to generate the necessary files for custom training. This tutorial assumes a Linux system.

C.1 Installing Darknet, DarkHelp, DarkMark

Darknet is currently hosted on a GitHub repository and maintained, as of time of writing, by Alexey Bochkovskiy: https://github.com/AlexeyAB/darknet. The requirements are as follows:

- CMake version 3.18 or more recent,
- CUDA version 10.2 or more recent,
- OpenCV version 2.4 or more recent,
- cuDNN version 8.0.2 or more recent,
- a GPU with compute capabilities 3.0 or better [69]
- Git installed.

Installation of Darknet is as simple as pulling the Darknet repository to a source folder and editing the Makefile for the appropriate flags.

```
mkdir ~/src && cd ~/src
git clone https://github.com/AlexeyAB/darknet
cd darknet
```

```
vim Makefile # Use whatever IDE you'd like to set the GPU, CUDNN,
    CUDNN_HALF, OPENCV and LIBSO flags to 1
make
sudo cp libdarknet.so /usr/local/lib
sudo cp include/darknet.h /usr/local/include
sudo ldconfig
```

To check correct installation, try running the darknet executable file:

```
    ./darknet
```

The output should be:

```
   usage: ./darknet <function>
```

DarkHelp and DarkMark are two helper library and applications for Darknet. As documentation and examples for the original C Darknet are scarce, these libraries are very helpful to integrate YOLO-based models into a custom C++ application or to custom train a model.

https://www.ccoderun.ca/DarkHelp/api/Building.html provides the instructions to install DarkHelp and are relayed to here:

```
cd ~/src
git clone https://github.com/stephanecharette/DarkHelp.git
cd DarkHelp
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
make package
sudo dpkg -i darkhelp-*.deb
```

https://www.ccoderun.ca/darkmark/Building.html provides instructions to install DarkMark, a helper tool to annotate and verify training data.

```
cd ~/src
git clone https://github.com/stephanecharette/DarkMark.git
cd DarkMark
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
```

```
make
make package
sudo dpkg -i darkmark*.deb
```

C.2 Start training

After annotating and verifying the training data with DarkMark it is possible to generate scripts that will initiate GPU training of the model. The Darknet GitHub repository offers the following instructions:

- Change batch to 64,
- change subdivisions to 16,
- change max_batches 2000 times the number of classes but not under the number of training images and at least 6000,
- set network size, width and height, to $416, 416$ or any multiple of 32.

After executing the scripts generated by DarkMark, model training will begin. The model can then be integrated in a C++ application via DarkHelp.

C.3 Tutorial Example

For tutorial purposes, a YOLOv4 model to detect a Magic: The Gathering card will be trained. A trivial dataset was aggregated by recording a short video of the object to be detected: a Lightning Bolt card from the trading card game Magic: The Gathering. 50 frames from the video recording were annotated using DarkMark.

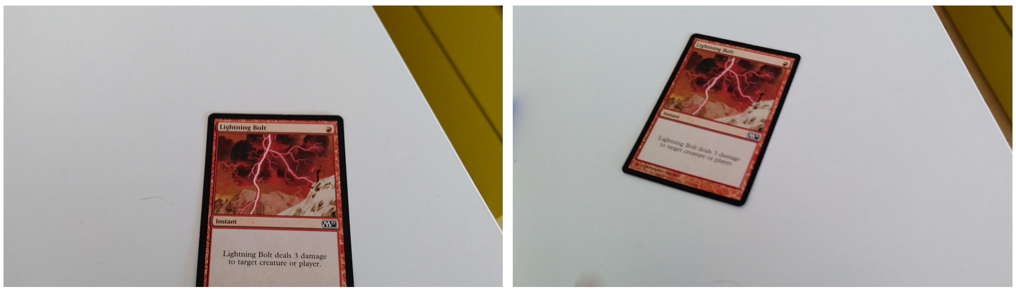


Figure C.1: Training dataset images to be annotated.

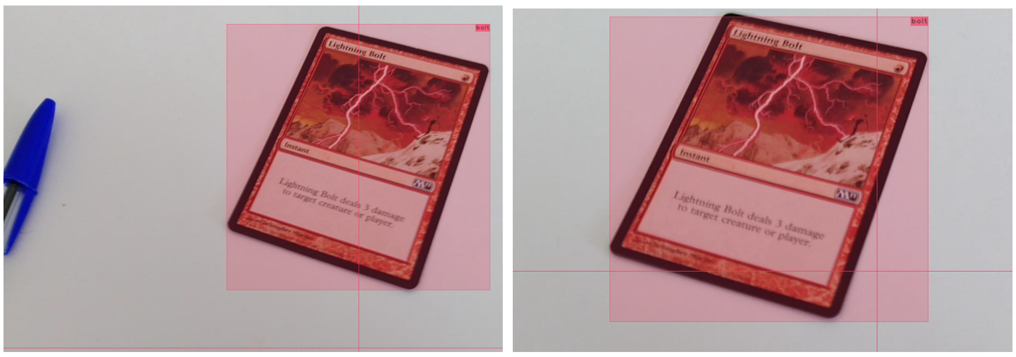


Figure C.2: Annotated training dataset images.

After annotating the dataset, right-click and select “Create darknet files” and define training hyperparameters. In this tutorial example, the instructions in the Darknet Github repository, described above, were followed. Training can begin by executing the bash command:

```
./bolt_training.sh
```

After training, the model can be implemented in a C++ program with the use of the DarkHelp library. Here follows a program that takes a USB camera video feed as the input to the trained neural network and displays the results in a OpenCV window.

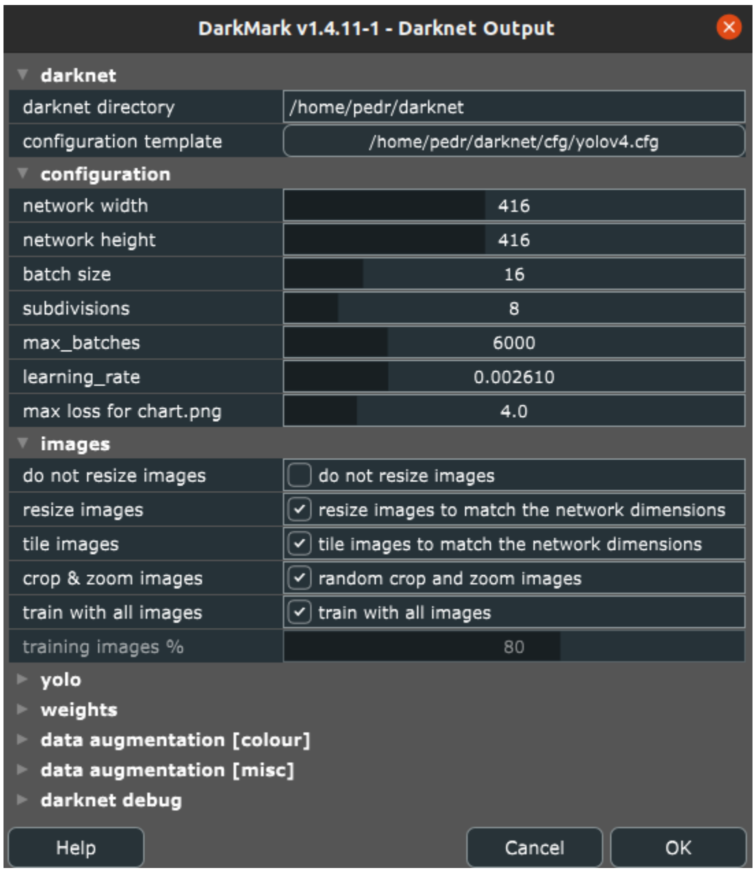
```
#include <DarkHelp.hpp>
#include <iostream>


int main(int argc, char *argv[]) {
  DarkHelp dh_tinyyolo("bolt.cfg", "bolt_final.weights", "bolt.names");
  cv::Mat mat, dst;
  cv::Size size(416, 416);

  cv::VideoCapture camera(0);
  cv::Mat img_capture;

  if (!camera.isOpened()) {
    std::cout << "Camera isn't open.\n";
    return -1;
  }

  while (1) {
    camera >> img_capture;
    auto results = dh_tinyyolo.predict(img_capture);
    mat = dh_tinyyolo.annotate();
```



darknet

darknet directory	/home/pedr/darknet
configuration template	/home/pedr/darknet/cfg/yolov4.cfg

configuration

network width	416
network height	416
batch size	16
subdivisions	8
max_batches	6000
learning_rate	0.002610
max loss for chart.png	4.0

images

do not resize images	☐ do not resize images
resize images	✓ resize images to match the network dimensions
tile images	✓ tile images to match the network dimensions
crop & zoom images	✓ random crop and zoom images
train with all images	✓ train with all images
training images %	80

yolo

weights

data augmentation [colour]

data augmentation [misc]

darknet debug

Figure C.3: Training hyperparameters as defined in DarkMark.

```
    cv::resize(mat, dst, size, 0, 0, cv::INTER_LINEAR);
    cv::imshow("Prediction", dst);
    cv::waitKey(20);
  }
}
```

To build this program, the OpenCV library, libdarknet.so and libdarkhelp.so must be linked. The following "CMakeLists.txt" file generates the necessary Makefile to build an executable named "bolt" in a folder named "darkhelptest":

```
CMAKE_MINIMUM_REQUIRED (VERSION 3.0)

PROJECT (darkhelptest C CXX)

SET (CMAKE_BUILD_TYPE Release)
SET (CMAKE_CXX_STANDARD 17)
SET (CMAKE_CXX_STANDARD_REQUIRED ON)

ADD_DEFINITIONS ("-Wall -Wextra -Werror -Wno-unused-parameter")

```

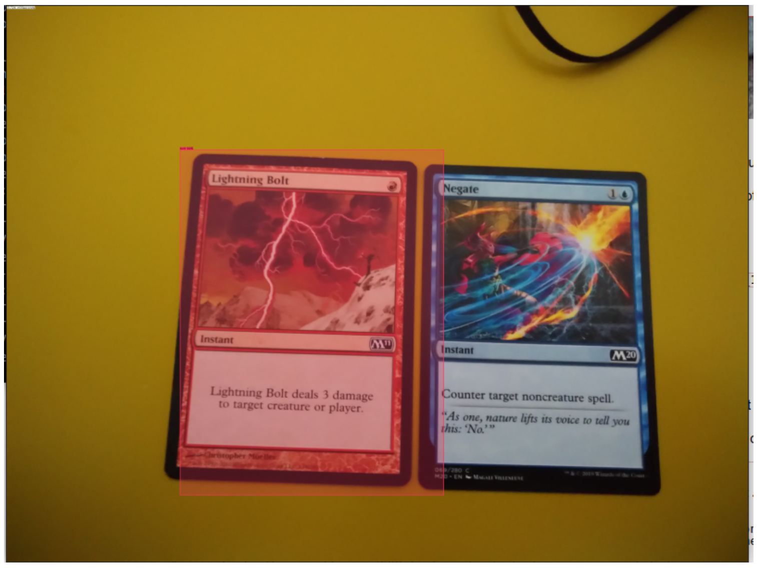

Lightning Bolt
Instant
Lightning Bolt deals 3 damage to target creature or player.
Negate
Instant
Counter target noncreature spell.
"As one, nature lifts its voice to tell you this: 'No.'"

Figure C.4: Test of the trained tutorial Yolov4 model.

```
FIND_PACKAGE (Threads    REQUIRED)
FIND_PACKAGE (OpenCV    REQUIRED)
FIND_LIBRARY (DARKHELP    darkhelp)
FIND_LIBRARY (DARKNET    darknet    )

INCLUDE_DIRECTORIES (${OpenCV_INCLUDE_DIRS})

FILE (GLOB SOURCE *.cpp)
LIST (SORT SOURCE)

ADD_EXECUTABLE ( bolt ${SOURCE})
TARGET_LINK_LIBRARIES (bolt Threads::Threads ${DARKHELP} ${DARKNET} ${
    OpenCV_LIBS})
```

Execute the command

```
cmake .
```

in the folder containing "CMakeLists.txt". Then execute

```
make
```

to build the "bolt" executable file. Test the program by executing it with

```
./bolt
```

Appendix D

Results of custom TinyYOLOv4 model

Here follows some examples of the results obtained via the custom trained TinyYOLOv4 neural network model for detection of facial masks. It can be asserted that the color of the facial mask, environmental lighting, facial hair among other factors can influence the results of the model and cause erroneous predictions.

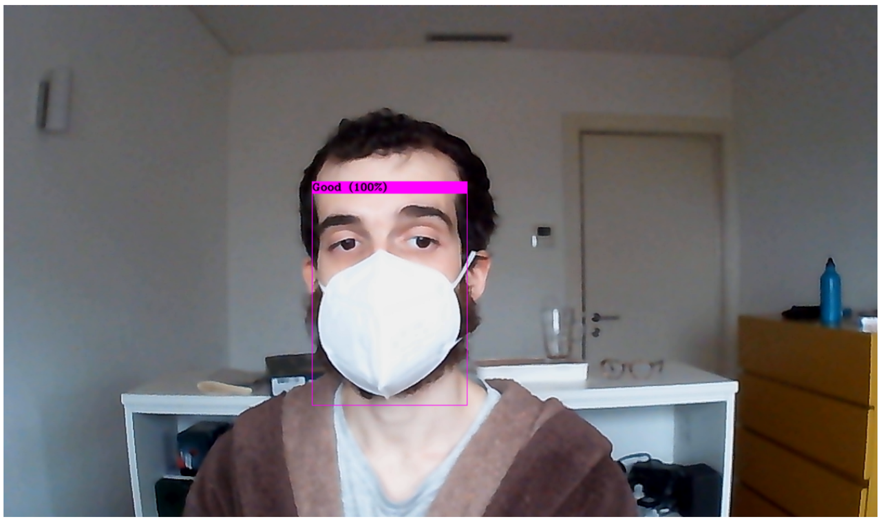


Figure D.1: The model predicts a masked face.

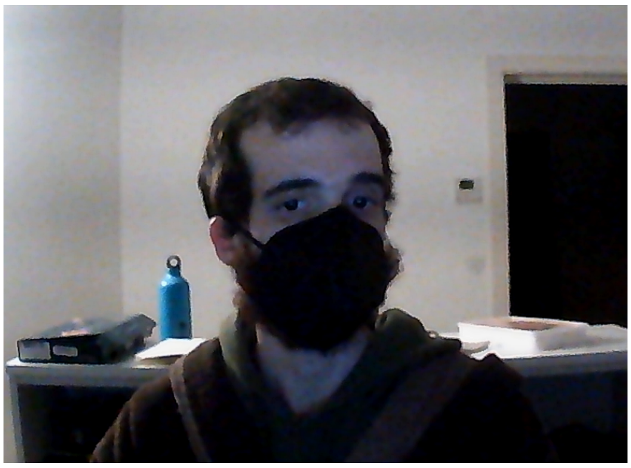


Figure D.2: The model does not predict a correctly worn facial mask, possibly due to its color.

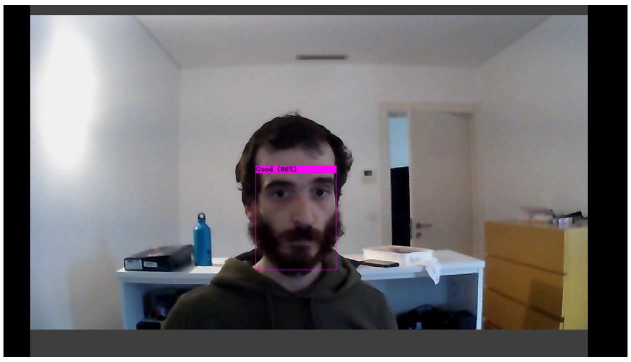


Figure D.3: The model wrongly predicts a correctly worn facial mask due to the subject's facial hair with a confidence score of 86%.

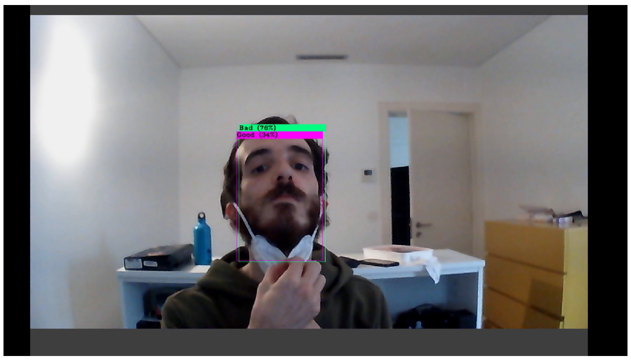


Figure D.4: The model predicts two overlapped faces one with a facial mask and one without a facial mask with confidence scores 34% and 76%, respectively.